

COP 4610L: Applications in the Enterprise Fall 2006

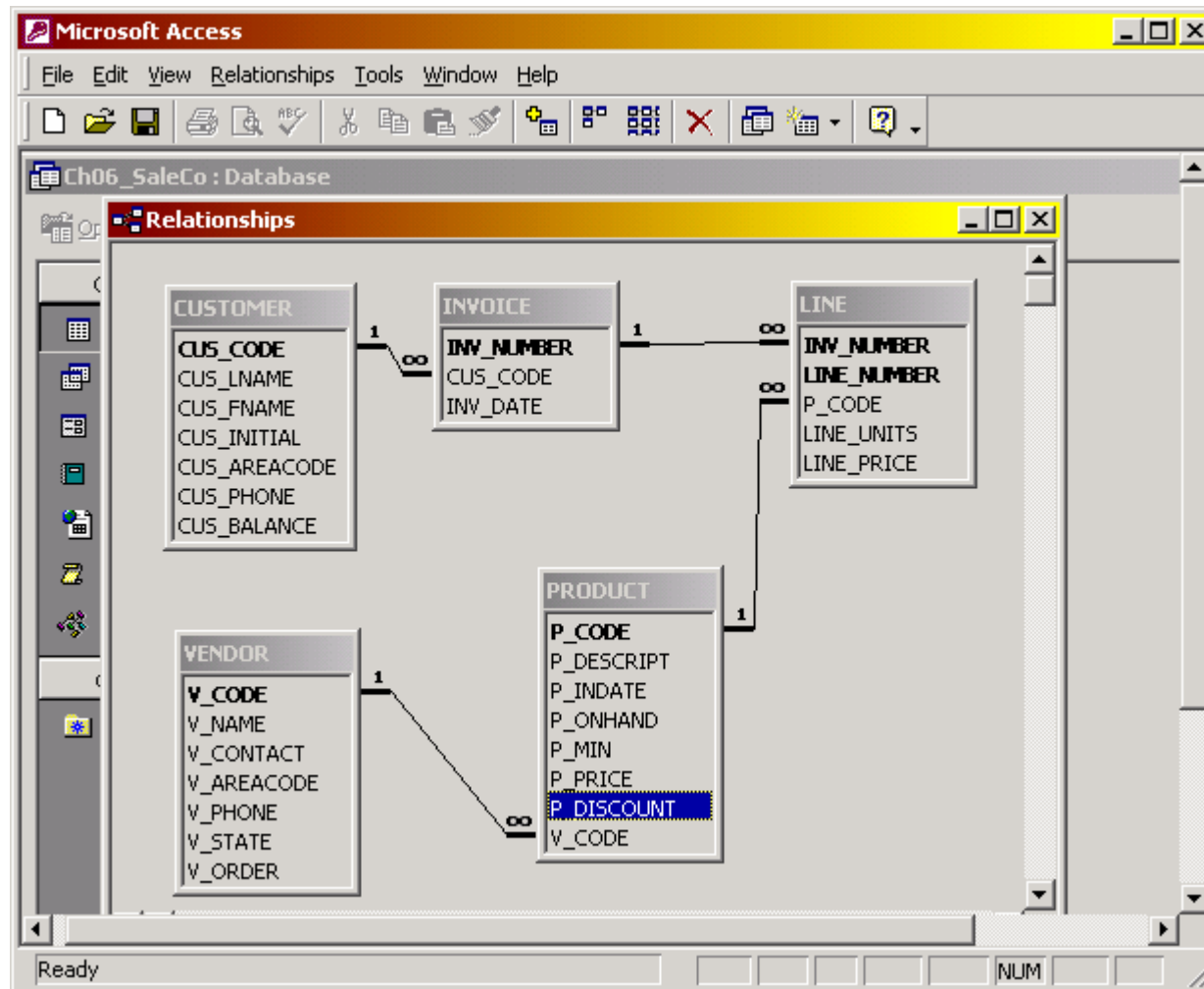
Introduction To SQL – Part 2

Instructor : Mark Llewellyn
markl@cs.ucf.edu
CSB 242, 823-2790
<http://www.cs.ucf.edu/courses/cop4610/fall2006>

School of Electrical Engineering and Computer Science
University of Central Florida



An Example Database



Advanced SELECT Queries

- One of the most important advantages of SQL is its ability to produce complex free-form queries.
- The logical operators that were illustrated in the last set of notes work just as well in the query environment.
- In addition, SQL provides useful functions that count, find minimum and maximum values, calculate averages, and so on.
- Even better, SQL allows the user to limit queries to only those entries having no duplicates or entries whose duplicates may be grouped.
- We'll illustrate several of these features over the next few pages.



Ordering A Listing

- The ORDER BY clause is especially useful if the listing order is important to you. T

- The syntax is:

```
SELECT columnlist  
FROM tablelist  
[ WHERE conditionlist ]  
[ ORDER BY columnlist [ASC | DESC] ] ;
```

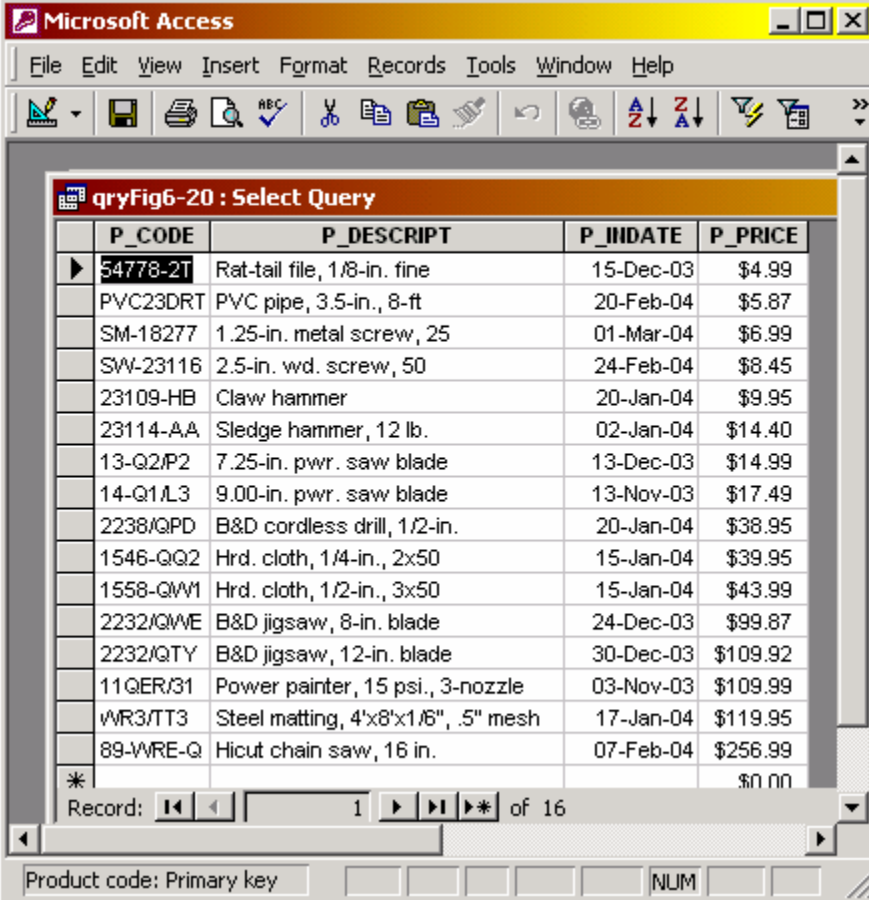
- If the ordering column contains nulls, they are either listed first or last depending on the RDBMS.
- The ORDER BY clause must always be listed last in the SELECT command sequence.
- Although you have the option of specifying the ordering type, either ascending or descending – the default order is ascending.



Ordering A Listing (cont.)

- The query shown below lists the contents of the PRODUCT table listed by P_PRICE in ascending order:

```
SELECT
    P_CODE, P_DESCRIPT,
    P_INDATE, P_PRICE
FROM PRODUCT
ORDER BY P_PRICE;
```



The screenshot shows the Microsoft Access interface with a query window titled 'qryFig6-20 : Select Query'. The query results are displayed in a table with the following columns: P_CODE, P_DESCRIPT, P_INDATE, and P_PRICE. The records are sorted by P_PRICE in ascending order. The first record is highlighted.

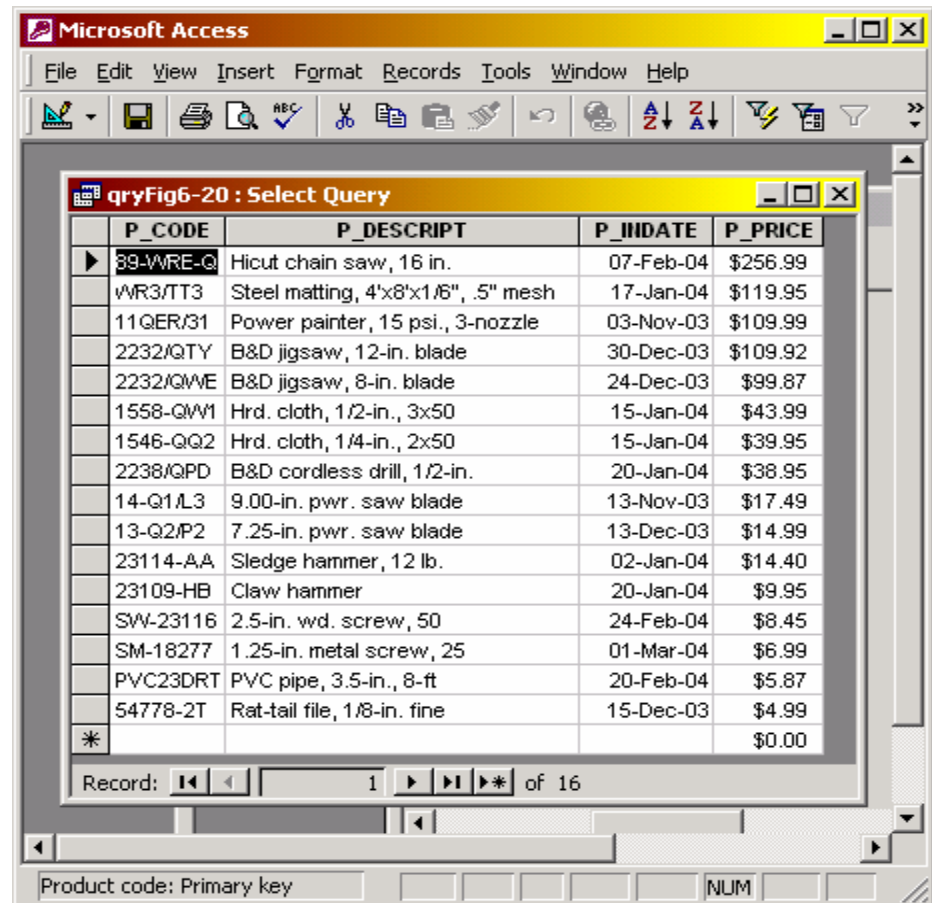
P_CODE	P_DESCRIPT	P_INDATE	P_PRICE
54778-2T	Rat-tail file, 1/8-in. fine	15-Dec-03	\$4.99
PVC23DRT	PVC pipe, 3.5-in., 8-ft	20-Feb-04	\$5.87
SM-18277	1.25-in. metal screw, 25	01-Mar-04	\$6.99
SW-23116	2.5-in. wd. screw, 50	24-Feb-04	\$8.45
23109-HB	Claw hammer	20-Jan-04	\$9.95
23114-AA	Sledge hammer, 12 lb.	02-Jan-04	\$14.40
13-Q2/P2	7.25-in. pwr. saw blade	13-Dec-03	\$14.99
14-Q1/L3	9.00-in. pwr. saw blade	13-Nov-03	\$17.49
2238/QPD	B&D cordless drill, 1/2-in.	20-Jan-04	\$38.95
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15-Jan-04	\$39.95
1558-QW1	Hrd. cloth, 1/2-in., 3x50	15-Jan-04	\$43.99
2232/QWE	B&D jigsaw, 8-in. blade	24-Dec-03	\$99.87
2232/QTY	B&D jigsaw, 12-in. blade	30-Dec-03	\$109.92
11QER/31	Power painter, 15 psi., 3-nozzle	03-Nov-03	\$109.99
WR3/TT3	Steel matting, 4'x8'x1/6", .5" mesh	17-Jan-04	\$119.95
89-WRE-Q	Hicut chain saw, 16 in.	07-Feb-04	\$256.99
*			\$0.00



Ordering A Listing (cont.)

- The query shown below lists the contents of the PRODUCT table listed by P_PRICE in descending order:

```
SELECT
    P_CODE, P_DESCRIPT,
    P_INDATE, P_PRICE
FROM PRODUCT
ORDER BY P_PRICE DESC;
```



The screenshot shows the Microsoft Access interface with a query window titled 'qryFig6-20 : Select Query'. The query results are displayed in a table with the following columns: P_CODE, P_DESCRIPT, P_INDATE, and P_PRICE. The records are sorted in descending order of P_PRICE. The first record is highlighted with a mouse cursor.

P_CODE	P_DESCRIPT	P_INDATE	P_PRICE
89-WRE-Q	Hicut chain saw, 16 in.	07-Feb-04	\$256.99
WR3/TT3	Steel matting, 4'x8'x1/6", .5" mesh	17-Jan-04	\$119.95
11QER/31	Power painter, 15 psi., 3-nozzle	03-Nov-03	\$109.99
2232/QTY	B&D jigsaw, 12-in. blade	30-Dec-03	\$109.92
2232/QWE	B&D jigsaw, 8-in. blade	24-Dec-03	\$99.87
1558-QW1	Hrd. cloth, 1/2-in., 3x50	15-Jan-04	\$43.99
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15-Jan-04	\$39.95
2238/QPD	B&D cordless drill, 1/2-in.	20-Jan-04	\$38.95
14-Q1/L3	9.00-in. pwr. saw blade	13-Nov-03	\$17.49
13-Q2/P2	7.25-in. pwr. saw blade	13-Dec-03	\$14.99
23114-AA	Sledge hammer, 12 lb.	02-Jan-04	\$14.40
23109-HB	Claw hammer	20-Jan-04	\$9.95
SWV-23116	2.5-in. wd. screw, 50	24-Feb-04	\$8.45
SM-18277	1.25-in. metal screw, 25	01-Mar-04	\$6.99
PVC23DRT	PVC pipe, 3.5-in., 8-ft	20-Feb-04	\$5.87
54778-2T	Rat-tail file, 1/8-in. fine	15-Dec-03	\$4.99
*			\$0.00

Record: 1 of 16

Product code: Primary key NUM



Cascading Order Sequences

- Ordered listings are used frequently. For example, suppose you want to create a phone directory of employees. It would be helpful if you could produce an ordered sequence (last name, first name, middle initial) in three stages:
 1. ORDER BY last name.
 2. Within last names, ORDER BY first name.
 3. Within the order created in Step 2, ORDER BY middle initial.
- A multi-level ordered sequence is called a **cascading order sequence**, and is easily created by listing several attributes, separated by commas, after the ORDER BY clause.
- This concept is illustrated in the next couple of slides.



Cascading Order Sequences (cont.)

Microsoft Access

File Edit View Insert Format Records Tools Window Help

EMPLOYEE : Table

EMP_NUM	EMP_TITLE	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_DOB	EMP_HIRE_DATE	EMP_YEARS	EMP_AREACODE	EMP_PHONE
100	Mr.	Kolmycz	George	D	15-Jun-42	15-Mar-85	18	615	324-5456
101	Ms.	Lewis	Rhonda	G	19-Mar-65	25-Apr-86	16	615	324-4472
102	Mr.	Vandam	Rhett		14-Nov-58	20-Dec-90	12	901	675-8993
103	Ms.	Jones	Anne	M	16-Oct-74	28-Aug-94	8	615	898-3456
104	Mr.	Lange	John	P	08-Nov-71	20-Oct-94	8	901	504-4430
105	Mr.	Williams	Robert	D	14-Mar-75	08-Nov-98	4	615	890-3220
106	Mrs.	Smith	Jeanine	K	12-Feb-68	05-Jan-89	14	615	324-7883
107	Mr.	Diante	Jorge	D	21-Aug-74	02-Jul-94	8	615	890-4567
108	Mr.	Wiesenbach	Paul	R	14-Feb-66	18-Nov-92	10	615	897-4358
109	Mr.	Smith	George	K	18-Jun-61	14-Apr-89	13	901	504-3339
110	Mrs.	Genkazi	Leighla	W	19-May-70	01-Dec-90	12	901	569-0093
111	Mr.	Washington	Rupert	E	03-Jan-66	21-Jun-93	9	615	890-4925
112	Mr.	Johnson	Edward	E	14-May-61	01-Dec-83	19	615	898-4387
113	Ms.	Smythe	Melanie	P	15-Sep-70	11-May-99	3	615	324-9006
114	Ms.	Brandon	Marie	G	02-Nov-56	15-Nov-79	23	901	882-0845
115	Mrs.	Saranda	Hermine	R	25-Jul-72	23-Apr-93	9	615	324-5505
116	Mr.	Smith	George	A	08-Nov-65	10-Dec-88	14	615	890-2984
*	0						0		

Record: 1 of 17

Employee number. (Primary key)

Employee Table

NUM



Cascading Order Sequences (cont.)

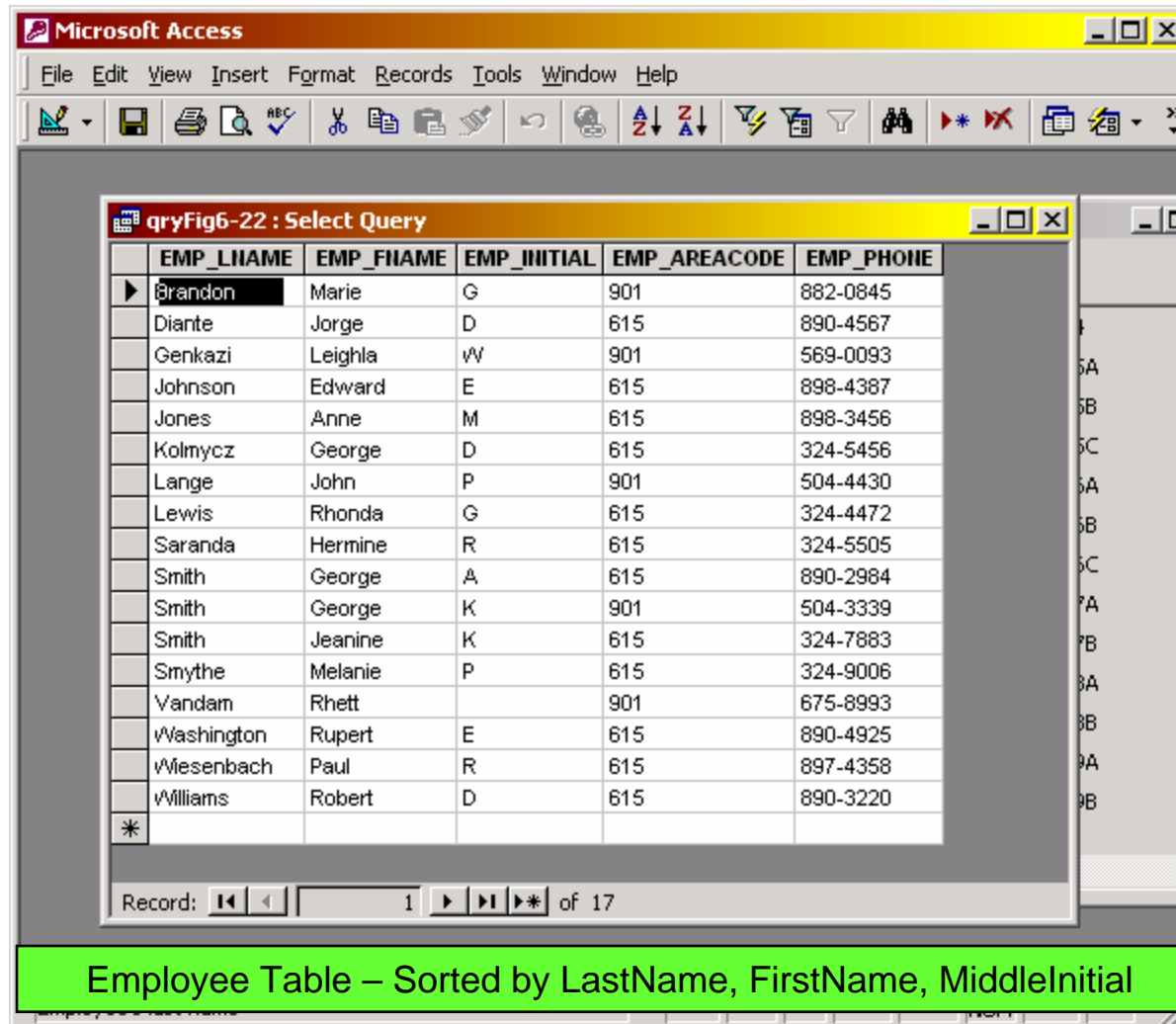
- To create the phonebook type ordering from the EMPLOYEE table, we can execute the following SQL query:

```
SELECT EMP_LNAME, EMP_FNAME, EMP_INITIAL, EMP_AREACODE, EMP_PHONE  
FROM EMPLOYEE  
ORDER BY EMP_LNAME, EMP_FNAME, EMP_INITIAL;
```

- This query would produce the result shown on the next slide.



Cascading Order Sequences (cont.)



The screenshot shows a Microsoft Access window titled "Microsoft Access" with a menu bar (File, Edit, View, Insert, Format, Records, Tools, Window, Help) and a toolbar. A query window titled "qryFig6-22 : Select Query" is open, displaying a table with the following columns: EMP_LNAME, EMP_FNAME, EMP_INITIAL, EMP_AREACODE, and EMP_PHONE. The table contains 17 rows of employee data, sorted by last name, first name, and middle initial. The first row is highlighted. The status bar at the bottom of the query window indicates "Record: 1 of 17".

EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_AREACODE	EMP_PHONE
Brandon	Marie	G	901	882-0845
Diante	Jorge	D	615	890-4567
Genkazi	Leighla	vW	901	569-0093
Johnson	Edward	E	615	898-4387
Jones	Anne	M	615	898-3456
Kolmycz	George	D	615	324-5456
Lange	John	P	901	504-4430
Lewis	Rhonda	G	615	324-4472
Saranda	Hermine	R	615	324-5505
Smith	George	A	615	890-2984
Smith	George	K	901	504-3339
Smith	Jeanine	K	615	324-7883
Smythe	Melanie	P	615	324-9006
Vandam	Rhett		901	675-8993
Washington	Rupert	E	615	890-4925
Wiesenbach	Paul	R	615	897-4358
Williams	Robert	D	615	890-3220
*				

Record: 1 of 17

Employee Table – Sorted by LastName, FirstName, MiddleInitial



Additional Uses of the ORDER BY Clause

- You can use the ORDER BY clause in conjunction with other SQL commands as well.
- For example, note the use of restrictions on date and price in the following command sequence:

```
SELECT P_DESCRIPT, V_CODE, P_INDATE, P_PRICE  
FROM PRODUCT  
WHERE P_INDATE < '21-Jan-2004' AND P_PRICE <= 50.00  
ORDER BY V_CODE, P_PRICE DESC;
```

- The result of this query is shown on the next slide:



Additional Uses of the ORDER BY Clause (cont.)

P_DESCRIPT	V_CODE	P_IIDATE	P_PRICE
Sledge hammer, 12 lb.		02-Jan-04	\$14.40
Claw hammer	21225	20-Jan-04	\$9.95
9.00-in. pwr. saw blade	21344	13-Nov-03	\$17.49
7.25-in. pwr. saw blade	21344	13-Dec-03	\$14.99
Rat-tail file, 1/8-in. fine	21344	15-Dec-03	\$4.99
Hrd. cloth, 1/2-in., 3x50	23119	15-Jan-04	\$43.99
Hrd. cloth, 1/4-in., 2x50	23119	15-Jan-04	\$39.95
B&D cordless drill, 1/2-in.	25595	20-Jan-04	\$38.95
*	0		\$0.00

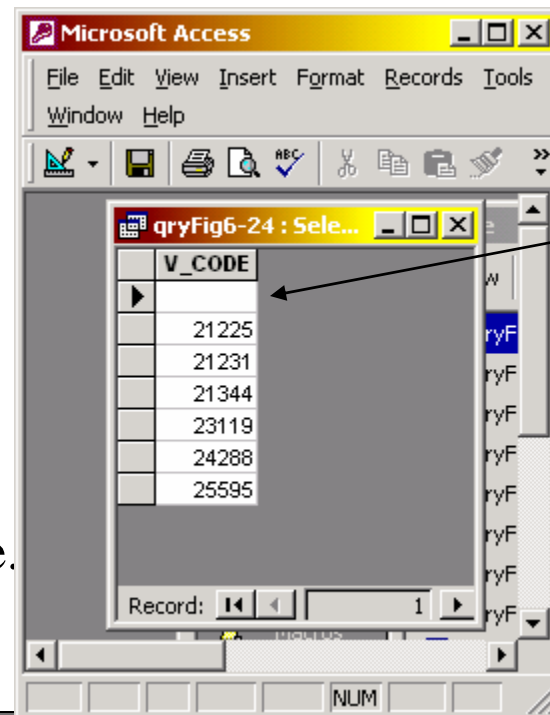


Listing Unique Values

- How many different vendors are currently represented in the PRODUCT table? A simple listing (SELECT command) is not very useful in answering this query, particularly if the table contained several thousand rows and we would have to manually sift out the vendor codes.
- Fortunately, SQL's DISTINCT clause is designed to produce a list of only those values that are different from one another.
- For example, the command:

```
SELECT DISTINCT V_CODE  
FROM PRODUCT;
```

will yield on the different (distinct) vendor codes (V_CODE) that are encountered in the PRODUCT table.



Oracle puts the null V_CODE at the bottom of the list while Access will put it at the top. You can, of course, using the ORDER BY clause.



Grouping Results

- Frequency distributions can be created quickly and easily using the GROUP BY clause within the SELECT statement.

- The syntax is:

```
SELECT columnlist
FROM tablelist
[WHERE conditionlist ]
[GROUP BY columnlist ]
[HAVING condtionlist ]
[ORDER BY columnlist [ASC | DESC] ] ;
```

- The GROUP BY clause is generally used when you have attribute columns combined with aggregate functions in the SELECT statement.
- For example, to determine the minimum price for each sales code, use the following statement shown on the next page.



Grouping Results (cont.)

- The query is:

```
SELECT P_SALECODE, MIN(P_PRICE)
FROM PRODUCT
GROUP BY P_SALECODE;
```

P_SALECODE	Expr1001
	\$50.23
1	\$104.90
2	\$39.95



Grouping Results (cont.)

- When using the GROUP BY clause with a SELECT statement, the following rules must be observed:
 1. The SELECT's *columnlist* must include a combination of column names and aggregate functions.
 2. The GROUP BY clause's *columnlist* must include all non-aggregate function columns specified in the SELECT's *columnlist*. If required, you could also group by any aggregate function columns that appear in the SELECT's *columnlist*.
 3. The GROUP BY clause *columnlist* can include any column from the tables in the FROM clause of the SELECT statement, even if they do not appear in the SELECT's *columnlist*.



The GROUP BY Feature's HAVING Clause

- A particularly useful extension of the GROUP BY clause is the HAVING clause.
- Basically, HAVING operates like the WHERE clause in the SELECT statement. However, the WHERE clause applies to columns and expressions for individual rows, while the HAVING clause is applied to the output of a GROUP BY operation.
- For example, suppose you want to generate a listing of the number of products in the inventory supplied by each vendor, but you want to limit the listing to the products whose prices average below \$10.00. The first part of this requirement is satisfied with the help of the GROUP BY clause, the second part of the requirement will be accomplished with the HAVING clause.
- The complete query and results are shown on the next page.



The GROUP BY Feature's HAVING Clause (cont.)

```
SELECT PRODUCT_2.V_CODE, Count(PRODUCT_2.P_CODE) AS CountOfP_CODE, Avg(PRODUCT_2.P_PRICE) AS AvgOfP_PRICE
FROM PRODUCT_2
GROUP BY PRODUCT_2.V_CODE
HAVING (((Avg(PRODUCT_2.P_PRICE))<10));
```

The query

V_CODE	Expr1001	Expr1002
21225	2	\$8.47
21231	1	\$8.45

The results



Virtual Tables: Creating Views

- Recall that the output of a relational operator (like SELECT in SQL) is another relations (or table).
- Using our sample database as an example, suppose that at the end of each business day, we would like to get a list of all products to reorder, which is the set of all products whose quantity on hand is less than some threshold value (minimum quantity).
- Rather than typing the same query at the end of every day, wouldn't it be better to permanently save that query in the database?
- To do this is the function of a relational view. In SQL a **view** is a table based on a SELECT query. That query can contain columns, computed columns, aliases, and aggregate functions from one or more tables.
- The tables on which the view is based are called **base tables**.
- Views are created in SQL using the CREATE VIEW command. Views are not available in MySQL 4.1, but will be a new feature in MySQL 5.0.



Virtual Tables: Creating Views (cont.)

- The syntax of the CREATE VIEW command is:

```
CREATE VIEW viewname AS SELECT query
```

- The CREATE VIEW statement is a DDL command that stores the subquery specification, i.e., the SELECT statement used to generate the virtual table in the data dictionary.

- An example:

```
CREATE VIEW PRODUCT_3 AS  
SELECT P_DESCRIPT, P_ONHAND, P_PRICE  
FROM PRODUCT  
WHERE P_PRICE > 50.00;
```

- Note: The CREATE VIEW command is not directly supported in Access. To create a view in Access, you just need to create an SQL query and then save it.



Virtual Tables: Creating Views (cont.)

- A relational view has several special characteristics:
 1. You can use the name of a view anywhere a table name is expected in an SQL statement.
 2. Views are dynamically updated. That is, the view is re-created on demand each time it is invoked.
 3. Views provide a level of security in the database because the view can restrict users to only specified columns and specified rows in a table.
 4. Views may also be used as the basis for reports. The view definition shown below creates a summary of total product cost and quantity on hand statistics grouped by vendor:

```
CREATE VIEW SUMPRDXVEN AS
  SELECT V_CODE, SUM(P_ONHAND*P_PRICE) AS TOTCOST,
         MAX(P_ONHAND) AS MAXQTY, MIN(P_ONHAND) AS MINQTY,
         AVG(P_ONHAND) AS AVGGTY
  FROM PRODUCT
  GROUP BY V_CODE;
```



Joining Database Tables

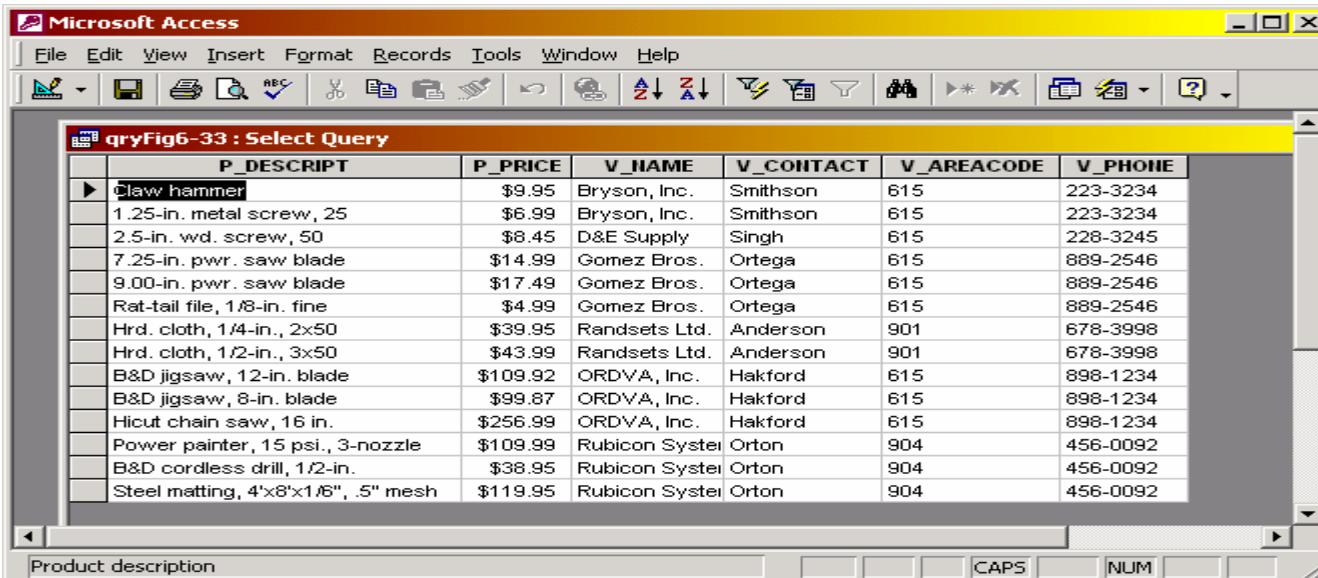
- The ability to combine (join) tables on common attributes is perhaps the most important distinction between a relational database and other types of databases.
- In SQL, a join is performed whenever data is retrieved from more than one table at a time.
- To join tables, you simply enumerate the tables in the FROM clause of the SELECT statement. The RDBMS will create the Cartesian product of every table specified in the FROM clause.
- To effect a natural join, you must specify the linking on the common attributes in the WHERE clause. This is called the **join condition**.
- The join condition is generally composed of an equality comparison between the foreign key and the primary key in the related tables.



Joining Database Tables (cont.)

- Suppose we want to join the **VENDOR** and **PRODUCT** tables. **V_CODE** is the foreign key in the **PRODUCT** table and the primary key in the **VENDOR** table, the join condition occurs on this attribute.

```
SELECT PRODUCT.P_DESCRIPTION, PRODUCT.P_PRICE, VENDOR.V_NAME  
        VENDOR.V_CONTACT, VENDOR.V_AREACODE, VENDOR.V_PHONE  
FROM PRODUCT, VENDOR  
WHERE PRODUCT.V_CODE = VENDOR.V_CODE;
```



P_DESCRIPTION	P_PRICE	V_NAME	V_CONTACT	V_AREACODE	V_PHONE
Claw hammer	\$9.95	Bryson, Inc.	Smithson	615	223-3234
1.25-in. metal screw, 25	\$6.99	Bryson, Inc.	Smithson	615	223-3234
2.5-in. wd. screw, 50	\$8.45	D&E Supply	Singh	615	228-3245
7.25-in. pwr. saw blade	\$14.99	Gomez Bros.	Ortega	615	889-2546
9.00-in. pwr. saw blade	\$17.49	Gomez Bros.	Ortega	615	889-2546
Rat-tail file, 1/8-in. fine	\$4.99	Gomez Bros.	Ortega	615	889-2546
Hrd. cloth, 1/4-in., 2x50	\$39.95	Randssets Ltd.	Anderson	901	678-3998
Hrd. cloth, 1/2-in., 3x50	\$43.99	Randssets Ltd.	Anderson	901	678-3998
B&D jigsaw, 12-in. blade	\$109.92	ORDVA, Inc.	Hakford	615	898-1234
B&D jigsaw, 8-in. blade	\$99.87	ORDVA, Inc.	Hakford	615	898-1234
Hicut chain saw, 16 in.	\$256.99	ORDVA, Inc.	Hakford	615	898-1234
Power painter, 15 psi., 3-nozzle	\$109.99	Rubicon System	Orton	904	456-0092
B&D cordless drill, 1/2-in.	\$38.95	Rubicon System	Orton	904	456-0092
Steel matting, 4'x8'x1/8", .5" mesh	\$119.95	Rubicon System	Orton	904	456-0092

Qualified names are normally only required where the same attribute appears in more than one of the joined relations.



Joining Database Tables (cont.)

- If you do not specify a join condition in the WHERE clause, a Cartesian product results. Using our sample database, the PRODUCT table contains 16 tuples (rows) and the VENDOR table contains 11 tuples, which results in a Cartesian product that contains $16 \times 11 = 176$ tuples. Most of these tuples (as you can see from the proper result on the previous page) are garbage!
- When joining three or more tables, you need to specify a join condition for each pair of tables. The number of join conditions will always be $N-1$ where N is the number of tables listed in the FROM clause.
- Be careful not to create circular join conditions. For example, if table A is related to table B, table B is related to table C, and table C is also related to table A, create only two join conditions: join A with B and B with C. Do not join C with A!



Recursive Joins

- An **alias** can be used to identify the source table from which data is taken for a query. For example:

```
SELECT P_DESCRIPTION, P_PRICE, V_NAME, V_CONTACT, V_AREACODE, V_PHONE  
FROM PRODUCT P, VENDOR V  
WHERE P.V_CODE = V.V_CODE  
ORDER BY P_PRICE;
```

Creating an alias. In Access add the keyword AS before the alias.

- An alias is especially useful when a table must be joined with itself, called a **recursive join**.
- For example, using the EMPLOYEE table we would like to generate a list of all employees along with the name of their manager. Without using an alias this query is not possible, since even qualified attribute names are not unique.



Recursive Joins (cont.)

The screenshot displays two windows from Microsoft Access. The top window, titled 'qryFig6-36 : Select Query', contains the following SQL code:

```
SELECT E.EMP_MGR, M.EMP_LNAME, E.EMP_NUM, E.EMP_LNAME  
FROM EMP AS E, EMP AS M  
WHERE E.EMP_MGR=M.EMP_NUM  
ORDER BY E.EMP_MGR;
```

A blue callout box with an arrow points to the 'EMP AS M' part of the FROM clause, containing the text: 'Creating an alias using Access notation.'

The bottom window, also titled 'qryFig6-36 : Select Query', shows the results of the query in a data grid. The grid has four columns: EMP_MGR, M.EMP_LNAME, EMP_NUM, and E.EMP_LNAME. The data is as follows:

EMP_MGR	M.EMP_LNAME	EMP_NUM	E.EMP_LNAME
100	Kolmycz	112	Johnson
100	Kolmycz	103	Jones
100	Kolmycz	102	Vandam
100	Kolmycz	101	Lewis
105	Williams	115	Saranda
105	Williams	113	Smythe
105	Williams	111	Washington
105	Williams	107	Diante
105	Williams	106	Smith
105	Williams	104	Lange
108	Wiesenbach	116	Smith
108	Wiesenbach	114	Brandon
108	Wiesenbach	110	Genkazi
108	Wiesenbach	109	Smith

The status bar at the bottom of the data grid window shows 'Record: 1 of 14' and 'Employee's Manager'.



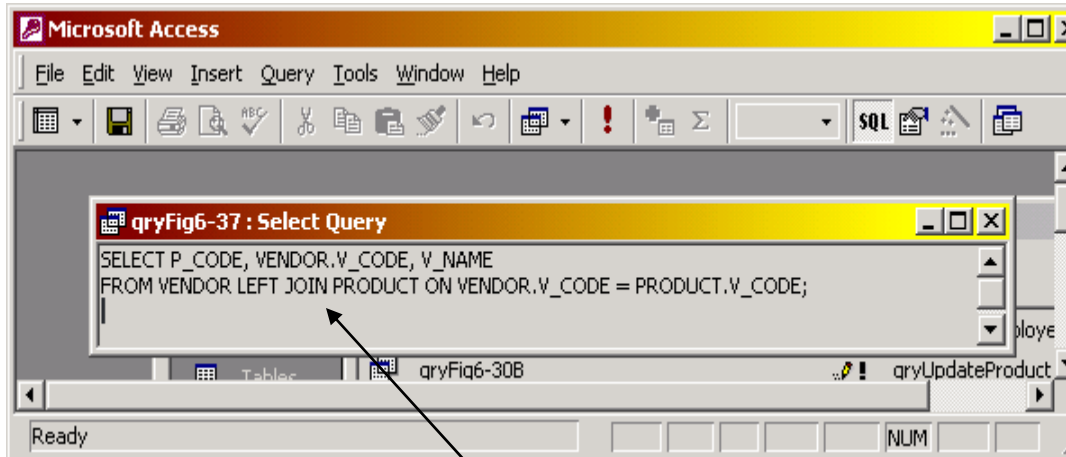
Outer Joins

- The query results shown on page 23 resulted from the natural join of the PRODUCT and VENDOR tables. Notice that there are 14 product rows listed in this output. If you compare these results with the PRODUCT table itself (see SQL part 1 notes page 46) you will notice that there are two missing products. Why? The reason is that the two missing products have null values in the V_CODE attribute in the PRODUCT table. Because there is no matching null “value” in the VENDOR table’s V_CODE attribute, they do not appear in the final output based on the join.
- To include such rows in the final join output, we’ll need to use an outer join.
- Recall that there are three basic types of outer joins, left outer joins, right outer joins, and full outer joins. Given tables A and B, A left outer join B gives all matching rows (on the join condition) plus all unmatched rows in A. A right outer join B gives all matching rows (on the join condition) plus all unmatched rows in B. We’ll look at full outer joins later.



Left Outer Joins

- To include the null valued V_CODE tuples from the PRODUCT table in the final output, we'll need to issue the following query:



Note: The word “outer” does not appear in the query. It is simply either a left join or a right join, the outer is implied.

P_CODE	V_CODE	V_NAME
23109-HE	21225	Bryson, Inc.
SM-18277	21225	Bryson, Inc.
	21226	SuperLoo, Inc.
SW-23116	21231	D&E Supply
13-Q2/P2	21344	Gomez Bros.
14-Q1/L3	21344	Gomez Bros.
54778-2T	21344	Gomez Bros.
	22567	Dome Supply
1546-QQ2	23119	Randsets Ltd.
1558-QW1	23119	Randsets Ltd.
	24004	Brackman Bros.
2232/QTY	24288	ORDVA, Inc.
2232/QWE	24288	ORDVA, Inc.
89-WRE-Q	24288	ORDVA, Inc.
	25443	B&K, Inc.
	25501	Damal Supplies
11QER/31	25595	Rubicon Systems
2238/QPD	25595	Rubicon Systems
WR3/TT3	25595	Rubicon Systems
*		



Left Outer Joins (cont.)

P_CODE	V_CODE
11QER/31	25595
13-Q2/P2	21344
14-Q1/L3	21344
1546-QQ2	23119
1558-QW1	23119
2232/QTY	24288
2232/QWE	24288
2238/QPD	25595
23109-HB	21225
23114-AA	
54778-2T	21344
89-WRE-Q	24288
PVC23DRT	
SM-18277	21225
SW-23116	21231
WR3/TT3	25595
*	0

V_CODE	V_NAME
21225	Bryson, Inc.
21226	SuperLoo, Inc.
21231	D&E Supply
21344	Gomez Bros.
22567	Dome Supply
23119	Randsets Ltd.
24004	Brackman Bros.
24288	ORDVA, Inc.
25443	B&K, Inc.
25501	Damal Supplies
25595	Rubicon Systems
*	0

P_CODE	V_CODE	V_NAME
23109-HB	21225	Bryson, Inc.
SM-18277	21225	Bryson, Inc.
	21226	SuperLoo, Inc.
	21231	D&E Supply
SW-23116	21231	D&E Supply
13-Q2/P2	21344	Gomez Bros.
14-Q1/L3	21344	Gomez Bros.
54778-2T	21344	Gomez Bros.
	22567	Dome Supply
1546-QQ2	23119	Randsets Ltd.
1558-QW1	23119	Randsets Ltd.
	24004	Brackman Bros.
2232/QTY	24288	ORDVA, Inc.
2232/QWE	24288	ORDVA, Inc.
89-WRE-Q	24288	ORDVA, Inc.
	25443	B&K, Inc.
	25501	Damal Supplies
11QER/31	25595	Rubicon Systems
2238/QPD	25595	Rubicon Systems
WR3/TT3	25595	Rubicon Systems
*	0	

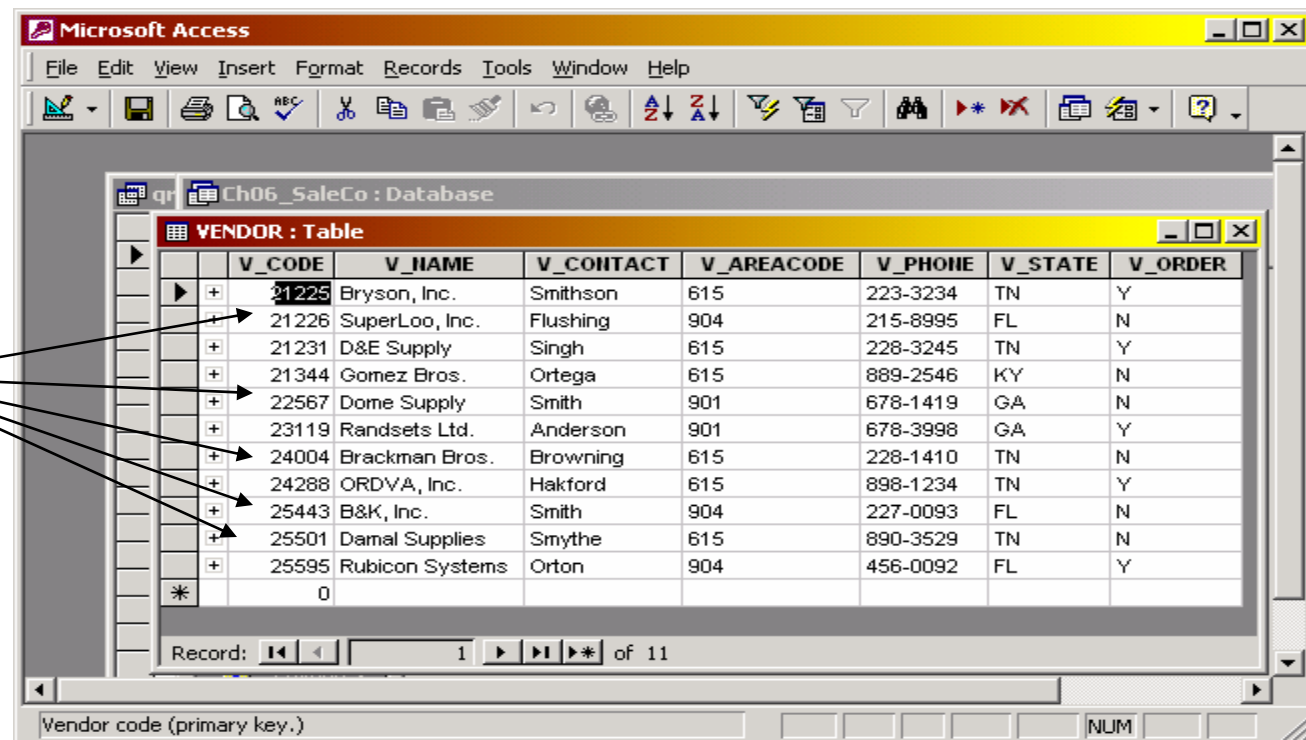
Results shows all rows from VENDOR with all matching rows from PRODUCT (left outer join).



Right Outer Joins

- The VENDOR table is shown below. Notice that there are rows in this table in which the V_CODE does not match any of the V_CODE values in the PRODUCT table.

These vendors do not appear in the PRODUCT table



V_CODE	V_NAME	V_CONTACT	V_AREACODE	V_PHONE	V_STATE	V_ORDER
21225	Bryson, Inc.	Smithson	615	223-3234	TN	Y
21226	SuperLoo, Inc.	Flushing	904	215-8995	FL	N
21231	D&E Supply	Singh	615	228-3245	TN	Y
21344	Gomez Bros.	Ortega	615	889-2546	KY	N
22567	Dome Supply	Smith	901	678-1419	GA	N
23119	Randssets Ltd.	Anderson	901	678-3998	GA	Y
24004	Brackman Bros.	Browning	615	228-1410	TN	N
24288	ORDVA, Inc.	Hakford	615	898-1234	TN	Y
25443	B&K, Inc.	Smith	904	227-0093	FL	N
25501	Damal Supplies	Smythe	615	890-3529	TN	N
25595	Rubicon Systems	Orton	904	456-0092	FL	Y
*	0					



Right Outer Joins (cont.)

The right outer join shows all PRODUCT rows with all matching VENDOR rows.

P_CODE	V_CODE	V_NAME
23114-AA		
PVC23DRT		
23109-HB	21225	Bryson, Inc.
SM-18277	21225	Bryson, Inc.
SW-23116	21231	D&E Supply
13-Q2/P2	21344	Gomez Bros.
14-Q1/L3	21344	Gomez Bros.
54778-2T	21344	Gomez Bros.
1546-QQ2	23119	Randssets Ltd.
1558-QW1	23119	Randssets Ltd.
2232/QTY	24288	ORDVA, Inc.
2232/QWE	24288	ORDVA, Inc.
89-WRE-Q	24288	ORDVA, Inc.
11QER/31	25595	Rubicon Systems
2238/QPD	25595	Rubicon Systems
wWR3/TT3	25595	Rubicon Systems
*		



Right Outer Joins (cont.)

Microsoft Access

File Edit View Insert Format Records Help

Tools Window Help

Projection on PRODUCT : S...

P_CODE	V_CODE
11QER/31	25595
13-Q2/P2	21344
14-Q1/L3	21344
1546-QQ2	23119
1558-QW1	23119
2232/QTY	24288
2232/QWE	24288
2238/QPD	25595
23109-HB	21225
23114-AA	
54778-2T	21344
89-WRE-Q	24288
PVC23DRT	
SM-18277	21225
SW-23116	21231
WR3/TT3	25595
*	0

Record: 1 of 1

Microsoft Access

File Edit View Insert Format Records Tools Window Help

Ch06_SaleCo : Database

qryFig6-38 : Select Query

P_CODE	V_CODE	V_NAME
23114-AA		
PVC23DRT		
23109-HB	21225	Bryson, Inc.
SM-18277	21225	Bryson, Inc.
SW-23116	21231	D&E Supply
13-Q2/P2	21344	Gomez Bros.
14-Q1/L3	21344	Gomez Bros.
54778-2T	21344	Gomez Bros.
1546-QQ2	23119	Randsets Ltd.
1558-QW1	23119	Randsets Ltd.
2232/QTY	24288	ORDVA, Inc.
2232/QWE	24288	ORDVA, Inc.
89-WRE-Q	24288	ORDVA, Inc.
11QER/31	25595	Rubicon Systems
2238/QPD	25595	Rubicon Systems
WR3/TT3	25595	Rubicon Systems
*		

Record: 1 of 16

Product code: P

Result shows all rows from PRODUCT with all matching rows from VENDOR (right outer join)



Relational Set Operators

- Recall that relational algebra is set-oriented and includes many set operators such as union, intersection, and set difference. Recall too, that the terms, sets, tables and relations are interchangeable in the relational world.
- As with pure relational algebra, the set operators only work with union-compatible relations. In SQL, this means that the names of the attributes must be the same and their data types must be identical. This is an area where different RDBMSs vary widely in what is meant by union-compatible. For example, some RDBMSs will consider the data types VARCHAR(35) and VARCHAR(15) compatible because, although they have different length, the underlying base type is the same. Other RDBMSs will not consider these two data types as compatible. You'll need to experiment with your RDBMS to see what is compatible and what isn't.



Union Operator

- Suppose that our company has bought another company and management wants to make sure that the acquired company's customer list is properly merged with the existing company customer list. Since it is quite possible that some customers have purchased from both companies, the two lists may contain common customers. Management does not want any duplicates in the customer list.
- The SQL UNION query automatically removes duplicate rows from the operand relations. If you wish to include duplicate rows in the result use the UNION ALL command.
- The syntax of a UNION query is: `query UNION query`
- Basically, the UNION statement combines the output of two SELECT queries. Remember that the output of the two SELECT queries must be union compatible.
- To illustrate the UNION query, let's combine our original customer list with the new customer list as shown on the next couple of pages.



Union Operator (cont.)

The image displays two Microsoft Access windows. The top window, titled 'CUSTOMER : Table', shows a table with 10 records. The bottom window, titled 'CUSTOMER_2 : Table', shows a table with 7 records. Both windows are in Datasheet View.

CUSTOMER : Table

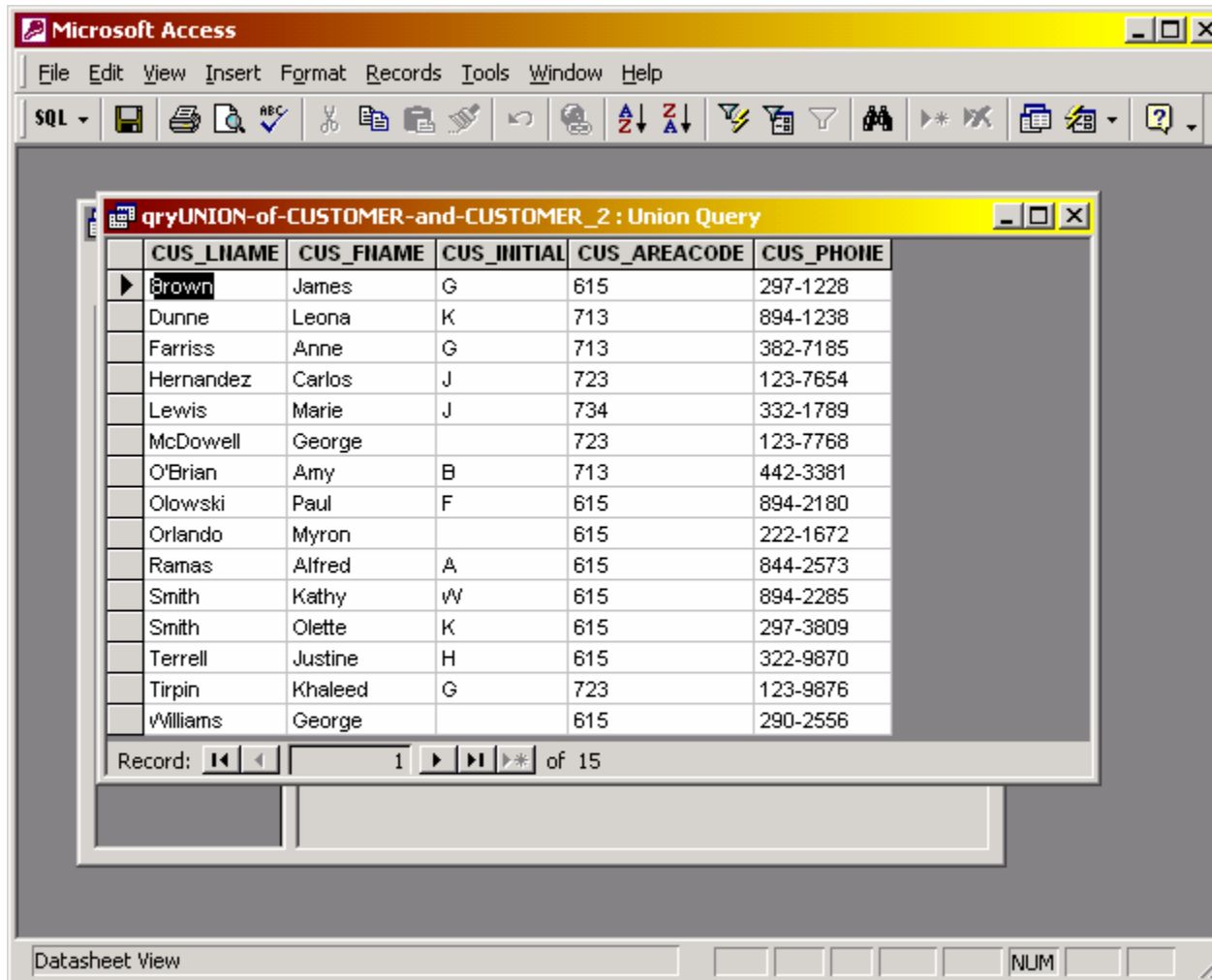
CUS_CODE	CUS_LIAME	CUS_FIAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE	CUS_BALANCE
10010	Ramas	Alfred	A	615	844-2573	\$0.00
10011	Dunne	Leona	K	713	894-1238	\$0.00
10012	Smith	Kathy	vW	615	894-2285	\$345.86
10013	Olowski	Paul	F	615	894-2180	\$536.75
10014	Orlando	Myron		615	222-1672	\$0.00
10015	O'Brian	Amy	B	713	442-3381	\$0.00
10016	Brown	James	G	615	297-1228	\$221.19
10017	vWilliams	George		615	290-2556	\$768.93
10018	Farriss	Anne	G	713	382-7185	\$216.55
10019	Smith	Olette	K	615	297-3809	\$0.00

CUSTOMER_2 : Table

CUS_CODE	CUS_LIAME	CUS_FIAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE
345	Terrell	Justine	H	615	322-9870
347	Olowski	Paul	F	615	894-2180
351	Hernandez	Carlos	J	723	123-7654
352	McDowell	George		723	123-7768
365	Tirpin	Khaleed	G	723	123-9876
368	Lewis	Marie	J	734	332-1789
369	Dunne	Leona	K	713	894-1238



Union Operator (cont.)



The screenshot shows the Microsoft Access interface with a window titled "qryUNION-of-CUSTOMER-and-CUSTOMER_2 : Union Query". The window displays a table with the following columns: CUS_LIAME, CUS_FIAME, CUS_INITIAL, CUS_AREACODE, and CUS_PHONE. The data is as follows:

CUS_LIAME	CUS_FIAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE
Brown	James	G	615	297-1228
Dunne	Leona	K	713	894-1238
Farriss	Anne	G	713	382-7185
Hernandez	Carlos	J	723	123-7654
Lewis	Marie	J	734	332-1789
McDowell	George		723	123-7768
O'Brian	Amy	B	713	442-3381
Olowski	Paul	F	615	894-2180
Orlando	Myron		615	222-1672
Ramas	Alfred	A	615	844-2573
Smith	Kathy	w	615	894-2285
Smith	Olette	K	615	297-3809
Terrell	Justine	H	615	322-9870
Tirpin	Khaleed	G	723	123-9876
Williams	George		615	290-2556

The status bar at the bottom of the window indicates "Record: 1 of 15" and "NUM". The view is set to "Datasheet View".

The result of the UNION of the CUSTOMER and CUSTOMER_2 tables.

Customer names Dunne and Olowski appear in both original tables and thus appear only once in the union result.



Union ALL Operator

CUS_LIAME	CUS_FIAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE
Ramas	Alfred	A	615	844-2573
Dunne	Leona	K	713	894-1238
Smith	Kathy	W	615	894-2285
Olowski	Paul	F	615	894-2180
Orlando	Myron		615	222-1672
O'Brian	Amy	B	713	442-3381
Brown	James	G	615	297-1228
Williams	George		615	290-2556
Farriss	Anne	G	713	382-7185
Smith	Olette	K	615	297-3809
Terrell	Justine	H	615	322-9870
Olowski	Paul	F	615	894-2180
Hernandez	Carlos	J	723	123-7654
McDowell	George		723	123-7768
Tirpin	Khaleed	G	723	123-9876
Lewis	Marie	J	734	332-1789
Dunne	Leona	K	713	894-1238

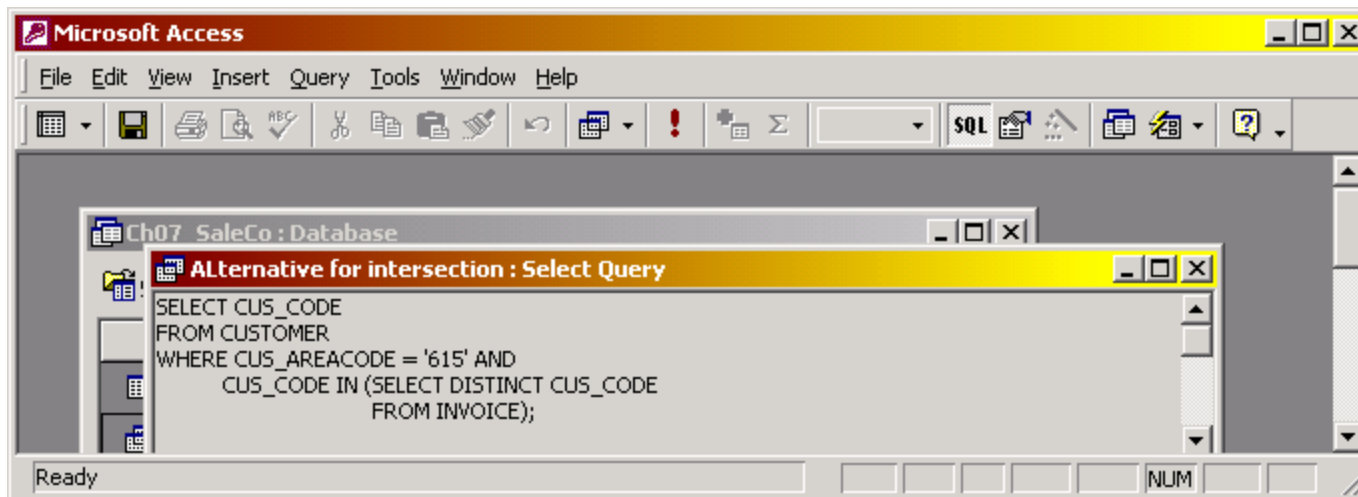
The result of the UNION ALL of the CUSTOMER and CUSTOMER_2 tables.

Customer names Dunne and Olowski appear twice since duplicates are not removed in this form of UNION.



Intersect Operator

- The syntax of an INTERSECT query is: `query INTERESCT query`
- Access does not support the INTERSECT statement. To effect an intersection in Access you need to use the IN operator.



Intersect Operator

The screenshot displays three Microsoft Access windows. The top-left window shows the 'CUSTOMER : Table' with columns CUS_CODE, CUS_LNAME, CUS_FNAME, CUS_INITIAL, CUS_AREACODE, and CUS_PHONE. The top-right window shows the 'INVOICE : Table' with columns IIV_NUMBER, CUS_CODE, and IIV_DATE. The bottom window shows the 'Alternative for intersection : Select Query' with columns CUS_CODE. A blue callout box points to the query results.

CUSTOMER : Table

CUS_CODE	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE
10010	Ramas	Alfred	A	615	844-257
10011	Dunne	Leona	K	713	894-123
10012	Smith	Kathy	W	615	894-228
10013	Olowski	Paul	F	615	894-218
10014	Orlando	Myron		615	222-167
10015	O'Brian	Amy	B	713	442-338
10016	Brown	James	G	615	297-122
10017	Williams	George		615	297-255
10018	Farriss	Anne	G	713	
10019	Smith	Olette	K	615	

INVOICE : Table

IIV_NUMBER	CUS_CODE	IIV_DATE
1001	10014	16-Jan-04
1002	10011	16-Jan-04
1003	10012	16-Jan-04
1004	10011	17-Jan-04
1005	10018	17-Jan-04
1006	10014	17-Jan-04
1007	10015	17-Jan-04
1008	10011	17-Jan-04
0	0	

Alternative for intersection : Select Query

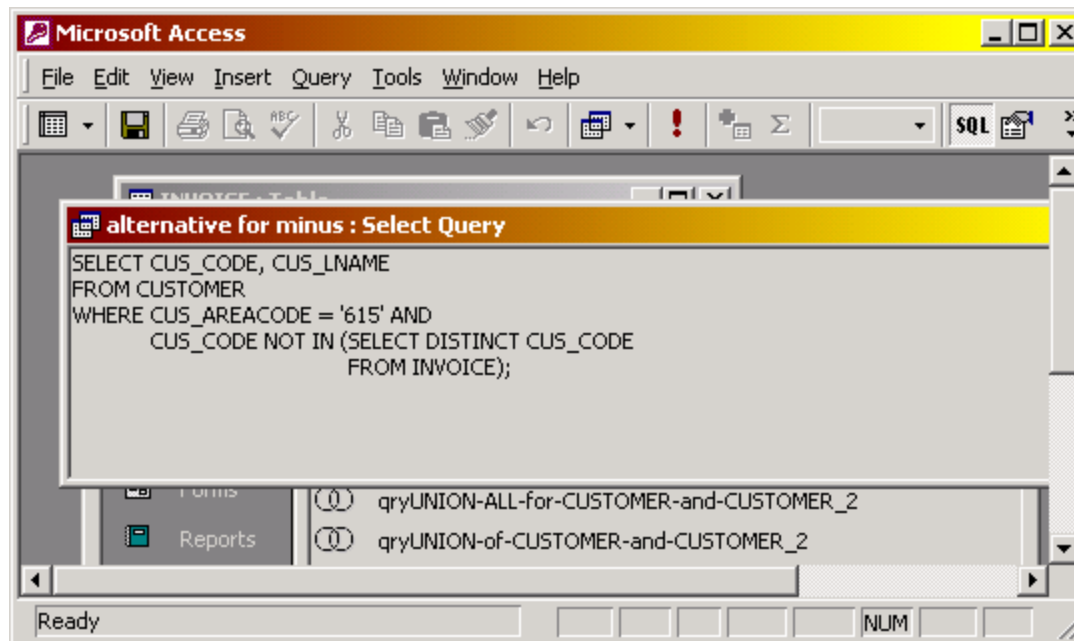
CUS_CODE
10012
10014
0

Results of intersection of the two tables shown above (query on the previous page).



Set Difference Operator

- The syntax of a (set difference) MINUS query is: `query MINUS query`
- Access does not support the MINUS statement. To effect a set difference in Access you need to use the NOT IN operator.
- Most RDBMSs name the MINUS operation EXCEPT.



Set Difference Operator (cont.)

The image displays three overlapping Microsoft Access windows. The top-left window shows the 'CUSTOMER : Table' with 10 records. The top-right window shows the 'INVOICE : Table' with 8 records. The bottom window shows a query titled 'alternative for minus : Select Qu...' with 5 records, which are the results of a set difference operation. A blue callout box points to this query window.

CUSTOMER : Table

CUS_CODE	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_AREACODE
10010	Ramas	Alfred	A	615
10011	Dunne	Leona	K	713
10012	Smith	Kathy	w	615
10013	Olowski	Paul	F	615
10014	Orlando	Myron		615
10015	O'Brian	Amy	B	713
10016	Brown	James	G	615
10017	Williams	George		615
10018	Farriss	Anne	G	713
10019	Smith	Olette	K	615

INVOICE : Table

IIV_NUMBER	CUS_CODE	IIV_DATE
1001	10014	16-Jan-04
1002	10011	16-Jan-04
1003	10012	16-Jan-04
1004	10011	17-Jan-04
1005	10018	17-Jan-04
1006	10014	17-Jan-04
1007	10015	17-Jan-04
1008	10011	17-Jan-04

alternative for minus : Select Qu...

CUS_CODE	CUS_LNAME
10010	Ramas
10013	Olowski
10016	Brown
10017	Williams
10019	Smith

Results of the set difference query from the previous page



SQL Join Operations

- The SQL join operations merge rows from two tables and returns the rows that:
 1. Have common values in common columns (natural join) or,
 2. Meet a given join condition (equality or inequality) or,
 3. Have common values in common columns or have no matching values (outer join).
- We've already examined the basic form of an SQL join which occurs when two tables are listed in the FROM clause and the WHERE clause specifies the join condition.
- An example of this basic form of the join is shown on the next page.



SQL Join Operations (cont.)

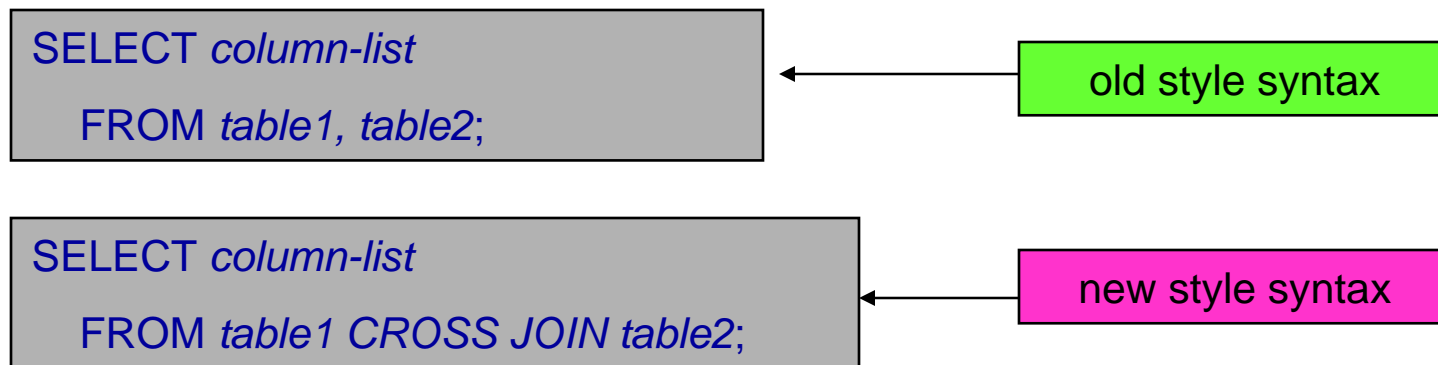
```
SELECT P_CODE, P_DESCRIPT, P_PRICE, V_NAME  
FROM PRODUCT, VENDOR  
WHERE PRODUCT.V_CODE = VENDOR.V_CODE;
```

- The FROM clause indicates which tables are to be joined. If three or more tables are specified, the join operation takes place two tables at a time, starting from left to right.
- The join condition is specified in the WHERE clause. In the example, a natural join is effected on the attribute V_CODE.
- The SQL join syntax shown above is sometimes referred to as an “old-style” join.
- The tables on pages 55 and 56, summarize the SQL join operations.



SQL Cross Join Operation

- A **cross join** in SQL is equivalent to a Cartesian product in standard relational algebra. The cross join syntax is:



SQL Natural Join Operation

- The **natural join** syntax is:

```
SELECT column-list  
FROM table1 NATURAL JOIN table2;
```

new style syntax

- The natural join will perform the following tasks:
 - Determine the common attribute(s) by looking for attributes with identical names and compatible data types.
 - Select only the rows with common values in the common attribute(s).
 - If there are no common attributes, return the cross join of the two tables.



SQL Natural Join Operation (cont.)

- The syntax for the old-style natural join is:

```
SELECT column-list
FROM table1, table2
WHERE table1.C1 = table2.C2;
```

old style syntax

- One important difference between the natural join and the “old-style” syntax is that the natural join does not require the use of a table qualifier for the common attributes. The two SELECT statements shown on the next page are equivalent.



SQL Natural Join Operation (cont.)

```
SELECT CUS_NUM, CUS_LNAME,  
       INV_NUMBER, INV_DATE  
FROM   CUSTOMER, INVOICE  
WHERE  CUSTOMER.CUS_NUM = INVOICE.CUS_NUM;
```

old style
syntax

```
SELECT CUS_NUM, CUS_LNAME,  
       INV_NUMBER, INV_DATE  
FROM   CUSTOMER NATURAL JOIN INVOICE;
```

old style
syntax



Join With Using Clause

- A second way to express a join is through the **USING** keyword. This query will return only the rows with matching values in the column indicated in the **USING** clause. The column listed in the **USING** clause must appear in both tables.
- The syntax is:

```
SELECT column-list  
FROM table1 JOIN table2 USING (common-column);
```



Join With Using Clause (cont.)

- An example:

```
SELECT INV_NUMBER, P_CODE, P_DESCRIPT, LINE_UNITS,  
       LINE_PRICE  
FROM INVOICE JOIN LINE USING (INV_NUMBER)  
       JOIN PRODUCT USING (P_CODE);
```

- As was the case with the natural join command, the `JOIN USING` does not required the use of qualified names (qualified table names). In fact, Oracle 9i will return an error if you specify the table name in the `USING` clause.



Join On Clause

- Both the NATURAL JOIN and the JOIN USING commands use common attribute names in joining tables.
- Another way to express a join when the tables have no common attribute names is to use the JOIN ON operand. This query will return only the rows that meet the indicated condition. The join condition will typically include an equality comparison expression of two columns. The columns may or may not share the same name, but must obviously have comparable data types.
- The syntax is:

```
SELECT column-list  
FROM table1 JOIN table2 ON join-condition;
```



Join On Clause (cont.)

- An example:

```
SELECT INVOICE.INV_NUMBER, P_CODE, P_DESCRIPT, LINE_UNITS, LINE_PRICE  
FROM INVOICE JOIN LINE ON INVOICE.INV_NUMBER = LINE.INV_NUMBER  
JOIN PRODUCT ON LINE.P_CODE = PRODUCT.P_CODE;
```

- Notice in the example query, that unlike the NATURAL JOIN and the JOIN USING operation, the JOIN ON clause requires the use of table qualifiers for the common attributes. If you do not specify the table qualifier you will get a “column ambiguously defined” error message.
- Keep in mind that the JOIN ON syntax allows you to perform a join even when the tables do not share a common attribute name.



Join On Clause (cont.)

- For example, to generate a list of all employees with the manager's name you can use the recursive query shown below which utilizes the JOIN ON clause.

```
SELECT E.EMP_MGR, M.EMP_LNAME, E.EMP_NUM, E.EMP_LNAME  
FROM EMP E JOIN EMP M ON E.EMP_MGR = M.EMP_NUM  
ORDER BY E.EMP_MGR;
```



Outer Joins

- We saw the forms for the LEFT OUTER JOIN and the RIGHT OUTER JOIN in the previous set of notes.
- There is also a FULL OUTER JOIN operation in SQL. A full outer join returns not only the rows matching the join condition (that is, rows with matching values in the common column(s)), but also all the rows with unmatched values in either side table.
- The syntax of a full outer join is:

```
SELECT column-list  
FROM table1 FULL [OUTER] JOIN table2 ON join-condition;
```



Outer Joins (cont.)

- The following example will list the product code, vendor code, and vendor name for all products and include all the product rows (products without matching vendors) and also all vendor rows (vendors without matching products):

```
SELECT P_CODE, VENDOR.V_CODE, V_NAME  
FROM VENDOR FULL OUTER JOIN PRODUCT  
ON VENDOR.V_CODE = PRODUCT.V_CODE;
```



Summary of SQL JOIN Operations

Join Classification	Join Type	SQL Syntax Example	Description
Cross	CROSS JOIN	SELECT * FROM T1, T2;	Old style. Returns the Cartesian product of T1 and T2
		SELECT * FROM T1 CROSS JOIN T2;	New style. Returns the Cartesian product of T1 and T2.
Inner	Old Style JOIN	SELECT * FROM T1, T2 WHERE T1.C1 = T2.C1	Returns only the rows that meet the join condition in the WHERE clause – old style. Only rows with matching values are selected.
	NATURAL JOIN	SELECT * FROM T1 NATURAL JOIN T2	Returns only the rows with matching values in the matching columns. The matching columns must have the same names and similar data types.
	JOIN USING	SELECT * FROM T1 JOIN T2 USING (C1)	Returns only the rows with matching values in the columns indicated in the USING clause.
	JOIN ON	SELECT * FROM T1 JOIN T2 ON T1.C1 = T2.C1	Returns only the rows that meet the join condition indicated in the ON clause.



Summary of SQL JOIN Operations (cont.)

Join Classification	Join Type	SQL Syntax Example	Description
Outer	LEFT JOIN	SELECT * FROM T1 LEFT OUTER JOIN T2 ON T1.C1= T2.C1	Returns rows with matching values and includes all rows from the left table (T1) with unmatched values.
	RIGHT JOIN	SELECT * FROM T1 RIGHT OUTER JOIN T2 ON T1.C1= T2.C1	Returns rows with matching values and includes all rows from the right table (T2) with unmatched values.
	FULL JOIN	SELECT * FROM T1 FULL OUTER JOIN T2 ON T1.C1= T2.C1	Returns rows with matching values and includes all rows from both tables (T1 and T2) with unmatched values.



Subqueries and Correlated Queries

- The use of joins allows a RDBMS go get information from two or more tables. The data from the tables is processed simultaneously.
- It is often necessary to process data based on other processed data. Suppose, for example, that you want to generate a list of vendors who provide products. (Recall that not all vendors in the VENDOR table have provided products – some of them are only potential vendors.)
- The following query will accomplish our task:

```
SELECT V_CODE, V_NAME  
FROM VENDOR  
WHERE V_CODE NOT IN (SELECT V_CODE FROM PRODUCT);
```



Subqueries and Correlated Queries (cont.)

- A subquery is a query (SELECT statement) inside a query.
- A subquery is normally expressed inside parentheses.
- The first query in the SQL statement is known as the outer query.
- The second query in the SQL statement is known as the inner query.
- The inner query is executed first.
- The output of the inner query is used as the input for the outer query.
- The entire SQL statement is sometimes referred to as a nested query.



Subqueries and Correlated Queries (cont.)

- A subquery can return:
 1. One single value (one column and one row). This subquery can be used anywhere a single value is expected. For example, in the right side of a comparison expression.
 2. A list of values (one column and multiple rows). This type of subquery can be used anywhere a list of values is expected. For example, when using the IN clause.
 3. A virtual table (multi-column, multi-row set of values). This type of subquery can be used anywhere a table is expected. For example, in the FROM clause.
 4. No value at all, i.e., NULL. In such cases, the output of the outer query may result in an error or null empty set, depending on where the subquery is used (in a comparison, an expression, or a table set).



Correlated Queries

- A correlated query (really a subquery) is a subquery that contains a reference to a table that also appears in the outer query.
- A correlated query has the following basic form:

```
SELECT * FROM table1 WHERE col1 = ANY  
  (SELECT col1 FROM table2  
   WHERE table2.col2 = table1.col1);
```

- Notice that the subquery contains a reference to a column of `table1`, even though the subquery's `FROM` clause doesn't mention `table1`. Thus, query execution requires a look outside the subquery, and finds the table reference in the outer query.



WHERE Subqueries

- The most common type of subquery uses an inner SELECT subquery on the right hand side of a WHERE comparison expression.
- For example, to find all products with a price greater than or equal to the average product price, the following query would be needed:

```
SELECT P_CODE, P_PRICE  
FROM PRODUCT  
WHERE P_PRICE >= (SELECT AVG(P_PRICE)  
FROM PRODUCT);
```



WHERE Subqueries (cont.)

- Subqueries can also be used in combination with joins.
- The query below lists all the customers that ordered the product “Claw hammer”.

```
SELECT DISTINCT CUS_CODE, CUS_LNAME, CUYS_FNAME
FROM CUSTOMER JOIN INVOICE USING (CUS_CODE)
             JOIN LINE USING (INV_NUMBER)
             JOIN PRODUCT USING (P_CODE)
WHERE P_CODE = (SELECT P_CODE
                FROM PRODUCT
                WHERE P_DESCRIPT = "Claw hammer");
```



WHERE Subqueries (cont.)

- Notice that the previous query could have been written as:

```
SELECT DISTINCT CUS_CODE, CUS_LNAME, CUYS_FNAME
FROM CUSTOMER JOIN INVOICE USING (CUS_CODE)
             JOIN LINE USING (INV_NUMBER)
             JOIN PRODUCT USING (P_CODE)
WHERE P_DESCRIPT = 'Claw hammer');
```

- However, what would happen if two or more product descriptions contain the string “Claw hammer”?
 - You would get an error message because only a single value is expected on the right hand side of this expression.



IN Subqueries

- To handle the problem we just saw, the IN operand must be used.
- The query below lists all the customers that ordered any kind of hammer or saw.

```
SELECT DISTINCT CUS_CODE, CUS_LNAME, CUYS_FNAME
FROM CUSTOMER JOIN INVOICE USING (CUS_CODE)
              JOIN LINE USING (INV_NUMBER)
              JOIN PRODUCT USING (P_CODE)
WHERE P_CODE IN (SELECT P_CODE
                 FROM PRODUCT
                 WHERE P_DESCRIPT LIKE '%hammer%'
                 OR P_DESCRIPT LIKE '%saw%');
```



HAVING Subqueries

- It is also possible to use subqueries with a HAVING clause.
- Recall that the HAVING clause is used to restrict the output of a GROUP BY query by applying a conditional criteria to the grouped rows.
- For example, the following query will list all products with the total quantity sold greater than the average quantity sold.

```
SELECT DISTINCT P_CODE, SUM(LINE_UNITS)
FROM LINE
GROUP BY P_CODE
HAVING SUM(LINE_UNITS) > (SELECT AVG(LINE_UNITS)
                           FROM LINE);
```



Multi-row Subquery Operators: ANY and ALL

- The IN subquery uses an equality operator; that is, it only selects those rows that match at least one of the values in the list. What happens if you need to do an inequality comparison of one value to a list of values?
- For example, suppose you want to know what products have a product cost that is greater than all individual product costs for products provided by vendors from Florida.

```
SELECT P_CODE, P_ONHAND*P_PRICE
FROM PRODUCT
WHERE P_ONHAND*P_PRICE > ALL (SELECT P_ONHAND*P_PRICE
                               FROM PRODUCT
                               WHERE V_CODE IN (SELECT V_CODE
                                                FROM VENDOR
                                                WHERE V_STATE= 'FL'));
```



FROM Subqueries

- In all of the cases of subqueries we've seen so far, the subquery was part of a conditional expression and it always appeared on the right hand side of an expression. This is the case for WHERE, HAVING, and IN subqueries as well as for the ANY and ALL operators.
- Recall that the FROM clause specifies the table(s) from which the data will be drawn. Because the output of a SELECT statement is another table (or more precisely, a “virtual table”), you could use a SELECT subquery in the FROM clause.
- For example, suppose that you want to know all customers who have purchased products 13-Q2/P2 and 23109-HB. Since all product purchases are stored in the LINE table, it is easy to find out who purchased any given product just by searching the P_CODE attribute in the LINE table. However, in this case, you want to know all customers who purchased both, not just one.
- The query on the next page accomplishes this task.



FROM Subqueries (cont.)

```
SELECT DISTINCT CUSTOMER.CUS_CODE      , CUSTOMER.LNAME
FROM CUSTOMER, (SELECT INVOICE.CUS_CODE
                 FROM INVOICE NATURAL JOIN LINE
                 WHERE P_CODE = '13-Q2/P2') CP1,
              (SELECT INVOICE.CUS_CODE
                 FROM INVOICE NATURAL JOIN LINE
                 WHERE P_CODE = '23109-HB') CP2
WHERE CUSTOMER.CUS_CODE = CP1.CUS_CODE
AND CP1.CUS_CODE = CP2.CUS_CODE;
```



Subqueries in MySQL

- The ability to handle subqueries like we've just examined was not available in MySQL until version 4.1.
- If you are using a version of MySQL earlier than 4.1 you will need to download the latest version (5.0) before you begin to work on the next assignment which will involve the execution of subqueries.
- There are a number of other enhancements that became active with version 4.1 that are extremely useful and we will examine a number of these over the coming days.



Subqueries in MySQL (cont.)

- Subqueries are also useful in optimizing queries as they can be used to eliminate more costly join operations.
- Consider the following general query:

```
SELECT DISTINCT table1.col1  
FROM table1, table2  
WHERE table1.col1 = table2.col1;
```

- This query can be more efficiently expressed using subqueries as:

```
SELECT DISTINCT col1  
FROM table1  
WHERE table1.col1 IN (SELECT col1  
                      FROM table2);
```

