An Example Database

[Diagram of a database schema showing relationships between tables such as CUSTOMER, INVOICE, LINE, PRODUCT, and VENDOR.]
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.
- 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:

```sql
SELECT P_CODE, P_DESCRIPT, P_INDATE, P_PRICE
FROM PRODUCT
ORDER BY P_PRICE;
```
Ordering A Listing (cont.)

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

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

Employee Table

<table>
<thead>
<tr>
<th>EMP_NUM</th>
<th>EMP_TITLE</th>
<th>EMP_LNAME</th>
<th>EMP_FNAME</th>
<th>EMP_INITIAL</th>
<th>EMPDOB</th>
<th>EMP_HIRE_DATE</th>
<th>EMP_YEARS</th>
<th>EMP_AREA_CODE</th>
<th>EMP_PHONE</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Mr.</td>
<td>Kolmycz</td>
<td>George</td>
<td>D</td>
<td>15-Jun-42</td>
<td>15-Mar-85</td>
<td>18</td>
<td>615</td>
<td>324-5456</td>
</tr>
<tr>
<td>101</td>
<td>Ms.</td>
<td>Lewis</td>
<td>Rhonda</td>
<td>G</td>
<td>19-Mar-65</td>
<td>25-Apr-86</td>
<td>16</td>
<td>615</td>
<td>324-4472</td>
</tr>
<tr>
<td>102</td>
<td>Mr.</td>
<td>Vandam</td>
<td>Rhett</td>
<td>M</td>
<td>14-Nov-58</td>
<td>20-Dec-90</td>
<td>12</td>
<td>901</td>
<td>675-8993</td>
</tr>
<tr>
<td>103</td>
<td>Ms.</td>
<td>Jones</td>
<td>Anne</td>
<td>M</td>
<td>16-Oct-74</td>
<td>28-Aug-94</td>
<td>8</td>
<td>615</td>
<td>898-3456</td>
</tr>
<tr>
<td>104</td>
<td>Mr.</td>
<td>Lange</td>
<td>John</td>
<td>P</td>
<td>08-Nov-71</td>
<td>20-Oct-94</td>
<td>8</td>
<td>901</td>
<td>504-4430</td>
</tr>
<tr>
<td>105</td>
<td>Mr.</td>
<td>Williams</td>
<td>Robert</td>
<td>R</td>
<td>14-Mar-75</td>
<td>00-Nov-90</td>
<td>4</td>
<td>015</td>
<td>090-3220</td>
</tr>
<tr>
<td>106</td>
<td>Mrs.</td>
<td>Smith</td>
<td>Jeanine</td>
<td>K</td>
<td>12-Feb-68</td>
<td>05-Jan-89</td>
<td>14</td>
<td>615</td>
<td>324-7883</td>
</tr>
<tr>
<td>107</td>
<td>Mr.</td>
<td>Dianette</td>
<td>Jorge</td>
<td>D</td>
<td>21-Aug-74</td>
<td>02-Jul-94</td>
<td>8</td>
<td>615</td>
<td>890-4867</td>
</tr>
<tr>
<td>108</td>
<td>Mr.</td>
<td>Viesenberg</td>
<td>Paul</td>
<td>R</td>
<td>14-Feb-66</td>
<td>18-Nov-92</td>
<td>10</td>
<td>615</td>
<td>897-4356</td>
</tr>
<tr>
<td>109</td>
<td>Mr.</td>
<td>Smith</td>
<td>George</td>
<td>K</td>
<td>18-Jun-61</td>
<td>14-Apr-89</td>
<td>13</td>
<td>901</td>
<td>504-3339</td>
</tr>
<tr>
<td>110</td>
<td>Mrs.</td>
<td>Genkazi</td>
<td>Leighla</td>
<td>V</td>
<td>19-May-70</td>
<td>01-Dec-90</td>
<td>12</td>
<td>901</td>
<td>569-0093</td>
</tr>
<tr>
<td>111</td>
<td>Mr.</td>
<td>Washington</td>
<td>Rupert</td>
<td>E</td>
<td>03-Jan-66</td>
<td>21-Jun-93</td>
<td>9</td>
<td>615</td>
<td>890-4925</td>
</tr>
<tr>
<td>112</td>
<td>Mr.</td>
<td>Johnson</td>
<td>Edward</td>
<td>E</td>
<td>14-May-61</td>
<td>01-Dec-83</td>
<td>19</td>
<td>615</td>
<td>898-4387</td>
</tr>
<tr>
<td>113</td>
<td>Ms.</td>
<td>Smythe</td>
<td>Melanie</td>
<td>P</td>
<td>15-Sep-70</td>
<td>11-May-99</td>
<td>3</td>
<td>615</td>
<td>324-9006</td>
</tr>
<tr>
<td>114</td>
<td>Ms.</td>
<td>Brandon</td>
<td>Marie</td>
<td>G</td>
<td>02-Nov-58</td>
<td>15-Nov-73</td>
<td>23</td>
<td>901</td>
<td>882-0845</td>
</tr>
<tr>
<td>115</td>
<td>Mrs.</td>
<td>Saranda</td>
<td>Herrine</td>
<td>R</td>
<td>25-Jul-72</td>
<td>23-Apr-93</td>
<td>9</td>
<td>615</td>
<td>324-5505</td>
</tr>
<tr>
<td>116</td>
<td>Mr.</td>
<td>Smith</td>
<td>George</td>
<td>A</td>
<td>08-Nov-65</td>
<td>10-Dec-85</td>
<td>14</td>
<td>615</td>
<td>890-2964</td>
</tr>
</tbody>
</table>
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_AREAACODE, 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.)

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:

```sql
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.)

![Microsoft Access screenshot](image)

**qryFig6-23: Select Query**

<table>
<thead>
<tr>
<th>P_DESCRIBE</th>
<th>V_CODE</th>
<th>P_INDATE</th>
<th>P_PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sledge hammer, 12 lb.</td>
<td>02-Jan-04</td>
<td>$14.40</td>
<td></td>
</tr>
<tr>
<td>Claw hammer</td>
<td>02-Jan-04</td>
<td>$9.95</td>
<td></td>
</tr>
<tr>
<td>9.00-in. pwr. saw blade</td>
<td>13-Nov-03</td>
<td>$17.49</td>
<td></td>
</tr>
<tr>
<td>7.25-in. pwr. saw blade</td>
<td>13-Dec-03</td>
<td>$14.99</td>
<td></td>
</tr>
<tr>
<td>Rat-tail file, 1/8-in. tine</td>
<td>15-Dec-03</td>
<td>$4.99</td>
<td></td>
</tr>
<tr>
<td>Hrd. cloth, 1/2-in., 3x50</td>
<td>15-Jan-04</td>
<td>$43.99</td>
<td></td>
</tr>
<tr>
<td>Hrd. cloth, 1/4-in., 2x50</td>
<td>15-Jan-04</td>
<td>$39.95</td>
<td></td>
</tr>
<tr>
<td>B&amp;D cordless drill, 1/2-in.</td>
<td>20-Jan-04</td>
<td>$38.95</td>
<td></td>
</tr>
</tbody>
</table>

Record: 14 of 8
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.
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 conditionlist ]
    [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;
```
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.)

The query

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

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.
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_OHANDB) AS MINQTY,
           AVG(P_ONHAND) AS AVGQTY
    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.

```sql
SELECT  PRODUCT.P_DESCRIPT, PRODUCT.P_PRICE, VENDOR.V_NAME,
        VENDOR.V_CONTACT, VENDOR.V_AREA_CODE, VENDOR.V_PHONE
FROM  PRODUCT, VENDOR
WHERE  PRODUCT.V_CODE = VENDOR.V_CODE;
```

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_DESCRIP, P_PRICE, V_NAME, V_CONTACT, V_AREA_CODE, 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.)

Creating an alias using Access notation.
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 Day 17 page 45) 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:

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

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.
Left Outer Joins (cont.)

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.
Right Outer Joins (cont.)

The right outer join shows all PRODUCT rows with all matching VENDOR rows.
Right Outer Joins (cont.)

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.)

<table>
<thead>
<tr>
<th>CUS_CODE</th>
<th>CUS_LNAME</th>
<th>CUS_FNAME</th>
<th>CUS_INITIAL</th>
<th>CUS_AREAPLACE</th>
<th>CUS_PHONE</th>
<th>CUS_BAS +/-</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>Ranas</td>
<td>Alfred</td>
<td>A</td>
<td>615</td>
<td>844-2572</td>
<td>$0.00</td>
</tr>
<tr>
<td>10001</td>
<td>Dunne</td>
<td>Leona</td>
<td>K</td>
<td>713</td>
<td>894-1238</td>
<td>$0.00</td>
</tr>
<tr>
<td>10012</td>
<td>Smith</td>
<td>Kathy</td>
<td>W</td>
<td>615</td>
<td>894-2268</td>
<td>$345.86</td>
</tr>
<tr>
<td>10013</td>
<td>Olowskis</td>
<td>Paul</td>
<td>F</td>
<td>615</td>
<td>894-2180</td>
<td>$536.75</td>
</tr>
<tr>
<td>10014</td>
<td>Orlando</td>
<td>Myron</td>
<td></td>
<td>615</td>
<td>222-1672</td>
<td>$0.00</td>
</tr>
<tr>
<td>10015</td>
<td>O'Brian</td>
<td>Amy</td>
<td>D</td>
<td>713</td>
<td>442-3301</td>
<td>$0.00</td>
</tr>
<tr>
<td>10016</td>
<td>Brown</td>
<td>James</td>
<td>G</td>
<td>615</td>
<td>297-1228</td>
<td>$221.19</td>
</tr>
<tr>
<td>10017</td>
<td>Williams</td>
<td>George</td>
<td></td>
<td>615</td>
<td>290-2556</td>
<td>$789.93</td>
</tr>
<tr>
<td>10018</td>
<td>Farris</td>
<td>Anne</td>
<td>G</td>
<td>713</td>
<td>382-7185</td>
<td>$216.55</td>
</tr>
<tr>
<td>10019</td>
<td>Smith</td>
<td>Ototto</td>
<td>K</td>
<td>615</td>
<td>207-3800</td>
<td>$0.00</td>
</tr>
</tbody>
</table>

Record: 14 / 7 of 7
Union Operator (cont.)

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

Customer names Dunne and Olowsk appear in both original tables and thus appear only once in the union result.
Union ALL Operator

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:

- Access does not support the INTERSECT statement. To effect an intersection in Access you need to use the IN operator.

```sql
SELECT CUS_CODE
FROM CUSTOMER
WHERE CUS_AREA_CODE = '615' AND
CUS_CODE IN (SELECT DISTINCT CUS_CODE
FROM INVOICE);
```
Intersect Operator

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.)

Results of the set difference query from the previous page.