

# COP 4710: Database Systems Spring 2006

## Chapter 5 – Introduction To SQL – Part 1

Instructor : Mark Llewellyn  
markl@cs.ucf.edu  
CSB 242, 823-2790  
<http://www.cs.ucf.edu/courses/cop4710/spr2006>

School of Electrical Engineering and Computer Science  
University of Central Florida



# History of SQL

- SQL, pronounced “S-Q-L” by some and “sequel” by others (mostly old-timers), has become the de facto standard language for creating and querying relational databases. In fact, ANSI has declared that the official pronunciation for SQL is “*es queue el*”.
- It has been accepted by ANSI (American National Standards Institute) and ISO (International Standards Organization) as well as being a FIPS (Federal Information Processing Standard).
- Between 1974 and 1979, workers at the IBM Research Laboratory in San Jose, California undertook the development of System R. This was shortly after Codd’s classic paper defining the relational database was published. The goal of the System R project was to demonstrate the feasibility of implementing the relational model in a DBMS. They used a language named SEQUEL (Structured English QUery Language), which was a descendent of SQUARE (Specifying QUeries As Relational Expressions), both of which were developed at IBM, San Jose.
- SEQUEL was renamed to SQL during this project due to a trademark dispute with the Hawker-Siddeley aircraft company in the UK.



# History of SQL (cont.)

- System R itself was never produced commercially, but directly led to the development of SQL/DS (1981 running under DOS/VE OS, a VM version followed in 1982) which was IBM's first commercial relational DBMS.
- IBM however, did not produce the first commercial implementation of a relational DBMS. That honor went to Oracle (Relational Software) in 1979.
- Today, the relational DBMS system of virtually all vendors is based on SQL.
- Each vendor provides all the standard features of SQL. Most vendors also provide additional features of their own, called extensions to standard SQL. These extensions lead to portability issues when moving SQL-based applications across various RDBMS. Vendors attempt to distinguish their SQL versions through these extensions.



# History of SQL (cont.)

- The current version of ANSI standard for SQL is SQL-2003 (also referred to as SQL3 as was SQL-99). This standard has also been accepted by ISO.
- Although many different extensions of SQL exist, we'll look at the core SQL that will be found on any RDBMS that you will encounter. Whether you use Oracle, Microsoft SQL Server, IBM's DB2, Microsoft Access, MySQL, or any other well-established RDBMS, you'll be able to get up to speed on that system with the information in this set of notes.



# SQL

- SQL is a complete relational database language in the sense that it contains both a data definition language (DDL) and a data manipulation language (DML). It also contains a data control language (DCL) used for authorization and security.
- We'll examine components of all three parts of SQL.
- If you use Microsoft Access, for example, you'll need to know less about the DDL side of SQL than you will if you use Oracle 9i or MySQL.
- The table on the following pages summarize the commands in the DDL portion of SQL. The entries in the table do not correspond to the order in which you will use the commands, but simply give a quick summary of those available. The table does not contain a complete listing of the commands in the DDL portion of SQL.



# Summary of SQL DDL Commands

Command or Option	Description
CREATE SCHEMA AUTHORIZATION	Creates a database schema
CREATE TABLE	Creates a new table in the user's DB schema
NOT NULL	Constraint that ensures a column will not have null values
UNIQUE	Constraint that ensures a column will not have duplicate values
PRIMARY KEY	Defines a primary key for a table
FOREIGN KEY	Defines a foreign key for a table
DEFAULT	Defines a default value for a column (when no value is given)
CHECK	Constraint used to validate data in a column
CREATE INDEX	Creates an index for a table
CREATE VIEW	Creates a dynamic subset of rows/columns from 1 or more tables
ALTER TABLE	Modifies a table's definition: adds/deletes/updates attributes or constraints
DROP TABLE	Permanently deletes a table (and thus its data) from the DB schema
DROP INDEX	Permanently deletes an index
DROP VIEW	Permanently deletes a view



# The DDL Component Of SQL

- Before you can use a RDMS two tasks must be completed: (1) create the database structure, and (2) create the tables that will hold the end-user data.
- Completion of the first task involves the construction of the physical files that hold the database. The RDBMS will automatically create the data dictionary tables and create a default database administrator (DBA).
  - Creating the physical files requires interaction between the host OS and the RDBMS. Therefore, creating the database structure is the one feature that tends to differ substantially from one RDBMS to another.
- With the exception of the creation of the database, most RDBMS vendors use SQL that deviates very little from ANSI standard SQL. Nevertheless, you might occasionally encounter minor syntactic differences. For example, most RDBMSs require that any SQL command be ended with a semicolon. However, some SQL implementations do not use a semicolon. I'll try to point out most of the common syntactic differences, or at least the ones of which I am aware.



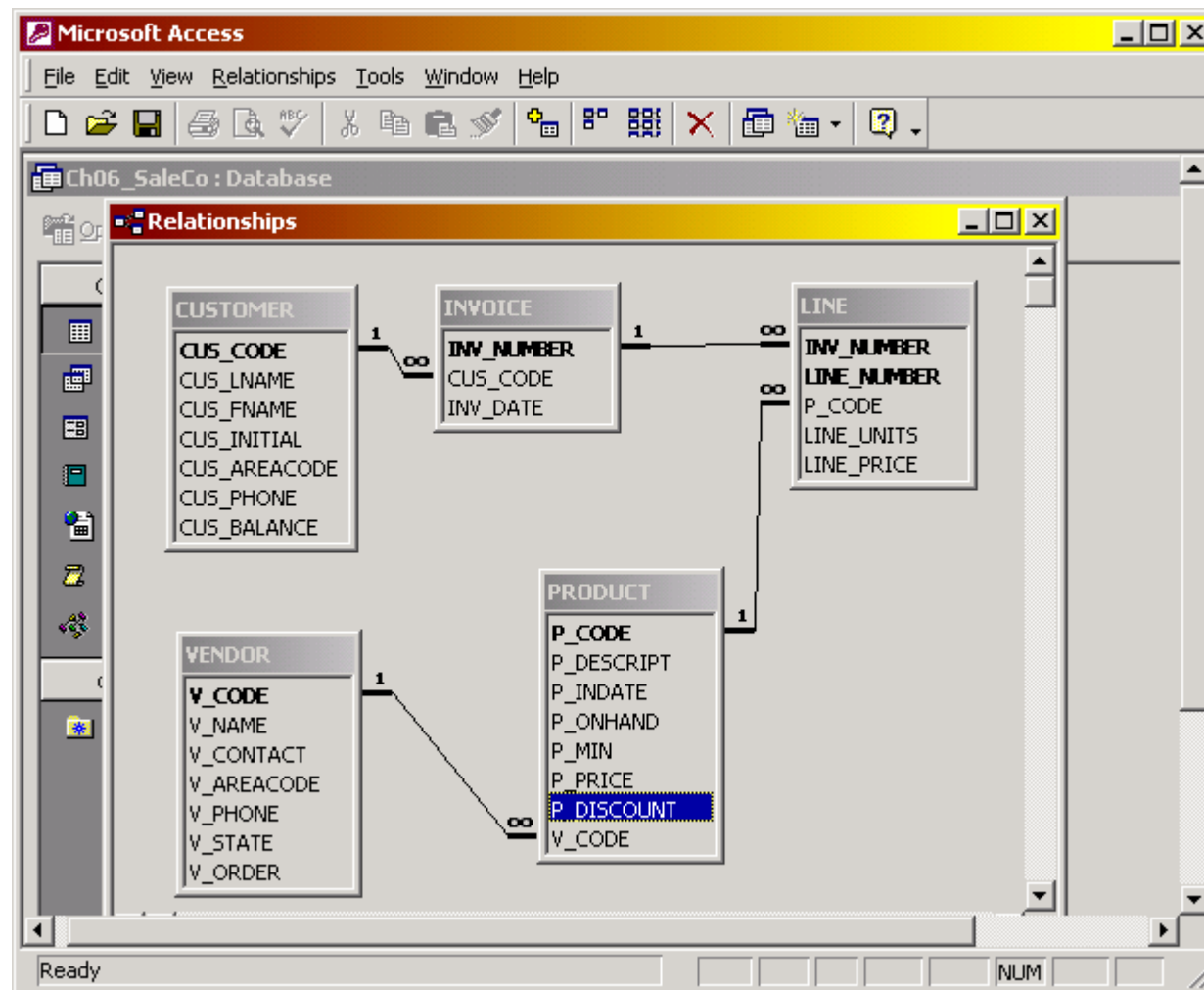
# Use Of DDL Commands In SQL

- We'll use the database shown on the next page for illustrating the DDL commands of SQL. This database is a bit more involved than our supplier-parts-jobs-ships database, but its along the same lines. The business rules that apply to this database are:
  1. A customer may generate many invoices. Each invoice is generated by one customer.
  2. An invoice contains one or more invoice lines. Each invoice line is associated with one invoice.
  3. Each invoice line references one product. A product may be found in many invoice lines. You can sell more than one hammer to more than one customer.
  4. A vendor may supply many products. Some vendors may not supply any products,
  5. If a product is vendor-supplied, that product is supplied by only one vendor.
  6. Some products are not supplied by a vendor, they may be made "in-house" or obtained through other means.





# An Example Database



# SQL Syntax Notation

Notation	Description
CAPITALS	Required SQL command keyword
<i>italics</i>	An end-user provided parameter – normally required
{a   b   ... }	A mandatory parameter, use one from option list
[...]	An optional parameter – everything in brackets is optional
<i>tablename</i>	The name of a table
<i>column</i>	The name of an attribute in a table
<i>data type</i>	A valid data type definition
<i>constraint</i>	A valid constraint definition
<i>condition</i>	A valid conditional expression – evaluates to true or false
<i>columnlist</i>	One or more column names or expressions separated by commas
<i>tablelist</i>	One or more table names separated by commas
<i>conditionlist</i>	One or more conditional expressions separated by logical operators
<i>expression</i>	A simple value (e.g., 76 or 'married') or a formula (e.g., price-10)



# Creating Table Structures Using SQL

- The CREATE TABLE syntax is:

```
CREATE TABLE tablename (  
    column1    data type    [constraint] [,  
    column2    data type    [constraint] ] [,  
    PRIMARY KEY (column1 [,column2] ) ] [,  
    FOREIGN KEY (column1 [,column2] ) REFERENCES tablename ] [,  
    CONSTRAINT constraint ] ) ;
```



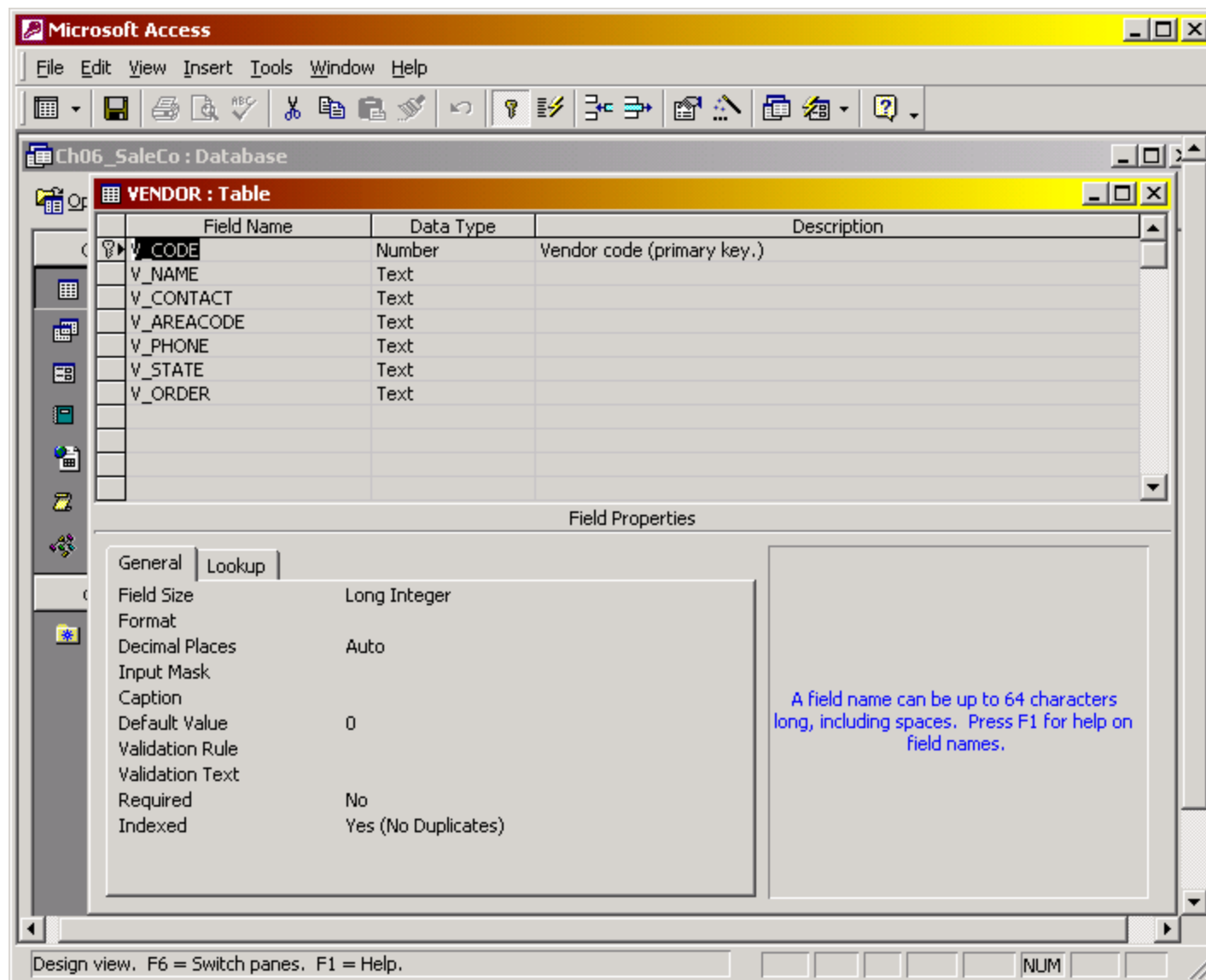
# Example – Table Creation in SQL

- As an example, let's create the VENDOR table as described on page 11.

```
CREATE TABLE VENDOR (  
    V_CODE          INTEGER          NOT NULL          UNIQUE,  
    V_NAME          VARCHAR(35)      NOT NULL,  
    V_CONTACT       VARCHAR(15)      NOT NULL,  
    V_AREACODE      CHAR(3)          NOT NULL,  
    V_PHONE         CHAR(8)          NOT NULL,  
    V_STATE         CHAR(2)          NOT NULL,  
    V_ORDER         CHAR(1)          NOT NULL,  
    PRIMARY KEY ( V_CODE));
```



# The VENDOR Table in Access



## Creating the Vendor Table in MySQL

C:\ Command Prompt (2) - mysql -u root -p

```
C:\Program Files\MySQL 4.1.9\MySQL Server 4.1\bin>mysql -u root -p
Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2 to server version: 4.1.9

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
```

```
mysql> create database sales;
Query OK, 1 row affected (0.06 sec)
```

Create database and switch to it

```
mysql> use sales;
Database changed
mysql> create table vendor (
  -> v_code int(9) NOT NULL,
  -> v_name text,
  -> v_contact text,
  -> v_areacode text,
  -> v_phonec text,
  -> v_state text,
  -> v_order text,
  -> primary key (v_code)
  -> );
```

Define table schema

```
Query OK, 0 rows affected (0.05 sec)
```

```
mysql> show tables;
+-----+
| Tables_in_sales |
+-----+
| vendor           |
+-----+
1 row in set (0.00 sec)
```

Show tables in the database

```
mysql> describe vendor
-> ;
```

Field	Type	Null	Key	Default	Extra
v_code	int(9)		PRI	0	
v_name	text	YES		NULL	
v_contact	text	YES		NULL	
v_areacode	text	YES		NULL	
v_phonec	text	YES		NULL	
v_state	text	YES		NULL	
v_order	text	YES		NULL	

Description of the Vendor table schema

```
7 rows in set (0.02 sec)
```

```
mysql>
```



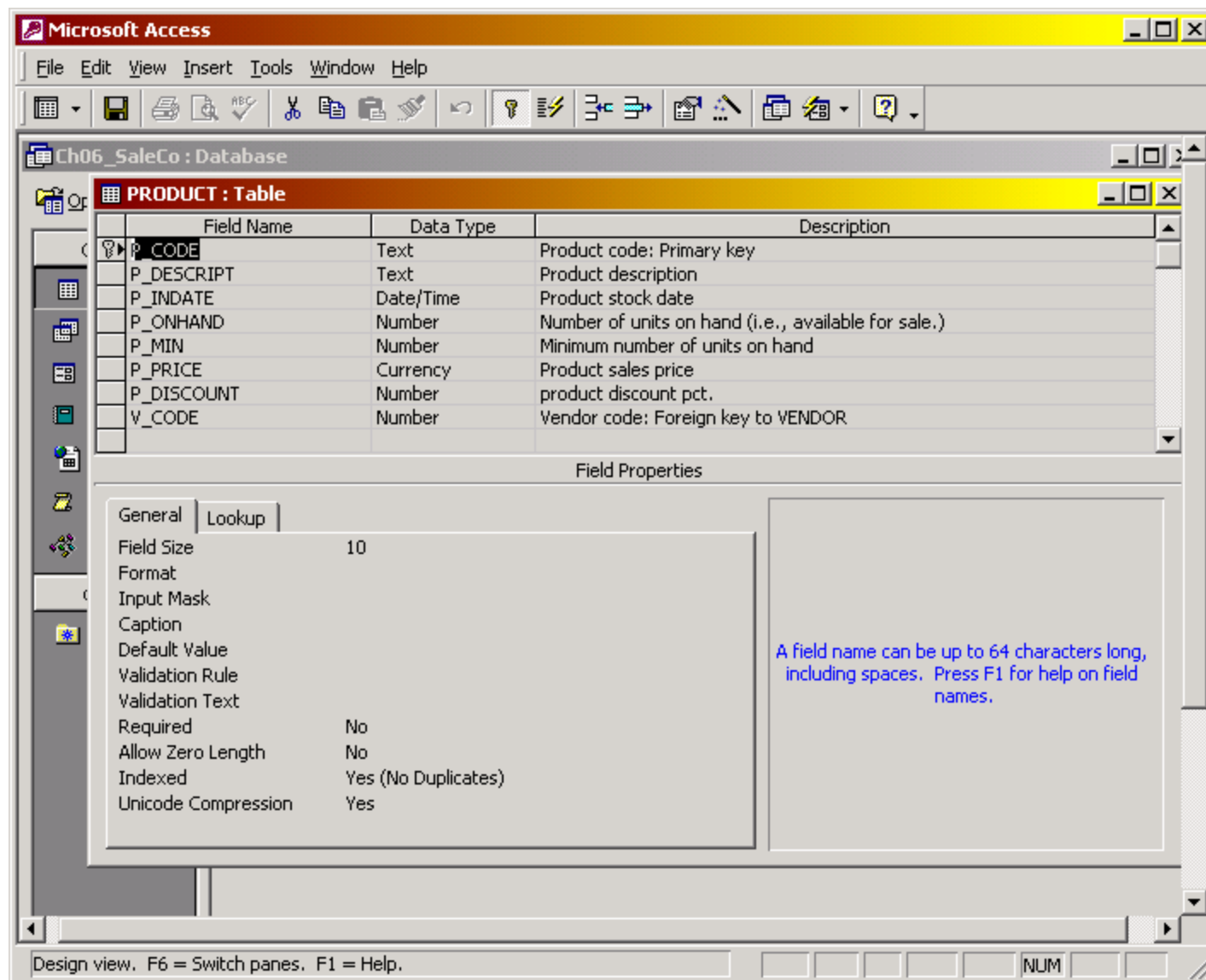
# Example – Table Creation

- Now let's create the PRODUCT table as described on page 11.

```
CREATE TABLE PRODUCT (  
    P_CODE          VARCHAR(10)    NOT NULL        UNIQUE,  
    P_DESCRIPT      VARCHAR(35)    NOT NULL,  
    P_INDATE        DATE           NOT NULL,  
    P_ONHAND        SMALLINT       NOT NULL,  
    P_MIN           SMALLINT       NOT NULL,  
    P_PRICE         NUMBER(8,2)    NOT NULL,  
    P_DISCOUNT     NUMBER(4,2)    NOT NULL,  
    V_CODE          INTEGER,  
    PRIMARY KEY ( P_CODE),  
    FOREIGN KEY (V_CODE) REFERENCES VENDOR ON UPDATE CASCADE);
```



# The PRODUCT Table in Access





# Example – Table Creation

- Now let's create the CUSTOMER table as described on page 11.

CREATE TABLE **CUSTOMER** (

CUS\_CODE                      NUMBER                      PRIMARY KEY,

CUS\_LNAME                    VARCHAR(15)                NOT NULL,

CUS\_FNAME                    VARCHAR(15)                NOT NULL,

CUS\_INITIAL                  CHAR(1),

CUS\_AREACODE   CHAR(3)                      DEFAULT '615' NOT NULL  
CHECK (CUS\_AREACODE IN ('615', '713', '931'));

CUS\_PHONE                    CHAR(8)                    NOT NULL,

CUS\_BALANCE                  NUMBER(9,2)                DEFAULT 0.00,

CONSTRAINT CUS\_UI1 UNIQUE (CUS\_LNAME, CUS\_FNAME));

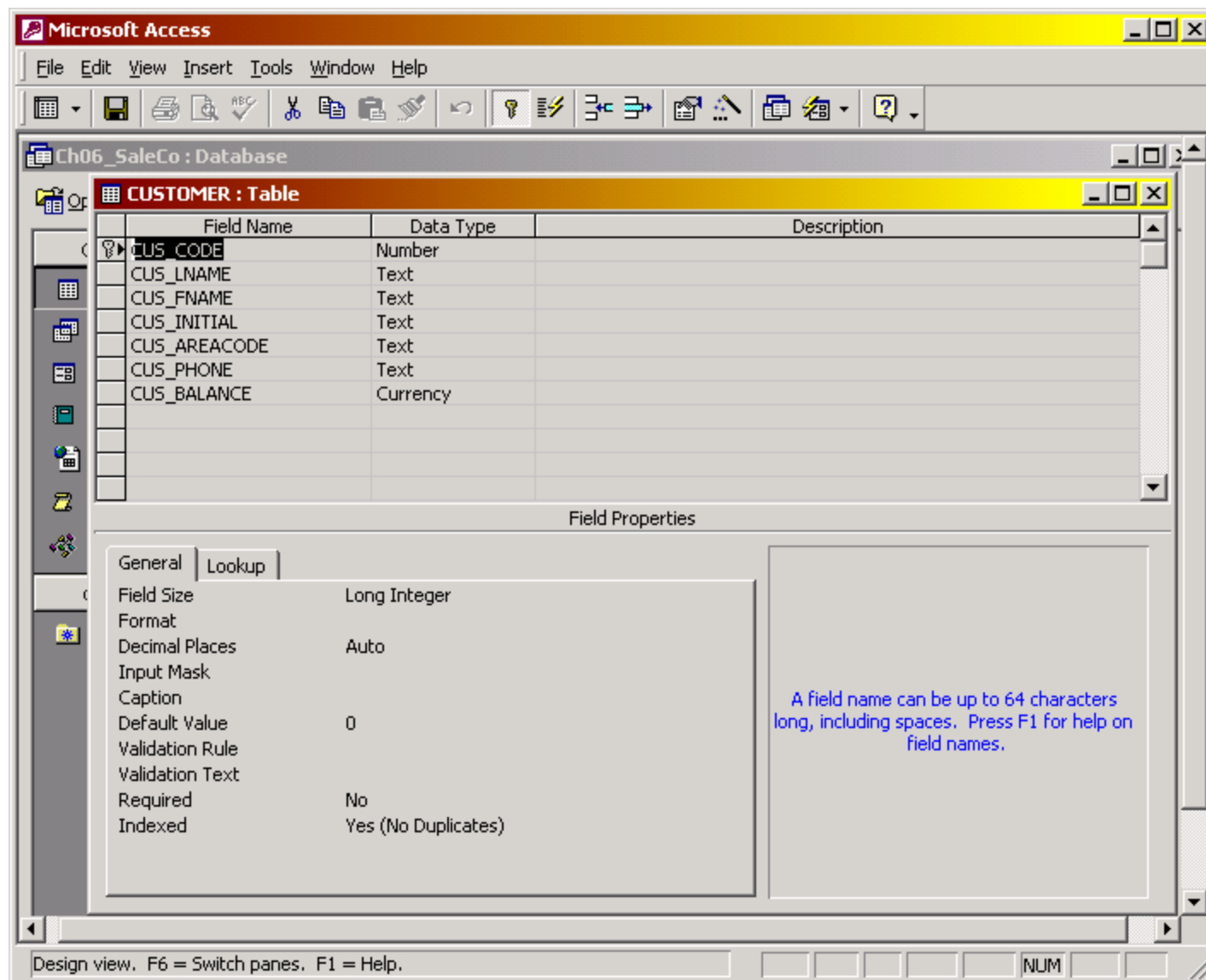
Column  
constraint

Table  
constraint

Creates a unique index constraint named CUS\_UI1  
on the customer's last name and first name.



# The CUSTOMER Table in Access



# Example – Table Creation

- Now let's create the INVOICE table as described on page 11.

```
CREATE TABLE INVOICE (  
    INV_NUMBER    NUMBER           PRIMARY KEY,  
    CUS_CODE      NUMBER           NOT NULL, REFERENCES CUSTOMER(CUS_CODE)  
    INV_DATE      DATE             DEFAULT SYSDATE NOT NULL,  
    CONSTRAINT INV_CK1 CHECK (INV_DATE > TO_DATE('01-JAN-2002', 'DD-MON-YYYY')));
```

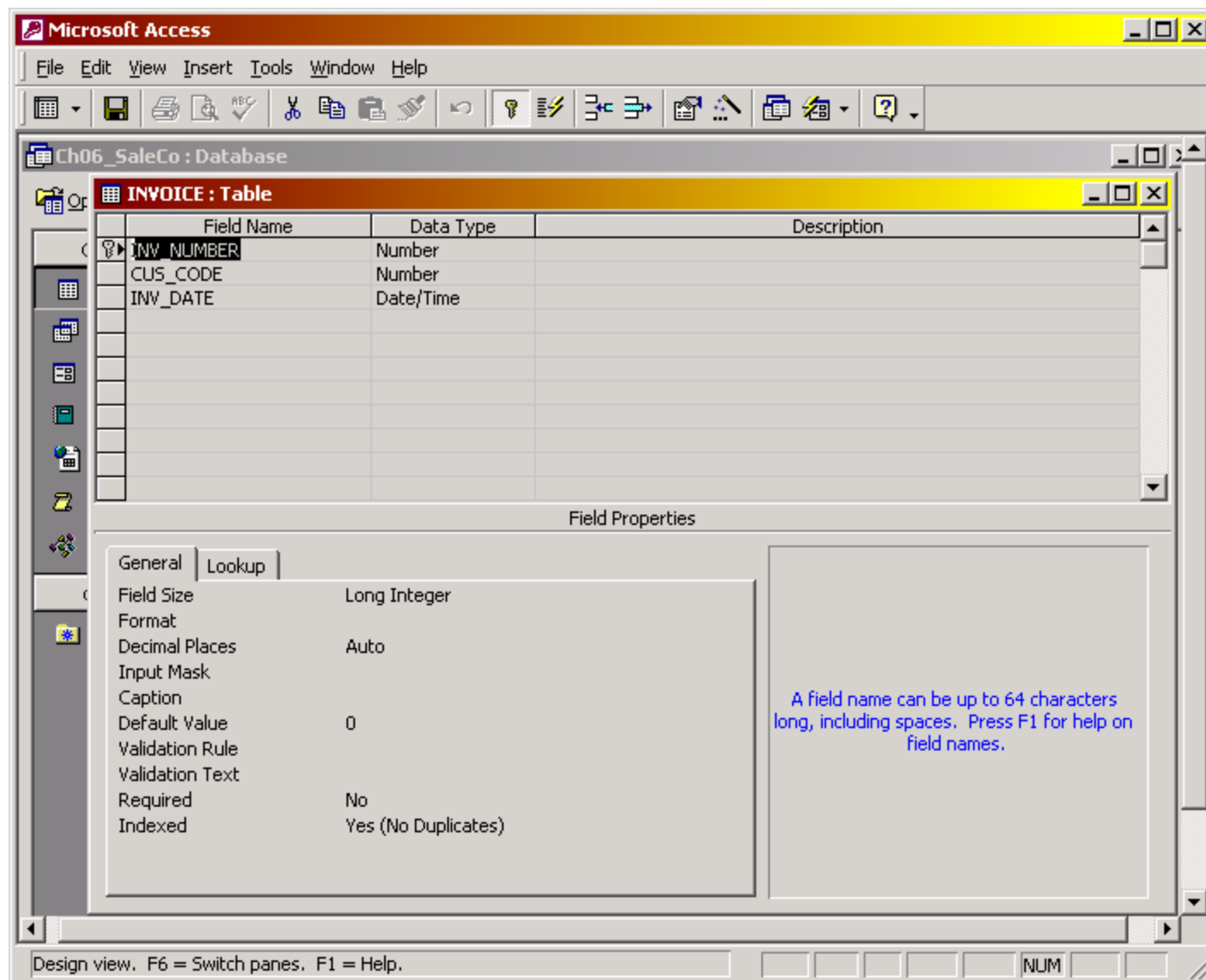
Alternative way to define a foreign key

Special function that returns today's date

Check constraint is used to validate that the invoice date is greater than January 1, 2002. The TO\_DATE function requires two parameters, the literal date and the date format used.



# The INVOICE Table in Access



# Example – Table Creation

- As a final example of table creation, let's create the LINE table as described on page 11.

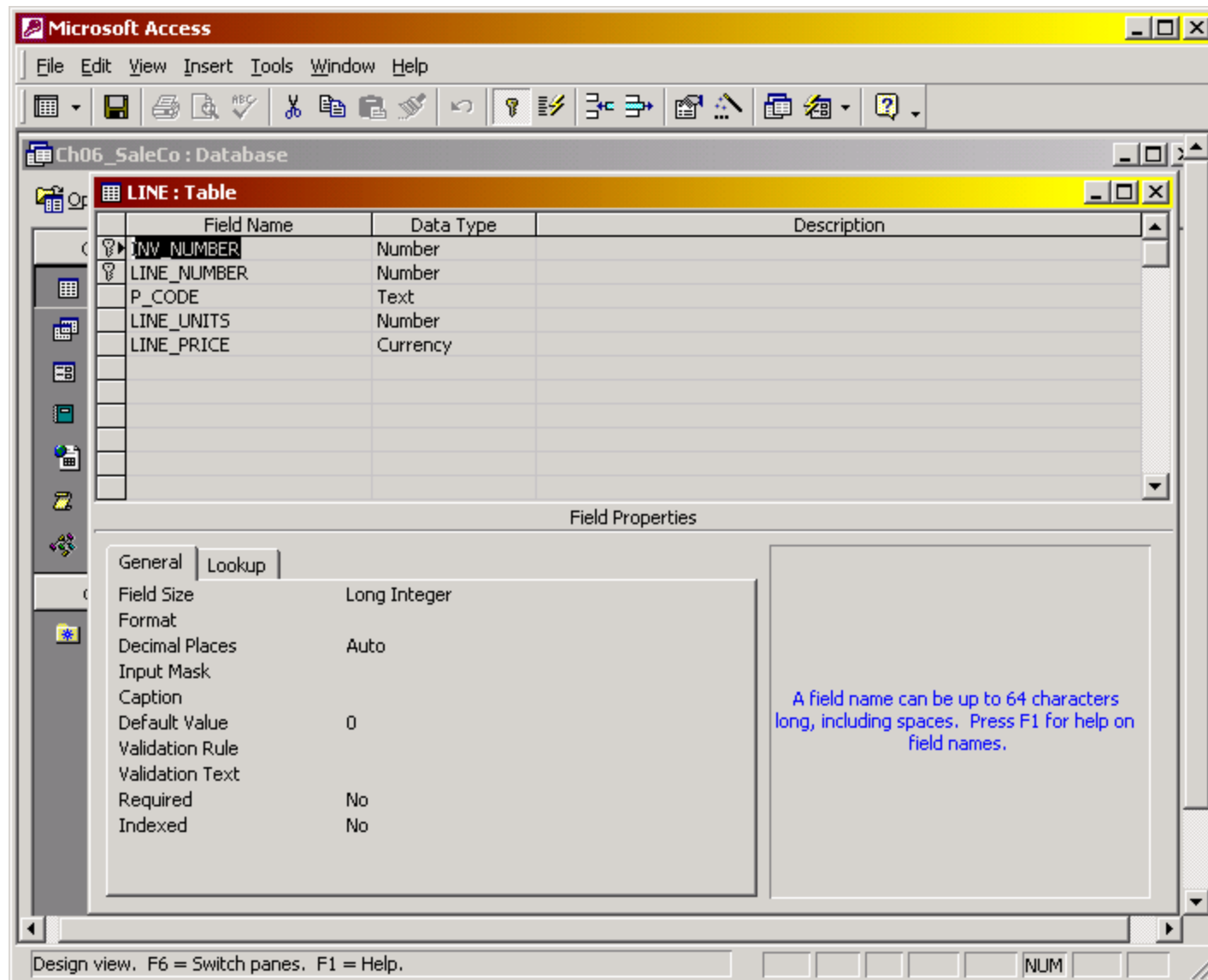
```
CREATE TABLE LINE (  
    INV_NUMBER      NUMBER          NOT NULL,  
    LINE_NUMBER     NUMBER(2,0)     NOT NULL,  
    P_CODE          VARCHAR(10)     NOT NULL,  
    LINE_UNITS      NUMBER(9,2)     DEFAULT 0.00 NOT NULL,  
    LINE_PRICE      NUMBER(9,2)     DEFAULT 0.00 NOT NULL,  
    PRIMARY KEY (INV_NUMBER, LINE_NUMBER),  
    FOREIGN KEY (INV_NUMBER) REFERENCES INVOICE ON DELETE CASCADE  
    FOREIGN KEY (P_CODE) REFERENCES PRODUCT(P_CODE),  
    CONSTRAINT LINE_UI1 UNIQUE(INV_NUMBER, P_CODE));
```

The use of ON DELETE CASCADE is recommended for weak entities to ensure that the deletion of a row in the strong entity automatically triggers the deletion of the corresponding rows in the dependent weak entity.

Table constraint prevents the duplication of an invoice line.



# The LINE Table in Access



# Some Notes On Table Creation

- Given our sample database, the PRODUCT table contains a foreign key that references the VENDOR table. Thus, the VENDOR table must be created first. In general, the table on the “1” side of a 1:M relationship must be created before the table on the “M” side can be created.
- In Oracle 9i, if you use the PRIMARY KEY designation you do not specify the NOT NULL and UNIQUE specifications. In fact, you will get an error message if you do so.
- ON UPDATE CASCADE is part of the ANSI standard but several RDBMSs do not support it. Oracle is one which does not support this specification.
- If the primary key is a composite key, all of the attributes of the key are contained within a set of parentheses and are separated by commas. For example, the table LINE on page 11 would have its primary key defined as:

PRIMARY KEY (inv\_number, line\_number).



# Steps For Table Creation

1. Identify the appropriate data type, including length, precision, and scale, if required, for each attributed.
2. Identify those columns that should accept null values. Column controls that indicate a column cannot be null are established when a table is created and are enforced for every update of the table when data are entered.
3. Identify those columns that need to be unique. Where a column is designated as **UNIQUE**, that column or set of columns is a candidate key. Although a table can have multiple candidate keys, only one candidate key may be designated as the **PRIMARY KEY**. Any columns designated as the **PRIMARY KEY** are implicitly assumed to be designated **NOT NULL**.
4. Identify all primary key-foreign key mates. Foreign keys can be established at the time of table creation or subsequently by altering the table. The parent table in such a parent-child relationship should be created first so that the child table will reference an existing parent table when it is created. The column constraint **REFERENCES** can be used to enforce referential integrity.





# Steps For Table Creation (cont.)

5. Determine the values to be inserted in any column for which a default value is desired. DEFAULT is used to define a value that is automatically inserted when no value is inserted during data entry.
6. Identify any columns for which domain specification may be state that are more constrained than those established by a data type. Using CHECK as a column constraint, it is possible to establish validation rules for values to be inserted into the database.
7. Create the table and any desired indices using the CREATE TABLE and CREATE INDEX statements.



# Some Notes On Table Creation (cont.)

- Support for referential constraints varies widely from RDBMS to RDBMS.
  - MS Access, SQL Server, and Oracle support ON DELETE CASCADE.
  - MS Access and SQL Server, support ON UPDATE CASCADE.
  - Oracle does not support ON UPDATE CASCADE.
  - Oracle supports SET NULL.
  - MS Access and SQL Server do not support SET NULL.
- MS Access does not support ON DELETE CASCADE or ON UPDATE CASCADE at the SQL line level, however, it does support it through the relationship window interface.

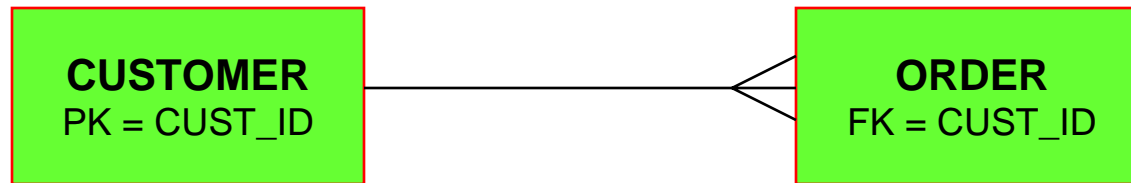


# Creating Data Integrity Controls

- In order to establish referential integrity controls between two tables with a 1:M relationship, the primary key of the table on the one side will be referenced by a column in the table on the many side of the relationship.
- Referential integrity means that a value in the matching column on the many side must correspond to a value in the primary key for some row in the table on the one side or be NULL.
- The REFERENCES clause prevents a foreign key value from being added if it is not already a valid value in the referenced primary key column, but there are other integrity issues that must be considered and handled.



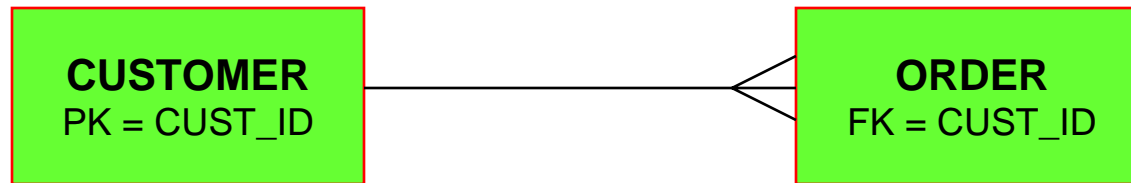
# Creating Data Integrity Controls (cont.)



- Consider the case shown above: If a **CUST\_ID** is changed, the connection between that customer and orders placed by that customer will be ruined (i.e., lost). The **REFERENCES** clause prevents making such a change in the foreign key value, but not in the primary key value.
- This problem could probably be handled by asserting that primary key values cannot be changed once they are established. In this case, updates to the Customer table would be handled using the **ON UPDATE RESTRICT** clause. Then any update that would delete or change a primary key value will be rejected unless no foreign key references that value in any child table.



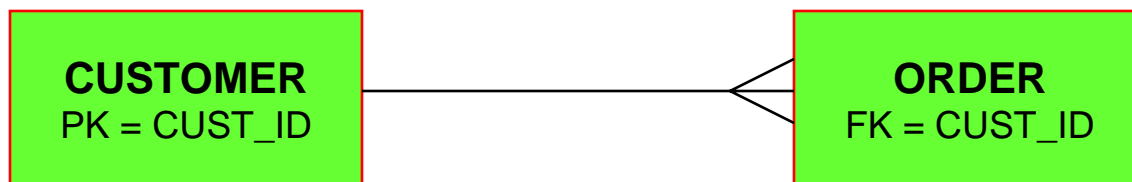
# Creating Data Integrity Controls (cont.)



- Another way to handle the change is to pass the update on to the child table(s) using the **ON UPDATE CASCADE** option. In this case, if a **CUST\_ID** is changed in the **CUSTOMER** table, the change will flow through (cascade) to the **ORDER** table.
- A third solution to this problem is to allow the update on the **CUSTOMER** table, but to change the involved **CUST\_ID** in the **ORDER** table to **NULL** by using the **ON UPDATE SET NULL** option. This would result in losing the connection between the order and the customer, which is not a desirable effect.



# Summary of Creating Data Integrity Controls



## **RESTRICTED UPDATE:**

Meaning in example:

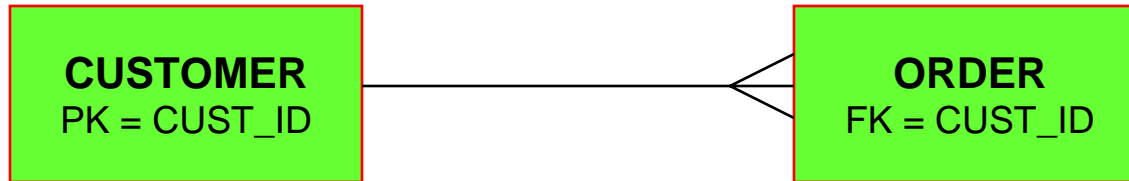
CUST\_ID can only be deleted if it is not found in the ORDER table.

Form:

```
CREATE TABLE customer
(cust_id INTEGER DEFAULT 'C999' NOT NULL
 CONSTRAINT cust_pk PRIMARY KEY (cust_id),
 ON UPDATE RESTRICT);
```



## Summary of Creating Data Integrity Controls (cont.)



### **CASCADED UPDATE:**

#### Meaning in example:

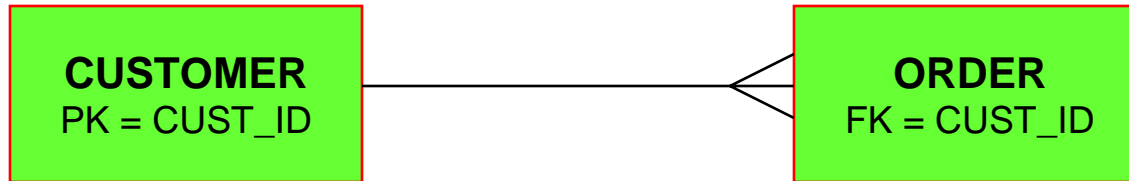
Changing a CUST\_ID in the CUSTOMER table will result in that value changing in the ORDER table.

#### Form:

```
CREATE TABLE customer
(cust_id INTEGER DEFAULT 'C999' NOT NULL
 CONSTRAINT cust_pk PRIMARY KEY (cust_id),
 ON UPDATE CASCADE);
```



## Summary of Creating Data Integrity Controls (cont.)



### SET NULL UPDATE:

#### Meaning in example:

Changing a **CUST\_ID** in the **CUSTOMER** table any **CUST\_ID** in the **ORDER** table that matches the old customer id will be set to **NULL**.

#### Form:

```
CREATE TABLE customer
(cust_id INTEGER DEFAULT 'C999' NOT NULL
 CONSTRAINT cust_pk PRIMARY KEY (cust_id),
 ON UPDATE SET NULL);
```





## Summary of Creating Data Integrity Controls (cont.)



### SET DEFAULT UPDATE:

#### Meaning in example:

Changing a CUST\_ID in the CUSTOMER table any CUST\_ID in the ORDER table that matches the old customer id will be set to a predefined value.

#### Form:

```
CREATE TABLE customer
(cust_id INTEGER DEFAULT 'C999' NOT NULL
CONSTRAINT cust_pk PRIMARY KEY (cust_id),
ON UPDATE SET DEFAULT);
```



# The DML Portion of SQL

- The DML portion of SQL can be viewed as two separate components which overlap in certain areas. The two components are the non-query DML commands and the query DML commands.
- Non-query DML commands allow you to populate tables (INSERT), modify data in tables (UPDATE), delete data from tables (DELETE) as well as make changes permanent (COMMIT) and undo changes (to some extent with ROLLBACK).
- The query DML commands essentially consist of a single statement (SELECT) with many different optional clauses.
- We'll look at the non-query part of the DML first.



# Summary of SQL DML Commands

Command or Option	Description
INSERT	Inserts row(s) into a table
SELECT	Selects attributes from rows in one or more tables or views
WHERE	Restricts the selection of rows based on a conditional expression
GROUP BY	Groups the selected rows based on one or more attributes
HAVING	Restricts the selection of grouped rows based on a condition
ORDER BY	Orders the selected rows
UPDATE	Modifies attribute values in one or more of a table's rows
DELETE	Deletes one or more rows from a table
COMMIT	Permanently saves data changes
ROLLBACK	Restores data to their original values
<i>Comparison Operators</i>	
=, <, >, <=, >=, <>	Used in conditional expressions
<i>Logical Operators</i>	
AND, OR, NOT	Used in conditional expressions



# Summary of SQL DML Commands (cont.)

Command or Option	Description
<i>Special Operators</i>	<i>used in conditional expressions</i>
BETWEEN	Checks whether an attributes values is within a range
IS NULL	Checks whether an attribute value is null
LIKE	Checks whether an attribute value matches a given string pattern
IN	Checks whether an attribute value matches any value within a value list
EXISTS	Checks if a subquery returns any rows or not
DISTINCT	Limits values to unique values, i.e., eliminates duplicates
<i>Aggregate Functions</i>	<i>used with SELECT to return mathematical summaries on columns</i>
COUNT	Returns the number of rows with non-null values for a given column
MIN	Returns the minimum attribute value found in a given column
MAX	Returns the maximum attribute value found in a given column
SUM	Returns the sum of all values for a given column
AVG	Returns the average of all values for a given column



# Adding Rows To Tables

- SQL requires the use of the INSERT command to enter data into a table.
- The syntax of the INSERT command is:

```
INSERT INTO tablename  
VALUES (value1, value 2, ...value n);
```



# Example - Adding Rows To Tables

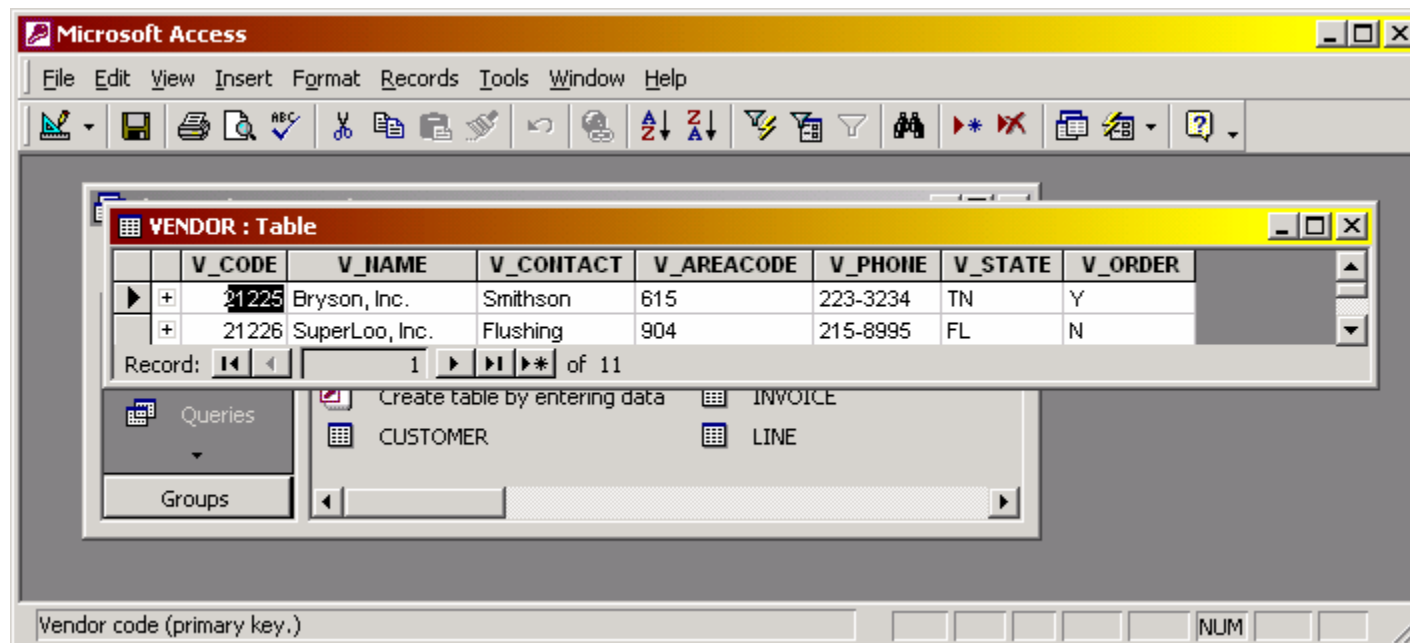
- In order to add the two rows to the VENDOR table shown below, we would need to execute the following two SQL commands:

```
INSERT INTO VENDOR
```

```
VALUES (21225, 'Bryson, Inc.', 'Smithson', '615', '223-3234', 'TN', 'Y');
```

```
INSERT INTO VENDOR
```

```
VALUES (21226, 'SuperLoo, Inc.', 'Flushing', '904', '215-8995', 'FL', 'N');
```

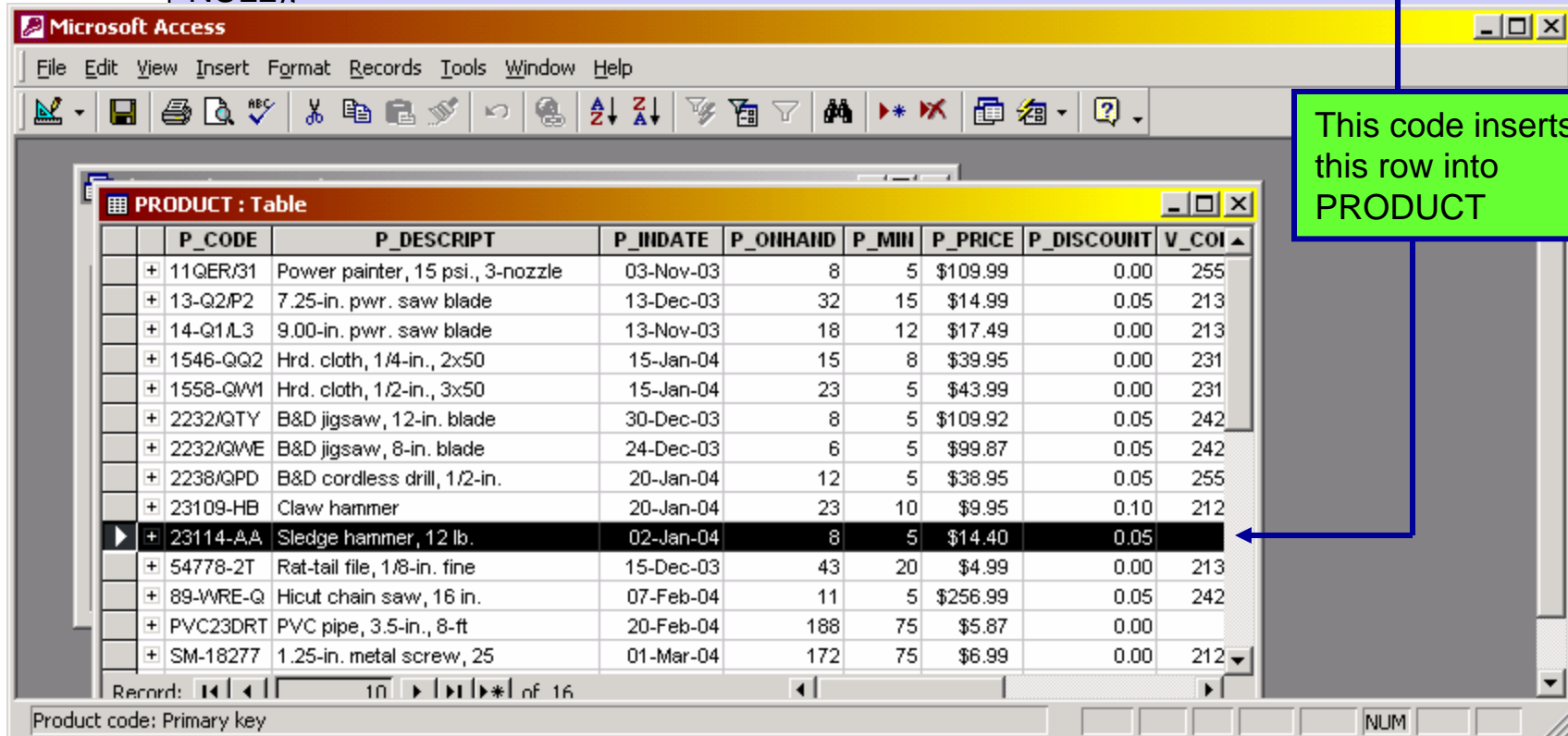


# Example - Adding Rows With Nulls To Tables

- If an attribute in a row has no value (i.e., is null) you would use the following syntax to enter the row into the table:

```
INSERT INTO PRODUCT
```

```
VALUES ('23114-AA', 'Sledge hammer, 12 lb.', '02-Jan-02', 8, 5, 14.40, 0.05,  
NULL);
```



The screenshot shows the Microsoft Access interface with the 'PRODUCT : Table' window open. The table contains the following data:

	P_CODE	P_DESCRIPT	P_INDATE	P_ONHAND	P_MIN	P_PRICE	P_DISCOUNT	V_COI
+	11QER/31	Power painter, 15 psi., 3-nozzle	03-Nov-03	8	5	\$109.99	0.00	255
+	13-Q2/P2	7.25-in. pwr. saw blade	13-Dec-03	32	15	\$14.99	0.05	213
+	14-Q1/L3	9.00-in. pwr. saw blade	13-Nov-03	18	12	\$17.49	0.00	213
+	1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15-Jan-04	15	8	\$39.95	0.00	231
+	1558-QW1	Hrd. cloth, 1/2-in., 3x50	15-Jan-04	23	5	\$43.99	0.00	231
+	2232/QTY	B&D jigsaw, 12-in. blade	30-Dec-03	8	5	\$109.92	0.05	242
+	2232/QWE	B&D jigsaw, 8-in. blade	24-Dec-03	6	5	\$99.87	0.05	242
+	2238/QPD	B&D cordless drill, 1/2-in.	20-Jan-04	12	5	\$38.95	0.05	255
+	23109-HB	Claw hammer	20-Jan-04	23	10	\$9.95	0.10	212
▶	23114-AA	Sledge hammer, 12 lb.	02-Jan-04	8	5	\$14.40	0.05	
+	54778-2T	Rat-tail file, 1/8-in. fine	15-Dec-03	43	20	\$4.99	0.00	213
+	89-WRE-Q	Hicut chain saw, 16 in.	07-Feb-04	11	5	\$256.99	0.05	242
+	PVC23DRT	PVC pipe, 3.5-in., 8-ft	20-Feb-04	188	75	\$5.87	0.00	
+	SM-18277	1.25-in. metal screw, 25	01-Mar-04	172	75	\$6.99	0.00	212

The new row (23114-AA) is highlighted. A green callout box with the text 'This code inserts this row into PRODUCT' points to the new row. The status bar at the bottom shows 'Record: 10 of 16' and 'Product code: Primary key'.



## Example - Adding Rows With Optional Values To Tables

- There may be occasions on which more than one attribute is optional (i.e., can be null). Rather than declaring each attribute as NULL in the INSERT command, you can just indicate the attributes that have required values.
- This is done by listing the attribute names for which values are being inserted inside parentheses after the table name.
- For the purposes of example, suppose that only the P\_CODE and P\_DESCRIPT are required attributes in the PRODUCT table. If this is the case, then either of the following syntactic forms could be used:

```
INSERT INTO PRODUCT
```

```
VALUES ('23114-AA', 'Sledge hammer, 12 lb.', NULL, NULL, NULL, NULL, NULL, NULL);
```

-or-

```
INSERT INTO PRODUCT(P_CODE, P_DESCRIPT)
```

```
VALUES('23114-AA', 'Sledge hammer, 12 lb.');
```





# Deleting Rows From A Table

- It is easy to use SQL to delete a row from a table. This is handled via the DELETE command.
- The syntax of the DELETE command is:

```
DELETE FROM tablename  
[WHERE conditionlist];
```

- To delete a row of a table based on a primary key value you would use a command such as:

```
DELETE FROM PRODUCT  
WHERE P_CODE = '23114-AA';
```



## Deleting Rows From A Table (cont.)

- Deletion also works to remove potentially multiple rows from a table.
  - For example, suppose that we want to delete every product from the PRODUCT table where the value of the P\_MIN attribute is equal to 5. To accomplish this you would issue the following command:

```
DELETE FROM PRODUCT  
  
WHERE P_MIN = 5;
```

- DELETE is a set-oriented command. This means that since the WHERE condition is optional, if it is not specified, all rows from the specified table will be deleted!



# Updating the Rows of a Table

- To modify the data within a table the UPDATE command is used.
- The syntax of the UPDATE command is:

```
UPDATE    tablename  
    SET columnname = expression [, columnname = expression ]  
    [ WHERE conditionlist ];
```

- Notice that the WHERE condition is optional in the UPDATE command. If the WHERE condition is omitted, then the update is applied to all rows of the specified table.



## Updating the Rows of a Table (cont.)

- As an example, suppose that we want to modify the P\_INDATE from December 13, 2003 to January 18, 2004 in the second row of the PRODUCT table. We need to use the primary key value 13-Q2/P2 to locate the correct row of the table, which gives the following command syntax:

```
UPDATE PRODUCT
  SET P_INDATE = '18-Jan-2004'
  WHERE P_CODE = '13-Q2/P2';
```

- If more than one attribute is to be updated in a row, the updates are separated by commas:

```
UPDATE PRODUCT
  SET P_INDATE = '18-JAN-2004', P_PRICE = 16.99, P_MIN = 10
  WHERE P_CODE = '13-Q2/P2';
```



# Saving Changes to a Table

- Any changes made to the table contents are not physically saved into the underlying physical table (the file system) until a COMMIT command has been executed.
- Depending on the sophistication of the system on which you are working, if the power should fail during the updating of a table (or database in general), before the COMMIT command was executed, your modifications are simply lost. More sophisticated systems will be able to recover from such disasters, but for small PC-based systems you'd better have a UPS installed!
- The syntax for the COMMIT command is:

```
COMMIT [ tablename ];
```

-or-

```
COMMIT; //saves all changes made in any modified tables
```



# Restoring Table Contents

- If you have not yet used the COMMIT command to permanently store the changes in the database, you can restore the database to its previous state (i.e., the one that was the result of the last COMMIT) with the ROLLBACK command.
- ROLLBACK undoes any changes made and brings the data back to the values that existed before the changes were made.
- The syntax for the ROLLBACK command is:

```
ROLLBACK;
```

- MS Access does not support ROLLBACK! Some RDBMSs like Oracle automatically COMMIT data changes when issuing DDL commands, so ROLLBACK won't do anything on these systems.
- ROLLBACK rolls back everything since the last COMMIT, which means that even changes that you might not want undone will be if no commit has been issued.



# Summary of SQL Non-Query DML Commands

- As you can see, data entry is rather cumbersome in SQL.
- End-user applications are best created with utilities that generate attractive and easy to use input screens.

The screenshot shows a Microsoft Access window titled 'Microsoft Access - [PRODUCT]'. The menu bar includes File, Edit, View, Insert, Format, Records, Tools, Window, and Help. The toolbar contains various icons for file operations, editing, and data manipulation. The main window displays a form titled 'PRODUCT Table Data View and Data Entry'. The form has a purple header and a light blue body. It contains several input fields for data entry:

Product code:	11QER/31
Description:	Power painter, 15 psi., 3-nozzle
Stock date::	03-Nov-03
Units on hand:	8
Minimum units:	5
Price:	\$109.99
Discount rate:	0.00
Vendor code:	25595

To the right of the input fields is a cartoon illustration of a duck holding a paintbrush, standing next to a red pedestal holding a computer monitor. Below the illustration is the text 'Duck Data Entry System'. At the bottom of the form is a button labeled 'Close the product form'. The status bar at the bottom of the window shows 'Record: 1 of 16' and 'Product code: Primary key'.



# Query Portion of the DML of SQL

- The query portion of the DML of SQL consists of a single command called the SELECT command.
- The syntax of the SELECT command is:

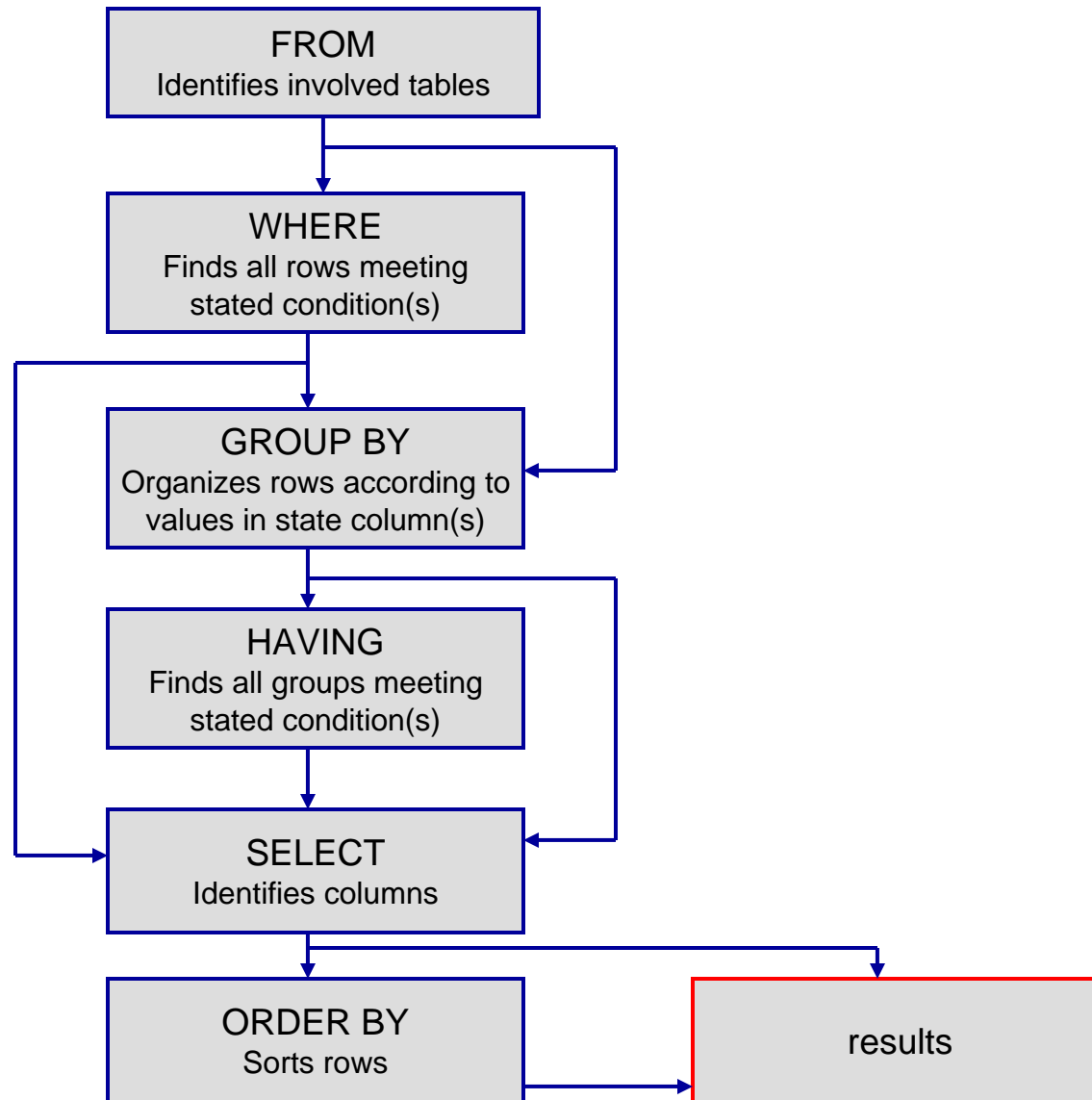
```
SELECT [ ALL | DISTINCT] columnlist  
FROM tablelist  
[ WHERE condition ]  
[ GROUP BY columnlist ]  
[ HAVING condition ]  
[ ORDER BY columnlist ];
```

- We'll examine most of the features of the SELECT command, starting with simple queries and working our way toward more complex queries. I'll continue to use the same database that we've developed in this set of notes.





# SELECT Statement Processing Order



# Simple Selection Queries in SQL

- Perhaps the simplest query to form is that which retrieves every row from some specified table.
- For example, suppose that we wanted to list every attribute value in every row of the PRODUCT table. In other words, to view this table. The following command will accomplish this task:

```
SELECT P_CODE, P_DESCRIPT, P_INDATE, P_ONHAND, P_MIN,  
       P_PRICE, P_DISCOUNT, V_CODE  
FROM PRODUCT;
```

-Or-

```
SELECT *  
FROM PRODUCT;
```

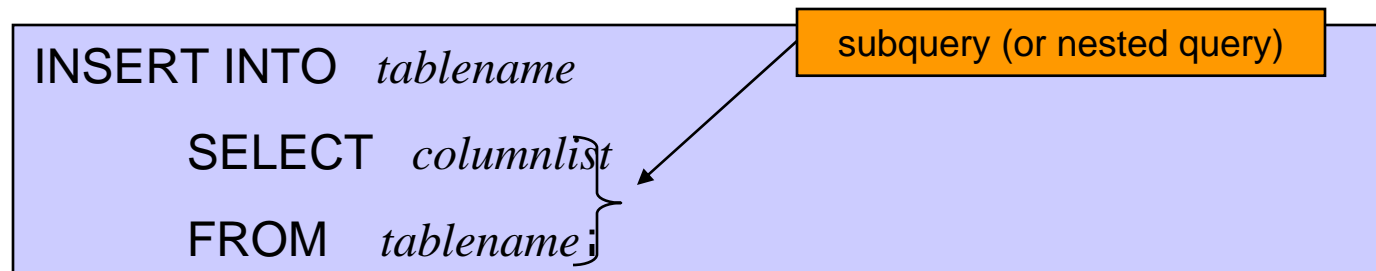
\* is a wildcard character representing all attributes in a table



# Inserting Table Rows with a Select Subquery

- Although this is technically a non-query DML operation, it also includes a query command, so I've included an example here before we move on to more complex query expressions.
- SQL allows you to enter rows into a table using the data from another table as the populating basis. The syntax for this type of insert command is:

```
INSERT INTO tablename
    SELECT columnlist
    FROM tablename;
```



- The inner query is always executed first by the RDBMS and the values extracted by the inner query will be used as input to the outer query (in this case the INSERT command). The values returned by the inner query must match the attributes and data types of the table in the INSERT statement.



# Selection Queries With Conditional Restrictions

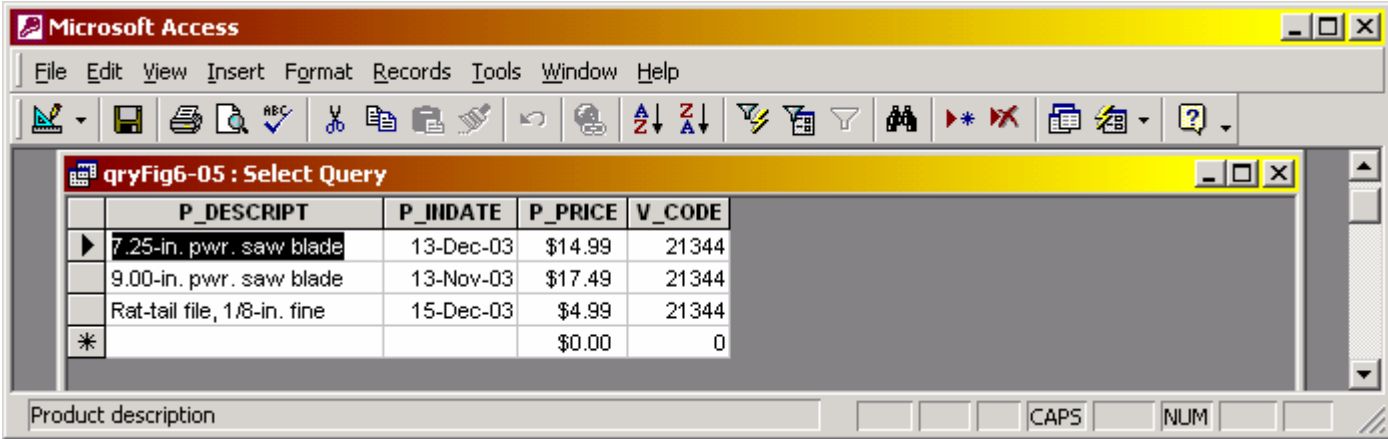
- You can select partial table contents by placing restrictions on the rows to be included in the result. This is accomplished using the WHERE clause:

```
SELECT columnlist  
FROM tablelist  
WHERE conditionlist ;
```

- The SELECT statement will retrieve all rows that match the specified condition(s) specified in the WHERE clause.

- For example: 

```
SELECT P_DESCRIPT, P_INDATE, P_PRICE, V_CODE  
FROM PRODUCT  
WHERE V_CODE = 21344;
```



The screenshot shows the Microsoft Access application window. The title bar reads 'Microsoft Access'. The menu bar includes File, Edit, View, Insert, Format, Records, Tools, Window, and Help. The toolbar contains various icons for file operations, editing, and querying. The main window displays a query named 'qryFig6-05 : Select Query'. The query results are shown in a table with the following data:

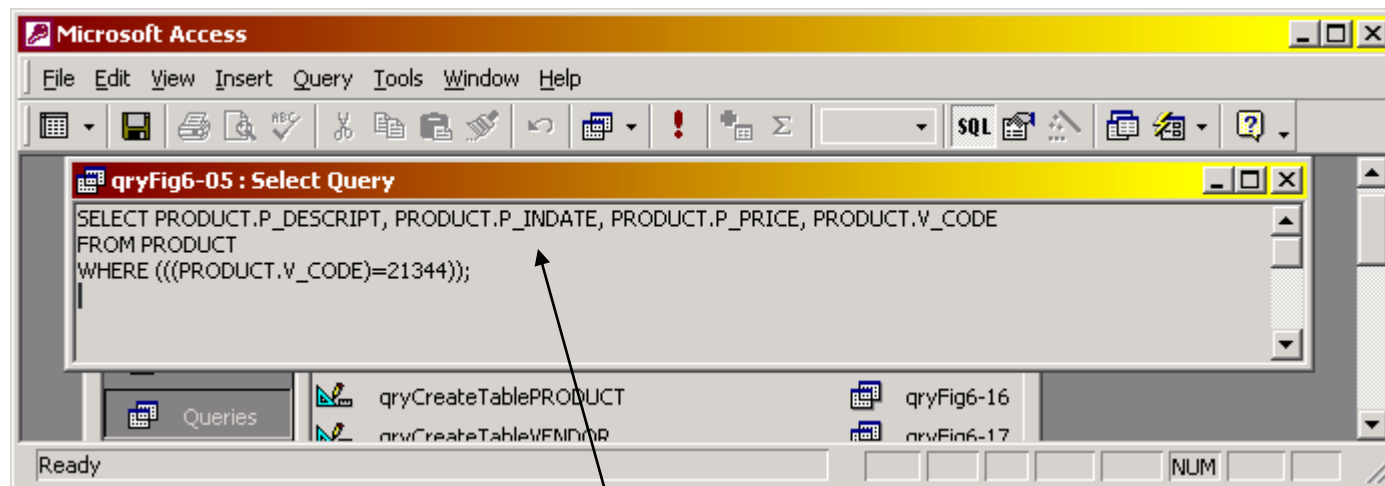
	P_DESCRIPT	P_INDATE	P_PRICE	V_CODE
▶	7.25-in. pwr. saw blade	13-Dec-03	\$14.99	21344
	9.00-in. pwr. saw blade	13-Nov-03	\$17.49	21344
	Rat-tail file, 1/8-in. fine	15-Dec-03	\$4.99	21344
*			\$0.00	0

At the bottom of the window, there is a text box labeled 'Product description' and several buttons, including 'CAPS' and 'NUM'.



## Note on Access QBE Interface for SQL

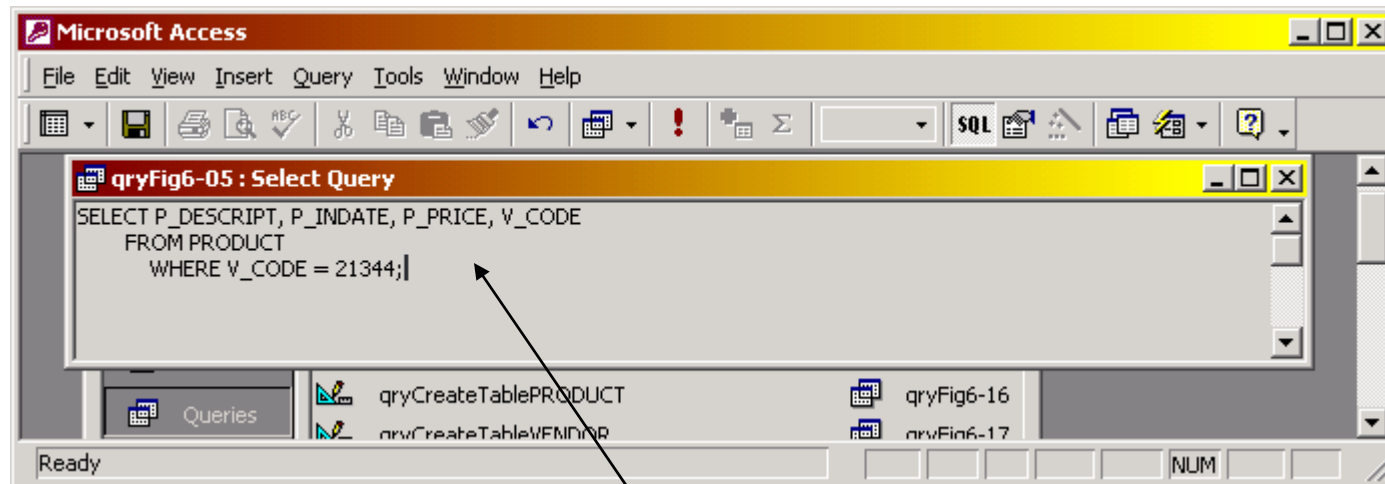
- Microsoft Access provides the Access QBE query generator. Although Access QBE generates its own “native” version of SQL, you can also elect to type standard SQL in the Access SQL window as shown on the next page.



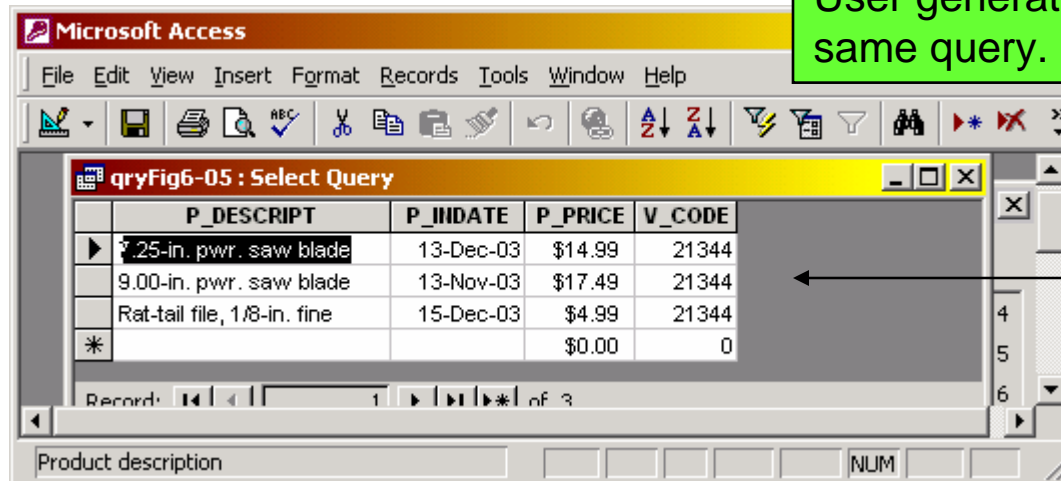
Access QBE “native” SQL code for the query on the previous page.



# Note on Access QBE Interface for SQL



User generated SQL code for the same query.



Results of the user generated SQL code showing the same set of tuples as before in the result.



# Conditional Restrictions in SQL Queries

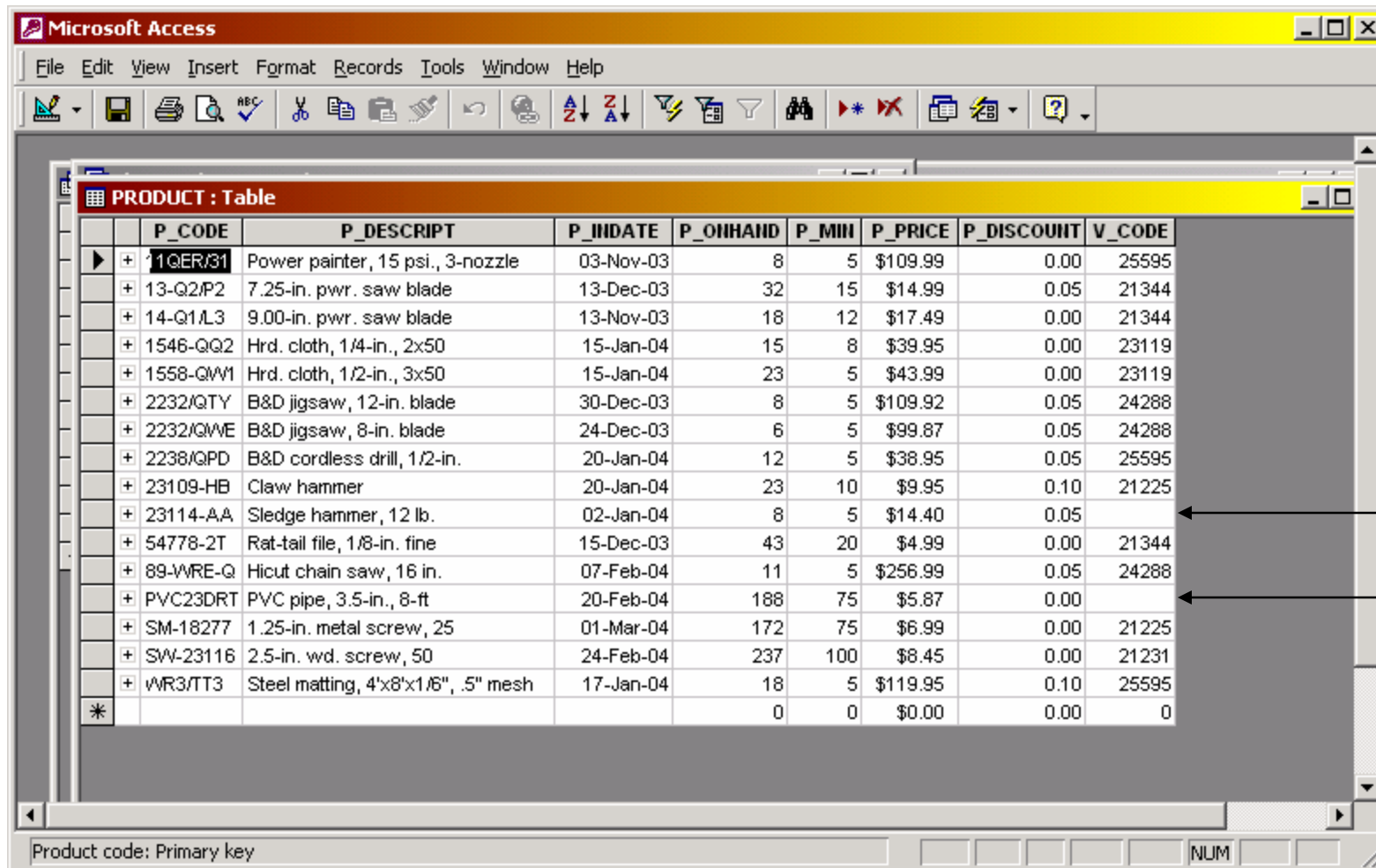
- The SQL command structure provides almost limitless query flexibility. Numerous conditional restrictions may be placed on the selected table contents.
- Unless specifically testing for attribute values which are null, SQL does not include rows for which a selected attribute value is null in the result.
- Consider the following query:

```
SELECT P_DESCRIPT, P_INDATE, P_PRICE, V_CODE  
FROM PRODUCT  
WHERE V_CODE <> 21344;
```

- The PRODUCT table is shown on the next page and the output from this query is shown on the following page. Notice that rows 10 and 13 in the PRODUCT table do not appear in the results of this query.



# Conditional Restrictions in SQL Queries (cont.)



Microsoft Access

File Edit View Insert Format Records Tools Window Help

PRODUCT : Table

	P_CODE	P_DESCRIPT	P_IIDATE	P_ONHAND	P_MIH	P_PRICE	P_DISCOUNT	V_CODE
+	11QER/31	Power painter, 15 psi., 3-nozzle	03-Nov-03	8	5	\$109.99	0.00	25595
+	13-Q2/P2	7.25-in. pwr. saw blade	13-Dec-03	32	15	\$14.99	0.05	21344
+	14-Q1/L3	9.00-in. pwr. saw blade	13-Nov-03	18	12	\$17.49	0.00	21344
+	1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15-Jan-04	15	8	\$39.95	0.00	23119
+	1558-QMVI	Hrd. cloth, 1/2-in., 3x50	15-Jan-04	23	5	\$43.99	0.00	23119
+	2232/QTY	B&D jigsaw, 12-in. blade	30-Dec-03	8	5	\$109.92	0.05	24288
+	2232/QMVE	B&D jigsaw, 8-in. blade	24-Dec-03	6	5	\$99.87	0.05	24288
+	2238/QPD	B&D cordless drill, 1/2-in.	20-Jan-04	12	5	\$38.95	0.05	25595
+	23109-HB	Claw hammer	20-Jan-04	23	10	\$9.95	0.10	21225
+	23114-AA	Sledge hammer, 12 lb.	02-Jan-04	8	5	\$14.40	0.05	
+	54778-2T	Rat-tail file, 1/8-in. fine	15-Dec-03	43	20	\$4.99	0.00	21344
+	89-WRE-Q	Hicut chain saw, 16 in.	07-Feb-04	11	5	\$256.99	0.05	24288
+	PVC23DRT	PVC pipe, 3.5-in., 8-ft	20-Feb-04	188	75	\$5.87	0.00	
+	SM-18277	1.25-in. metal screw, 25	01-Mar-04	172	75	\$6.99	0.00	21225
+	SW-23116	2.5-in. wd. screw, 50	24-Feb-04	237	100	\$8.45	0.00	21231
+	WR3/TT3	Steel matting, 4'x8'x1/8", .5" mesh	17-Jan-04	18	5	\$119.95	0.10	25595
*				0	0	\$0.00	0.00	0

Product code: Primary key

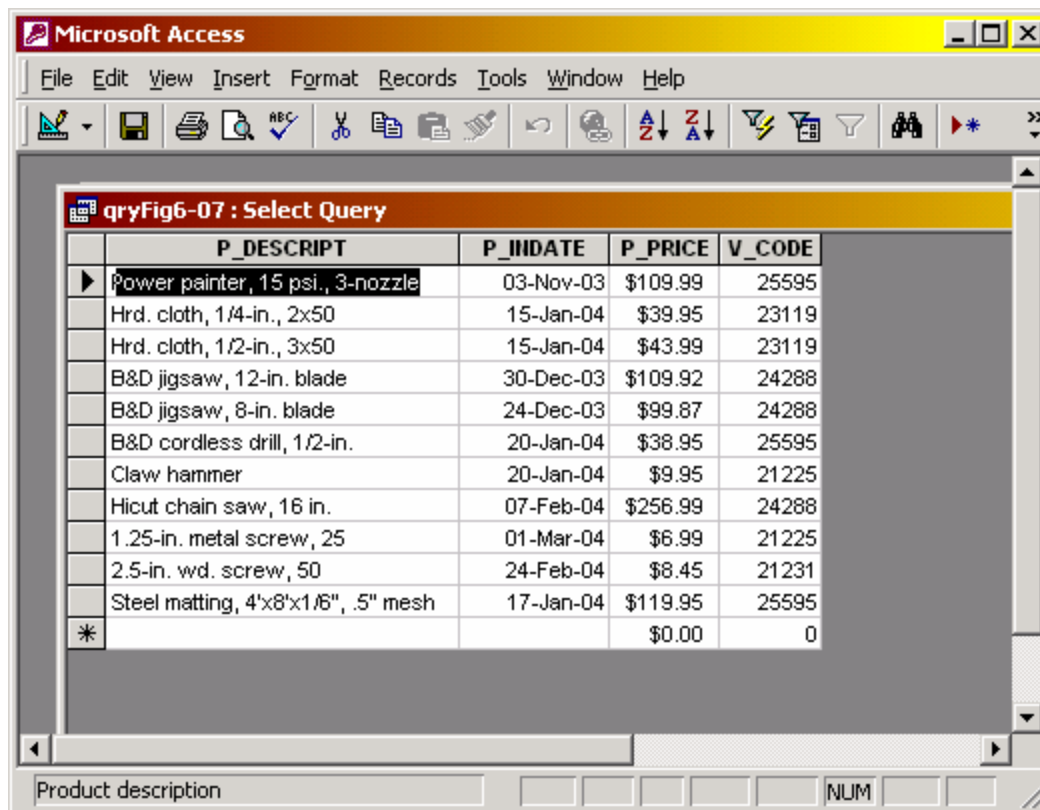
NUM

These two rows do not appear in the result on the following page.





# Conditional Restrictions in SQL Queries (cont.)



The screenshot shows the Microsoft Access application window with a menu bar (File, Edit, View, Insert, Format, Records, Tools, Window, Help) and a toolbar. The main window displays a query titled "qryFig6-07 : Select Query". The query results are shown in a table with the following columns: P\_DESCRIPT, P\_INDATE, P\_PRICE, and V\_CODE. The data is as follows:

P_DESCRIPT	P_INDATE	P_PRICE	V_CODE
Power painter, 15 psi., 3-nozzle	03-Nov-03	\$109.99	25595
Hrd. cloth, 1/4-in., 2x50	15-Jan-04	\$39.95	23119
Hrd. cloth, 1/2-in., 3x50	15-Jan-04	\$43.99	23119
B&D jigsaw, 12-in. blade	30-Dec-03	\$109.92	24288
B&D jigsaw, 8-in. blade	24-Dec-03	\$99.87	24288
B&D cordless drill, 1/2-in.	20-Jan-04	\$38.95	25595
Claw hammer	20-Jan-04	\$9.95	21225
Hicut chain saw, 16 in.	07-Feb-04	\$256.99	24288
1.25-in. metal screw, 25	01-Mar-04	\$6.99	21225
2.5-in. wd. screw, 50	24-Feb-04	\$8.45	21231
Steel matting, 4'x8'x1/8", .5" mesh	17-Jan-04	\$119.95	25595
*		\$0.00	0

At the bottom of the window, there is a status bar with the text "Product description" and a "NUM" button.

Results of the query:

```
SELECT P_SDESCRIPT,  
       P_INDATE, P_PRICE,  
       V_CODE  
FROM PRODUCT  
WHERE  
       V_CODE <> 21344;
```



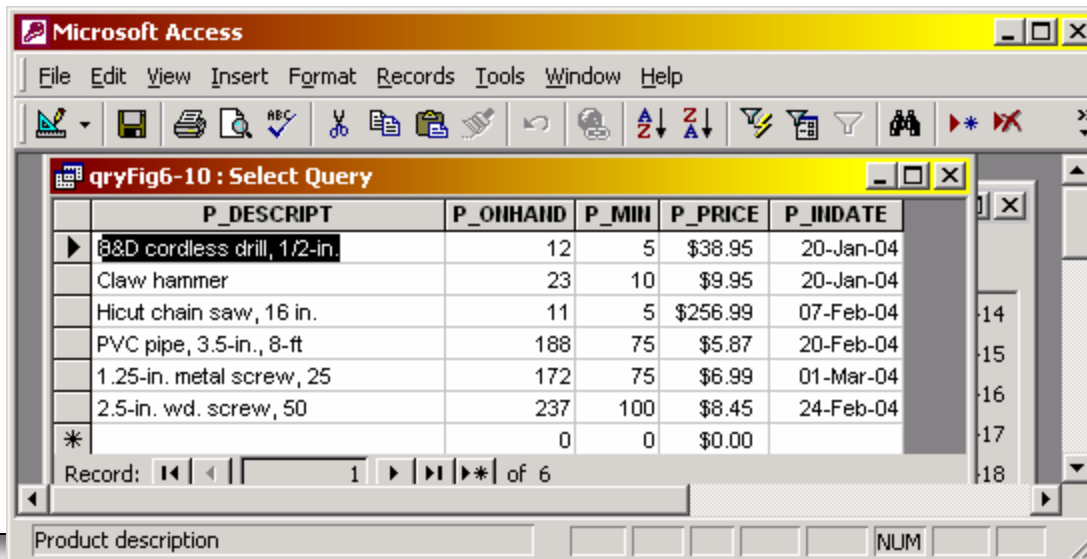
# Comparisons Involving Dates in SQL Queries

- Date procedures are often more software-specific than most other SQL procedures. For example, the query to list all the rows in which the inventory stock dates occur on or after January 20, 2004, will look like this:

```
SELECT P_DESCRIPT, P_ONHAND, P_MIN, P_PRICE, P_INDATE
FROM PRODUCT
WHERE P_INDATE >= "20-Jan-2004";
```

- Note that in Access the delimiters for dates is #, so in Access this query would look like:

```
SELECT P_DESCRIPT, P_ONHAND, P_MIN, P_PRICE, P_INDATE
FROM PRODUCT
WHERE P_INDATE >= #20-Jan-2004#;
```



# Using Computed Columns and Column Aliases

- Suppose that your query needs to determine a value which is not physically stored in the database but is calculated from data that is in the database.
- For example, let's suppose that we want to determine the total value of each of the products currently held in inventory. Logically, this determination requires the multiplication of each product's quantity on hand by its current price. The SQL query for this is shown below and the resulting output is on the next page.

```
SELECT P_DESCRIPT, P_ONHAND, P_PRICE, P_ONHAND * P_PRICE AS TOTVALUE  
FROM PRODUCT
```

SQL will accept any valid expression in the computed columns that apply to the attributes in any of the tables specified in the FROM clause. Note that Access will automatically add an Expr label to all computed columns. Oracle uses the actual expression to label the computed column.

Standard SQL permits the use of aliases for any column in a SELECT statement. The alias for any column is preceded by the keyword AS.



# Using Computed Columns and Column Aliases (cont.)

Microsoft Access

File Edit View Insert Format Records Tools Window Help

qryFig6-12 : Select Query

P_DESCRIPTION	P_ONHAND	P_PRICE	TOTVALUE
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	\$1,011.77
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	\$2,826.89
PVC pipe, 3.5-in., 8-ft	188	\$5.87	\$1,103.56
1.25-in. metal screw, 25	172	\$6.99	\$1,202.28
2.5-in. wd. screw, 50	237	\$8.45	\$2,002.65
Steel matting, 4'x8'x1/8", .5" mesh	18	\$119.95	\$2,159.10
*	0	\$0.00	

Product description

NUM

The computed column with its alias.



# Using A Computed Column an Alias and Date Arithmetic in a Single Query

- Suppose that we want to get a list of “out-of-warranty” products. In this case, let’s assume that we’ve arbitrarily defined out-of-warranty products as those that have been stored more than 90 days. Therefore, the P\_INDATE is at least 90 days less than the current date. The Access version of this query is shown below followed by the Oracle version, with the resulting output shown on the next page.

## Access Version

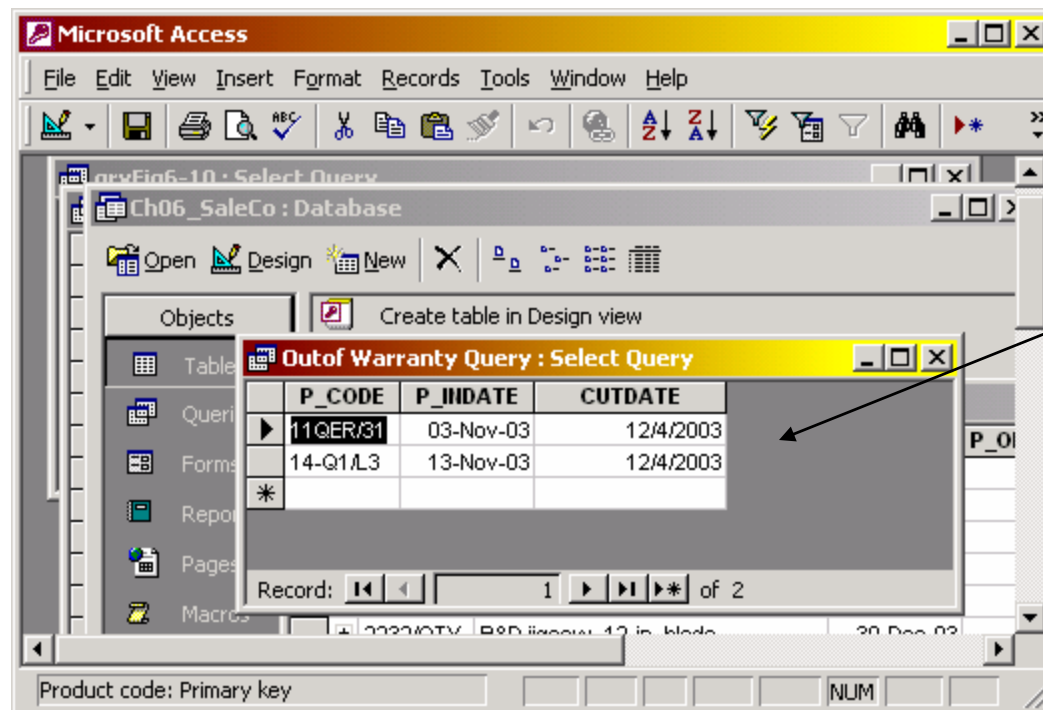
```
SELECT P_CODE, P_INDATE, DATE() - 90 AS CUTDATE  
FROM PRODUCT  
WHERE P_INDATE <= DATE() - 90;
```

## Oracle Version

```
SELECT P_CODE, P_INDATE, SYSDATE - 90 AS CUTDATE  
FROM PRODUCT  
WHERE P_INDATE <= SYSDATE - 90;
```



# Using A Computed Column an Alias and Date Arithmetic in a Single Query



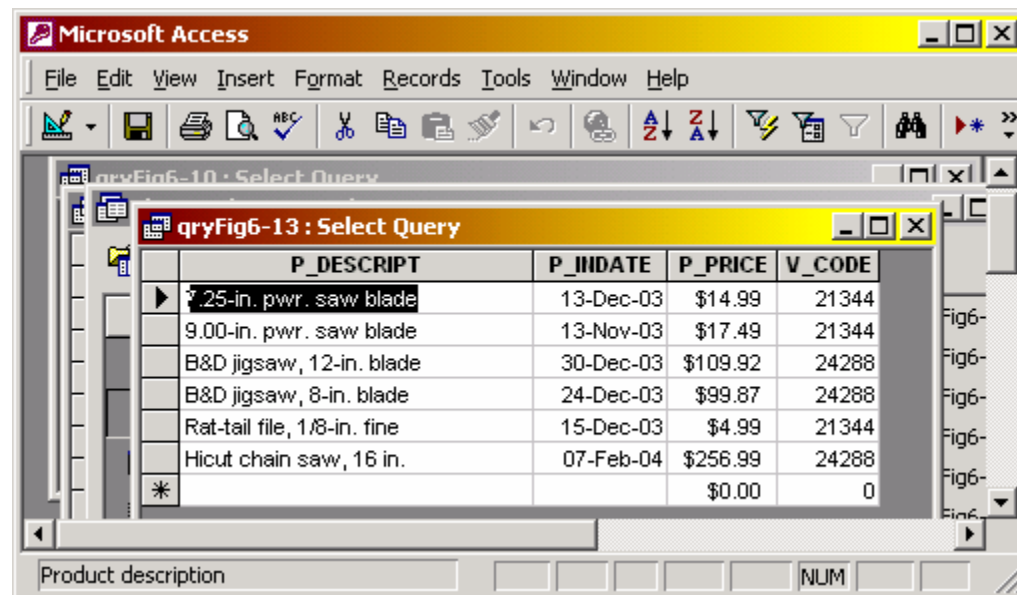
Verify that these are the only two products that are out of range for the warranty by checking the dates of products in the PRODUCTS table on page 45.



# Using The Logical Operators AND, OR, and NOT

- In the real world, a search of data normally involves multiple conditions. SQL allows you to express multiple conditions in a single query through the use of logical operators.
- The logical operators supported by SQL are: AND, OR, and NOT.
- Suppose you want a list of the table of PRODUCTS for either V\_CODE = 21344 or V\_CODE = 24288. The SQL query to accomplish this is:

```
SELECT  P_DESCRIPT,  
        P_INDATE,  
        P_PRICE,  
        V_CODE  
FROM PRODUCT  
WHERE  
    V_CODE = 21344  
OR  
    V_CODE = 24288;
```



The screenshot shows the Microsoft Access application window. The title bar reads "Microsoft Access". The menu bar includes File, Edit, View, Insert, Format, Records, Tools, Window, and Help. The toolbar contains various icons for file operations and data manipulation. The main window displays a query named "qryFig6-13 : Select Query". The query results are shown in a table with the following columns: P\_DESCRIPT, P\_INDATE, P\_PRICE, and V\_CODE. The data rows are as follows:

P_DESCRIPT	P_INDATE	P_PRICE	V_CODE
25-in. pwr. saw blade	13-Dec-03	\$14.99	21344
9.00-in. pwr. saw blade	13-Nov-03	\$17.49	21344
B&D jigsaw, 12-in. blade	30-Dec-03	\$109.92	24288
B&D jigsaw, 8-in. blade	24-Dec-03	\$99.87	24288
Rat-tail file, 1/8-in. fine	15-Dec-03	\$4.99	21344
Hicut chain saw, 16 in.	07-Feb-04	\$256.99	24288
*		\$0.00	0

At the bottom of the window, there is a status bar with the text "Product description" and a numeric field labeled "NUM".



# Special Operators in SQL

- ANSI standard SQL allows the use of special operators in conjunction with the WHERE clause. These special operators include:

**BETWEEN** – Used to check whether an attribute value is within a range.

**IS NULL** – Used to determine if an attribute value is null.

**LIKE** – Used to match an attribute value to a string pattern. Many wildcard options are available.

**IN** – Used to determine if an attribute value is within a list of values.

**EXISTS** – Used to determine if a subquery returns an empty set or not.





# The BETWEEN Special Operator

- Suppose that we want to see a listing for all products whose prices are between \$50 and \$100. The BETWEEN operator can be used for this query expression.

```
SELECT *  
FROM PRODUCT  
WHERE P_PRICE BETWEEN 50.00 AND 100.00;
```

- If your RDBMS does not support BETWEEN you would need to express this query as:

```
SELECT *  
FROM PRODUCT  
WHERE P_PRICE > 50.00 AND P_PRICE < 100.00;
```



# The IS NULL Special Operator

- Suppose that we want to see a listing for all products that do not currently have a vendor assigned, i.e., V\_CODE = null. The null entries could be found with the following query expression.

```
SELECT P_CODE, P_DESCRIPT, V_CODE  
FROM PRODUCT  
WHERE V_CODE IS NULL;
```

- NOTE: SQL uses a special operator for testing for nulls. You cannot use a condition such as V\_CODE = NULL. The reason is that NULL is technically not a “value”, but a special property of an attribute that represents precisely the absence of any value at all.



# The LIKE Special Operator

- The LIKE special operator is used in conjunction with wildcards to find patterns within string attributes.
- Standard SQL allows you to use the percent sign (%) and underscore (\_) wildcard characters to make matches when the entire string is not known.

% means any and all following characters are eligible.

‘M%’ includes Mark, Marci, M-234x, etc.

\_ means any one character may be substituted for the underscore.

‘\_07-345-887\_’ includes 407-345-8871, 007-345-8875

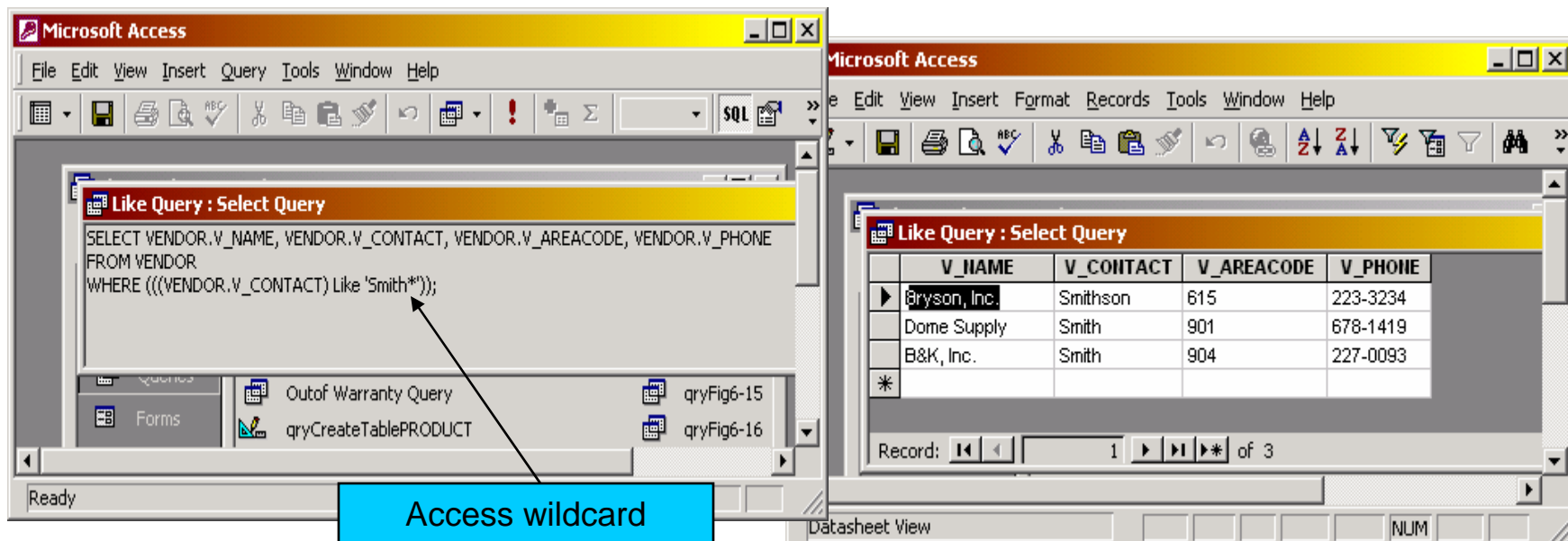
- Note: Access uses \* instead of % and ? instead of \_. Oracle searches are case-sensitive, Access searches are not.



# The LIKE Special Operator (cont.)

- Suppose that we would like to find all the VENDOR rows for contacts whose last names begin with Smith.

```
SELECT V_NAME, V_CONTACT, V_AREACODE, V_PHONE  
FROM VENDOR  
WHERE V_CONTACT LIKE 'Smith%';
```



# The IN Special Operator

- Many queries that would seem to require the use of the logical OR operator can be more easily handled with the help of the special operator IN.
- For example the query:

```
SELECT *  
FROM PRODUCT  
WHERE V_CODE = 21344 OR V_CODE = 24288;
```

can be handled more efficiently with:

```
SELECT *  
FROM PRODUCT  
WHERE V_CODE IN (21344, 24288);
```



# The IN Special Operator (cont.)

- The IN operator is especially valuable when it is used in conjunction with subqueries.
- For example, suppose you want to list the V\_CODE and V\_NAME of only those vendors that actually provide products. In this case, you could use a subquery within the IN operator to automatically generate the value list. The query expression would be:

```
SELECT V_CODE, V_NAME  
FROM VENDOR  
WHERE V_CODE IN ( SELECT V_CODE  
                  FROM PRODUCT);
```

- We'll look more closely at the IN operator later when we deal more in depth with subqueries.



# The EXISTS Special Operator

- The EXISTS operator can be used whenever there is a requirement to execute a command based on the result of another query. That is, if a subquery returns any rows, then run the main query, otherwise, don't. We'll see this operator in more detail when we look at subqueries in more depth.
- For example, suppose we want a listing of vendors, but only if there are products to order. The following query will accomplish our task.

```
SELECT *  
FROM VENDOR  
WHERE EXISTS ( SELECT *  
                FROM PRODUCT  
                WHERE P_ONHAND <= P_MIN);
```



# Advanced DDL Commands in SQL

- Most of the preceding notes covered the basics of table creation in SQL, including inserting, deleting, and updating rows in tables.
- Now we'll look at the SQL commands to alter a table's structure, by changing attribute characteristics and by adding columns. We'll also see how to add data into new or modified columns. We'll also see how to copy tables or parts of tables as well as deleting entire tables.





# The ALTER Command

- All changes in the table structure are made using the ALTER command, followed by a keyword that produces the specific change you want to make.
- There are three options for the keyword: ADD, MODIFY, and DROP.
  - **ADD** enables you to add a column to a table.
  - **MODIFY** enables you to change a column's characteristics.
  - **DROP** allows you to delete a column from a table. Most RDBMSs do not allow you to delete a column from a table, unless that column does not contain any values, because such an action may delete crucial data that are used by other tables.



# The ALTER Command (cont.)

- The basic syntax of the ALTER command is:

```
ALTER TABLE tablename  
{ADD | MODIFY} ( columnname datatype  
                [ {ADD | MODIFY} columnname datatype ] );
```

- The ALTER TABLE command can also be used to add table constraints. In that case the syntax would be as follows:

```
ALTER TABLE tablename  
ADD constraint [ ADD constraint];
```



# The ALTER Command (cont.)

- You can also use the ALTER TABLE command to remove a column or table constraint. The basic syntax of this form of the ALTER command is:

```
ALTER TABLE tablename  
DROP { PRIMARY KEY |  
      COLUMN columnname |  
      CONSTRAINT constraintname } ;
```

- Notice that when removing a constraint, you need to specify the name given to the constraint. This is one reason why it is always advisable to name the constraints in the CREATE TABLE or ALTER TABLE statements.



# Changing a Column's Data Type

- You can also use the ALTER TABLE command to change the data type of a column.
- For example, suppose we wanted to change the data type of V\_CODE attribute in the PRODUCT table from integer to character. The following SQL command would accomplish this task:

```
ALTER TABLE PRODUCT  
MODIFY (V_CODE CHAR(5));
```

- Most RDBMSs do not allow you to change the data type of an attribute unless the column to be changed is empty. For example, if we issued the SQL command above on the database we've been using and error message would be generated because the V\_CODE column already contains data. The reason for the error is simple. The V\_CODE attribute in PRODUCT references the V\_CODE attribute in VENDOR. If the data types don't match, there is a referential integrity violation, thus triggering the error message. If the V\_CODE column in PRODUCT was empty **and** the foreign key reference was not specified during the PRODUCT table's creation, then the SQL command above would execute properly.



# Changing a Column's Data Characteristics

- If the column to be changed already contains data, you can make any changes in the column's characteristics if those changes do not alter the data type.
- For example, if we wanted to increase the width of the P\_PRICE column from 8 digits to 9 digits, we would need to issue the following command:

```
ALTER TABLE PRODUCT  
MODIFY (P_PRICE DECIMAL(9,2));
```

- Many RDBMSs place restrictions on the types of changes to column characteristics that can occur. For example, Oracle will allow you to widen a column but will not allow you to narrow a column.



# Adding a Column to a Table

- You can alter an existing table by adding one or more columns.
- For example, suppose that we want to add a column to the PRODUCT table called P\_SALECODE that will allow us to decide if products that have been in inventory for a certain length of time should be placed on sale. Let's assume that the P\_SALECODE entries will be 1, 2, or 3, and we're not going to do arithmetic on the column so we'll make it a character.

```
ALTER TABLE PRODUCT  
ADD (P_SALECODE CHAR(1));
```



## Adding a Column to a Table (cont.)

- When adding a column, be careful not to include the NOT NULL clause for the new column. Doing so will cause an error message because when adding a new column to a table that already has rows, the existing rows will default to a value of null for the new column. Therefore, it is not possible to add the NOT NULL clause for this new column.
- You can add the NOT NULL clause to the table structure after all the data for the new column has been entered and the column no longer contains nulls.



# Deleting a Column from a Table

- Occasionally, you may want to modify a table by deleting a column.
- Suppose that we want to delete the V\_ORDER attribute from the VENDOR table. To accomplish this task we would use the following SQL command:

```
ALTER TABLE VENDOR  
DROP COLUMN V_ORDER;
```

- As before, some RDBMSs will impose restrictions on the deletion of an attribute. For example, most RDBMSs will not allow you to delete attributes that are involved in foreign key relationships, nor may you delete an attribute of a table that contains only that one attribute.





# Advanced Data Updates

- To make data entries in an existing row's columns, SQL uses the UPDATE command. The UPDATE command only updates data in existing rows.
- For example, to enter the P\_SALECODE value of '2' in the fourth row of the PRODUCT\_2 table, we need to use the UPDATE command in conjunction with the primary key value for that row. The following command produces this effect with the before and after results shown on the next page.

```
UPDATE PRODUCT_2  
  SET P_SALECODE = '2'  
 WHERE P_CODE = '1546-QQ2';
```



# Advanced Data Updates (cont.)

P\_SALECODE value before update

Microsoft Access

File Edit View Insert Format Records Tools Window Help

PRODUCT\_2 : Table

P_CODE	P_DESCRIPT	P_INDATE	P_ONHAND	P_MIN	P_PRICE	P_DISCOUNT	V_CODE	P_SALECODE
11QER/31	Power painter, 15 psi., 3-nozzle	03-Nov-03	8	5	\$109.99	0.00	25595	
13-Q2/P2	7.25-in. pwr. saw blade	13-Dec-03	32	15	\$14.99	0.05	21344	
14-Q1/L3	9.00-in. pwr. saw blade	13-Nov-03	18	12	\$17.49	0.00	21344	
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15-Jan-04	15	8	\$39.95	0.00	23119	
1558-QW1	Hrd. cloth, 1/2-in., 3x50	15-Jan-04	23	5	\$43.99	0.00	23119	
2232/QTY	B&D jigsaw, 12-in. blade	30-Dec-03	8	5	\$109.92	0.05	24288	1
2232/QW1	B&D jigsaw, 8-in. blade	24-Dec-03	6	5	\$99.87	0.05	24288	1
2238/QPD	B&D cordless drill, 1/2-in.	20-Jan-04	12	5	\$38.95	0.05	25595	
23109-HB	Claw hammer	20-Jan-04	23	10	\$9.95	0.10	21225	
23114-AA	Sledge hammer, 12 lb.	02-Jan-04	8	5	\$14.40	0.05		
54778-2T	Rat-tail file, 1/8-in. fine	15-Dec-03	43	20	\$4.99	0.00	21344	
89-WRE-Q	Hicut chain saw, 16 in.	07-Feb-04	11	5	\$256.99	0.05	24288	
PVC23DRT	PVC pipe, 3.5-in., 8-ft	20-Feb-04	188	75	\$5.87	0.00		
SM-18277	1.25-in. metal screw, 25	01-Mar-04	172	75	\$6.99	0.00	21225	

Product description

NUM



# Advanced Data Updates (cont.)

P\_SALECODE value after update

Microsoft Access

File Edit View Insert Format Records Tools Window Help

PRODUCT\_2 : Table

P_CODE	P_DESCRIPTION	P_INDATE	P_ONHAND	P_MIN	P_PRICE	P_DISCOUNT	V_CODE	P_SALECODE
11QER/31	Power painter, 15 psi., 3-nozzle	03-Nov-03	8	5	\$109.99	0.00	25595	
13-Q2/P2	7.25-in. pwr. saw blade	13-Dec-03	32	15	\$14.99	0.05	21344	
14-Q1/L3	9.00-in. pwr. saw blade	13-Nov-03	18	12	\$17.49	0.00	21344	
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15-Jan-04	15	8	\$39.95	0.00	23119	2
1558-QW1	Hrd. cloth, 1/2-in., 3x50	15-Jan-04	23	5	\$43.99	0.00	23119	
2232/QTY	B&D jigsaw, 12-in. blade	30-Dec-03	8	5	\$109.92	0.05	24288	1
2232/QWE	B&D jigsaw, 8-in. blade	24-Dec-03	6	5	\$99.87	0.05	24288	1
2238/QPD	B&D cordless drill, 1/2-in.	20-Jan-04	12	5	\$38.95	0.05	25595	
23109-HB	Claw hammer	20-Jan-04	23	10	\$9.95	0.10	21225	
23114-AA	Sledge hammer, 12 lb.	02-Jan-04	8	5	\$14.40	0.05		
54778-2T	Rat-tail file, 1/8-in. fine	15-Dec-03	43	20	\$4.99	0.00	21344	
89-WRE-Q	Hicut chain saw, 16 in.	07-Feb-04	11	5	\$256.99	0.05	24288	
PVC23DRT	PVC pipe, 3.5-in., 8-ft	20-Feb-04	188	75	\$5.87	0.00		
SM-18277	1.25-in. metal screw, 25	01-Mar-04	172	75	\$6.99	0.00	21225	

Product description

NUM



# Advanced Data Updates (cont.)

- Subsequent data can be entered in the same fashion, defining each entry location by its primary key (P\_CODE) and its column location (P\_SALECODE).
- The table on the previous page, also reflects the execution of the following SQL command:

```
UPDATE PRODUCT_2  
  SET P_SALECODE = '1'  
 WHERE P_CODE IN ('2232/QWE', '2232/QTY');
```



# Advanced Data Updates (cont.)

- Although the UPDATE sequences we've just seen allow you to enter values into specified table cells, the process is very cumbersome. Fortunately, there may be better method.
- If a relationship can be established between entries and the existing columns, that relationship may be used to assign values to their appropriate slots.
- For example, suppose that we want to place sales codes based on the P\_INDATE into the table using the following rule:
  - If P\_INDATE is before December 25, 2003, then P\_SALECODE = 2
  - If P\_INDATE between Jan. 16 and Feb 10, 2004, then P\_SALECODE = 1



# Advanced Data Updates (cont.)

- Given the rule from the previous page and using the PRODUCT table to effect the update, the following two command sequences will make all the appropriate updates to the P\_SALECODE in the PRODUCT table. The effect of these two updates on the PRODUCT table is shown on the next page.

```
UPDATE PRODUCT
```

```
  SET P_SALECODE = '2'
```

```
  WHERE P_INDATE < '25-Dec-2003';
```

```
UPDATE PRODUCT
```

```
  SET P_SALECODE = '1'
```

```
  WHERE P_INDATE >= '16-Jan-2004'
```

```
    AND P_INDATE <= '10-Feb-2004';
```



# Advanced Data Updates (cont.)

Microsoft Access

File Edit View Insert Format Records Tools Window Help

Ch06\_SaleCo : Database

PRODUCT\_3 : Table

P_CODE	P_DESCRIPT	P_INDATE	P_ONHAND	P_MIN	P_PRICE	P_DISCOUNT	V_CODE	P_SALECODE
11QER/31	Power painter, 15 psi., 3-nozzle	03-Nov-03	8	5	\$109.99	0.00	25595	2
13-Q2/P2	7.25-in. pwr. saw blade	13-Dec-03	32	15	\$14.99	0.05	21344	2
14-Q1/L3	9.00-in. pwr. saw blade	13-Nov-03	18	12	\$17.49	0.00	21344	2
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15-Jan-04	15	8	\$39.95	0.00	23119	
1558-QW1	Hrd. cloth, 1/2-in., 3x50	15-Jan-04	23	5	\$43.99	0.00	23119	
2232/QTY	B&D jigsaw, 12-in. blade	30-Dec-03	8	5	\$109.92	0.05	24288	
2232/QWE	B&D jigsaw, 8-in. blade	24-Dec-03	6	5	\$99.87	0.05	24288	2
2238/QPD	B&D cordless drill, 1/2-in.	20-Jan-04	12	5	\$38.95	0.05	25595	1
23109-HB	Claw hammer	20-Jan-04	23	10	\$9.95	0.10	21225	1
23114-AA	Sledge hammer, 12 lb.	02-Jan-04	8	5	\$14.40	0.05		
54778-2T	Rat-tail file, 1/8-in. fine	15-Dec-03	43	20	\$4.99	0.00	21344	2
89-WRE-Q	Hicut chain saw, 16 in.	07-Feb-04	11	5	\$256.99	0.05	24288	1
PVC23DRT	PVC pipe, 3.5-in., 8-ft	20-Feb-04	188	75	\$5.87	0.00		
SM-18277	1.25-in. metal screw, 25	01-Mar-04	172	75	\$6.99	0.00	21225	
SW-23116	2.5-in. wdd. screw, 50	24-Feb-04	237	100	\$8.45	0.00	21231	
WR3/TT3	Steel matting, 4'x8'x1/8", .5" mesh	17-Jan-04	18	5	\$119.95	0.10	25595	1
			0	0	\$0.00	0.00	0	

Record: 17 of 17

Product code: Primary key

NUM



## Advanced Data Updates (cont.)

- The arithmetic operators are particularly useful in data updates.
- For example, if the quantity on hand in the PRODUCT table has dropped below the minimum desirable value, you'll order more of the product. Suppose that we've ordered 20 more units of product 2232/QWE. When the 20 units arrive, we'll need to add them to the inventory in the following manner:

```
UPDATE PRODUCT
```

```
SET P_ONHAND = P_ONHAND + 20
```

```
WHERE P__CODE = '2232/QWE/;
```





# Advanced Data Updates (cont.)

- Suppose that you want to add 10 percent to the price of all products that have current prices below \$50. This can be accomplished with the following command.

```
UPDATE PRODUCT  
  SET P_PRICE = P_PRICE * 1.10  
  WHERE P__PRICE < 50.00;
```



# Copying Parts of Tables

- Although the database should be properly designed before it is implemented, there are times when it may be necessary to break up a table structure into several component parts (smaller tables).
- SQL allows for the copying of the contents of selected table columns so that the data in those columns need not be reentered manually into the newly created table(s).
- Suppose, for example, that we want to copy the P\_CODE, P\_DESCRIPT, and P\_PRICE columns from the PRODUCT table to a new table named PART.
- First, we'll need to create the PART table as shown on the next page.



# Copying Parts of Tables (cont.)

```
CREATE TABLE PART (  
    PART_CODE          CHAR(8) NOT NULL  UNIQUE,  
    PART_DESCRIPT      CHAR(35),  
    PART_PRICE         DECIMAL(8,2),  
    PRIMARY KEY (PART_CODE) );
```

- Note that the PART column names need not be identical to those in the original table, and that the new table need not have the same number of columns as the original table.
  - In this case, the first column in the PART table is PART\_CODE, rather than the original P\_CODE found in the PRODUCT table. And the PART table contains only three columns, rather than the seven columns found in the PRODUCT table.
  - However, column characteristics must match: you cannot copy a character-based attribute into a numeric structure, and vice versa.



# Copying Parts of Tables (cont.)

- Next, we need to add the rows to the new PART table using the PRODUCT table rows. To do this we'll need to use the INSERT command.
- Recall, the basic syntax of this command:

```
INSERT INTO target_tablename [(target_columnlist)]  
  SELECT source_columnlist  
  FROM source-tablename;
```

- The target-columnlist is required if the source-columnlist doesn't match all of the attribute names and characteristics of the target table (including the order of the columns!). Otherwise, you do not need to specify the target-columnlist.
  - In our example, we will need to specify the target-columnlist since we are changing the names of one or more of the columns.



## Copying Parts of Tables (cont.)

- To effect the desired update, we need to issue the following INSERT command:

```
INSERT INTO PART (PART_CODE, PART_DESCRIPT, PART_PRICE)  
  SELECT P_CODE, P_DESCRIPT, P_PRICE  
  FROM PRODUCT;
```

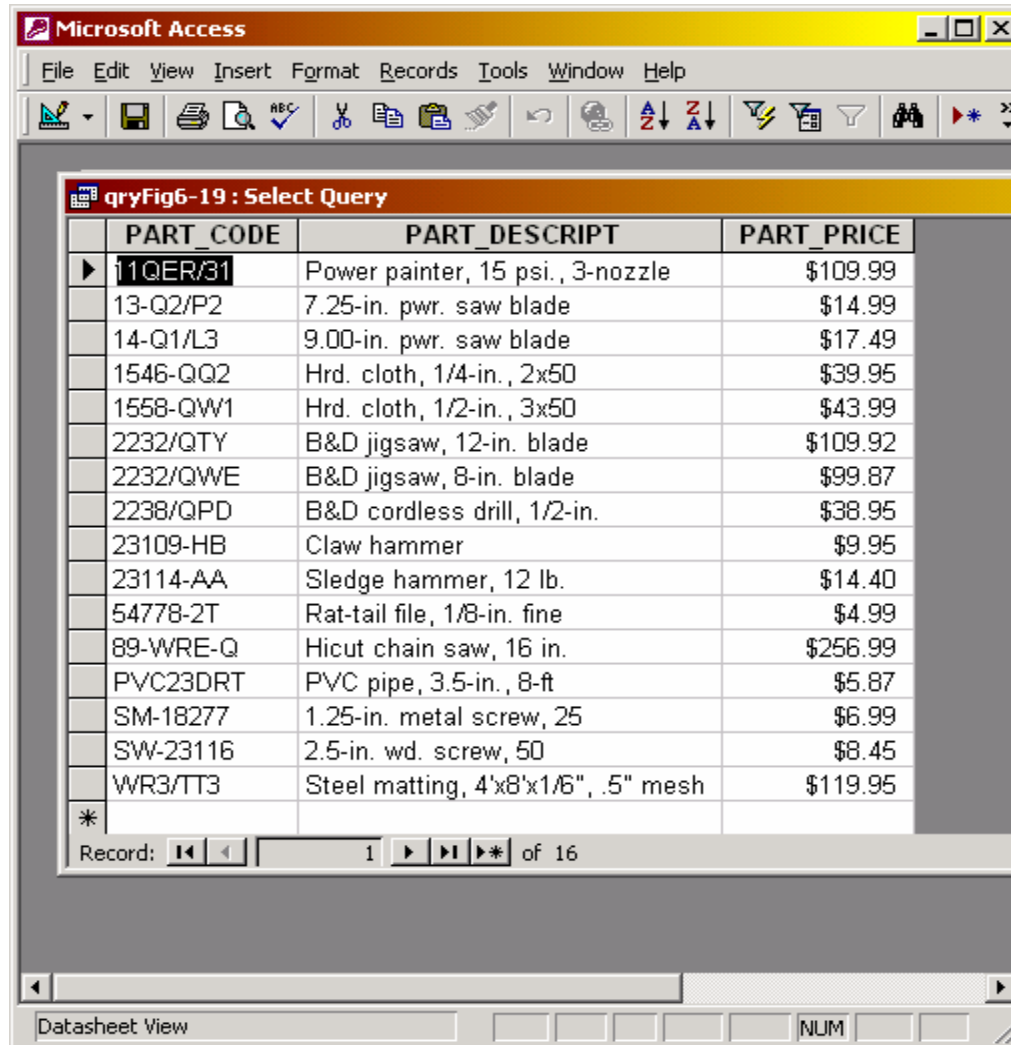
- The contents of the updated PARTS table can be viewed with the following query:

```
SELECT *  
  FROM PART;
```

- The results are shown on the next page.



# Copying Parts of Tables (cont.)



The screenshot shows the Microsoft Access application window. The title bar reads 'Microsoft Access'. The menu bar includes 'File', 'Edit', 'View', 'Insert', 'Format', 'Records', 'Tools', 'Window', and 'Help'. The toolbar contains various icons for file operations, editing, and data manipulation. The main window displays a query named 'qryFig6-19 : Select Query' in Datasheet View. The query results are shown in a table with three columns: 'PART\_CODE', 'PART\_DESCRIPT', and 'PART\_PRICE'. The first row is highlighted, showing '11QER/31' as the part code, 'Power painter, 15 psi., 3-nozzle' as the description, and '\$109.99' as the price. The status bar at the bottom indicates 'Record: 1 of 16' and 'Datasheet View'.

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

Results of copying parts of a table with renaming of the columns.



# Adding Primary and Foreign Key Designations

- Although we were able to create a new table based on an existing table in the previous example, the process is not without its problems. Primarily, the new PART table was created without the inclusion of the integrity rules of the table on which it was based. In particular, there is no primary key designated for the table shown on the previous page.
- To define the primary key for this table we need to use the ALTER command as shown below:

```
ALTER TABLE PRODUCT  
ADD PRIMARY KEY (P_CODE);
```



# Adding Primary and Foreign Key Designations (cont.)

- Quite aside from the fact that the integrity rules are not automatically transferred to a new table that derives its data from one or more other tables, there are several other scenarios that would leave you without entity and referential integrity enforcement.
- For example, you might have simply forgotten to define the primary and foreign keys when you created the tables.
- The integrity rules can be reestablished via the ALTER command as shown below:

```
ALTER TABLE PRODUCT  
  ADD PRIMARY KEY(P_CODE)  
  ADD FOREIGN KEY(V_CODE) REFERENCES VENDOR;
```





# Deleting a Table From the Database

- A table can be deleted from the database through the DROP command as shown below:

```
DROP TABLE PRODUCT;
```

- A table can only be dropped from a database if it is not participating as the “1” side of any relationships. If you attempt to delete such a table, the RDMS will issue an error message to indicate that a foreign key integrity violation has occurred.

