



INSTITUTE OF INDUSTRIAL AND INFORMATION TECHNOLOGY

INTRODUCTION TO SQL

Essentials of Database Management

OBJECTIVES

- ✗ Define terms
- ✗ Interpret history and role of SQL
- ✗ Define a database using SQL data definition language
- ✗ Write single table queries using SQL
- ✗ Establish referential integrity using SQL
- ✗ Discuss SQL:1999 and SQL:2008 standards

SQL OVERVIEW

- ✗ Structured Query Language
- ✗ The standard for relational database management systems (RDBMS)
- ✗ RDBMS: A database management system that manages data as a collection of tables in which all relationships are represented by common values in related tables

HISTORY OF SQL

- ✖ 1970–E. F. Codd develops relational database concept
- ✖ 1974-1979–System R with Sequel (later SQL) created at IBM Research Lab
- ✖ 1979–Oracle markets first relational DB with SQL
- ✖ 1981 – SQL/DS first available RDBMS system on DOS/VSE
- ✖ Others followed: INGRES (1981), IDM (1982), DG/SGL (1984), Sybase (1986)
- ✖ 1986–ANSI SQL standard released
- ✖ 1989, 1992, 1999, 2003, 2006, 2008–Major ANSI standard updates
- ✖ Current–SQL is supported by most major database vendors

PURPOSE OF SQL STANDARD

- ✖ Specify syntax/semantics for data definition and manipulation
- ✖ Define data structures and basic operations
- ✖ Enable portability of database definition and application modules
- ✖ Specify minimal (level 1) and complete (level 2) standards
- ✖ Allow for later growth/enhancement to standard (referential integrity, transaction management, user-defined functions, extended join operations, national character sets)

BENEFITS OF A STANDARDIZED RELATIONAL LANGUAGE

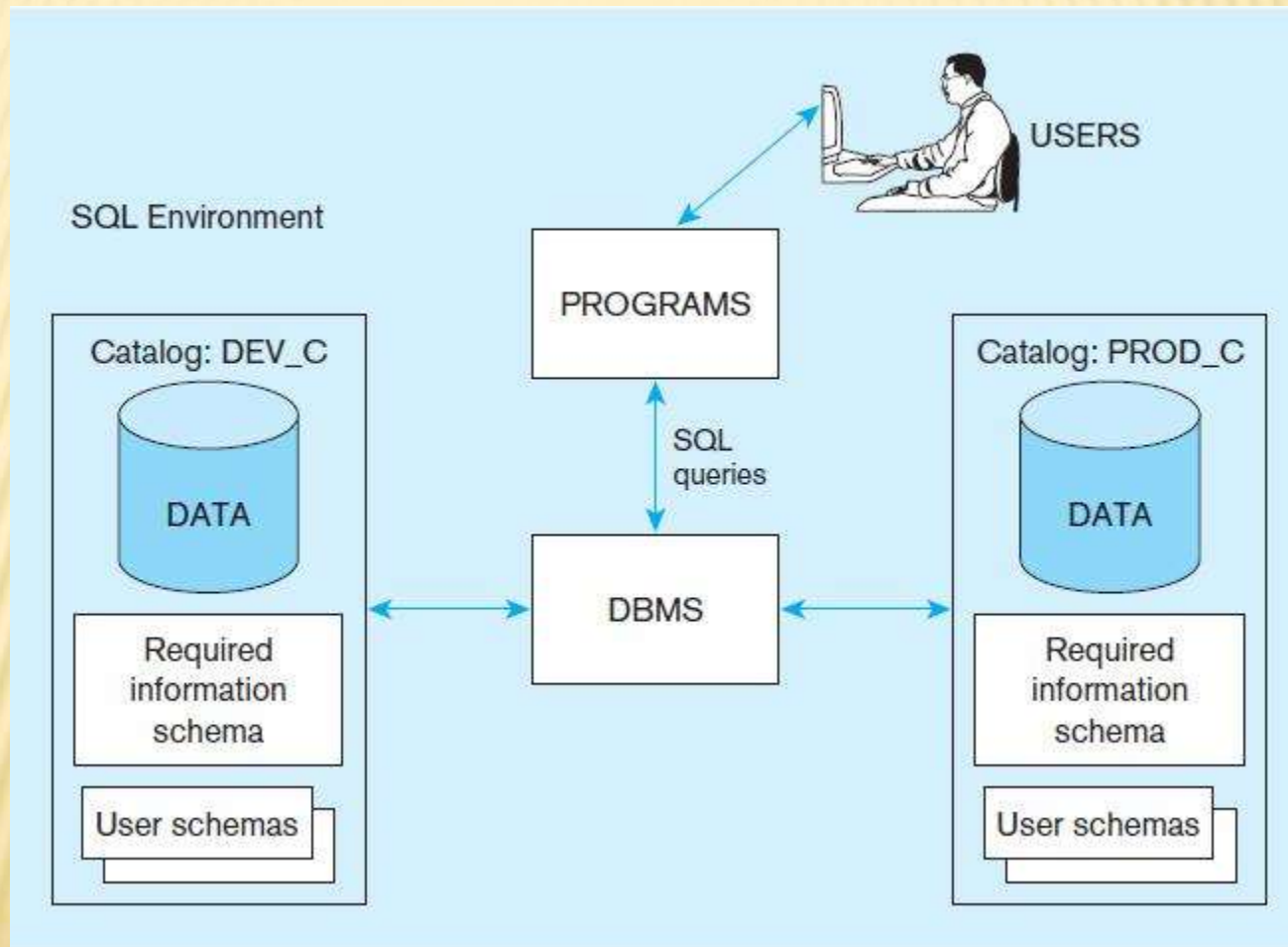
- ✗ Reduced training costs
- ✗ Productivity
- ✗ Application portability
- ✗ Application longevity
- ✗ Reduced dependence on a single vendor
- ✗ Cross-system communication

SQL ENVIRONMENT

- ✗ **Catalog**
 - + A set of schemas that constitute the description of a database
- ✗ **Schema**
 - + The structure that contains descriptions of objects created by a user (base tables, views, constraints)
- ✗ **Data Definition Language (DDL)**
 - + Commands that define a database, including creating, altering, and dropping tables and establishing constraints
- ✗ **Data Manipulation Language (DML)**
 - + Commands that maintain and query a database
- ✗ **Data Control Language (DCL)**
 - + Commands that control a database, including administering privileges and committing data

Figure 6-1

A simplified schematic of a typical SQL environment, as described by the SQL: 2008 standard



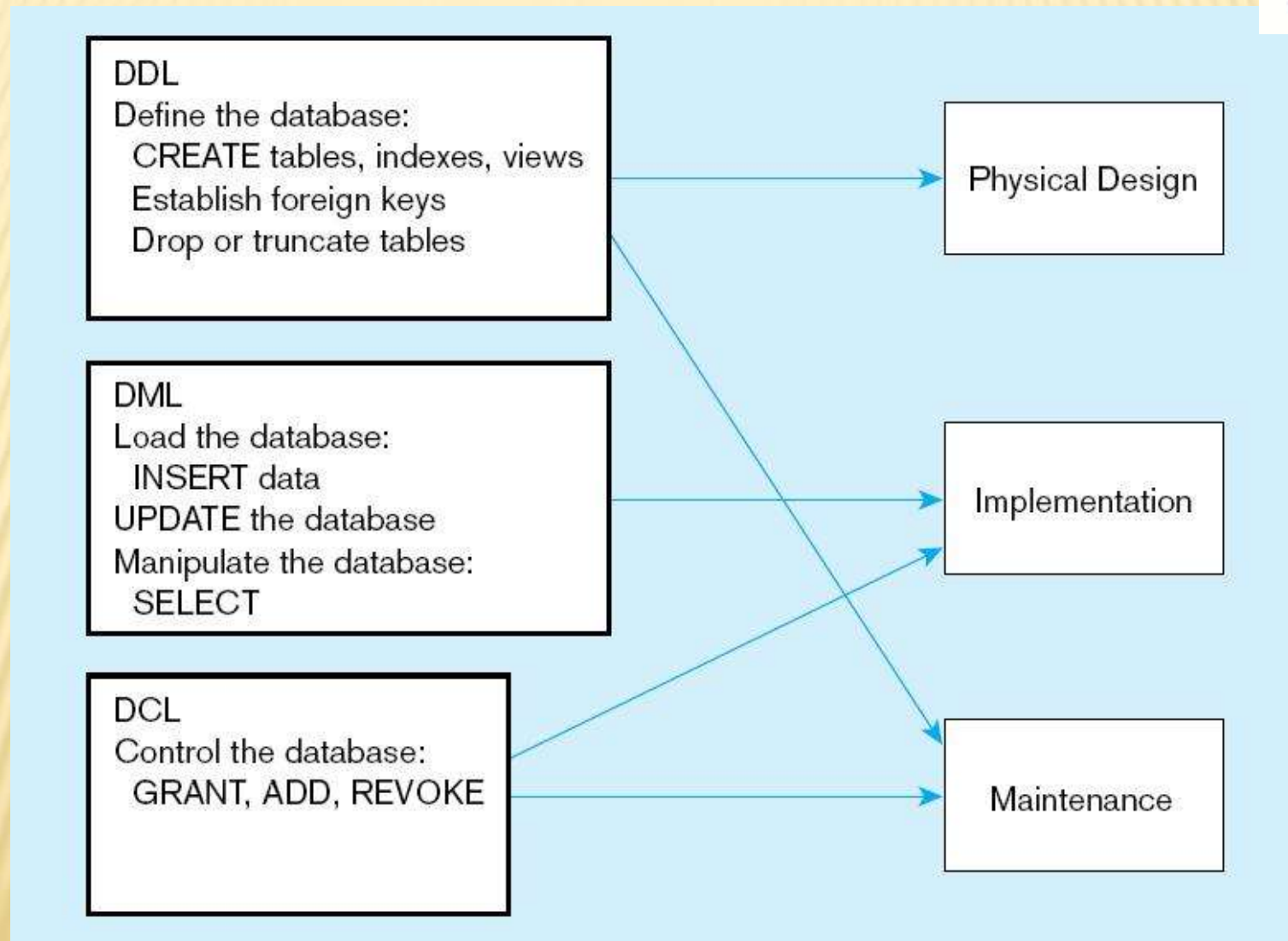
SQL DATA TYPES

TABLE 6-2 Sample SQL Data Types

String	CHARACTER (CHAR)	Stores string values containing any characters in a character set. CHAR is defined to be a fixed length.
	CHARACTER VARYING (VARCHAR or VARCHAR2)	Stores string values containing any characters in a character set but of definable variable length.
	BINARY LARGE OBJECT (BLOB)	Stores binary string values in hexadecimal format. BLOB is defined to be a variable length. (Oracle also has CLOB and NCLOB, as well as BFILE for storing unstructured data outside the database.)
Number	NUMERIC	Stores exact numbers with a defined precision and scale.
	INTEGER (INT)	Stores exact numbers with a predefined precision and scale of zero.
Temporal	TIMESTAMP TIMESTAMP WITH LOCAL TIME ZONE	Stores a moment an event occurs, using a definable fraction-of-a-second precision. Value adjusted to the user's session time zone (available in Oracle and MySQL)
Boolean	BOOLEAN	Stores truth values: TRUE, FALSE, or UNKNOWN.

Figure 6-4

DDL, DML, DCL, and the database development process



SQL DATABASE DEFINITION

- ✗ Data Definition Language (DDL)
- ✗ Major CREATE statements:
 - + CREATE SCHEMA–defines a portion of the database owned by a particular user
 - + CREATE TABLE–defines a new table and its columns
 - + CREATE VIEW–defines a logical table from one or more tables or views
- ✗ Other CREATE statements: CHARACTER SET, COLLATION, TRANSLATION, ASSERTION, DOMAIN

STEPS IN TABLE CREATION

1. Identify data types for attributes
2. Identify columns that can and cannot be null
3. Identify columns that must be unique (candidate keys)
4. Identify primary key–foreign key mates
5. Determine default values
6. Identify constraints on columns (domain specifications)
7. Create the table and associated indexes

Figure 6-5 General syntax for CREATE TABLE statement used in data definition language

```
CREATE TABLE tablename  
( {column definition [table constraint] } . . .  
[ON COMMIT {DELETE | PRESERVE} ROWS] );
```

where *column definition* ::=
column_name

```
    {domain name | datatype [(size)] }  
    [column_constraint_clause . . .]  
    [default value]  
    [collate clause]
```

and *table constraint* ::=

```
    [CONSTRAINT constraint_name]  
    Constraint_type [constraint_attributes]
```

THE FOLLOWING SLIDES CREATE TABLES FOR THIS ENTERPRISE DATA MODEL

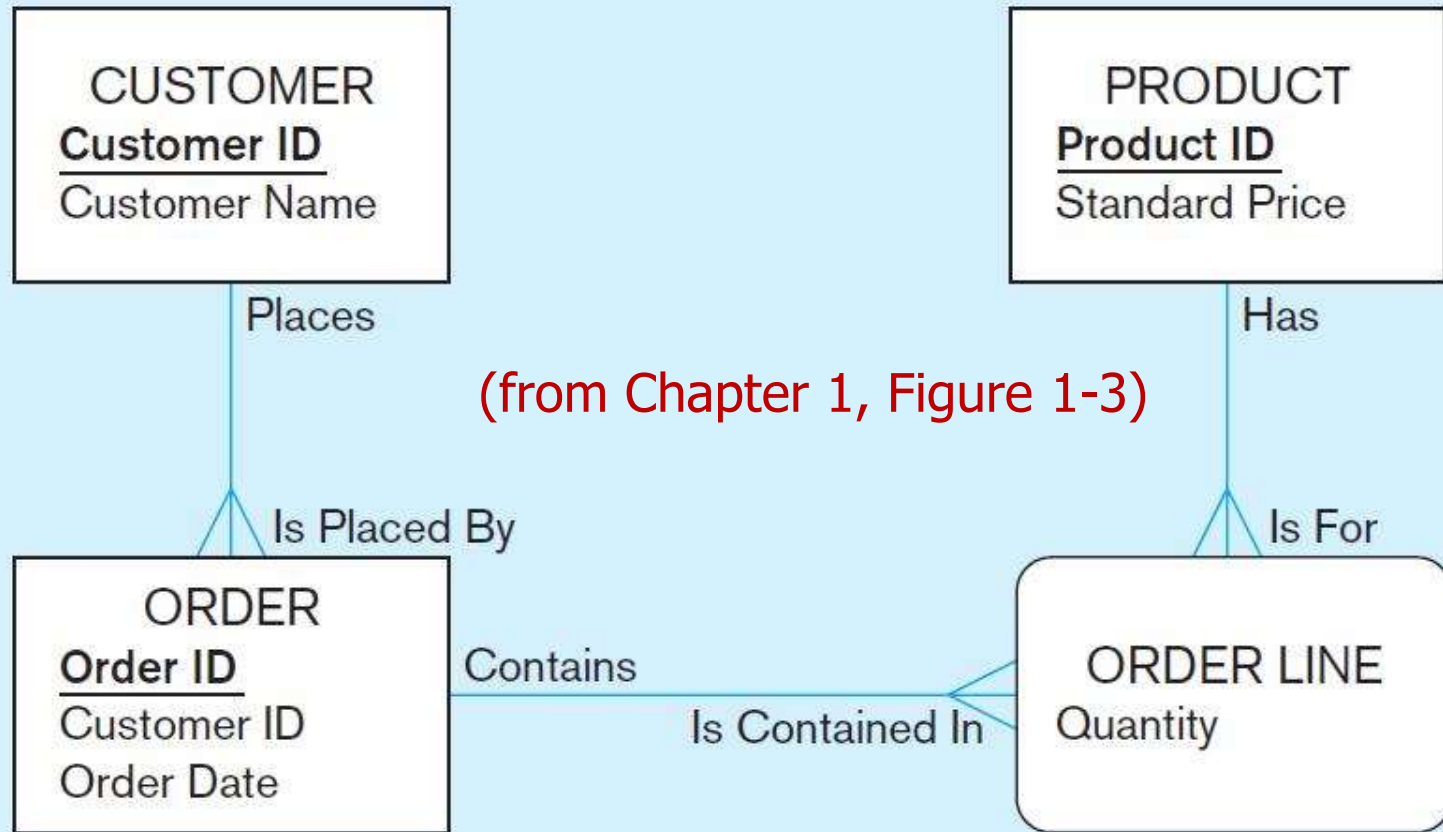


Figure 6-6 SQL database definition commands for Pine Valley Furniture Company (Oracle 11g)



Overall table definitions

```
CREATE TABLE Customer_T
    (CustomerID          NUMBER(11,0)    NOT NULL,
     CustomerName        VARCHAR2(25)    NOT NULL,
     CustomerAddress     VARCHAR2(30),
     CustomerCity        VARCHAR2(20),
     CustomerState       CHAR(2),
     CustomerPostalCode  VARCHAR2(9),
 CONSTRAINT Customer_PK PRIMARY KEY (CustomerID));
```

```
CREATE TABLE Order_T
    (OrderID             NUMBER(11,0)    NOT NULL,
     OrderDate           DATE DEFAULT SYSDATE,
     CustomerID         NUMBER(11,0),
 CONSTRAINT Order_PK PRIMARY KEY (OrderID),
 CONSTRAINT Order_FK FOREIGN KEY (CustomerID) REFERENCES Customer_T(CustomerID));
```

```
CREATE TABLE Product_T
    (ProductID          NUMBER(11,0)    NOT NULL,
     ProductDescription  VARCHAR2(50),
     ProductFinish      VARCHAR2(20)
                        CHECK (ProductFinish IN ('Cherry', 'Natural Ash', 'White Ash',
                                                  'Red Oak', 'Natural Oak', 'Walnut')),
     ProductStandardPrice DECIMAL(6,2),
     ProductLineID      INTEGER,
 CONSTRAINT Product_PK PRIMARY KEY (ProductID));
```

```
CREATE TABLE OrderLine_T
    (OrderID            NUMBER(11,0)    NOT NULL,
     ProductID          INTEGER         NOT NULL,
     OrderedQuantity    NUMBER(11,0),
 CONSTRAINT OrderLine_PK PRIMARY KEY (OrderID, ProductID),
 CONSTRAINT OrderLine_FK1 FOREIGN KEY (OrderID) REFERENCES Order_T(OrderID),
 CONSTRAINT OrderLine_FK2 FOREIGN KEY (ProductID) REFERENCES Product_T(ProductID));
```

Defining attributes and their data types

```
CREATE TABLE Product_T
```

(ProductID	NUMBER(11,0)	NOT NULL,
ProductDescription	VARCHAR2(50),	
ProductFinish	VARCHAR2(20)	

```
CHECK (ProductFinish IN ('Cherry', 'Natural Ash', 'White Ash',  
                          'Red Oak', 'Natural Oak', 'Walnut')),
```

ProductStandardPrice	DECIMAL(6,2),
ProductLineID	INTEGER,

```
CONSTRAINT Product_PK PRIMARY KEY (ProductID));
```

Non-nullable specification

```
CREATE TABLE Product_T
```

```
(ProductID NUMBER(11,0) NOT NULL,
```

```
ProductDescription VARCHAR2(50),
```

```
ProductFinish VARCHAR2(20)
```

```
CHECK (ProductFinish IN ('Cherry', 'Natural Ash', 'White Ash',  
'Red Oak', 'Natural Oak', 'Walnut')),
```

```
ProductStandardPrice DECIMAL(6,2),
```

```
ProductLineID INTEGER,
```

```
CONSTRAINT Product_PK PRIMARY KEY (ProductID));
```

Primary keys
can never have
NULL values

Identifying primary key

Non-nullable specifications

```
CREATE TABLE OrderLine_T
    (OrderID                NUMBER(11,0)    NOT NULL,
     ProductID              INTEGER         NOT NULL,
     OrderedQuantity        NUMBER(11,0),
     CONSTRAINT OrderLine_PK PRIMARY KEY (OrderID, ProductID),
     CONSTRAINT OrderLine_FK1 FOREIGN KEY (OrderID) REFERENCES Order_T(OrderID),
     CONSTRAINT OrderLine_FK2 FOREIGN KEY (ProductID) REFERENCES Product_T(ProductID));
```

Primary key

Some primary keys are composite—
composed of multiple attributes

Controlling the values in attributes

```

CREATE TABLE Order_T
    (OrderID                NUMBER(11,0)    NOT NULL,
     OrderDate              DATE DEFAULT SYSDATE,
     CustomerID             NUMBER(11,0),
 CONSTRAINT Order_PK PRIMARY KEY (OrderID),
 CONSTRAINT Order_FK FOREIGN KEY (CustomerID) REFERENCES Customer_T(CustomerID));

CREATE TABLE Product_T
    (ProductID              NUMBER(11,0)    NOT NULL,
     ProductDescription      VARCHAR2(50),
     ProductFinish           VARCHAR2(20)
     CHECK (ProductFinish IN ('Cherry', 'Natural Ash', 'White Ash',
                              'Red Oak', 'Natural Oak', 'Walnut')),
     ProductStandardPrice   DECIMAL(6,2),
     ProductLineID           INTEGER,
 CONSTRAINT Product_PK PRIMARY KEY (ProductID));
  
```

Default value

Domain constraint

Identifying foreign keys and establishing relationships

```
CREATE TABLE Customer_T
```

(CustomerID	NUMBER(11,0)	NOT NULL,
CustomerName	VARCHAR2(25)	NOT NULL,
CustomerAddress	VARCHAR2(30),	
CustomerCity	VARCHAR2(20),	
CustomerState	CHAR(2),	
CustomerPostalCode	VARCHAR2(9),	

Primary key of
parent table

```
CONSTRAINT Customer_PK PRIMARY KEY (CustomerID));
```

```
CREATE TABLE Order_T
```

(OrderID	NUMBER(11,0)	NOT NULL,
OrderDate	DATE DEFAULT SYSDATE,	
CustomerID	NUMBER(11,0),	

```
CONSTRAINT Order_PK PRIMARY KEY (OrderID),
```

```
CONSTRAINT Order_FK FOREIGN KEY (CustomerID) REFERENCES Customer_T(CustomerID));
```

Foreign key of dependent table

DATA INTEGRITY CONTROLS

- ✗ Referential integrity–constraint that ensures that foreign key values of a table must match primary key values of a related table in 1:M relationships
- ✗ Restricting:
 - + Deletes of primary records
 - + Updates of primary records
 - + Inserts of dependent records

CHANGING TABLES



- ✖ ALTER TABLE statement allows you to change column specifications:

```
ALTER TABLE table_name alter_table_action;
```

- ✖ Table Actions:

```
ADD [COLUMN] column_definition  
ALTER [COLUMN] column_name SET DEFAULT default-value  
ALTER [COLUMN] column_name DROP DEFAULT  
DROP [COLUMN] column_name [RESTRICT] [CASCADE]  
ADD table_constraint
```

- ✖ Example (adding a new column with a default value):

```
ALTER TABLE CUSTOMER_T  
ADD COLUMN CustomerType VARCHAR2 (2) DEFAULT "Commercial";
```

REMOVING TABLES

✕ DROP TABLE statement allows you to remove tables from your schema:

+ DROP TABLE CUSTOMER_T

INSERT STATEMENT

- ✗ Adds one or more rows to a table
- ✗ Inserting into a table

```
INSERT INTO Customer_T VALUES  
(001, 'Contemporary Casuals', '1355 S. Himes Blvd.', 'Gainesville', 'FL', 32601);
```

- ✗ Inserting a record that has some null attributes requires identifying the fields that actually get data

```
INSERT INTO Product_T (ProductID,  
ProductDescription, ProductFinish, ProductStandardPrice)  
VALUES (1, 'End Table', 'Cherry', 175, 8);
```

- ✗ Inserting from another table

```
INSERT INTO CaCustomer_T  
SELECT * FROM Customer_T  
WHERE CustomerState = 'CA';
```

CREATING TABLES WITH IDENTITY COLUMNS

```
CREATE TABLE Customer_T  
(CustomerID INTEGER GENERATED ALWAYS AS IDENTITY  
  (START WITH 1  
   INCREMENT BY 1  
   MINVALUE 1  
   MAXVALUE 10000  
   NO CYCLE),  
  CustomerName          VARCHAR2(25) NOT NULL,  
  CustomerAddress        VARCHAR2(30),  
  CustomerCity           VARCHAR2(20),  
  CustomerState          CHAR(2),  
  CustomerPostalCode     VARCHAR2(9),  
  CONSTRAINT Customer_PK PRIMARY KEY (CustomerID);
```

Introduced with SQL:2008

Inserting into a table does not require explicit customer ID entry or field list

```
INSERT INTO CUSTOMER_T VALUES ( 'Contemporary Casuals',  
'1355 S. Himes Blvd.', 'Gainesville', 'FL', 32601);
```

DELETE STATEMENT

- ✗ Removes rows from a table
- ✗ Delete certain rows
 - + **DELETE FROM CUSTOMER_T WHERE CUSTOMERSTATE = 'HI';**
- ✗ Delete all rows
 - + **DELETE FROM CUSTOMER_T;**

UPDATE STATEMENT

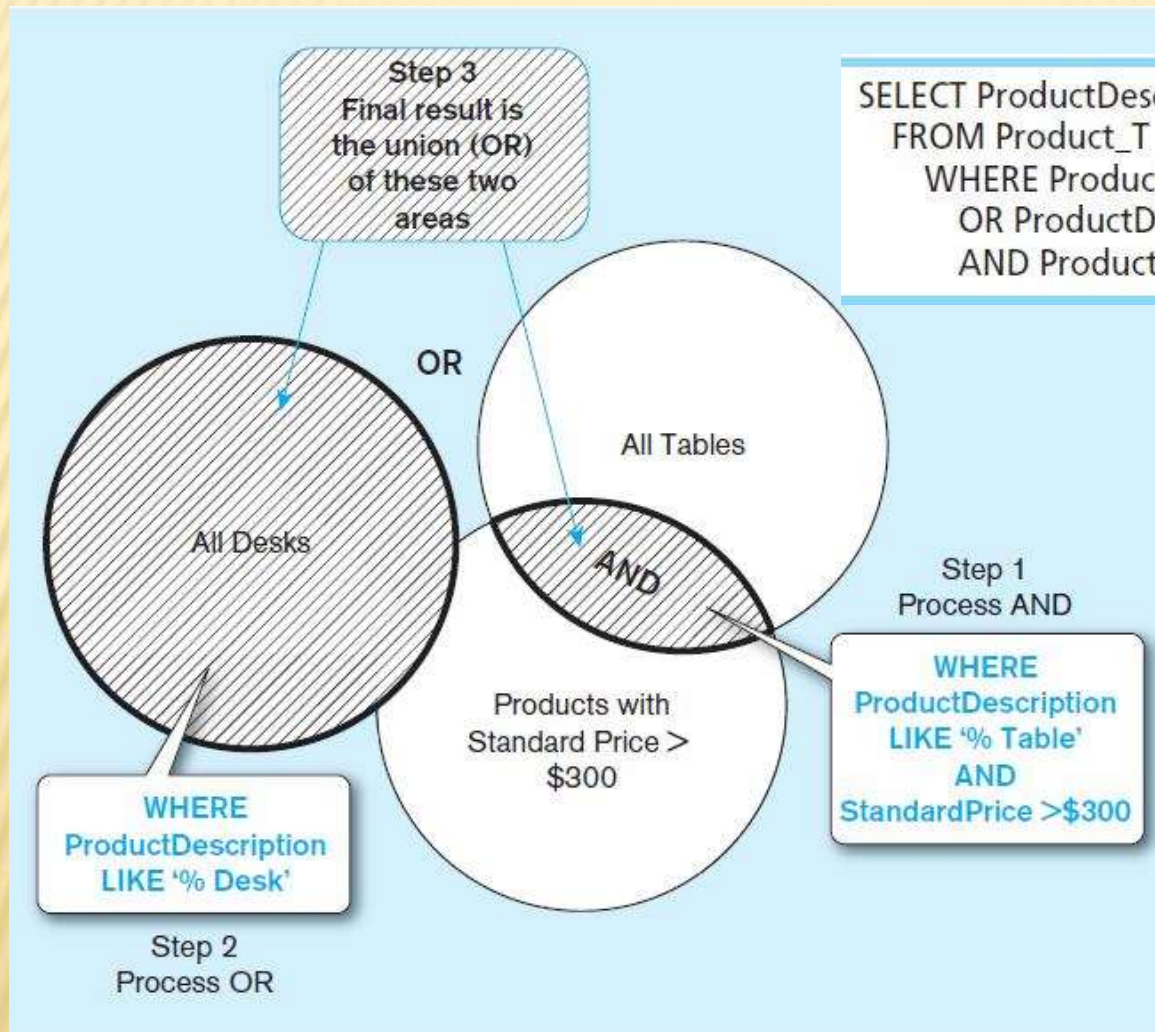
- ✗ Modifies data in existing rows

```
UPDATE Product_T  
SET ProductStandardPrice = 775  
WHERE ProductID = 7;
```

SCHEMA DEFINITION

- ✗ Control processing/storage efficiency:
 - + Choice of indexes
 - + File organizations for base tables
 - + File organizations for indexes
 - + Data clustering
 - + Statistics maintenance
- ✗ Creating indexes
 - + Speed up random/sequential access to base table data
 - + Example
 - ✗ `CREATE INDEX NAME_IDX ON CUSTOMER_T(CUSTOMERNAME)`
 - ✗ This makes an index for the CUSTOMERNAME field of the CUSTOMER_T table

Figure 6-7 Boolean query A without use of parentheses



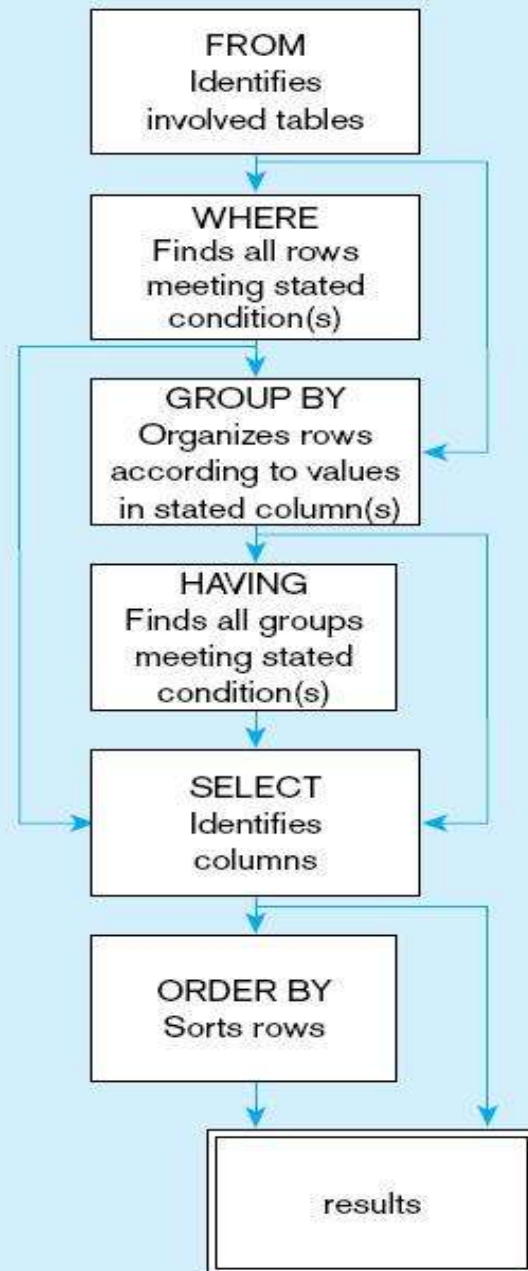
```
SELECT ProductDescription, ProductFinish, ProductStandardPrice  
FROM Product_T  
WHERE ProductDescription LIKE '%Desk'  
OR ProductDescription LIKE '%Table'  
AND ProductStandardPrice > 300;
```

By default,
processing order
of Boolean
operators is NOT,
then AND, then
OR

SELECT STATEMENT

- ✗ Used for queries on single or multiple tables
- ✗ Clauses of the SELECT statement:
 - + **SELECT**
 - ✗ List the columns (and expressions) to be returned from the query
 - + **FROM**
 - ✗ Indicate the table(s) or view(s) from which data will be obtained
 - + **WHERE**
 - ✗ Indicate the conditions under which a row will be included in the result
 - + **GROUP BY**
 - ✗ Indicate categorization of results
 - + **HAVING**
 - ✗ Indicate the conditions under which a category (group) will be included
 - + **ORDER BY**
 - ✗ Sorts the result according to specified criteria

Figure 6-9
SQL statement
processing
order (based
on van der
Lans, 2006
p.100)



SELECT EXAMPLE



- ✖ Find products with standard price less than \$275

```
SELECT ProductDescription, ProductStandardPrice  
FROM Product_T  
WHERE ProductStandardPrice < 275;
```

Table 6-3: Comparison Operators in SQL

TABLE 6-3 Comparison Operators in SQL

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to
!=	Not equal to

SELECT EXAMPLE USING ALIAS

- ✖ Alias is an alternative column or table name

```
SELECT CUST.CustomerName AS Name, CUST.CustomerAddress  
FROM ownerid.Customer_T AS Cust  
WHERE Name = 'Home Furnishings';
```

- ✖ Here, CUST is a table alias and Name is a column alias

SELECT EXAMPLE USING A FUNCTION

- ✗ Using the COUNT *aggregate function* to find totals

```
SELECT COUNT(*) FROM ORDERLINE_T  
WHERE ORDERID = 1004;
```

Note: with aggregate functions you can't have single-valued columns included in the SELECT clause, unless they are included in the GROUP BY clause.

SELECT EXAMPLE–BOOLEAN OPERATORS

- ✗ **AND**, **OR**, and **NOT** Operators for customizing conditions in WHERE clause

```
SELECT ProductDescription, ProductFinish, ProductStandardPrice  
FROM Product_T  
WHERE ProductDescription LIKE '%Desk'  
OR ProductDescription LIKE '%Table'  
AND ProductStandardPrice > 300;
```

Note: the **LIKE** operator allows you to compare strings using wildcards. For example, the % wildcard in '%Desk' indicates that all strings that have any number of characters preceding the word "Desk" will be allowed.

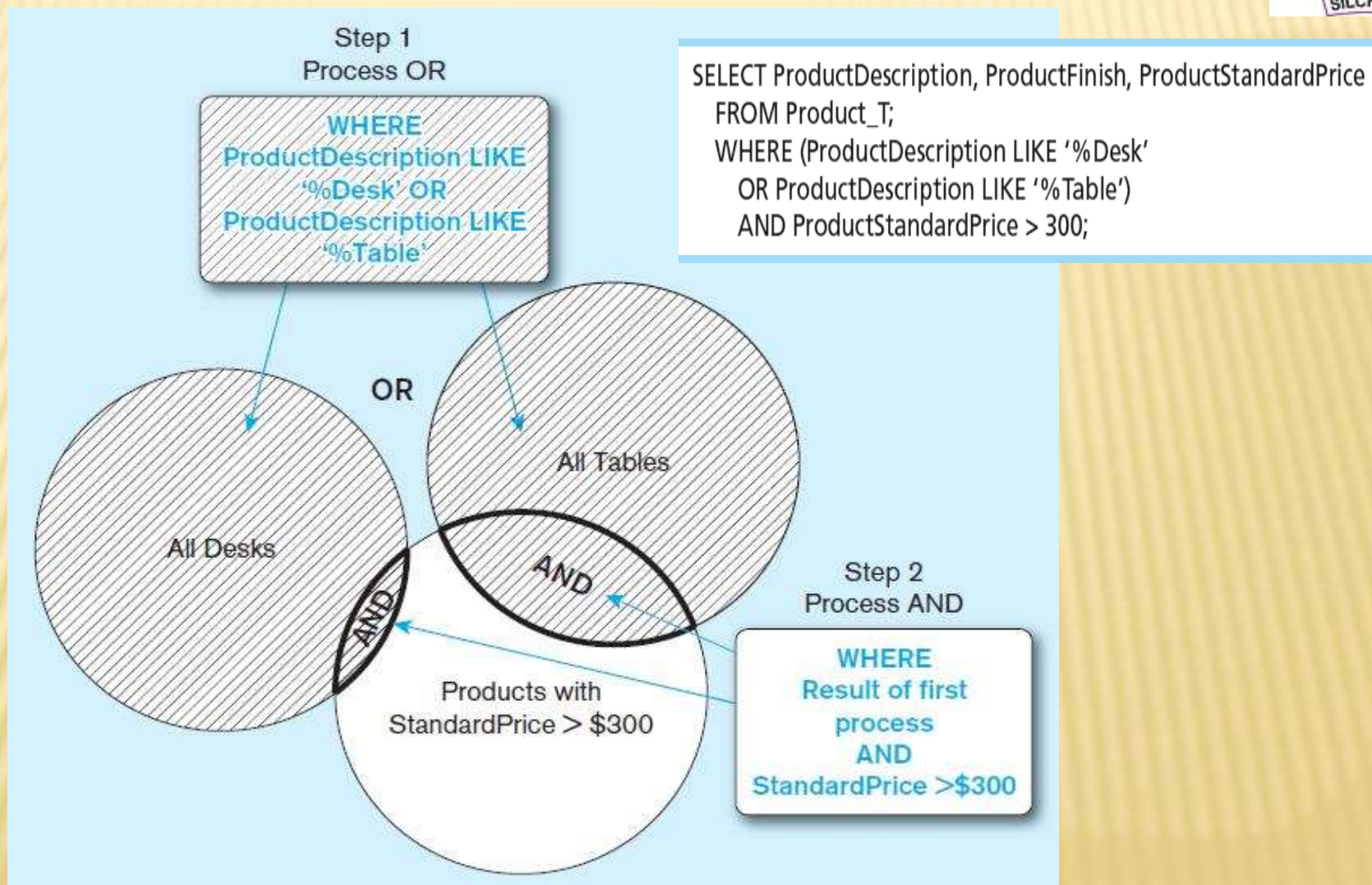
SELECT EXAMPLE–BOOLEAN OPERATORS

- ✗ With parentheses...these override the normal precedence of Boolean operators

```
SELECT ProductDescription, ProductFinish, ProductStandardPrice  
FROM Product_T;  
WHERE (ProductDescription LIKE '%Desk'  
      OR ProductDescription LIKE '%Table')  
      AND ProductStandardPrice > 300;
```

With parentheses, you can override normal precedence rules. In this case parentheses make the OR take place before the AND.

Figure 6-8 Boolean query B with use of parentheses



SORTING RESULTS WITH ORDER BY CLAUSE

- ✖ Sort the results first by STATE, and within a state by the CUSTOMER NAME

```
SELECT CustomerName, CustomerCity, CustomerState  
FROM Customer_T  
WHERE CustomerState IN ('FL', 'TX', 'CA', 'HI')  
ORDER BY CustomerState, CustomerName;
```

Note: the IN operator in this example allows you to include rows whose CustomerState value is either FL, TX, CA, or HI. It is more efficient than separate OR conditions.

CATEGORIZING RESULTS USING GROUP BY CLAUSE

✖ For use with aggregate functions

- + **Scalar aggregate**: single value returned from SQL query with aggregate function
- + **Vector aggregate**: multiple values returned from SQL query with aggregate function (via GROUP BY)

```
SELECT CustomerState, COUNT (CustomerState)
FROM Customer_T
GROUP BY CustomerState;
```

You can use single-value fields with aggregate functions if they are included in the GROUP BY clause.

QUALIFYING RESULTS BY CATEGORIES USING THE HAVING CLAUSE

✗ For use with GROUP BY

```
SELECT CustomerState, COUNT (CustomerState)
FROM Customer_T
GROUP BY CustomerState
HAVING COUNT (CustomerState) > 1;
```

Like a WHERE clause, but it operates on groups (categories), not on individual rows. Here, only those groups with total numbers greater than 1 will be included in final result.

USING AND DEFINING VIEWS

- ✗ Views provide users controlled access to tables
- ✗ Base Table–table containing the raw data
- ✗ Virtual Table–constructed automatically as needed; not maintained as real data
- ✗ Dynamic View
 - + A “virtual table” created dynamically upon request by a user
 - + No data actually stored; instead data from base table made available to user
 - + Based on SQL SELECT statement on base tables or other views
 - + Contents materialized as a result of a query

SAMPLE CREATE VIEW

Query: What are the data elements necessary to create an invoice for a customer?
Save this query as a view named Invoice_V.

```
CREATE VIEW Invoice_V AS
SELECT Customer_T.CustomerID, CustomerAddress, Order_T.OrderID,
Product_T.ProductID, ProductStandardPrice,
OrderedQuantity, and other columns as required
FROM Customer_T, Order_T, OrderLine_T, Product_T
WHERE Customer_T.CustomerID = Order_T.CustomerID
AND Order_T.OrderID = OrderLine_T.OrderID
AND Product_T.ProductID = OrderLine_T.ProductID;
```

ADVANTAGES OF VIEWS

- ✗ Simplify query commands
- ✗ Assist with data security (but don't rely on views for security, there are more important security measures)
- ✗ Enhance programming productivity
- ✗ Contain most current base table data
- ✗ Use little storage space
- ✗ Provide customized view for user
- ✗ Establish physical data independence

DISADVANTAGES OF VIEWS

- ✗ Use processing time each time view is referenced
- ✗ May or may not be directly updateable