Database Management Systems
Chapter 6: Introduction to SQL

UNIVERSITY OF MARYLAND

ROBERT H. SMITH SCHOOL OF BUSINESS

DR. ADAM LEE

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.

SQL Overview

- **S**tructured **Q**uery **L**anguage
- The standard for **r**elational **d**ata**b**ase **m**anagement **s**ystems (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 developed 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**, **2011** – Major ANSI standard updates.
- **Current** – SQL is supported by most major database vendors.

UNIVERSITY OF MARYLAND

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

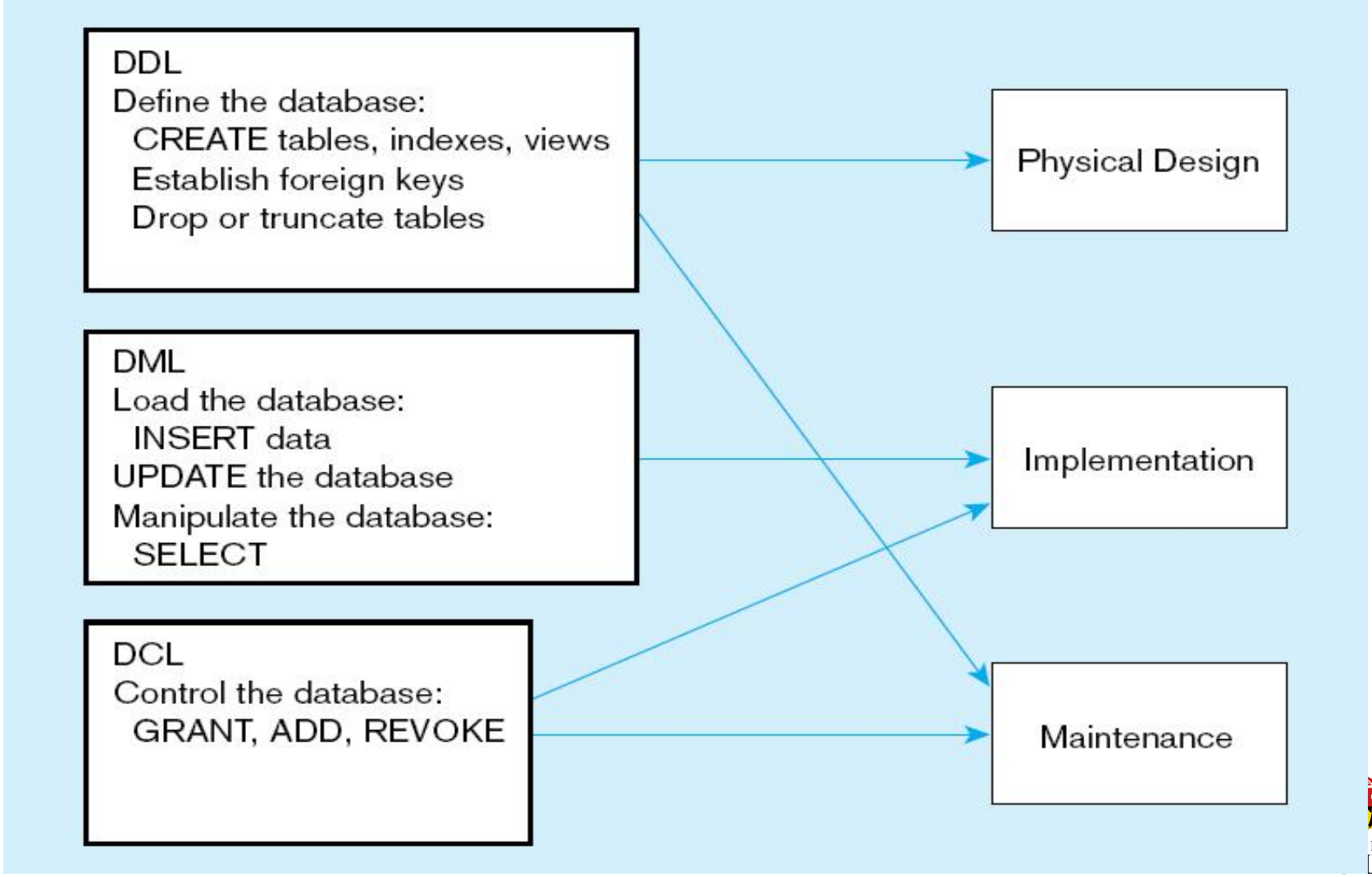
- **Catalog**:
  - A set of schemas that constitute the description of a database.
- **Schema**:
  - The structure that contains descriptions of objects created by a user (base tables, views, constraints).
- Data Definition Language (**DDL**):
  - Commands that define a database, including creating, 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.

UNIVERSITY OF MARYLAND

Table 6-2: 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:
  - **CREATE CHARACTER SET**
  - **CREATE ASSERTION**
  - **CREATE DOMAIN**

Steps in Table Creation

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

Figure 6-5: General Syntax for CREATE TABLE statement used in DDL

```
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]
```

Figure 1-3: Project Level Data Models

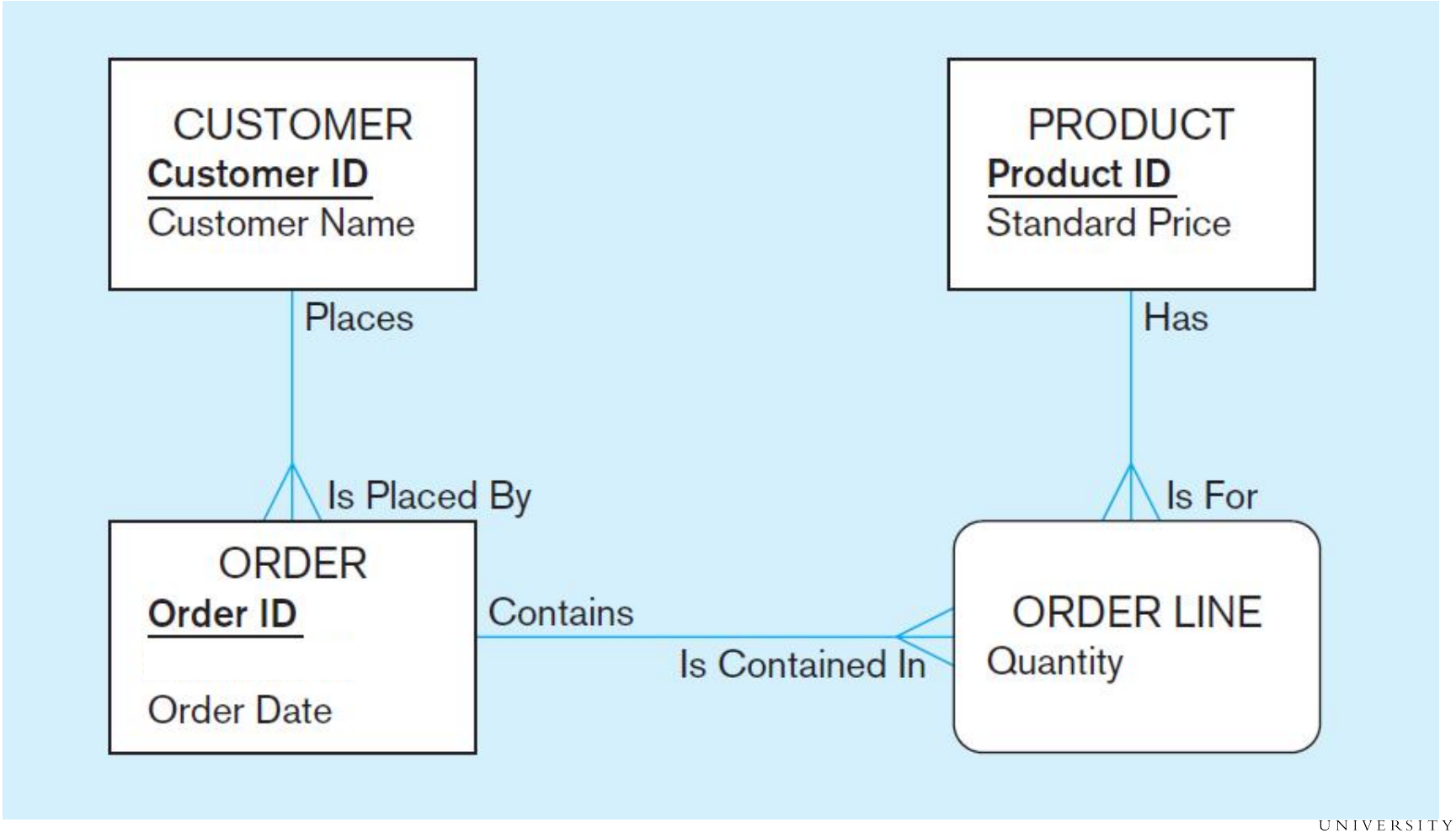


Figure 6-6: SQL Database Definition Commands (Pine Valley Furniture Company)

```
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));
```

Overall table definitions

Figure 6-6: Attributes and Data Types (Pine Valley Furniture Company)

Attribute (no space) DATA TYPE

```
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));
```

Figure 6-6: Required Attribute and Primary Key (Pine Valley Furniture Company)

```
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

Identifying primary key

Primary keys can never have NULL values – entity integrity.

UNIVERSITY OF MARYLAND

Figure 6-6: Composite Primary Key (Pine Valley Furniture Company)

```
CREATE TABLE OrderLine_T                                    Non-nullable specification
        (OrderID                    NUMBER(11,0)      NOT NULL,
         ProductID                  INTEGER           NOT NULL,
         OrderedQuantity            NUMBER(11,0),
CONSTRAINT OrderLine_PK PRIMARY KEY (OrderID, ProductID),   Identifying primary key
CONSTRAINT OrderLine_FK1 FOREIGN KEY (OrderID) REFERENCES Order_T(OrderID),
CONSTRAINT OrderLine_FK2 FOREIGN KEY (ProductID) REFERENCES Product_T(ProductID));
```

Some primary keys are composite – composed of multiple attributes.

Figure 6-6: Default Value and Domain (Pine Valley Furniture Company)

```
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

Figure 6-6: Foreign Keys and Relationships (Pine Valley Furniture Company)

Primary key of parent table

Foreign key of dependent table

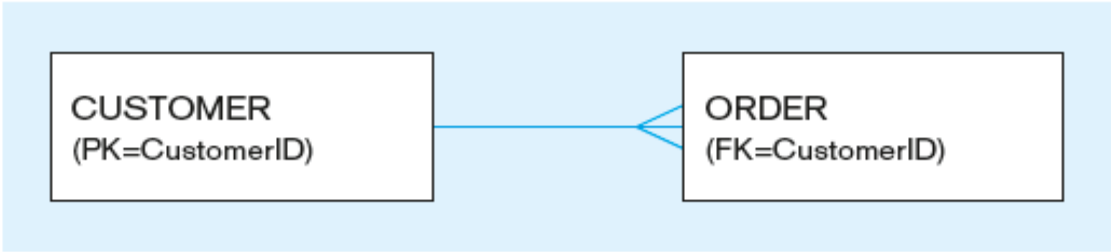
```
CREATE TABLE Customer_T
                (CustomerID          NUMBER(11,0)        NOT NULL,
                 CustomerName        VARCHAR2(25)        NOT NULL,
                 CustomerAddress     VARCHAR2(30),
                 CustomerCity        VARCHAR2(20),
                 CustomerState       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));
```

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.

Figure 6-7: Ensuring Data Integrity through Updates

Relational integrity is enforced via the primary key to foreign key match



Restricted Update: A customer ID can only be deleted if it is not found in ORDER table.

```
CREATE TABLE CustomerT
                (CustomerID          INTEGER DEFAULT '999'     NOT NULL,
                 CustomerName        VARCHAR(40)               NOT NULL,
                 . . .
CONSTRAINT Customer_PK PRIMARY KEY (CustomerID),
ON UPDATE RESTRICT);
```

Cascaded Update: Changing a customer ID in the CUSTOMER table will result in that value changing in the ORDER table to match.

```
. . . ON UPDATE CASCADE);
```

Set Null Update: When a customer ID is changed, any customer ID in the ORDER table that matches the old customer ID is set to NULL.

```
. . . ON UPDATE SET NULL);
```

Set Default Update: When a customer ID is changed, any customer ID in the ORDER tables that matches the old customer ID is set to a predefined default value.

```
. . . ON UPDATE SET DEFAULT);
```

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**;**

UNIVERSITY OF MARYLAND

UPDATE Statement

- Modifies data in existing rows:

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

MERGE Statement

```
MERGE INTO Product_T AS PROD
USING
(SELECT ProductID, ProductDescription, ProductFinish,
ProductStandardPrice, ProductLineID FROM Purchases_T) AS PURCH
    ON (PROD.ProductID = PURCH.ProductID)
WHEN MATCHED THEN UPDATE
    PROD.ProductStandardPrice = PURCH.ProductStandardPrice
WHEN NOT MATCHED THEN INSERT
    (ProductID, ProductDescription, ProductFinish, ProductStandardPrice,
    ProductLineID)
    VALUES(PURCH.ProductID, PURCH.ProductDescription,
    PURCH.ProductFinish, PURCH.ProductStandardPrice,
      PURCH.ProductLineID);
```

- Makes it easier to update a table … allows combination of **INSERT** and **UPDATE** in one statement.
- Useful for updating master tables with new data.

UNIVERSITY OF MARYLAND

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.

UNIVERSITY OF MARYLAND

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 6-10: SQL Processing Order

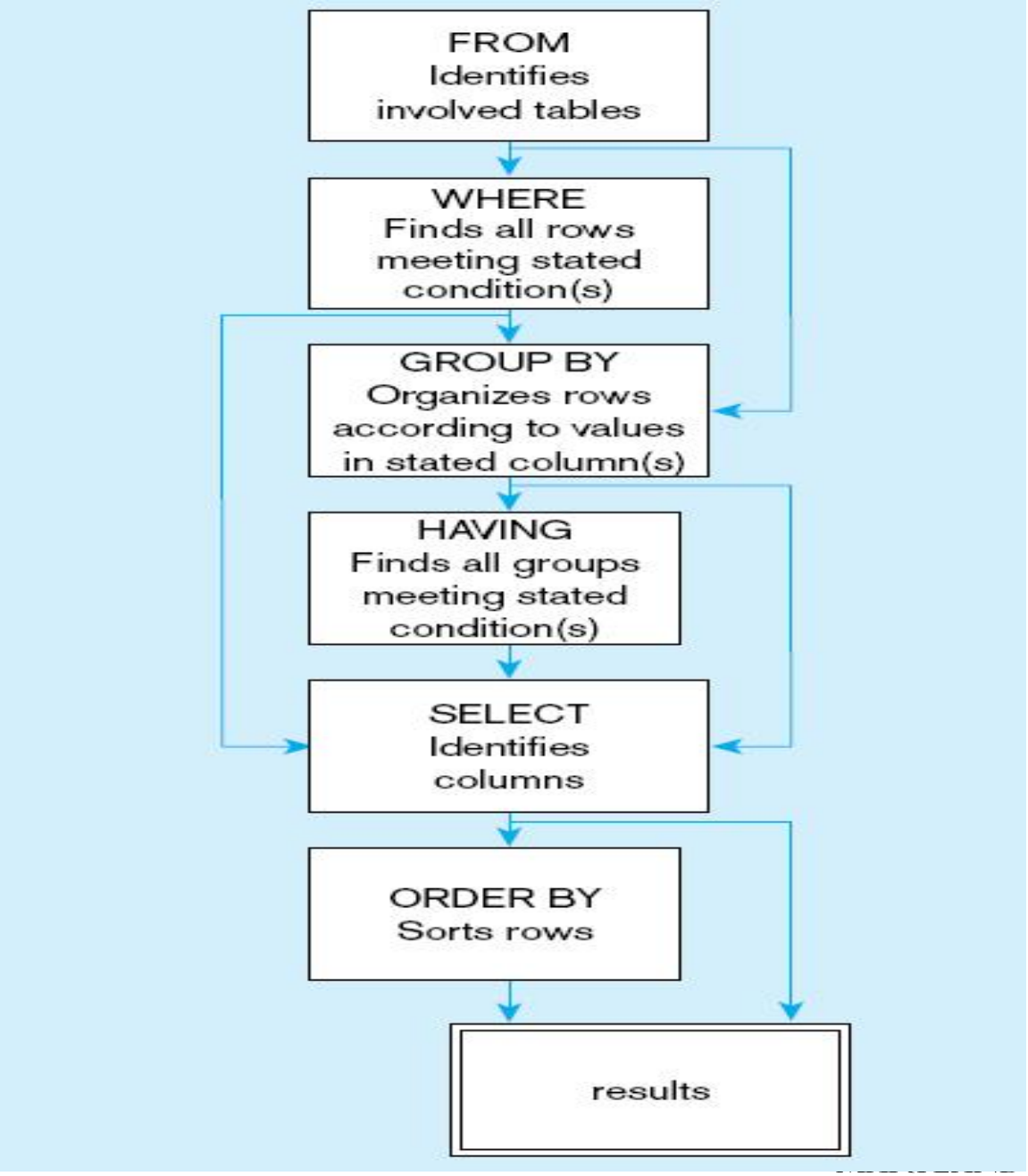


Table 6-3: Comparison Operators in SQL

- Find products with standard price less than $275:

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

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 Customer_V Cust
WHERE Name = 'Home Furnishings';
```

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:

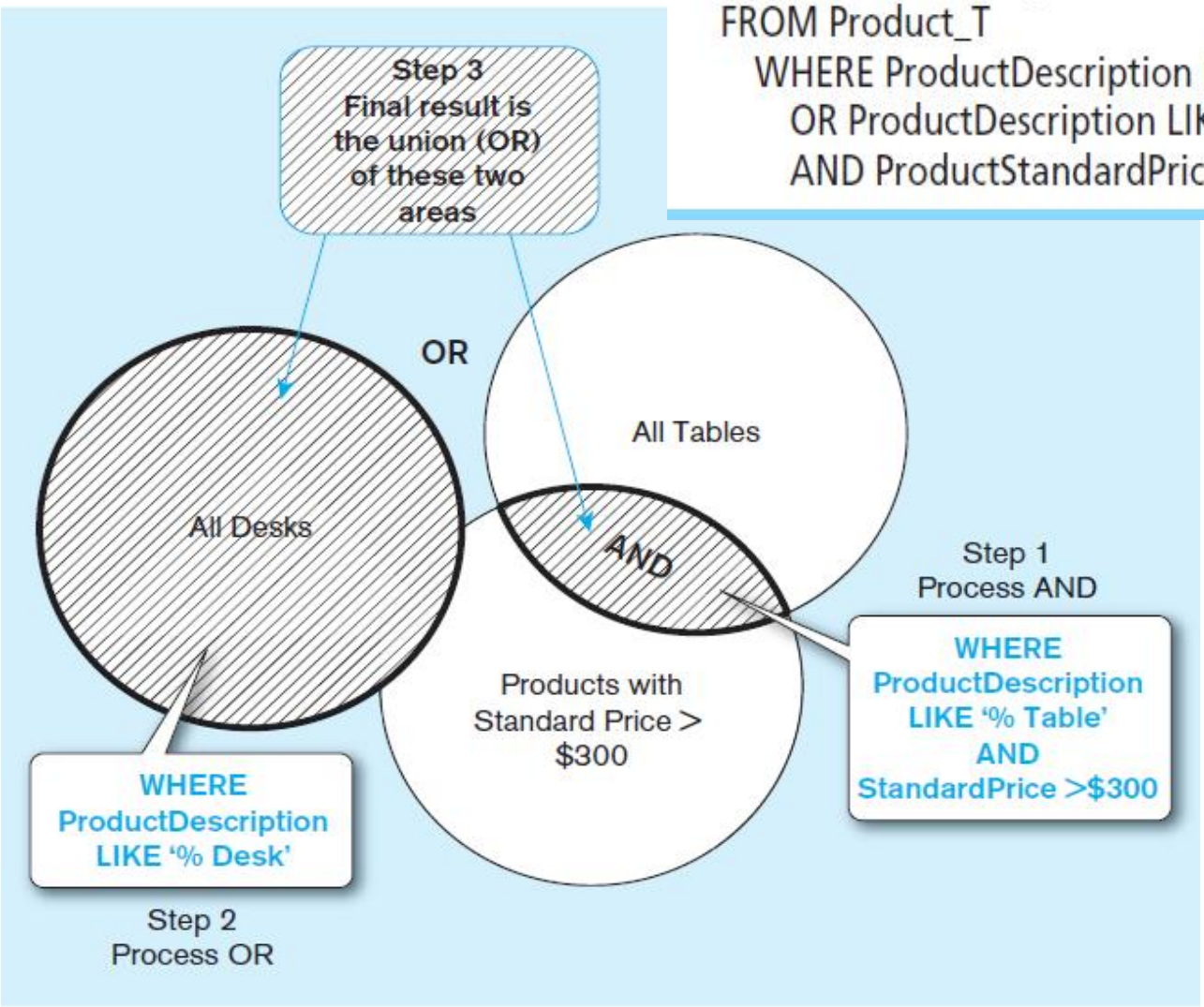
```
SELECT ProductDescription, ProductFinish, ProductStandardPrice
   FROM Product_T
      WHERE ProductDescription LIKE '%Desk'
         OR ProductDescription LIKE '%Table'
         AND ProductStandardPrice > 300;
```

- 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.

UNIVERSITY OF MARYLAND

Figure 6-8: Boolean Query 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 Example – BOOLEAN Operators

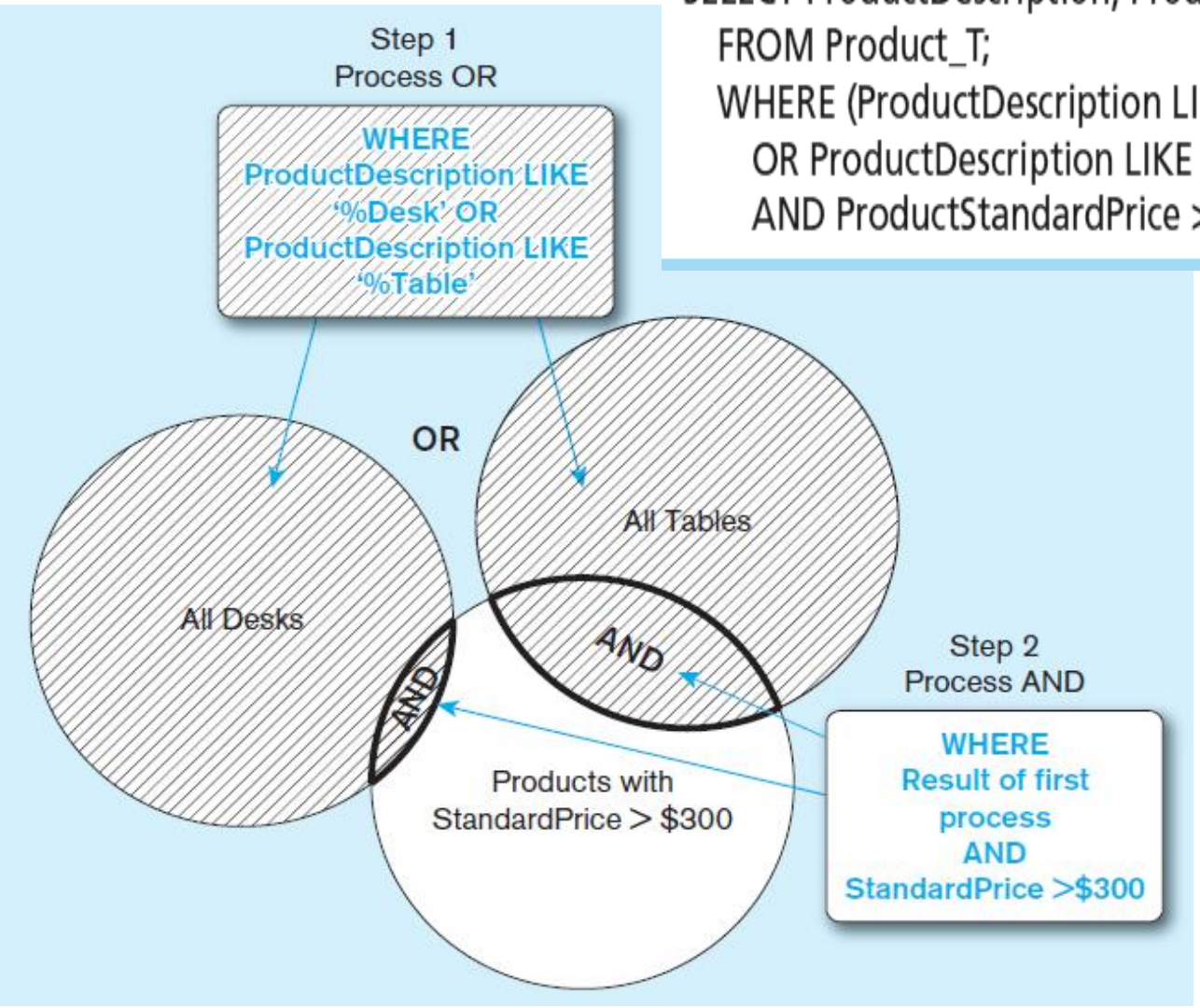
- With parentheses…these override the normal precedence of Boolean operators:

```
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 6-9: Boolean Query with Use of Parentheses

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



Sorting Results with ORDER BY Clause

- Sort the results first by CustomerState, and within a state by the CustomerName

```
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.

UNIVERSITY OF MARYLAND

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.
- **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.
- **Materialized View**:
  - Copy or replication of data.
  - Data actually stored.
  - Must be refreshed periodically to match corresponding base tables.

Sample CREATE VIEW

```
CREATE VIEW ExpensiveStuff_V
   AS
      SELECT ProductID, ProductDescription, ProductStandardPrice
         FROM Product_T
            WHERE ProductStandardPrice > 300
            WITH CHECK OPTION;
```

- View has a name.
- View is based on a **SELECT** statement.
- **CHECK OPTION** works only for updateable views and prevents updates that would create rows not included in the view.

UNIVERSITY OF MARYLAND

Advantages and Disadvantages of Views

- 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.

UNIVERSITY OF MARYLAND