Database Management System

Sumayyea Salahuddin (Lecturer)
Dept. of Computer Systems Eng.
UET Peshawar

Overview

- Introduction to Structured Query Language (SQL)
  - Environment
  - Data types
  - Data Definition Language
  - Data Manipulation Language

- Example

SQL

- Structured Query Language
- The standard for relational database management systems (RDBMS)
- SQL-92 Standard
- Purpose
  - Specify syntax/semantics for data definition and manipulation
  - Define data structures
  - Enable portability
  - Specify minimal (level 1) and complete (level 2) standards
  - Allow for later growth/enhancement to standard

Benefits of 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 description 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:2000n standards
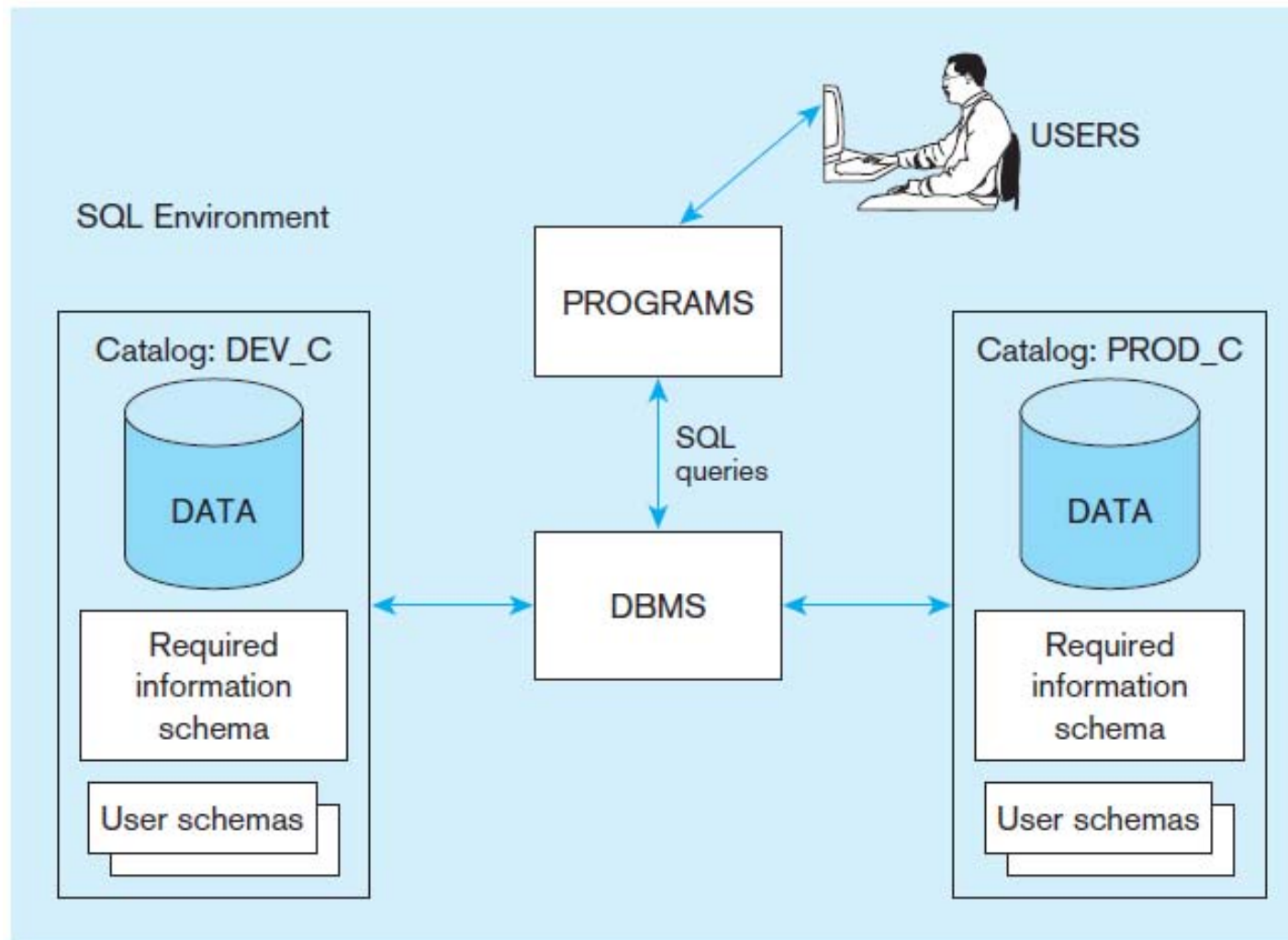
SQL Data types

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	TIMESTAMPTIMESTAMP 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.

SQL Languages
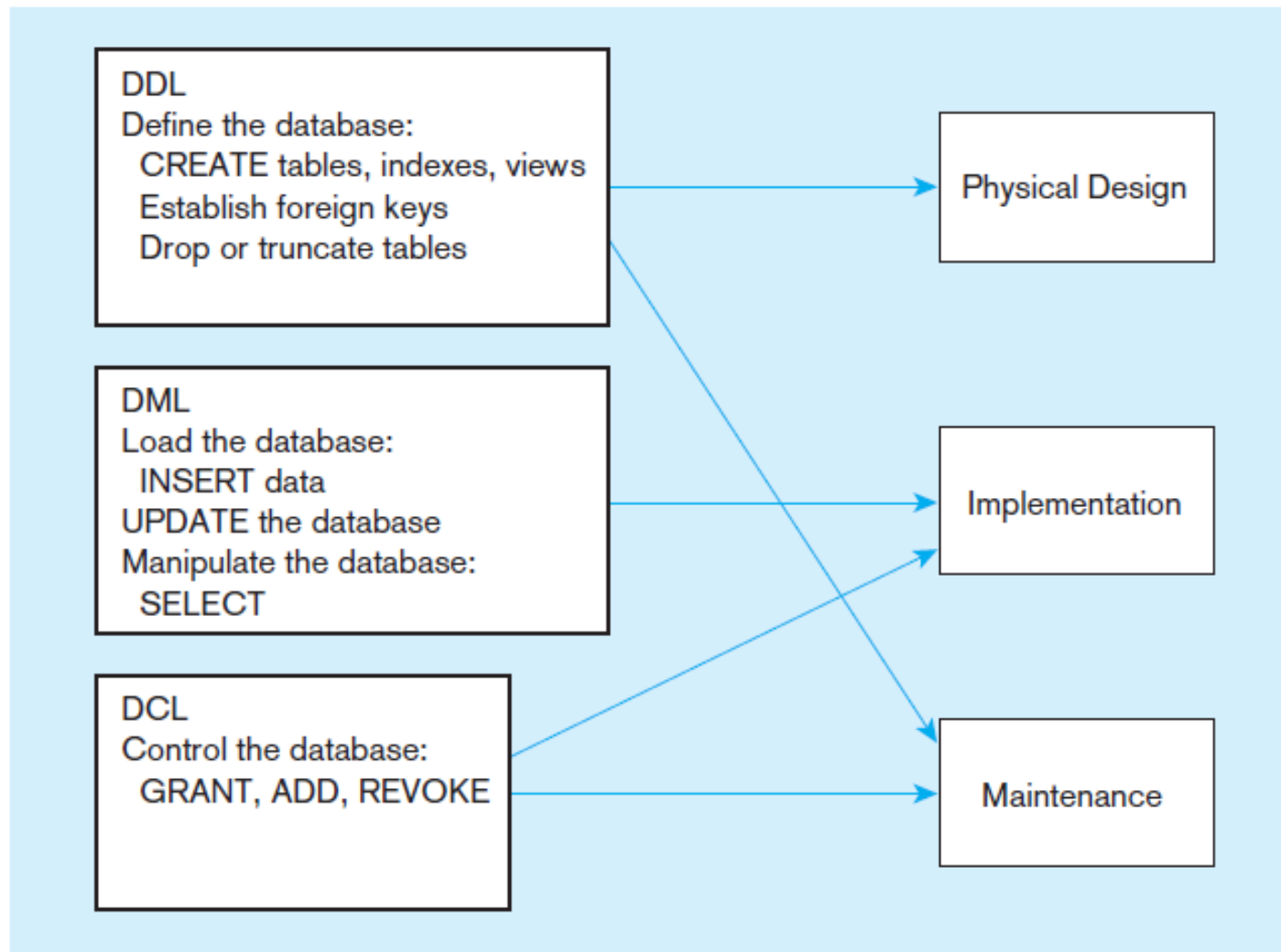
SQL Database Definition

- Data Definition Language (DDL)

- Major CREATE statements:

  – CREATE DATABASE: defines a portion of the database owned by a particular user

  – CREATE TABLE : defines a table and its columns

  – CREATE VIEW : defines a logical table from one or more views

- Other CREATE statements : CHARACTER SET, COLLATION, TRANSLATION, ASSERTION, DOMAIN

Table Creation

Steps in table creation:

Figure 6-5 General syntax for CREATE TABLE

```
CREATE TABLE tablename
( {column definition      [table constraint] } . , . .
[ON COMMIT {DELETE I P RESERVE} ROWS] );

where column definition ::=
column_name
        {domain namel d atatype [(size)] }
        [column_constraint_clause. . .]
        [default value]
        [collate clause]

and table constraint ::=
        [CONSTRAINT constraint_name]
        Constraint_type [constraint_attributes]
```
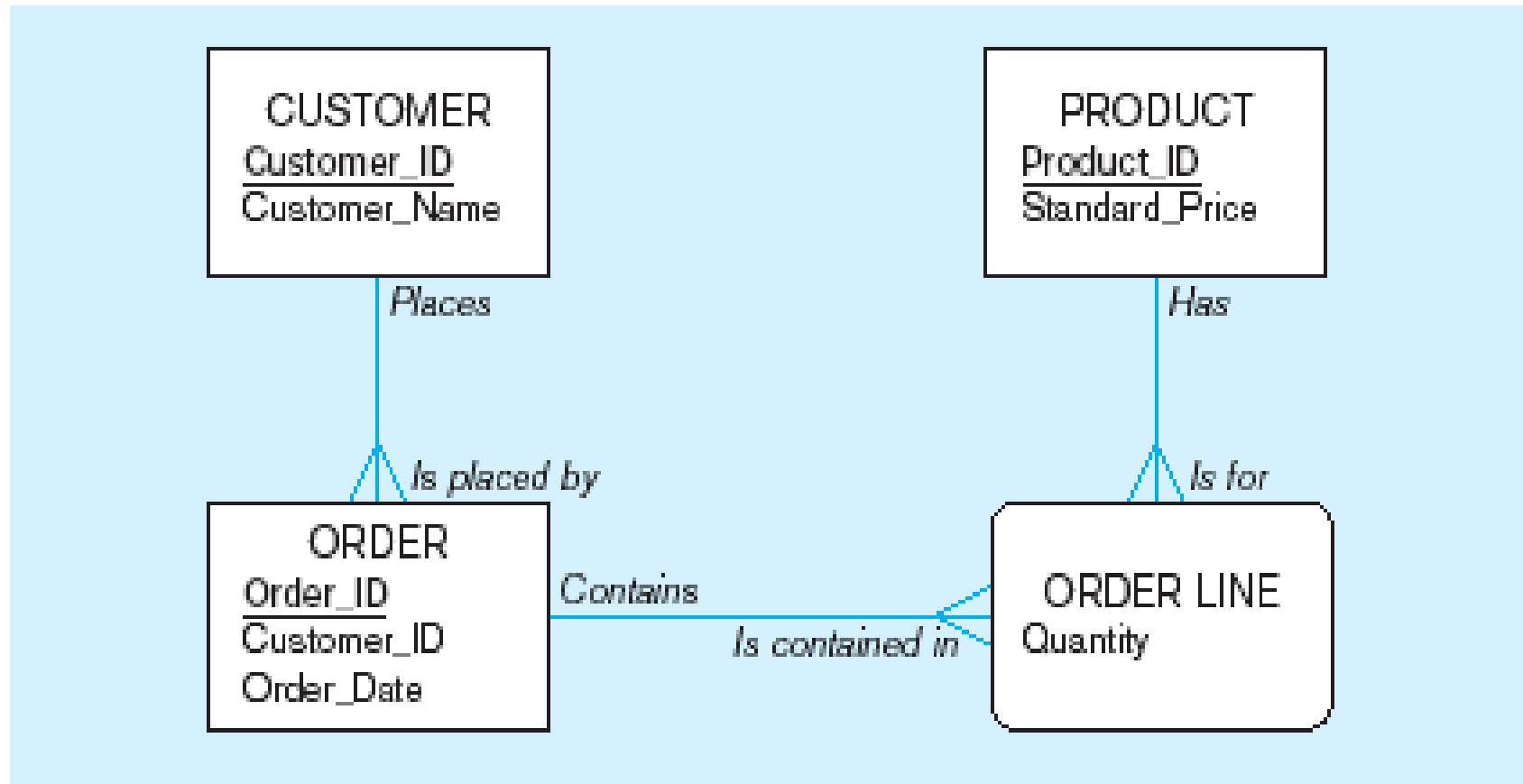
1. Identify data types for attributes

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

Sample Pine Valley Furniture Data Model

Sample Pine Valley Furniture Data

customers

orders

order lines

products

SQL Database Definition Commands

```
CREATE TABLE CUSTOMER_T
        (CUSTOMER_ID                    NUMBER(11, 0) NOT NULL,
         CUSTOMER_NAME                  VARCHAR2(25)  NOT NULL,
         CUSTOMER_ADDRESS               VARCHAR2(30),
         CITY                           VARCHAR2(20),
         STATE                          VARCHAR2(2),
         POSTAL_CODE                    VARCHAR2(9),
CONSTRAINT CUSTOMER_PK PRIMARY KEY (CUSTOMER_ID));

CREATE TABLE ORDER_T
        (ORDER_ID                       NUMBER(11, 0) NOT NULL,
         ORDER_DATE                     DATE DEFAULT SYSDATE,
         CUSTOMER_ID                    NUMBER(11, 0),
CONSTRAINT ORDER_PK PRIMARY KEY (ORDER_ID),
CONSTRAINT ORDER_FK FOREIGN KEY (CUSTOMER_ID) REFERENCES CUSTOMER_T(CUSTOMER_ID));

CREATE TABLE PRODUCT_T
        (PRODUCT_ID                     INTEGER       NOT NULL,
         PRODUCT_DESCRIPTION            VARCHAR2(50),
         PRODUCT_FINISH                 VARCHAR2(20)
                        CHECK (PRODUCT_FINISH IN ('Cherry', 'Natural Ash', 'White Ash',
                                'Red Oak', 'Natural Oak', 'Walnut')),
         STANDARD_PRICE                 DECIMAL(6,2),
         PRODUCT_LINE_ID                INTEGER,
CONSTRAINT PRODUCT_PK PRIMARY KEY (PRODUCT_ID));

CREATE TABLE ORDER_LINE_T
        (ORDER_ID                       NUMBER(11,0)  NOT NULL,
         PRODUCT_ID                     NUMBER(11,0)  NOT NULL,
         ORDERED_QUANTITY               NUMBER(11,0),
CONSTRAINT ORDER_LINE_PK PRIMARY KEY (ORDER_ID, PRODUCT_ID),
CONSTRAINT ORDER_LINE_FK1 FOREIGN KEY(ORDER_ID) REFERENCES ORDER_T(ORDER_ID),
CONSTRAINT ORDER_LINE_FK2 FOREIGN KEY (PRODUCT_ID) REFERENCES PRODUCT_T(PRODUCT_ID));
```

SQL Database Definition Commands (cont)

```
CREATE TABLE CUSTOMER_T
         (CUSTOMER_ID                NUMBER(11, 0) NOT NULL,
          CUSTOMER_NAME             VARCHAR2(25) NOT NULL,
          CUSTOMER_ADDRESS          VARCHAR2(30),
          CITY                      VARCHAR2(20),
          STATE                     VARCHAR2(2),
          POSTAL_CODE               VARCHAR2(9),
CONSTRAINT CUSTOMER_PK PRIMARY KEY (CUSTOMER_ID));

CREATE TABLE ORDER_T
         (ORDER_ID                   NUMBER(11, 0) NOT NULL,
          ORDER_DATE                 DATE DEFAULT SYSDATE,
          CUSTOMER_ID                NUMBER(11, 0),
CONSTRAINT ORDER_PK PRIMARY KEY (ORDER_ID),
CONSTRAINT ORDER_FK FOREIGN KEY (CUSTOMER_ID) REFERENCES CUSTOMER_T(CUSTOMER_ID));

CREATE TABLE PRODUCT_T
         (PRODUCT_ID                 INTEGER         NOT NULL,
          PRODUCT_DESCRIPTION        VARCHAR2(50),
          PRODUCT_FINISH             VARCHAR2(20)
                         CHECK (PRODUCT_FINISH IN ('Cherry', 'Natural Ash', 'White Ash',
                                'Red Oak', 'Natural Oak', 'Walnut')),
          STANDARD_PRICE             DECIMAL(6,2),
          PRODUCT_LINE_ID            INTEGER,
CONSTRAINT PRODUCT_PK PRIMARY KEY (PRODUCT_ID));

CREATE TABLE ORDER_LINE_T
         (ORDER_ID                   NUMBER(11,0) NOT NULL,
          PRODUCT_ID                 NUMBER(11,0) NOT NULL,
          ORDERED_QUANTITY           NUMBER(11,0),
CONSTRAINT ORDER_LINE_PK PRIMARY KEY (ORDER_ID, PRODUCT_ID),
CONSTRAINT ORDER_LINE_FK1 FOREIGN KEY(ORDER_ID) REFERENCES ORDER_T(ORDER_ID),
CONSTRAINT ORDER_LINE_FK2 FOREIGN KEY (PRODUCT_ID) REFERENCES PRODUCT_T(PRODUCT_ID));
```

Defining attributes and their data types

SQL Database Definition Commands (cont)

```
CREATE TABLE CUSTOMER_T
        (CUSTOMER_ID                    NUMBER(11, 0) NOT NULL,
         CUSTOMER_NAME                  VARCHAR2(25) NOT NULL,
         CUSTOMER_ADDRESS               VARCHAR2(30),
         CITY                           VARCHAR2(20),
         STATE                          VARCHAR2(2),
         POSTAL_CODE                    VARCHAR2(9),
CONSTRAINT CUSTOMER_PK PRIMARY KEY (CUSTOMER_ID));

CREATE TABLE ORDER_T
        (ORDER_ID                       NUMBER(11, 0) NOT NULL,
         ORDER_DATE                     DATE DEFAULT SYSDATE,
         CUSTOMER_ID                    NUMBER(11, 0),
CONSTRAINT ORDER_PK PRIMARY KEY (ORDER_ID),
CONSTRAINT ORDER_FK FOREIGN KEY (CUSTOMER_ID) REFERENCES CUSTOMER_T(CUSTOMER_ID));

CREATE TABLE PRODUCT_T
        (PRODUCT_ID                     INTEGER        NOT NULL,
         PRODUCT_DESCRIPTION            VARCHAR2(50),
         PRODUCT_FINISH                 VARCHAR2(20)
                            CHECK (PRODUCT_FINISH IN ('Cherry', 'Natural Ash', 'White Ash',
                                    'Red Oak', 'Natural Oak', 'Walnut')),
         STANDARD_PRICE                 DECIMAL(6,2),
         PRODUCT_LINE_ID                INTEGER,
CONSTRAINT PRODUCT_PK PRIMARY KEY (PRODUCT_ID));

CREATE TABLE ORDER_LINE_T
        (ORDER_ID                       NUMBER(11,0) NOT NULL,
         PRODUCT_ID                     NUMBER(11,0) NOT NULL,
         ORDERED_QUANTITY               NUMBER(11,0),
CONSTRAINT ORDER_LINE_PK PRIMARY KEY (ORDER_ID, PRODUCT_ID),
CONSTRAINT ORDER_LINE_FK1 FOREIGN KEY(ORDER_ID) REFERENCES ORDER_T(ORDER_ID),
CONSTRAINT ORDER_LINE_FK2 FOREIGN KEY (PRODUCT_ID) REFERENCES PRODUCT_T(PRODUCT_ID));
```

Non-nullable Specifications

Note: primary keys should not be null

SQL Database Definition Commands (cont)

```
CREATE TABLE CUSTOMER_T
        (CUSTOMER_ID              NUMBER(11, 0) NOT NULL,
         CUSTOMER_NAME            VARCHAR2(25)  NOT NULL,
         CUSTOMER_ADDRESS         VARCHAR2(30),
         CITY                     VARCHAR2(20),
         STATE                    VARCHAR2(2),
         POSTAL_CODE              VARCHAR2(9),
CONSTRAINT CUSTOMER_PK PRIMARY KEY (CUSTOMER_ID));
CREATE TABLE ORDER_T
        (ORDER_ID                 NUMBER(11, 0) NOT NULL,
         ORDER_DATE               DATE DEFAULT SYSDATE,
         CUSTOMER_ID              NUMBER(11, 0),
CONSTRAINT ORDER_PK PRIMARY KEY (ORDER_ID),
CONSTRAINT ORDER_FK FOREIGN KEY (CUSTOMER_ID) REFERENCES CUSTOMER_T(CUSTOMER_ID));
CREATE TABLE PRODUCT_T
        (PRODUCT_ID               INTEGER       NOT NULL,
         PRODUCT_DESCRIPTION      VARCHAR2(50),
         PRODUCT_FINISH           VARCHAR2(20)
                                  CHECK (PRODUCT_FINISH IN ('Cherry', 'Natural Ash', 'White Ash',
                                      'Red Oak', 'Natural Oak', 'Walnut')),
         STANDARD_PRICE           DECIMAL(6,2),
         PRODUCT_LINE_ID          INTEGER,
CONSTRAINT PRODUCT_PK PRIMARY KEY (PRODUCT_ID));
CREATE TABLE ORDER_LINE_T
        (ORDER_ID                 NUMBER(11,0)  NOT NULL,
         PRODUCT_ID               NUMBER(11,0)  NOT NULL,
         ORDERED_QUANTITY         NUMBER(11,0),
CONSTRAINT ORDER_LINE_PK PRIMARY KEY (ORDER_ID, PRODUCT_ID),
CONSTRAINT ORDER_LINE_FK1 FOREIGN KEY (ORDER_ID) REFERENCES ORDER_T(ORDER_ID),
CONSTRAINT ORDER_LINE_FK2 FOREIGN KEY (PRODUCT_ID) REFERENCES PRODUCT_T(PRODUCT_ID));
```

Identifying Primary keys

This is composite primary key

SQL Database Definition Commands (cont)

```
CREATE TABLE CUSTOMER_T
        (CUSTOMER_ID                NUMBER(11, 0) NOT NULL,
         CUSTOMER_NAME              VARCHAR2(25)  NOT NULL,
         CUSTOMER_ADDRESS           VARCHAR2(30),
         CITY                       VARCHAR2(20),
         STATE                      VARCHAR2(2),
         POSTAL_CODE                VARCHAR2(9),
CONSTRAINT CUSTOMER_PK PRIMARY KEY (CUSTOMER_ID));

CREATE TABLE ORDER_T
        (ORDER_ID                   NUMBER(11, 0) NOT NULL,
         ORDER_DATE                 DATE DEFAULT SYSDATE,
         CUSTOMER_ID                NUMBER(11, 0),
CONSTRAINT ORDER_PK PRIMARY KEY (ORDER_ID),
CONSTRAINT ORDER_FK FOREIGN KEY (CUSTOMER_ID) REFERENCES CUSTOMER_T(CUSTOMER_ID));

CREATE TABLE PRODUCT_T
        (PRODUCT_ID                 INTEGER       NOT NULL,
         PRODUCT_DESCRIPTION        VARCHAR2(50),
         PRODUCT_FINISH             VARCHAR2(20)
                        CHECK (PRODUCT_FINISH IN ('Cherry', 'Natural Ash', 'White Ash',
                                'Red Oak', 'Natural Oak', 'Walnut')),
         STANDARD_PRICE             DECIMAL(6,2),
         PRODUCT_LINE_ID            INTEGER,
CONSTRAINT PRODUCT_PK PRIMARY KEY (PRODUCT_ID));

CREATE TABLE ORDER_LINE_T
        (ORDER_ID                   NUMBER(11,0)  NOT NULL,
         PRODUCT_ID                 NUMBER(11,0)  NOT NULL,
         ORDERED_QUANTITY           NUMBER(11,0),
CONSTRAINT ORDER_LINE_PK PRIMARY KEY (ORDER_ID, PRODUCT_ID),
CONSTRAINT ORDER_LINE_FK1 FOREIGN KEY(ORDER_ID) REFERENCES ORDER_T(ORDER_ID),
CONSTRAINT ORDER_LINE_FK2 FOREIGN KEY (PRODUCT_ID) REFERENCES PRODUCT_T(PRODUCT_ID));
```

Identifying Foreign keys & establishing Relationships

SQL Database Definition Commands (cont)

```
CREATE TABLE CUSTOMER_T
        (CUSTOMER_ID            NUMBER(11, 0) NOT NULL,
         CUSTOMER_NAME          VARCHAR2(25)  NOT NULL,
         CUSTOMER_ADDRESS       VARCHAR2(30),
         CITY                   VARCHAR2(20),
         STATE                  VARCHAR2(2),
         POSTAL_CODE            VARCHAR2(9),
CONSTRAINT CUSTOMER_PK PRIMARY KEY (CUSTOMER_ID));

CREATE TABLE ORDER_T
        (ORDER_ID               NUMBER(11, 0) NOT NULL
         ORDER_DATE             DATE DEFAULT SYSDATE,
         CUSTOMER_ID            NUMBER(11, 0),
CONSTRAINT ORDER_PK PRIMARY KEY (ORDER_ID),
CONSTRAINT ORDER_FK FOREIGN KEY (CUSTOMER_ID) REFERENCES CUSTOMER_T(CUSTOMER_ID));

CREATE TABLE PRODUCT_T
        (PRODUCT_ID             INTEGER     NOT NULL,
         PRODUCT_DESCRIPTION    VARCHAR2(50),
         PRODUCT_FINISH         VARCHAR2(20)
                CHECK (PRODUCT_FINISH IN ('Cherry', 'Natural Ash', 'White Ash',
                        'Red Oak', 'Natural Oak', 'Walnut')),
         STANDARD_PRICE         DECIMAL(6,2),
         PRODUCT_LINE_ID        INTEGER,
CONSTRAINT PRODUCT_PK PRIMARY KEY (PRODUCT_ID));

CREATE TABLE ORDER_LINE_T
        (ORDER_ID               NUMBER(11,0)  NOT NULL,
         PRODUCT_ID             NUMBER(11,0)  NOT NULL,
         ORDERED_QUANTITY       NUMBER(11,0),
CONSTRAINT ORDER_LINE_PK PRIMARY KEY (ORDER_ID, PRODUCT_ID),
CONSTRAINT ORDER_LINE_FK1 FOREIGN KEY(ORDER_ID) REFERENCES ORDER_T(ORDER_ID),
CONSTRAINT ORDER_LINE_FK2 FOREIGN KEY (PRODUCT_ID) REFERENCES PRODUCT_T(PRODUCT_ID));
```

Default Values and Domain Constraints

SQL Database Definition Commands (cont)

```
CREATE TABLE CUSTOMER_T
        (CUSTOMER_ID                NUMBER(11, 0) NOT NULL,
         CUSTOMER_NAME              VARCHAR2(25)  NOT NULL,
         CUSTOMER_ADDRESS           VARCHAR2(30),
         CITY                       VARCHAR2(20),
         STATE                      VARCHAR2(2),
         POSTAL_CODE                VARCHAR2(9),
CONSTRAINT CUSTOMER_PK PRIMARY KEY (CUSTOMER_ID));
```

Overall Table Definitions

```
CREATE TABLE ORDER_T
        (ORDER_ID                   NUMBER(11, 0) NOT NULL,
         ORDER_DATE                 DATE DEFAULT SYSDATE,
         CUSTOMER_ID                NUMBER(11, 0),
CONSTRAINT ORDER_PK PRIMARY KEY (ORDER_ID),
CONSTRAINT ORDER_FK FOREIGN KEY (CUSTOMER_ID) REFERENCES CUSTOMER_T(CUSTOMER_ID));
```

```
CREATE TABLE PRODUCT_T
        (PRODUCT_ID                 INTEGER        NOT NULL,
         PRODUCT_DESCRIPTION        VARCHAR2(50),
         PRODUCT_FINISH             VARCHAR2(20)
                        CHECK (PRODUCT_FINISH IN ('Cherry', 'Natural Ash', 'White Ash',
                                'Red Oak', 'Natural Oak', 'Walnut')),
         STANDARD_PRICE             DECIMAL(6,2),
         PRODUCT_LINE_ID            INTEGER,
CONSTRAINT PRODUCT_PK PRIMARY KEY (PRODUCT_ID));
```

```
CREATE TABLE ORDER_LINE_T
        (ORDER_ID                   NUMBER(11,0)  NOT NULL,
         PRODUCT_ID                 NUMBER(11,0)  NOT NULL,
         ORDERED_QUANTITY           NUMBER(11,0),
CONSTRAINT ORDER_LINE_PK PRIMARY KEY (ORDER_ID, PRODUCT_ID),
CONSTRAINT ORDER_LINE_FK1 FOREIGN KEY(ORDER_ID) REFERENCES ORDER_T(ORDER_ID),
CONSTRAINT ORDER_LINE_FK2 FOREIGN KEY (PRODUCT_ID) REFERENCES PRODUCT_T(PRODUCT_ID));
```

Using and Defining Views

- Views provide users controlled access to tables
- Advantages of views:
  - Simplify query commands
  - Provide data security
  - Enhance programming productivity
- CREATE VIEW command

View Terminology

- Base Table
  - A 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
  - 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 the corresponding base tables

Sample CREATE VIEW

CREATE VIEW EXPENSIVE_STUFF_V **AS**
SELECT PRODUCT_ID, PRODUCT_NAME, UNIT_PRICE
FROM PRODUCT_T
WHERE UNIT_PRICE > 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

About Dynamic Views

TABLE 6-4 Pros and Cons of Using Dynamic Views

Positive Aspects	Negative Aspects
Simplify query commands	Use processing time re-creating the view each time it is referenced
Help provide data security and confidentiality	May or may not be directly updateable
Improve programmer productivity	
Contain most current base table data	
Use little storage space	
Provide a customized view for a user	
Establish physical data independence	

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

Ensuring Data Integrity through Updates
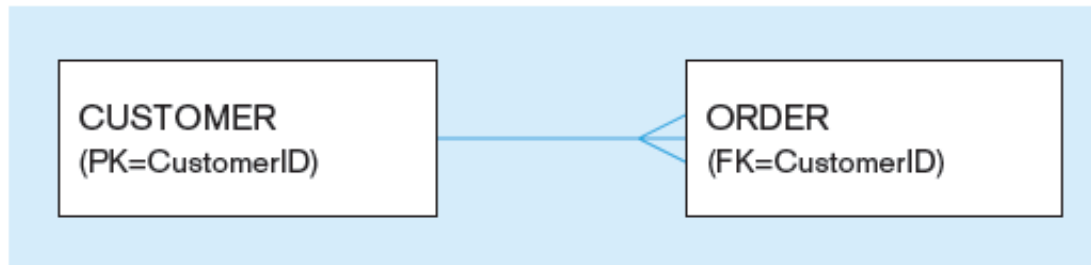
Relational integrity is enforced via the primary-key to foreign-key match

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 and Removing Tables

- ALTER TABLE statement allows you to change column specifications:
  - **ALTER TABLE** CUSTOMER_T **ADD** (TYPE VARCHAR(2))
- DROP TABLE statement allows you to remove tables from your schema:
  - **DROP TABLE** CUSTOMER_T

Alter Table

Syntax:

ALTER TABLE table_name alter_table_action;

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

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

Alter Table (Add Column)

```
mysql> describe employee;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| emp_id   | int(11)     | NO   | PRI | NULL    |       |
| emp_name | varchar(30) | NO   |     | NULL    |       |
| emp_job  | varchar(30) | NO   |     | NULL    |       |
| dep_ID   | int(5)      | YES  | MUL | NULL    |       |
+----------+-------------+------+-----+---------+-------+
4 rows in set (0.03 sec)

mysql> alter table employee add column (city varchar(30) not null default 'Peshawar');
Query OK, 0 rows affected (0.68 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> describe employee;
+----------+-------------+------+-----+----------+-------+
| Field    | Type        | Null | Key | Default  | Extra |
+----------+-------------+------+-----+----------+-------+
| emp_id   | int(11)     | NO   | PRI | NULL     |       |
| emp_name | varchar(30) | NO   |     | NULL     |       |
| emp_job  | varchar(30) | NO   |     | NULL     |       |
| dep_ID   | int(5)      | YES  | MUL | NULL     |       |
| city     | varchar(30) | NO   |     | Peshawar |       |
+----------+-------------+------+-----+----------+-------+
5 rows in set (0.08 sec)
```

Alter Table (Drop Column)

```
mysql> alter table employee drop column city;
Query OK, 0 rows affected (1.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> describe employee;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| emp_id   | int(11)     | NO   | PRI | NULL    |       |
| emp_name | varchar(30) | NO   |     | NULL    |       |
| emp_job  | varchar(30) | NO   |     | NULL    |       |
| dep_ID   | int(5)      | YES  | MUL | NULL    |       |
+----------+-------------+------+-----+---------+-------+
4 rows in set (0.01 sec)
```

Alter Table (Add/Drop Primary Key)

```
mysql> alter table employee drop primary key;
Query OK, 13 rows affected (1.31 sec)
Records: 13  Duplicates: 0  Warnings: 0

mysql> describe employee;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| emp_id   | int(11)     | NO   |     | NULL    |       |
| emp_name | varchar(30) | NO   |     | NULL    |       |
| emp_job  | varchar(30) | NO   |     | NULL    |       |
| dep_ID   | int(5)      | YES  | MUL | NULL    |       |
+----------+-------------+------+-----+---------+-------+
4 rows in set (0.14 sec)

mysql> alter table employee add primary key (emp_id);
Query OK, 0 rows affected (0.53 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> describe employee;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| emp_id   | int(11)     | NO   | PRI | NULL    |       |
| emp_name | varchar(30) | NO   |     | NULL    |       |
| emp_job  | varchar(30) | NO   |     | NULL    |       |
| dep_ID   | int(5)      | YES  | MUL | NULL    |       |
+----------+-------------+------+-----+---------+-------+
4 rows in set (0.01 sec)
```

Insert Statement

- Adds data to a table
- Inserting into a table
  - **INSERT INTO** CUSTOMER_T **VALUES** (001, 'CONTEMPORARY Casuals', '1335 S. Himes Blvd.', 'Gainesville', 'FL', 32601);
- Inserting a record that has some null attributes requires identifying the fields that actually get data
  - **INSERT INTO** PRODUNT_T (PRODUCT_ID, PRODUCT_DESCRIPTION, PRODUCT_FINISH, STANDARD_PRICE, PRODUCT_ON_HAND) **VALUES** (1, 'End Table', 'Cherry', 175, 8);
- Inserting from another table
  - **INSERT INTO** CA_CUSTOMER_T **SELECT * FROM** CUSTOMER_T **WHERE** STATE = 'CA';

Insert Statement (Cont.)

```
mysql> insert into department (dep_Name, dep_ID) values
    -> ('Industrial Engineering', 20);
Query OK, 1 row affected (0.15 sec)

mysql> select * from department;
+--------+---------------------------------+
| dep_ID | dep_Name                        |
+--------+---------------------------------+
|     10 | Computer Systems Engineering    |
|     20 | Industrial Engineering          |
|     40 | Chemical Engineering            |
|     50 | Mechanical Engineering          |
|     70 | Software Engineering            |
|     75 | Mechatronics Engineering        |
|     76 | Software Engineering            |
|     78 | Computer Science & Information   |
|    100 | Agriculture Engineering         |
|    105 | Electrical Engineering          |
+--------+---------------------------------+
10 rows in set (0.00 sec)
```

Insert Statement (Cont.)

```
mysql> select * from employee_mrdn;
Empty set (0.00 sec)

mysql> insert into employee_mrdn
    -> select * from employee
    -> where emp_domicile = 'mardan';
Query OK, 4 rows affected (0.14 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> select * from employee_mrdn;
+--------+----------+---------------------+--------+--------------+
| emp_id | emp_name | emp_job             | dep_ID | emp_domicile |
+--------+----------+---------------------+--------+--------------+
|      4 | CVA      | Lecturer            |     50 | Mardan       |
|      6 | ACB      | Assistant Professor |    105 | Mardan       |
|      8 | XZY      | Assistant Professor |     70 | Mardan       |
|      9 | ABS      | Assistant Professor |     70 | Mardan       |
+--------+----------+---------------------+--------+--------------+
4 rows in set (0.00 sec)
```

Delete Statement

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

Update Statement

- Modifies data in existing rows
    - **UPDATE** PRODUCT_T **SET** UNIT_PRICE = 775 **WHERE** PRODUCT_ID = 7;

The SELECT Statement

- Used for queries on single or multiple tables
- Clauses of the SELECT statement:
  - SELECT
    - List the column (and expressions) that should 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

SQL Statement Processing Order (Adapted from Van der Lans, p.100)

FROM
Identifies involved tables

WHERE
Finds all rows meeting stated condition(s)

GROUP BY
Organizes rows according to values in stated column(s)

HAVING
Finds all groups meeting stated condition(s)

SELECT
Identifies columns

ORDER BY
Sorts rows
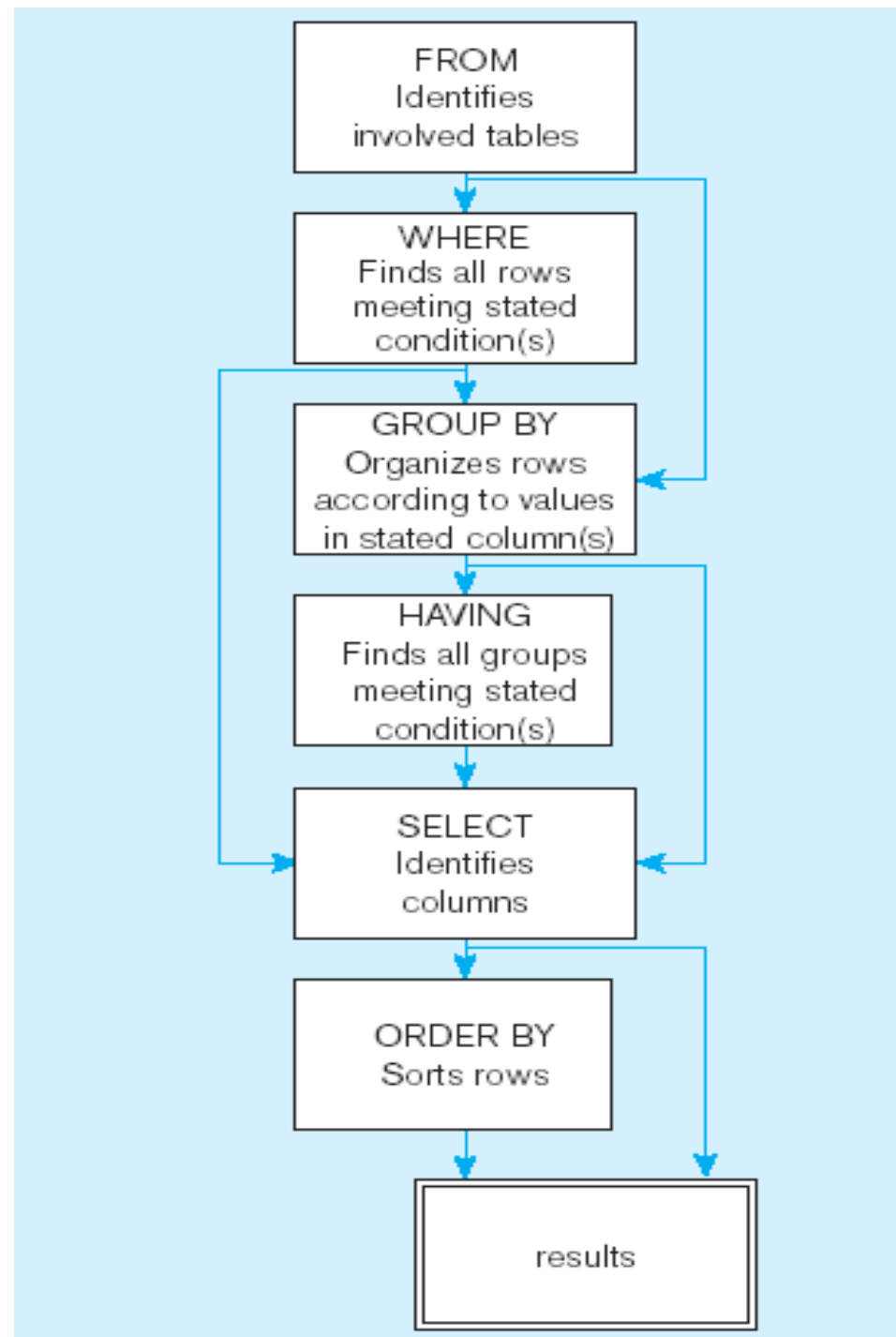
results

SELECT Example with Comparison Operator (<)

- Find products with standard price less than $275

  – SELECT PRODUCT_NAME, STANDARD_PRICE
    FROM PRODUCT_V
    WHERE STANDARD_PRICE <275;

Comparison Operators in SQL

Result:

PRODUCTDESCRIPTION	PRODUCTSTANDARDPRICE
End Table	175
Computer Desk	250
Coffee Table	200

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 with Comparison Operator (>)

Query: Which orders have been placed since 10/24/2010?

SELECT OrderID, OrderDate
  FROM Order_T
    WHERE OrderDate > '24-OCT-2010';

Result:

ORDERID	ORDERDATE
1007	27-OCT-10
1008	30-OCT-10
1009	05-NOV-10
1010	05-NOV-10

SELECT Example with Comparison Operator (!=)

Query: What furniture does Pine Valley carry that isn't made of cherry?

SELECT ProductDescription, ProductFinish
  FROM Product_T
    WHERE ProductFinish != 'Cherry';

Result:

PRODUCTDESCRIPTION	PRODUCTFINISH
Coffee Table	Natural Ash
Computer Desk	Natural Ash
Entertainment Center	Natural Maple
8-Drawer Desk	White Ash
Dining Table	Natural Ash
Computer Desk	Walnut

SELECT Example with ALIAS

- Alias is an alternative column or table name

    – **SELECT** CUSTOMER_Name **AS NAME**, CUSTOMER_ADDRESS
    FROM CUSTOMER_V
    WHERE NAME = 'Home Furnishings';

Result:

NAME	CUSTOMERADDRESS
Home Furnishings	1900 Allard Ave.

SELECT Example Using a Function

- Using the COUNT *aggregate function* to find totals

  – **SELECT COUNT(*)**
    FROM ORDER_LINE_V
    WHERE ORDER_ID = 1004;

  Result:

  COUNT (*)
  ─────────
      2

  Note: With aggregate functions you can't have single-valued columns included in the SELECT clause

SELECT Example Using Boolean Operators

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

  - **SELECT** PRODUCT_DESCRIPTION, PRODUCT_FINISH, STANDARD_PRICE

    FROM PRODUCT_V

    WHERE (PRODUCT_DESCRIPTION **LIKE** '**%**Desk' **OR** PRODUCT_DESCRIPTION **LIKE** '**%**Table') **AND** UNIT_PRICE>300;

Output of this query is shown in next slide (i.e. S 41)

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

Output

Query:

SELECT PRODUCT_DESCRIPTION, PRODUCT_FINISH, STANDARD_PRICE

FROM PRODUCT_V

WHERE (PRODUCT_DESCRIPTION **LIKE** '%Desk' **OR** PRODUCT_DESCRIPTION **LIKE** '%Table') **AND** UNIT_PRICE>300;

Result:

PRODUCTDESCRIPTION	PRODUCTFINISH	PRODUCTSTANDARDPRICE
Computer Desk	Natural Ash	375
Writer's Desk	Cherry	325
8-Drawer Desk	White Ash	750
Dining Table	Natural Ash	800
Computer Desk	Walnut	250

SELECT Example – Sorting Results with the ORDER BY Clause

- Sort the results first by STATE, and within a state by CUSTOMER_NAME

    - **SELECT** CUSTOMER_NAME, CITY, STATE
      FROM CUSTOMER_V
      WHERE STATE **IN** ('FL', 'TX', 'CA', 'HI')
      ORDER BY STATE, CUSTOMER_NAME;

Output of this query is shown in next slide (i.e. S 43)

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

Output

SELECT CUSTOMER_NAME, CITY, STATE
FROM CUSTOMER_V
WHERE STATE **IN** ('FL', 'TX', 'CA', 'HI')
ORDER BY STATE, CUSTOMER_NAME;

Result:

CUSTOMERNAME	CUSTOMERCITY	CUSTOMERSTATE
California Classics	Santa Clara	CA
Impressions	Sacramento	CA
Contemporary Casuals	Gainesville	FL
M and H Casual Furniture	Clearwater	FL
Seminole Interiors	Seminole	FL
Kaneohe Homes	Kaneohe	HI
Value Furniture	Plano	TX

7 rows selected.

SELECT Example – Categorizing Results Using the 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 STATE, COUNT(STATE)
FROM CUSTOMER_V
GROUP BY STATE;

Output of this query is shown in next slide (i.e. S 45)

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

Output

SELECT STATE, COUNT(STATE)
FROM CUSTOMER_V
GROUP BY STATE;

Result:

CUSTOMERSTATE	COUNT(CUSTOMERSTATE)
CA	2
CO	1
FL	3
HI	1
MI	1
NJ	2
NY	1
PA	1
TX	1
UT	1
WA	1

11 rows selected.

SELECT Example – Qualifying Results by Categories Using the HAVING Clause

- For use with GROUP BY

SELECT STATE, COUNT(STATE)
FROM CUSTOMER_V
GROUP BY STATE
HAVING COUNT(STATE) > 1;

Result:

CUSTOMERSTATE	COUNT(CUSTOMERSTATE)
CA	2
FL	3
NJ	2

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.

Creating Index

```
mysql> describe employee;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| emp_id    | int(11)     | NO   | PRI | NULL    |       |
| emp_name  | varchar(30) | NO   |     | NULL    |       |
| emp_job   | varchar(30) | NO   |     | NULL    |       |
| dep_ID    | int(5)      | YES  | MUL | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
4 rows in set (0.01 sec)

mysql> create index e_job on employee (emp_job);
Query OK, 0 rows affected (1.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> describe employee;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| emp_id    | int(11)     | NO   | PRI | NULL    |       |
| emp_name  | varchar(30) | NO   |     | NULL    |       |
| emp_job   | varchar(30) | NO   | MUL | NULL    |       |
| dep_ID    | int(5)      | YES  | MUL | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
4 rows in set (0.01 sec)
```

Dropping Index

```
mysql> drop index e_job on employee;
Query OK, 0 rows affected (0.25 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> describe employee;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| emp_id   | int(11)     | NO   | PRI | NULL    |       |
| emp_name | varchar(30) | NO   |     | NULL    |       |
| emp_job  | varchar(30) | NO   |     | NULL    |       |
| dep_ID   | int(5)      | YES  | MUL | NULL    |       |
+----------+-------------+------+-----+---------+-------+
4 rows in set (0.01 sec)
```

Use of BETWEEN Operator

Query: Which products in the Product table have a standard price between $200 and $300?

```
SELECT ProductDescription, ProductStandardPrice
  FROM Product_T
    WHERE ProductStandardPrice > 199 AND ProductStandardPrice < 301;
```

Result:

PRODUCTDESCRIPTION	PRODUCTSTANDARDPRICE
Coffee Table	200
Computer Desk	250

The same result will be returned by the following query.

Query: Which products in the PRODUCT table have a standard price between $200 and $300?

```
SELECT ProductDescription, ProductStandardPrice
  FROM Product_T
    WHERE ProductStandardPrice BETWEEN 200 AND 300;
```

Result: Same as previous query.

Use of IS NULL Operator

```
mysql> select * from employee;
+--------+----------+---------------------+--------+
| emp_id | emp_name | emp_job             | dep_ID |
+--------+----------+---------------------+--------+
|      1 | BCA      | Lecturer            |     10 |
|      2 | ABD      | Lecturer            |     35 |
|      3 | XYZ      | Assitant Professor  |     10 |
|      4 | CVA      | Lecturer            |     50 |
|      5 | VAC      | Assistant Professor |     50 |
|      6 | ACB      | Assistant Professor |     35 |
|      8 | XZY      | Assistant Professor |     70 |
|      9 | ABS      | Assistant Professor |     70 |
|     11 | FGJ      | Assistant Professor |     25 |
|     16 | OOP      | Lecturer            |   NULL |
|     17 | HHA      | Lab Engineer        |     35 |
|     18 | HBV      | Lab Engineer        |     70 |
|     19 | HCA      | Lab Engineer        |     35 |
+--------+----------+---------------------+--------+
13 rows in set (0.00 sec)

mysql> select * from employee where dep_ID is null;
+--------+----------+----------+--------+
| emp_id | emp_name | emp_job  | dep_ID |
+--------+----------+----------+--------+
|     16 | OOP      | Lecturer |   NULL |
+--------+----------+----------+--------+
1 row in set (0.04 sec)
```

More Aggregate Functions

Query: What is the average standard price for all products in inventory?

SELECT AVG (ProductStandardPrice) AS AveragePrice
FROM Product_T;

Result:

AVERAGEPRICE

440.625

Query: Display for each product the difference between its standard price and the overall average standard price of all products.

SELECT ProductStandardPrice – PriceAvg AS Difference
  FROM Product_T, (SELECT AVG(ProductStandardPrice) AS PriceAvg
    FROM Product_T);

Result:

DIFFERENCE

–240.63
–65.63
–265.63
–190.63
359.38
–115.63
209.38
309.38

Summary

- Introduced SQL, Data Definition & Manipulation Language, various commands along with examples