Objectives

• Definition of terms.
• Discuss advantages of standardized SQL.
• Define a database using SQL data definition language.
• Write single table queries using SQL.
• Establish referential integrity using SQL.
• Work with Views.
The Physical Design Stage of SDLC (Figures 2-4, 2-5 revisited)

Purpose – programming, testing, training, installation, documenting
Deliverable – operational programs, documentation, training materials, program/data structures

Database activity – physical database design and database implementation
SQL Overview

• SQL ≡ Structured Query Language.
• The standard for relational database management systems (RDBMS).
• SQL-99 and SQL: 2003 Standards – 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 a Standardized Relational Language

- Reduced training costs
- Productivity
- Application portability
- Application longevity
- Reduced dependence on a single vendor
- Cross-system communication
The 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 7-1 (page 294):
A simplified schematic of a typical SQL environment, as described by the SQL:2003 standard
Some SQL Data Types (from Oracle 9i)

- **String types**
  - CHAR(n) – fixed-length character data, n characters long
    Maximum length = 2000 bytes
  - VARCHAR2(n) – variable length character data, maximum 4000 bytes
  - LONG – variable-length character data, up to 4GB. Maximum 1 per table

- **Numeric types**
  - NUMBER(p,q) – general purpose numeric data type
  - INTEGER(p) – signed integer, p digits wide
  - FLOAT(p) – floating point in scientific notation with p binary digits precision

- **Date/time type**
  - DATE – fixed-length date/time in dd-mm-yy form
Figure 7-4 (PAGE 297):
DDL, DML, DCL, and the database development process

DDL
Define the database:
  CREATE tables, indexes, views
  Establish foreign keys
  Drop or truncate tables

Physical Design

DML
Load the database:
  INSERT data
  UPDATE the database
  Manipulate the database:
    SELECT

Implementation

DCL
Control the database:
  GRANT, ADD, REVOKE

Maintenance
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 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

Figure 7-5: General syntax for CREATE TABLE

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]

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
The following few slides create tables for this enterprise data model

**Figure 2-1** Segment from enterprise data model (Pine Valley Furniture Company)
Figure 7-6: SQL database definition commands for Pine Valley Furniture

```
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(8,2),
   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
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))
```
```sql
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));
```

**Non-null specification**

**Identifying primary key**

Primary keys can never have NULL values
Non-null specifications

```
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));
```

Some primary keys are composite – composed of multiple attributes
Controlling the values in attributes

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),
     CONSTRAINT PRODUCT_PK PRIMARY KEY (PRODUCT_ID),
     CHECK (PRODUCT_FINISH IN ('Cherry', 'Natural Ash', 'White Ash',
                                'Red Oak', 'Natural Oak', 'Walnut')))
Identifying foreign keys and establishing relationships

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));

Primary key of parent table
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.
Relational integrity is enforced via the primary-key to foreign-key match.
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
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(CUSTOMER_NAME)
    • This makes an index for the CUSTOMER_NAME field of the CUSTOMER_T table
Insert Statement

- Adds data to a table
- Inserting into a table
- Inserting a record that has some null attributes requires identifying the fields that actually get data
  - `INSERT INTO PRODUCT_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’;`
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;
SELECT Statement

- Used for queries on single or multiple tables.
- Clauses of the SELECT statement:
  - SELECT
    - List the columns (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
Figure 7-8:

SQL statement processing order

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

- Find products with standard price less than $275

```sql
SELECT PRODUCT_NAME, STANDARD_PRICE
FROM PRODUCT_V
WHERE STANDARD_PRICE < 275;
```

Table 7-3
Comparison Operators in SQL

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Not equal to</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal to</td>
</tr>
</tbody>
</table>
SELECT Example using Alias

• Alias is an alternative column or table name.

```
SELECT CUST.CUSTOMER AS NAME,
     CUST.CUSTOMER_ADDRESS
FROM CUSTOMER_V CUST
WHERE NAME = 'Home Furnishings';
```
SELECT Example Using a Function

- Using the COUNT aggregate function to find totals

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

Note: with aggregate functions you can’t have single-valued columns included in the SELECT clause
SELECT Example – 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;
```

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 – 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;
```

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
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;
```

Note: you can use single-value fields with aggregate functions if they are included in the GROUP BY clause.
SELECT Example –
Qualifying Results by Category Using the HAVING Clause

• For use with GROUP BY

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