Chapter 7:
Introduction to SQL

Modern Database Management
9th Edition

Jeffrey A. Hoffer, Mary B. Prescott, Heikki Topi

Objectives

- Definition of 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: 200n 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. 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
- 1986–ANSI SQL standard released
- Current–SQL is supported by most major database vendors
Purpose of SQL Standard

- 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
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
A simplified schematic of a typical SQL environment, as described by the SQL: 200n standard
### Some SQL Data types

**Table 7-2**

<table>
<thead>
<tr>
<th>Type</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>CHARACTER (CHAR)</td>
<td>Stores string values containing any characters in a character set. CHAR is defined to be a fixed length.</td>
</tr>
<tr>
<td></td>
<td>CHARACTER VARYING (VARCHAR)</td>
<td>Stores string values containing any characters in a character set, but of definable variable length.</td>
</tr>
<tr>
<td></td>
<td>BINARY LARGE OBJECT (BLOB)</td>
<td>Stores binary string values in hexadecimal format. BLOB is defined to be a variable length.</td>
</tr>
<tr>
<td>Number</td>
<td>NUMERIC</td>
<td>Stores exact numbers with a defined precision and scale.</td>
</tr>
<tr>
<td></td>
<td>INTEGER (INT)</td>
<td>Stores exact numbers with a predefined precision and scale of zero.</td>
</tr>
<tr>
<td>Temporal</td>
<td>TIMESTAMP</td>
<td>Stores a moment an event occurs, using a definable fraction of a second precision.</td>
</tr>
<tr>
<td>Boolean</td>
<td>BOOLEAN</td>
<td>Stores truth values, TRUE, FALSE, or UNKNOWN.</td>
</tr>
</tbody>
</table>
Figure 7-4
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 slides create tables for this enterprise data model

**CUSTOMER**
- Customer_ID
- Customer_Name

**PRODUCT**
- Product_ID
- Standard_Price

**ORDER**
- Order_ID
- Customer_ID
- Order_Date

**ORDER LINE**
- Quantity

The relationships are:
- **CUSTOMER** places **PRODUCT**
- **CUSTOMER** orders **ORDER**
- **ORDER** contains **ORDER LINE**
- **PRODUCT** is for **ORDER LINE**
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(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
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 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-nullable specification

Primary keys can never have NULL values

Identifying primary key
Non-nullable 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));

Primary key

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),
   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),
   CONSTRAINT PRODUCT_FK OUTER REFERENCES PRODUCT_T(PRODUCT_ID));
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
**Figure 7-7 Ensuring data integrity through updates**

![Diagram showing relationships between CUSTOMER and ORDER tables with primary-key to foreign-key match.](image)

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

```
CREATE TABLE CUSTOMER_T
    (CUSTOMER_ID INTEGER DEFAULT 'C999' NOT NULL,
    CUSTOMER_NAME VARCHAR(40) NOT NULL,
    ...
    CONSTRAINT CUSTOMER_PK PRIMARY KEY (CUSTOMER_ID),
    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`
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
  - `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 (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';`
Creating Tables with Identity Columns

New with SQL:2003

```
CREATE TABLE CUSTOMER_T
(CUSTOMER_ID INTEGER GENERATED ALWAYS AS IDENTITY
 (START WITH 1
 INCREMENT BY 1
 MINVALUE 1
 MAXVALUE 10000
 NO CYCLE),
CUSTOMER_NAME VARCHAR (25) NOT NULL,
CUSTOMER_ADDRESS VARCHAR (30),
CITY VARCHAR (20),
STATE VARCHAR (2),
POSTAL_CODE VARCHAR (9),
CONSTRAINT CUSTOMER_PK PRIMARY KEY (CUSTOMER_ID);
```

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 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;
Merge Statement

MERGE INTO PRODUCT_T AS PROD
USING
(SELECT PRODUCT_ID, PRODUCT_DESCRIPTION, PRODUCT_FINISH,
STANDARD_PRICE, PRODUCT_LINE_ID FROM PURCHASES T) AS PURCH
ON (PROD.PRODUCT_ID = PURCH.PRODUCT_ID)
WHEN MATCHED THEN UPDATE
    PROD.STANDARD_PRICE = PURCH.STANDARD_PRICE
WHEN NOT MATCHED THEN INSERT
    (PRODUCT_ID, PRODUCT_DESCRIPTION, PRODUCT_FINISH,
STANDARD_PRICE, PRODUCT_LINE_ID)
VALUES(PURCH.PRODUCT_ID, PURCH.PRODUCT_DESCRIPTION,
PURCH.PRODUCT_FINISH, PURCH.STANDARD_PRICE,
PURCH.PRODUCT_LINE_ID);

Makes it easier to update a table...allows combination of Insert and Update in one statement

Useful for updating master tables with new data

Chapter 7 © 2009 Pearson Education, Inc. Publishing as Prentice Hall
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 7-10
SQL statement processing order (adapted from van der Lans, p.100)
SELECT Example

- Find products with standard price less than $275

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

Table 7-3: Comparison Operators in SQL

<table>
<thead>
<tr>
<th>OPERATOR MEANING</th>
<th>Equivalent SQL</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>

Chapter 7 © 2009 Pearson Education, Inc. Publishing as Prentice Hall 31
SELECT Example Using Alias

- Alias is an alternative column or table name

```sql
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

```sql
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

```sql
SELECT PRODUCT_DESCRIPTION, PRODUCT_FINISH, STANDARD_PRICE
FROM PRODUCT_V
WHERE (PRODUCT_DESCRIPTION LIKE '%%Desk'
    OR PRODUCT_DESCRIPTION LIKE '%%Table')
    AND STANDARD_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.
Venn Diagram from Previous Query

- DESK
- TABLE
- STANDARD_PRICE > $300

AND
RESULT

OR
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 CUSTOMER_STATE, COUNT(CUSTOMER_STATE)
FROM CUSTOMER_V
GROUP BY CUSTOMER_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 Categories
Using the HAVING Clause

For use with GROUP BY

SELECT CUSTOMER_STATE, COUNT(CUSTOMER_STATE) FROM CUSTOMER_V
GROUP BY CUSTOMER_STATE
HAVING COUNT(CUSTOMER_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

Chapter 7 © 2009 Pearson Education, Inc. Publishing as Prentice Hall
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 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
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