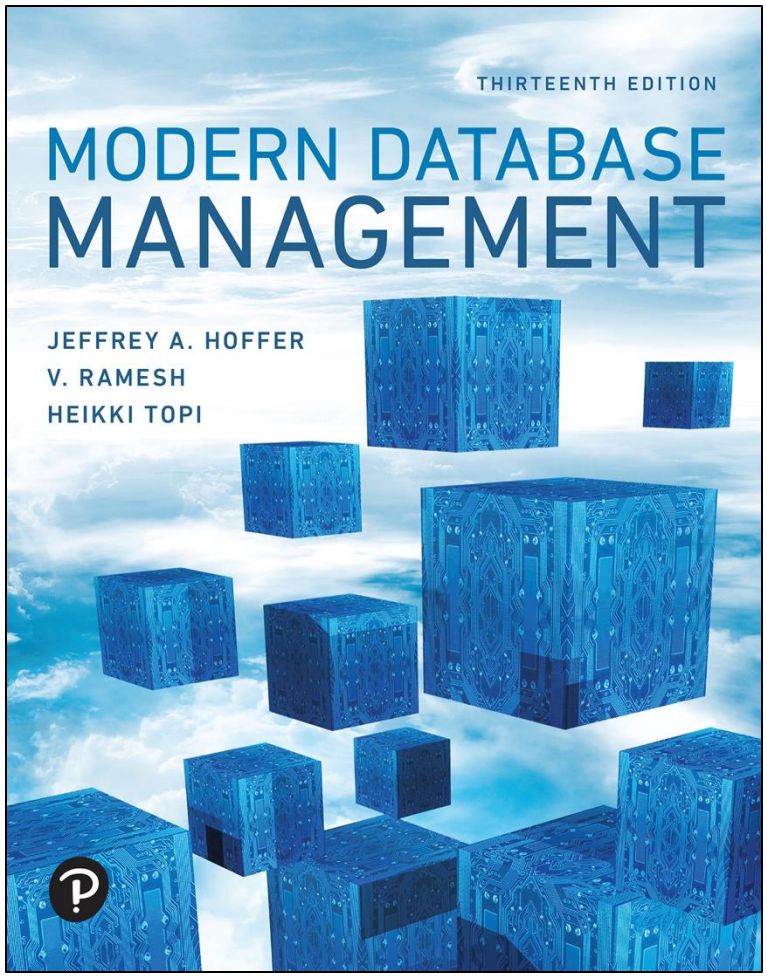
Modern Database Management

Thirteenth Edition



Chapter 5

Introduction to SQL

Learning Objectives

5.1 Define terms

5.2 Interpret the history and role of SQL in database development

5.3 Define a database using the SQL data definition language

5.4 Write single-table queries using SQL commands

5.5 Establish referential integrity using SQL

5.6 Discuss the SQL:1999 and SQL:2016 standards

SQL Overview

- Structured Query Language – often pronounced “Sequel”
- 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
- **1974-79** – 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
  - Major ANSI standard updates in 1989, 1992, 1999, 2003, 2006, 2008, 2011, 2016
- **Today** – SQL is supported by most major database vendors

Is SQL a standard? No longer certified by NIST.

Original 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)

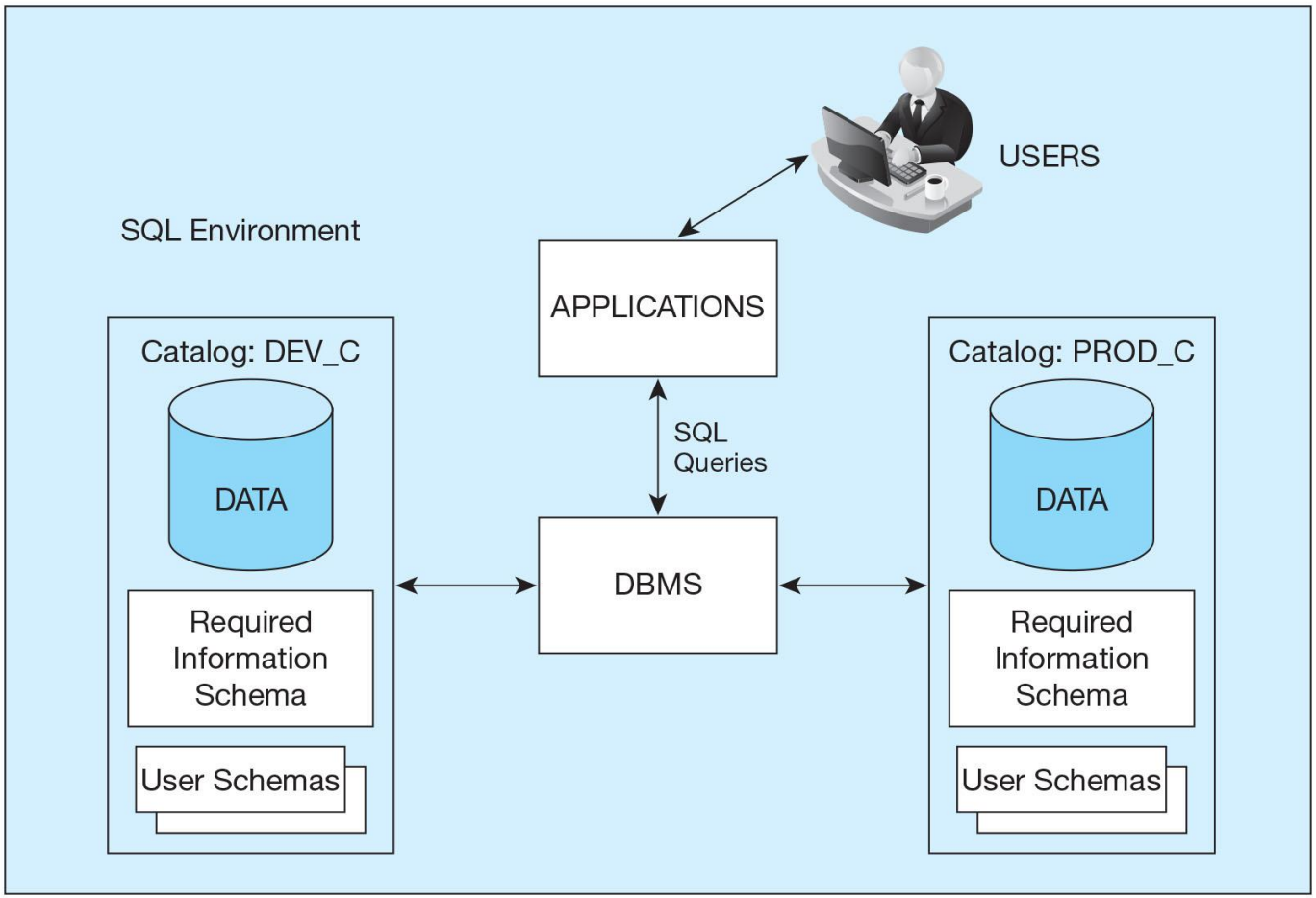
Benefits of a Standardized Relational Language

- 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

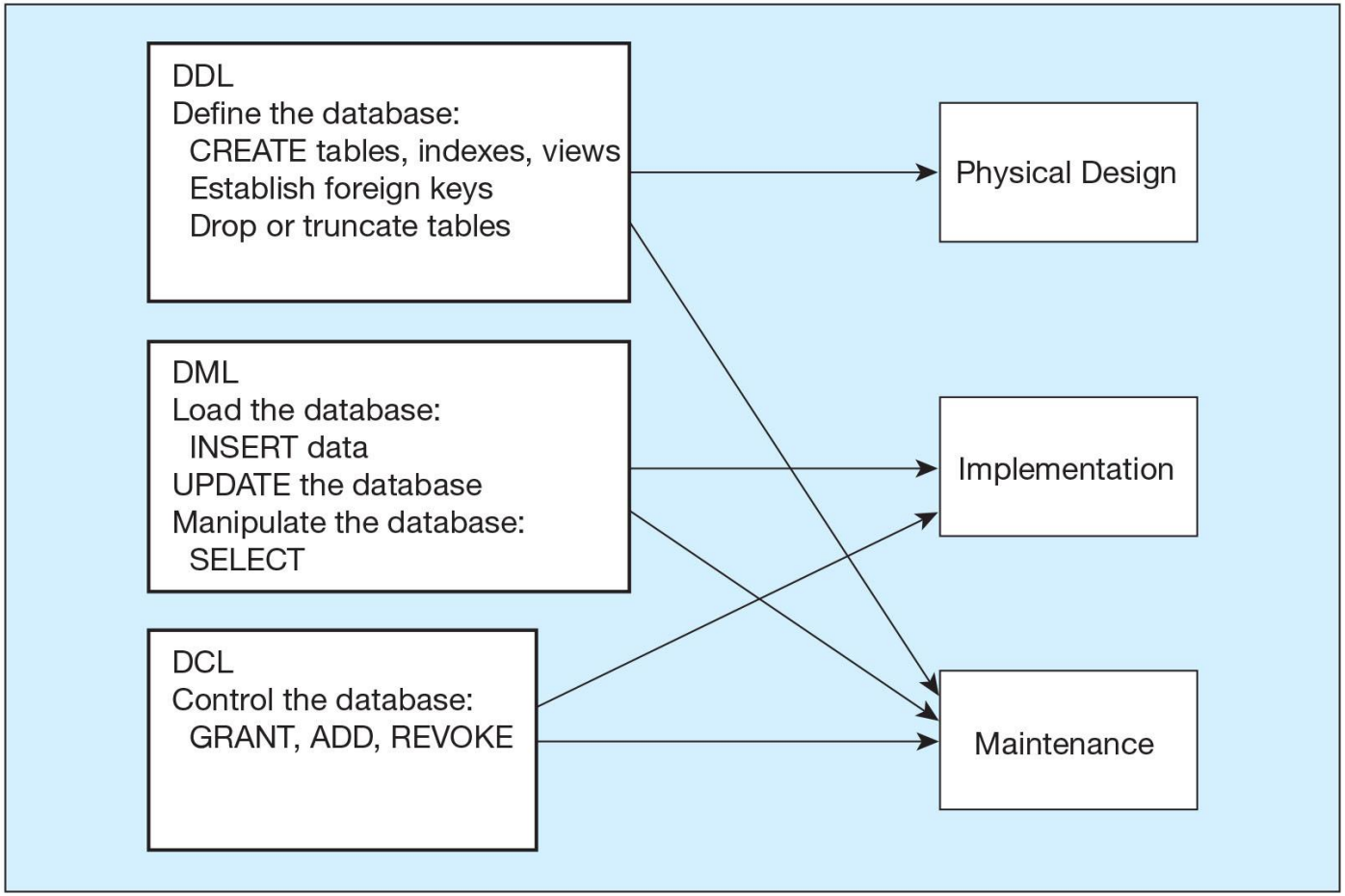
Figure 5-1 A Simplified Schematic of a Typical SQL Environment, as Described by the SQL:2016 Standards



SQL Data Types

- Strings
  - CHARACTER (n), VARYING CHARACTER (n)
- Binary
  - Binary Large Object (BLOB)
- Number
  - Numeric (precision, scale), Decimal (p, s), Integer
- Temporal
  - Timestamp, Timestamp with local time zone
- Boolean
  - True or False values

Figure 5-4 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

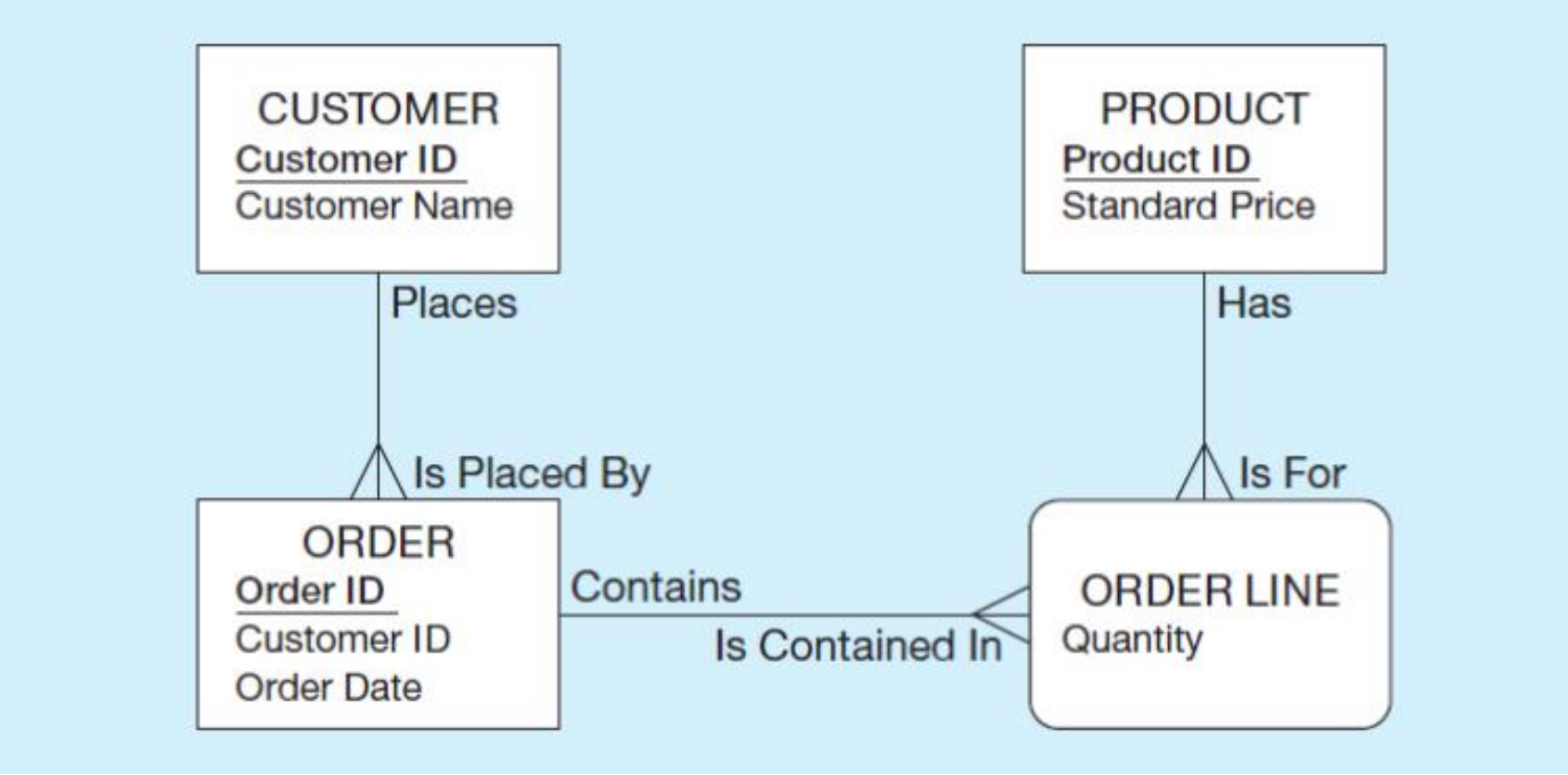
Steps in Table Creation

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 5-5 General Syntax of the 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



(from Chapter 1, Figure 1-3)

```
                    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 Specifications

Some primary keys are composite– composed of multiple attributes

```
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));
```

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));
```

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),
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));
```

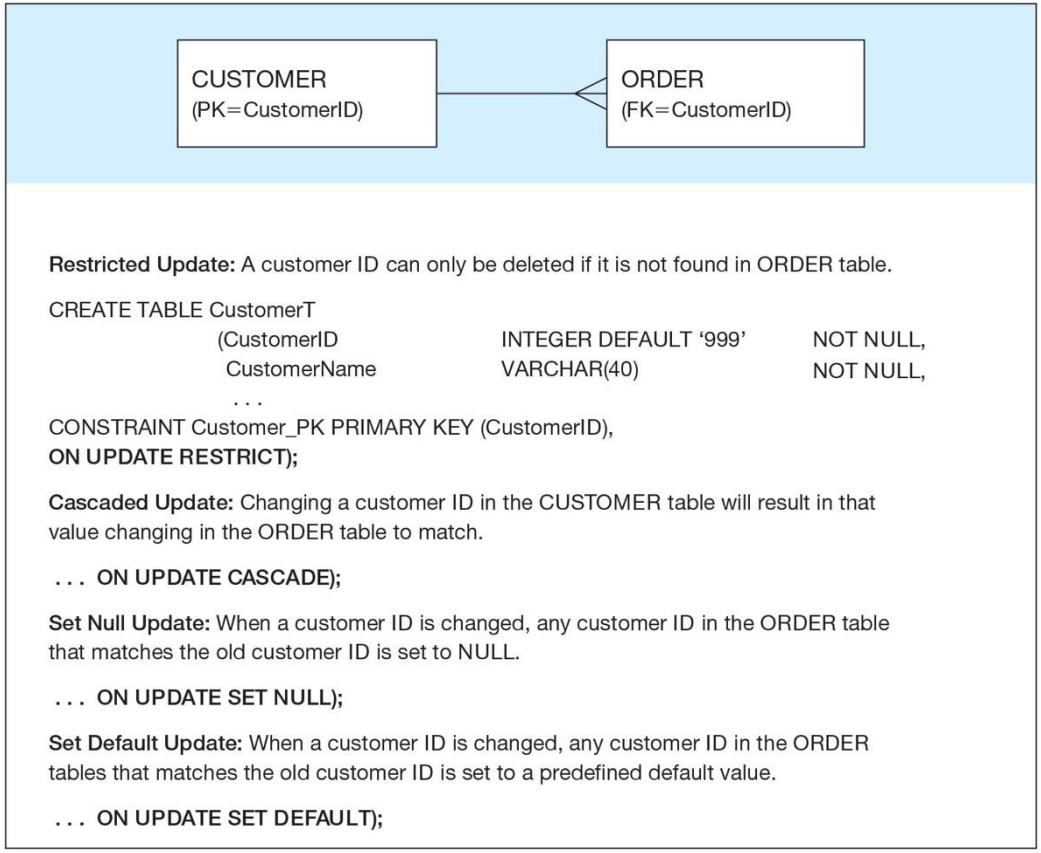
Primary key of parent table

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:N relationships
- Restricting:
  - Deletes of primary records
  - Updates of primary records
  - Inserts of dependent records

Figure 5-7 Ensuring Data Integrity Through Updates

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



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
- Example (adding a new column with a default value):
  - ALTER TABLE CUSTOMER_T ADD CustomerType VARCHAR2 (10) 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

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');

```
CREATE TABLE Customer_T
(CustomerID INTEGER GENERATED ALWAYS AS IDENTITY
      (START WITH 1
      INCREMENT BY 1
      MINVALUE 1
      MAXVALUE 10000
      NOCYCLE),
CustomerName            VARCHAR2(25) NOT NULL,
CustomerAddress         VARCHAR2(30),
CustomerCity            VARCHAR2(20),
CustomerState           CHAR(2),
CustomerPostalCode      VARCHAR2(9),
CONSTRAINT Customer_PK PRIMARY KEY (CustomerID));
```

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;

MERGE Statement

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

Useful for updating master tables with new data.

```
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);
```

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

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

SELECT Example

- Find products with standard price less than $275

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

- Comparison operators include
  - = 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_T **AS 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.

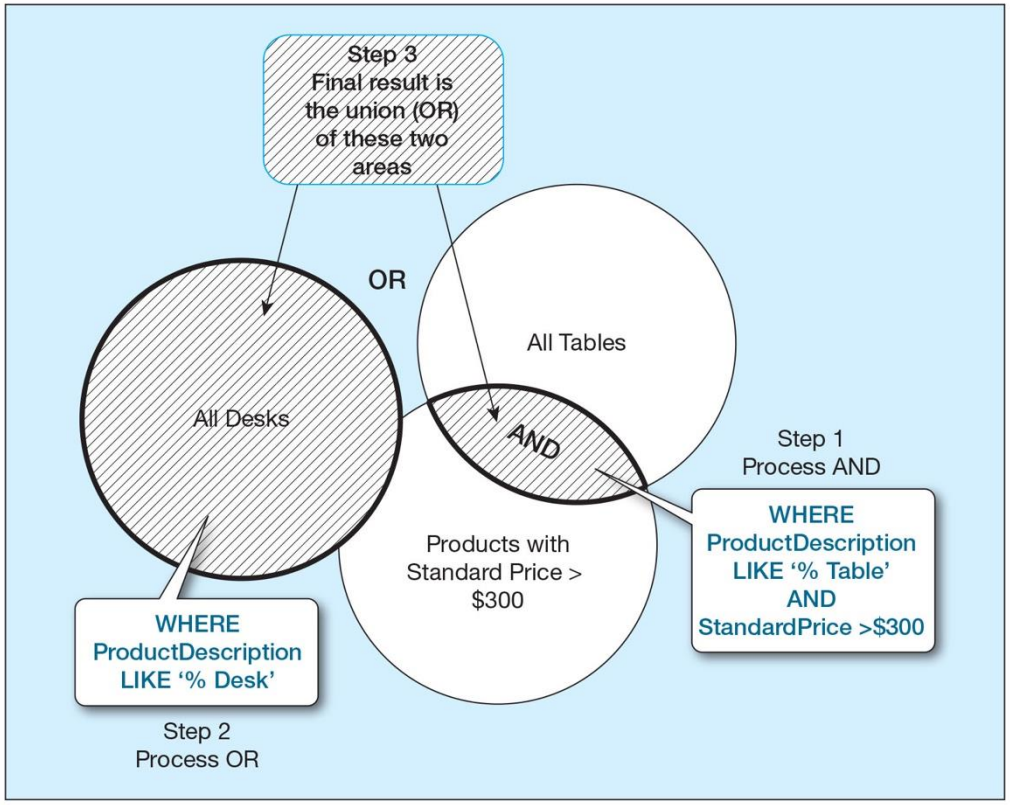
SELECT Example – Boolean Operators

- 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;
```

Figure 5-8 Boolean Query a Without the Use of Parentheses

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



SELECT Example–Boolean Operators

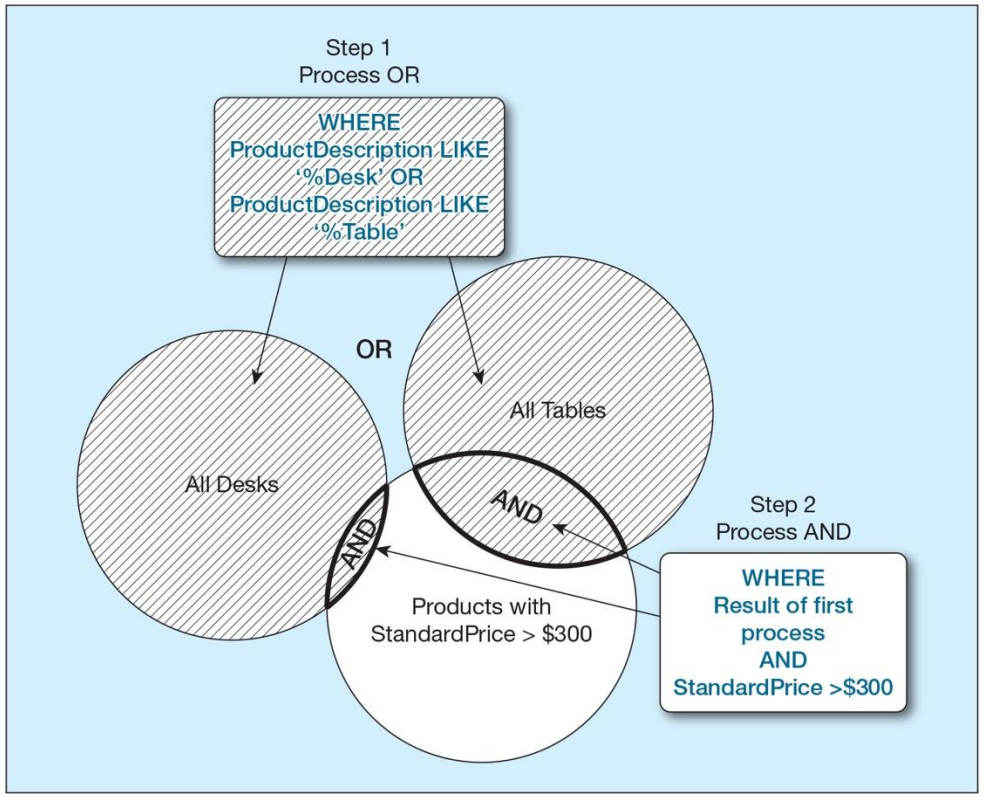
- With parentheses…these override the normal precedence of Boolean operators

```
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 5-9 Boolean Query B With Use of Parentheses

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



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;
```

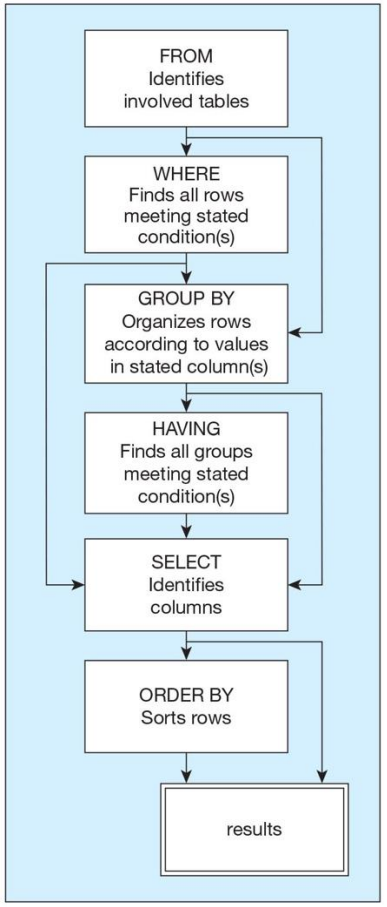
- Like a WHERE clause, but it operates on groups (categories), not on individual rows. Here, only those groups with total numbers greater than 1 will be included in the final result.

A Query With Both WHERE and HAVING

```
SELECT ProductFinish, AVG (ProductStandardPrice)

FROM Product_T

WHERE ProductFinish IN ('Cherry', 'Natural Ash', 'Natural
Maple', 'White Ash')

GROUP BY ProductFinish

HAVING AVG (ProductStandardPrice) < 750

ORDER BY ProductFinish;
```

Figure 5-10 SQL Statement Processing Order

SELECT [ALL/DISTINCT] column_list
FROM table list
[WHERE conditional expression]
[GROUP BY group_by_column_list]
[HAVING conditional expression]
[ORDER BY order_by_column_list]



(based on van der Lans, 2006, p. 100)

Copyright