Chapter 7: Introduction to SQL

Modern Database Management

8th Edition

Jeffrey A. Hoffer, Mary B. Prescott, Fred R. McFadden

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:2003 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
- 1989, 1992, 1999, 2003– Major ANSI standard updates
- 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

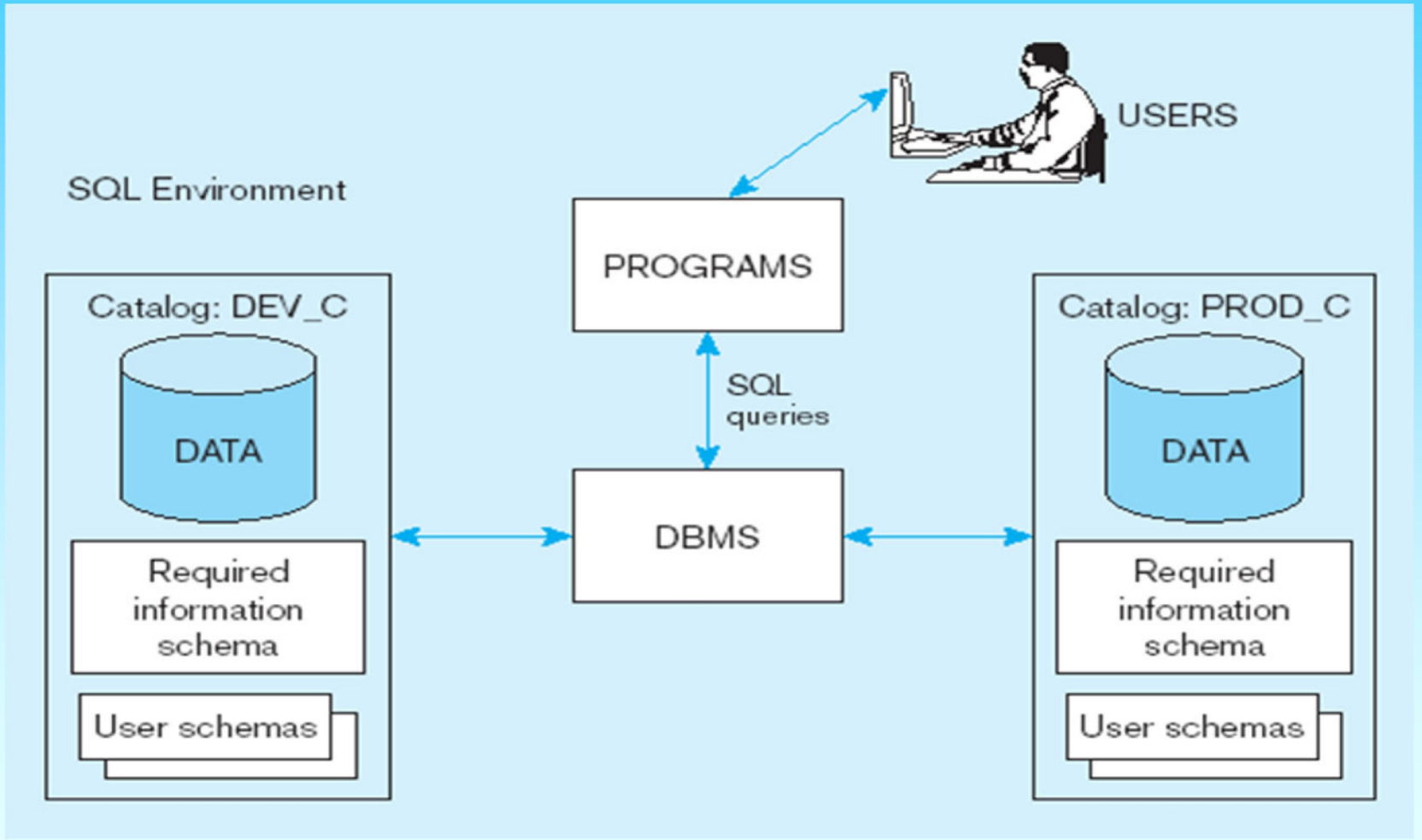
- Reduced training costs
- Productivity
- Application portability
- Application longevity
- Reduced dependence on a single vendor
- Cross-system communication

important

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-2003 standard

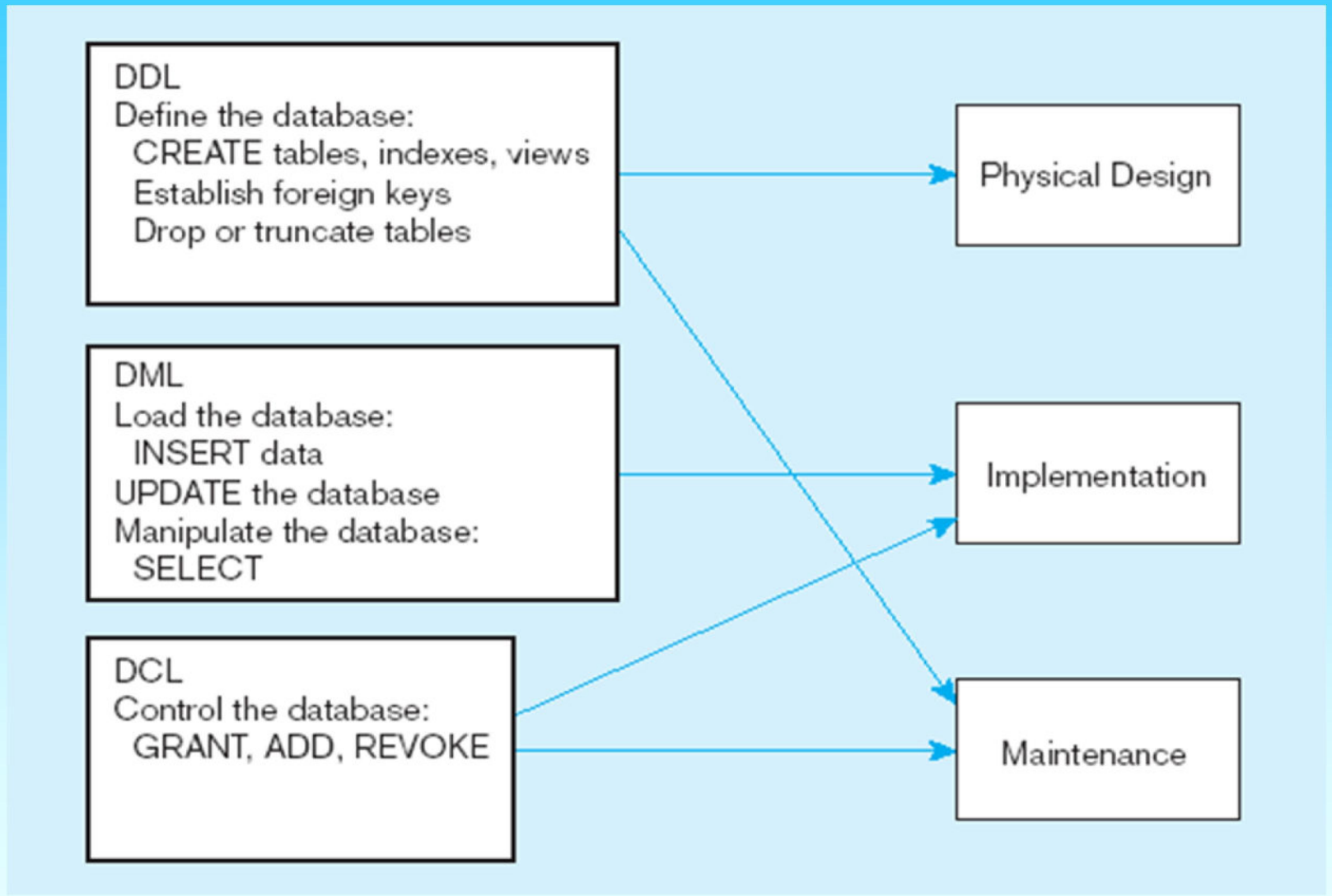


Some SQL Data types

Table 7-2 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)	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.
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	Stores a moment an event occurs, using a definable fraction of a second precision.
Boolean	BOOLEAN	Stores truth values, TRUE, FALSE, or UNKNOWN.

Figure 7-4
DDL, DML, DCL, and the database development process



SQL Database Definition

- Data Definition Language (DDL) ask
- 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

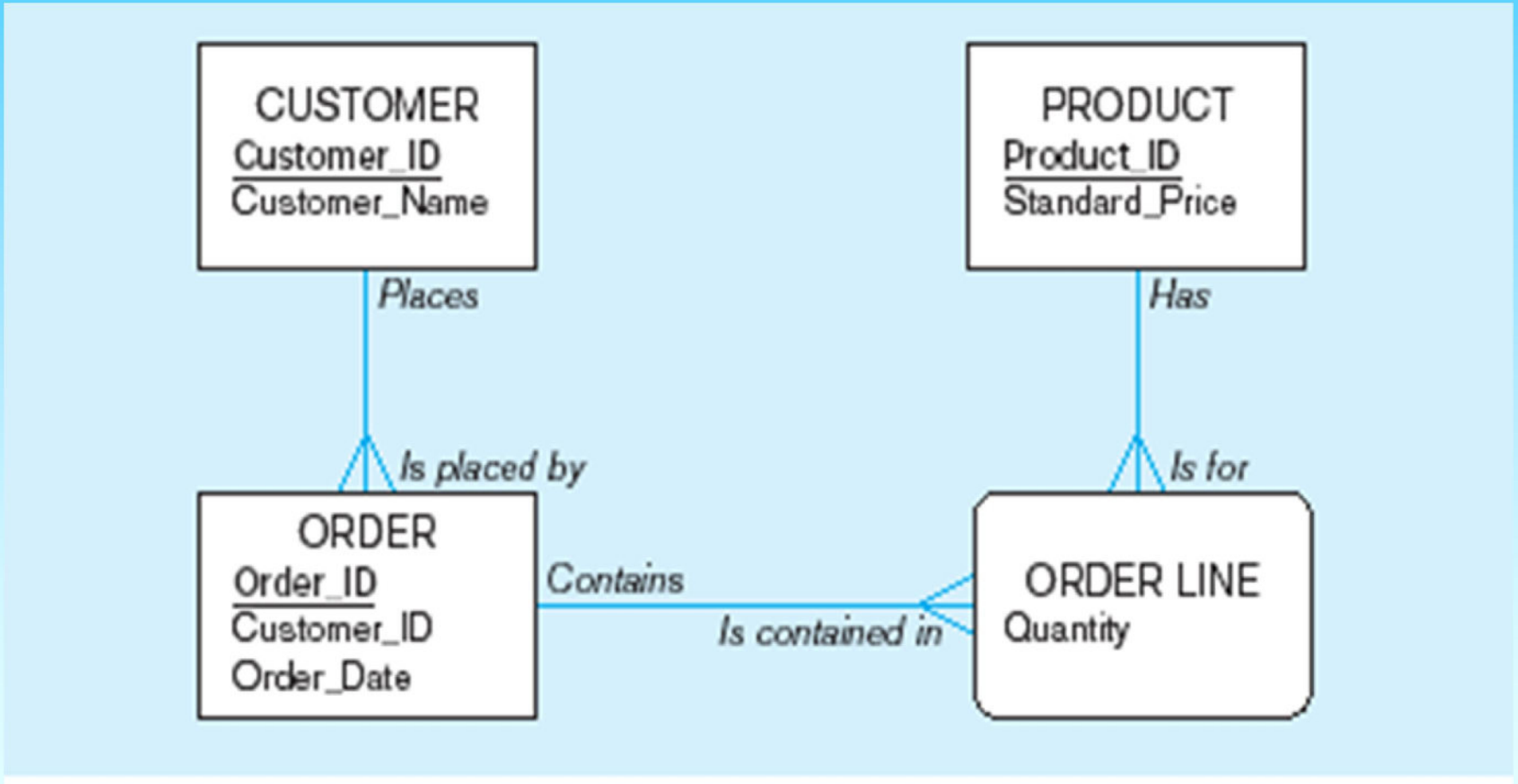


Figure 7-6 SQL database definition commands for Pine Valley Furniture

```
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));
```

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));
```

Defining attributes and their data types

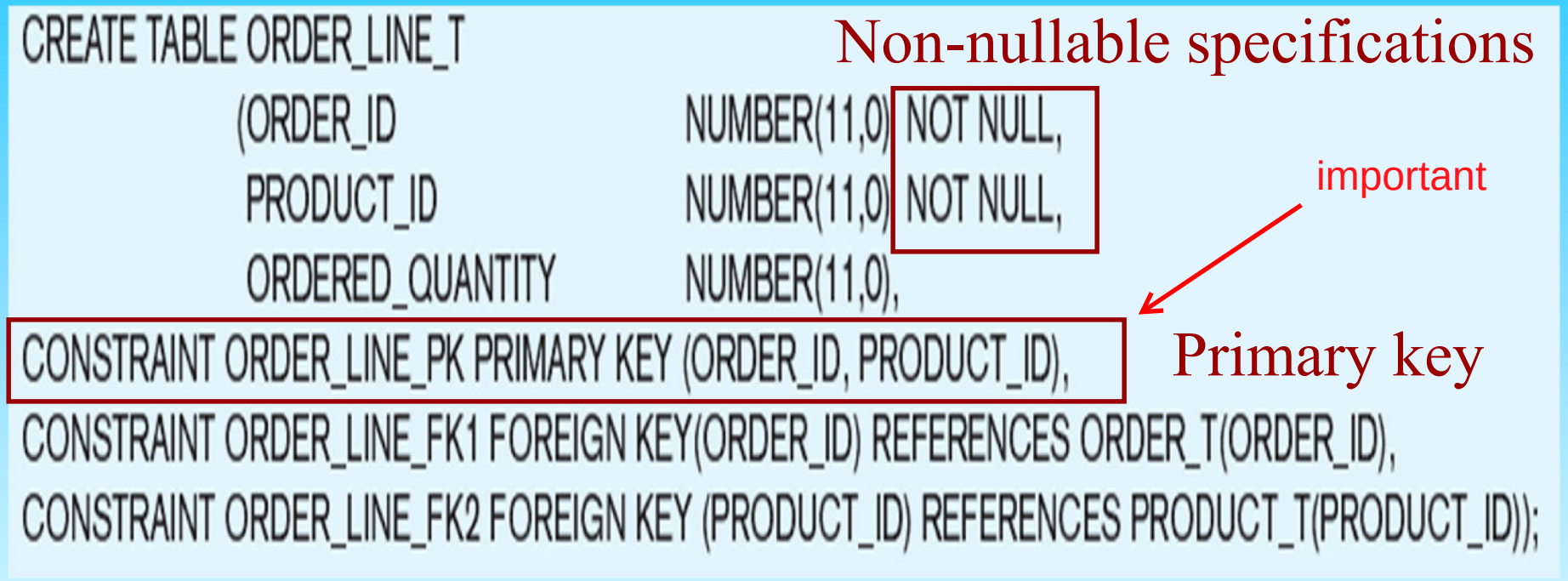
```
CREATE TABLE PRODUCT_T
        (PRODUCT_ID              INTEGER        NOT NULL,
         PRODUCT_DESCRIPTION     VARCHAR2(50),
         PRODUCT_FINISH          VARCHAR2(20)
                    CHECK (PRODUCT_FINISH IN ('Cherry', 'Natural Ash', 'White Ash',
                                'Red Oak', '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

Identifying primary key

Primary keys can never have NULL values



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,
```

Default value

good

good

Domain constraint

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

good

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

ask

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);

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

Changing and Removing Tables

- The ALTER TABLE statement allows you to rename an existing table.
- It can also be used to add, modify, or drop a column from an existing table
- DROP TABLE statement allows you to remove tables from your schema:
  - DROP TABLE CUSTOMER_T

Alter Table Statement

- **Renaming a table**
- The basic syntax for renaming a table is:

```
ALTER TABLE table_name
        RENAME TO new_table_name;
```

- **For example:**

```
ALTER TABLE suppliers
        RENAME TO vendors;
```

- This will rename the *suppliers* table to *vendors*.

Alter Table Statement

- **Adding column(s) to a table**
- To add a column to an existing table, the ALTER TABLE syntax is:

```
ALTER TABLE table_name
    ADD column_name column-definition;
```

- **For example:**

```
ALTER TABLE supplier
    ADD supplier_name  varchar2(50);
```

- This will add a column called *supplier_name* to the *supplier* table.

Alter Table Statement

- **Modifying column(s) in a table**
- To modify a column in an existing table, the ALTER TABLE syntax is:

```
ALTER TABLE table_name
    MODIFY column_name column_type;
```

- **For example:**

```
ALTER TABLE supplier
 MODIFY supplier_name  varchar2(100)  not null;
```

- This will modify the column called *supplier_name* to be a data type of varchar2(100) and force the column to not allow null values.

Alter Table Statement

- **Drop column(s) in a table**
- To drop a column in an existing table, the ALTER TABLE syntax is:

```
ALTER TABLE table_name
    DROP COLUMN column_name;
```

- **For example:**

```
ALTER TABLE supplier
    DROP COLUMN supplier_name;
```

- This will drop the column called *supplier_name* from the table called *supplier*.

Alter Table Statement

- **Rename column(s) in a table (NEW in Oracle 9i Release 2)**
- To rename a column in an existing table, the ALTER TABLE syntax is:

```
ALTER TABLE table_name
    RENAME COLUMN old_name to new_name;
```

- **For example:**

```
ALTER TABLE supplier
    RENAME COLUMN supplier_name to sname;
```

- This will rename the column called *supplier_name* to *sname*.

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

ask

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;

Creating Indexes

- Indexes are used to provide rapid access to tables data.
- Although, users don't directly refer to indexes, the RDMS recognizes which indexes would improve the query performance.
- Indexes can usually be created for primary and foreign keys (simple and composite)

Create Index

- To create an alphabetical index on customer name in Customer Table:

Create index Name_IDX on Customer_T(Customer_Name);

- once an index is created, it will be updated as data are entered, updated, or deleted.
- To remove Index:

  Drop Index Name_IDX;

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

Schema Definition

not important

- Control processing/storage efficiency
- Some techniques used to tune dbase performance:
  - Choosing to index keys to increase the speed of row selection, table joining, and row ordering.
  - Selecting File organizations for base tables that match type of processing (keeping table physically sorted by a frequently used sort key)
  - Selecting File organizations for indexes appropriate to the way the indexes are used.
  - Data clustering so that related rows of frequently joined tables are stored close together in secondary storage
  - Statistics maintenance about tables and their indexes so that DBMS can find the most efficient ways to perform various database operations.

- Creating indexes

until here

  - DBMS uses Indexes to Speed up random/sequential access to base table data
  - Although users do not directly refer to indexes when writing any SQL command, the DBMS recognizes which existing indexes would improve query performance
  - Indexes are usually created for both primary and foreign keys and both single and compound keys
  - Indexes could be in ascending or descending sequence
  - Example
    - CREATE INDEX NAME_IDX ON CUSTOMER_T(CUSTOMER_NAME)
    - This makes an index for the CUSTOMER_NAME field of the CUSTOMER_T table
  - To remove the index
    - Drop INDEX NAME_IDX