Database Implementation

(Dealing with SQL and advanced SQL)

- **Chapter 6**
  Introduction to SQL
- **Chapter 7**
  Advanced SQL

MDBM-Chapter 6

Introduction to SQL

Objectives

- Explain history, role and basics of SQL.
- Define a database using SQL data definition language.
- Write single table queries using SQL.
- Establish referential integrity using SQL.

SQL Concept

- **Structured Query Language (SQL)**: the standard computer language for creating and querying relational databases in a relational database management systems (RDBMS).

- **RDBMS**: A database management system that manages data as a collection of tables in which all the relationships are represented by common values in related tables.

History of SQL

- 1970: Edgar Codd developed relational database concept.
- 1974-1979: *System R* with Sequel (later SQL) created at IBM Research Lab.
- 1979: Oracle marketed the first relational DB with SQL.
- 1986: ANSI SQL standard released.
- Major ANSI standard updates:1989, **1992**, 1999, 2003, **2008** (Revisions are mostly backwards-compatible with older standards).
- Current: SQL (1992) is supported by most major database vendors; Oracle 11g provides full or partial conformance with Core SQL:2008.

Original Purposes 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.

Benefits of a Standardized Relational Language

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

SQL and Its Variations/Flavors

- Although SQL is an ANSI standard, there are many different versions of the SQL language.
- To be compliant with the ANSI standard, they all support at least the major commands (e.g., SELECT, UPDATE, DELETE, INSERT, WHERE) in a similar manner.
- Most of the SQL database programs also have their own proprietary extensions in addition to the SQL standard.
- Examples of SQL variations: SQL*PLUS (Oracle), Transact-SQL or T-SQL (Microsoft SQL Server), and MySQL (MySQL).

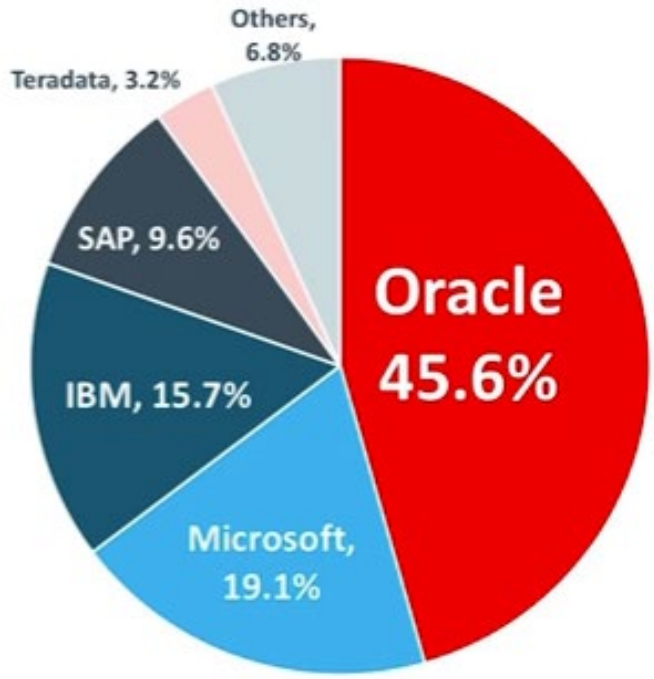
Market Share of Three Major Relational Database Vendors

- Oracle (over 44% in 2007, 48.8% in 2014, 45.6% in 2016).
- IBM DB2 (about 21% in 2007, 20.2% in 2014, 15.7% in 2016).
- Microsoft SQL Server (about 19% in 2007, 17.0% in 2014, 19.9% in 2016).

RDBMS Market Share

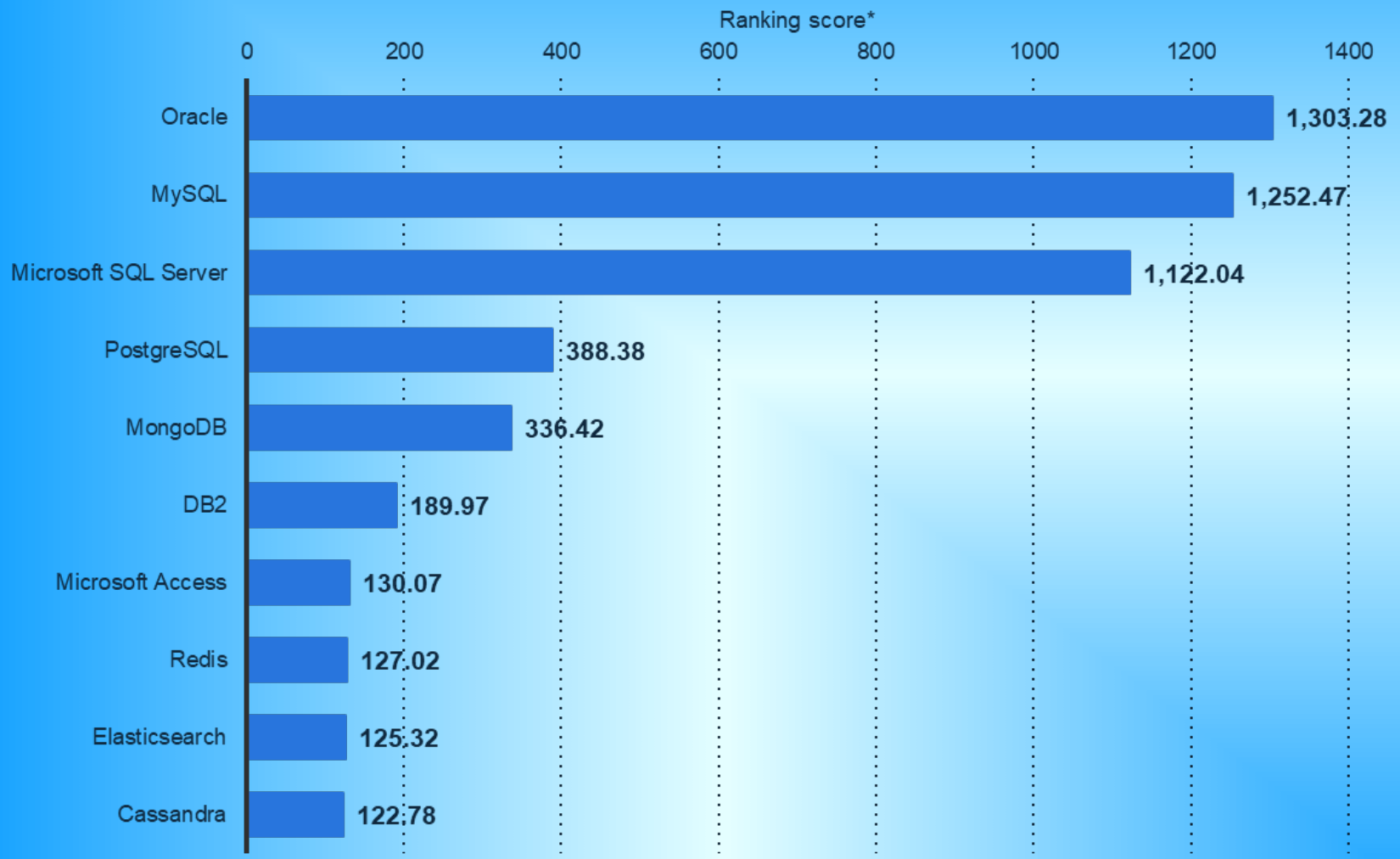
Oracle Continues Have Largest **RDBMS** Market Share by Wide Margin

More than Double Sales of Nearest Competitor



Graphic created by Oracle based on Gartner Research: Gartner "Market Share: All Software Markets, Worldwide 2015", March 31, 2016

Ranking of the most popular database management systems worldwide (Feb. 2018)



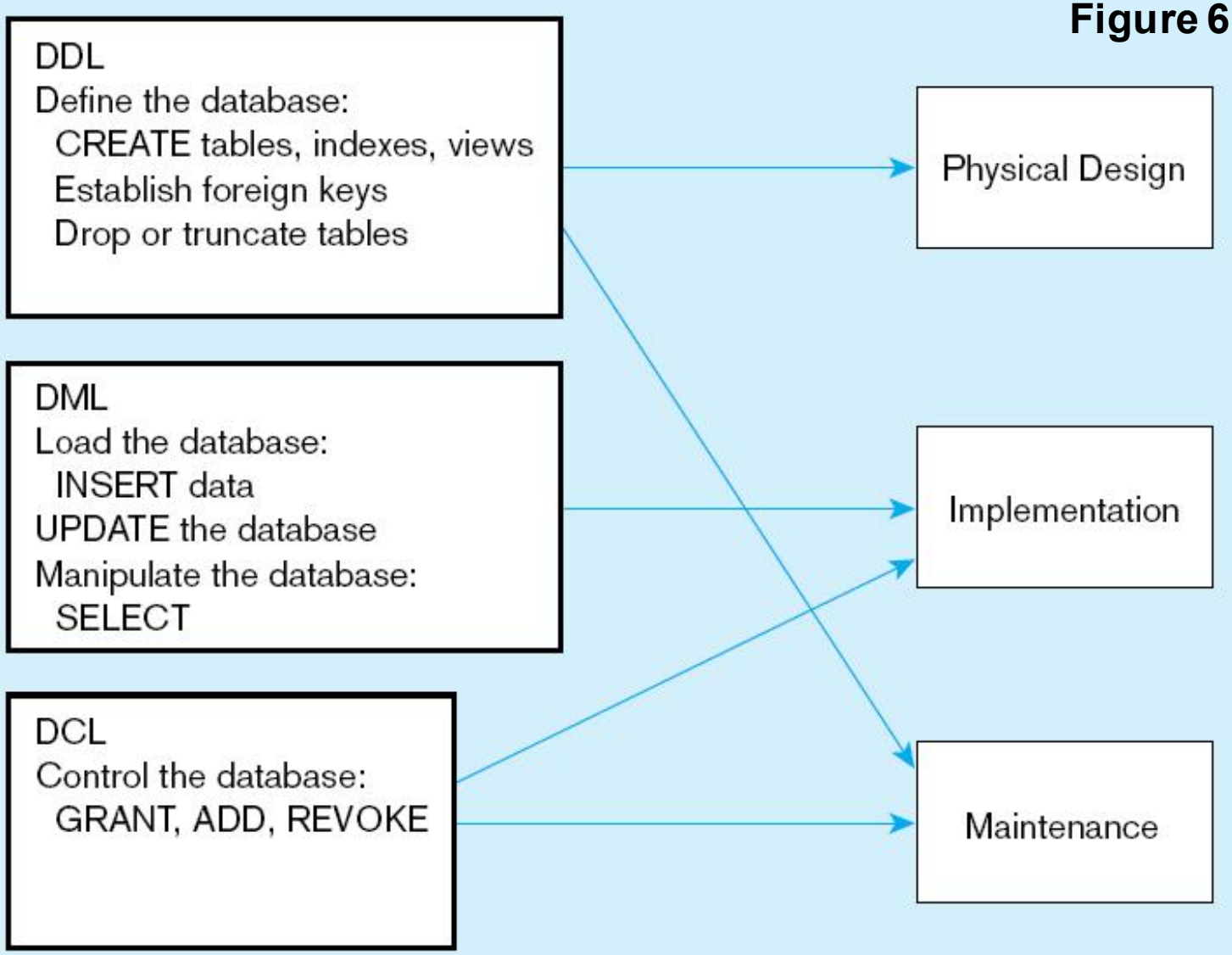
Source(s): DB-Engines; ID 809750

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, and 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, including updating, inserting, modifying and querying data
- Data Control Language (DCL)
  Commands that control a database, including administering privileges and committing (saving) data

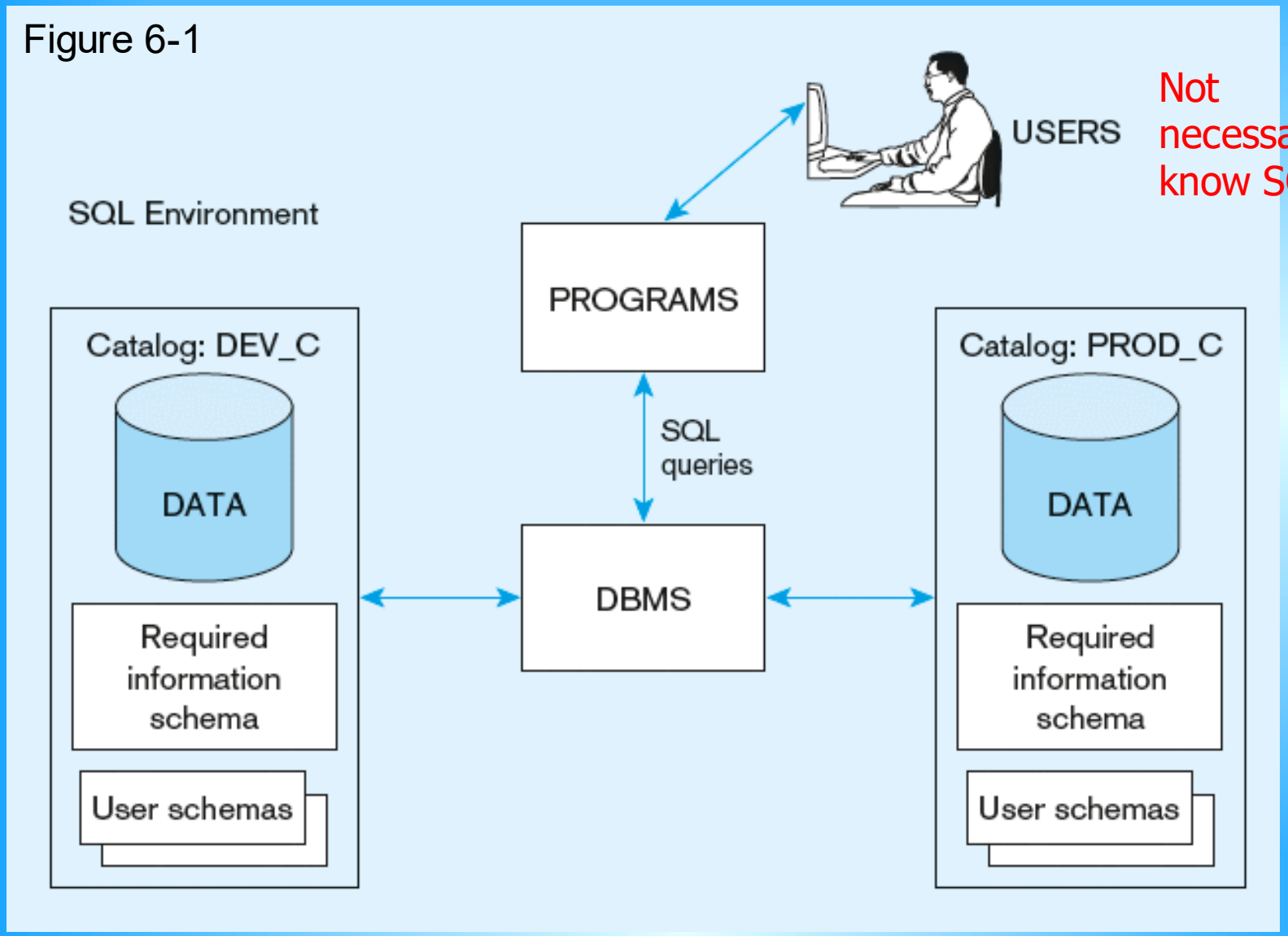
DDL, DML, DCL, and the database development process

Figure 6-4



A simplified schematic of a typical SQL environment, as described by the SQL: 200n standard

Figure 6-1



SQL Data Type Examples

(Each DBMS has a defined list of 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	TIMESTAMP TIMESTAMP 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.

Read the Supplemental Materials Posted on the Blackboard

- Oracle Database Data Types
- Oracle Schema Object Names and Qualifiers

Some Guidelines for Selecting Data Types

- The data type must bust be specified for each attribute when a table is created.
- Data values and expected use of the data affect the data type to use.
  - A unit price should use a numeric data type.
  - An address should use a string type.
- Numeric data (e.g., phone numbers) should be stored as character data if no mathematical operations are expected since character data are processed faster.
- Temporal data should be stored as date type rather than a string type, which will allow to use date/time calculation functions.
- Watch for (new) data types as SQL standards update, e.g., BIGINT, XML, CLOB, BLOB, and NLOB, etc.

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

CREATE SCHEMA (in Oracle)

Use the CREATE SCHEMA statement to create multiple tables and views and perform multiple grants in your own schema in a single transaction.

```
CREATE SCHEMA AUTHORIZATION oe
        CREATE TABLE new_product
                (color VARCHAR2(10) PRIMARY KEY, quantity NUMBER)
        CREATE VIEW new_product_view
                AS SELECT color, quantity FROM new_product
                WHERE color = 'RED'
        GRANT select ON new_product_view TO hr;
```

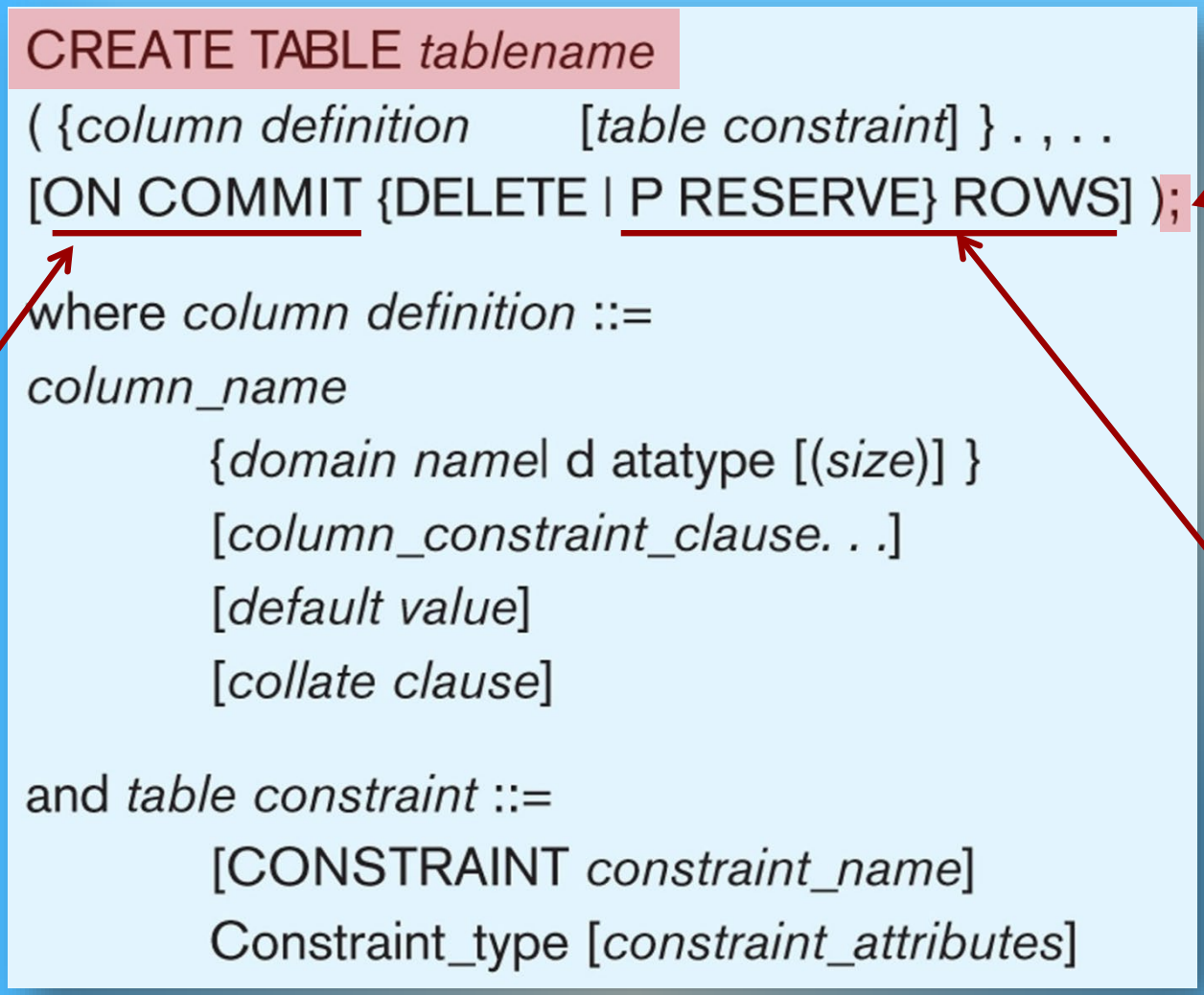
This statement does not actually create a schema. An Oracle database automatically creates a schema when you create a user. It lets you populate your schema with tables and views and grant privileges on those objects without having to issue multiple SQL statements in multiple transactions.

Table Creation

Steps in table creation:

1. Identify data types for attributes, including length, precision, and scale if required.
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 if any.
6. Identify constraints on columns (domain specifications) if any.
7. Create the table and associated indexes if any.

General syntax for CREATE TABLE statement used in data definition language (Figure 6-5)

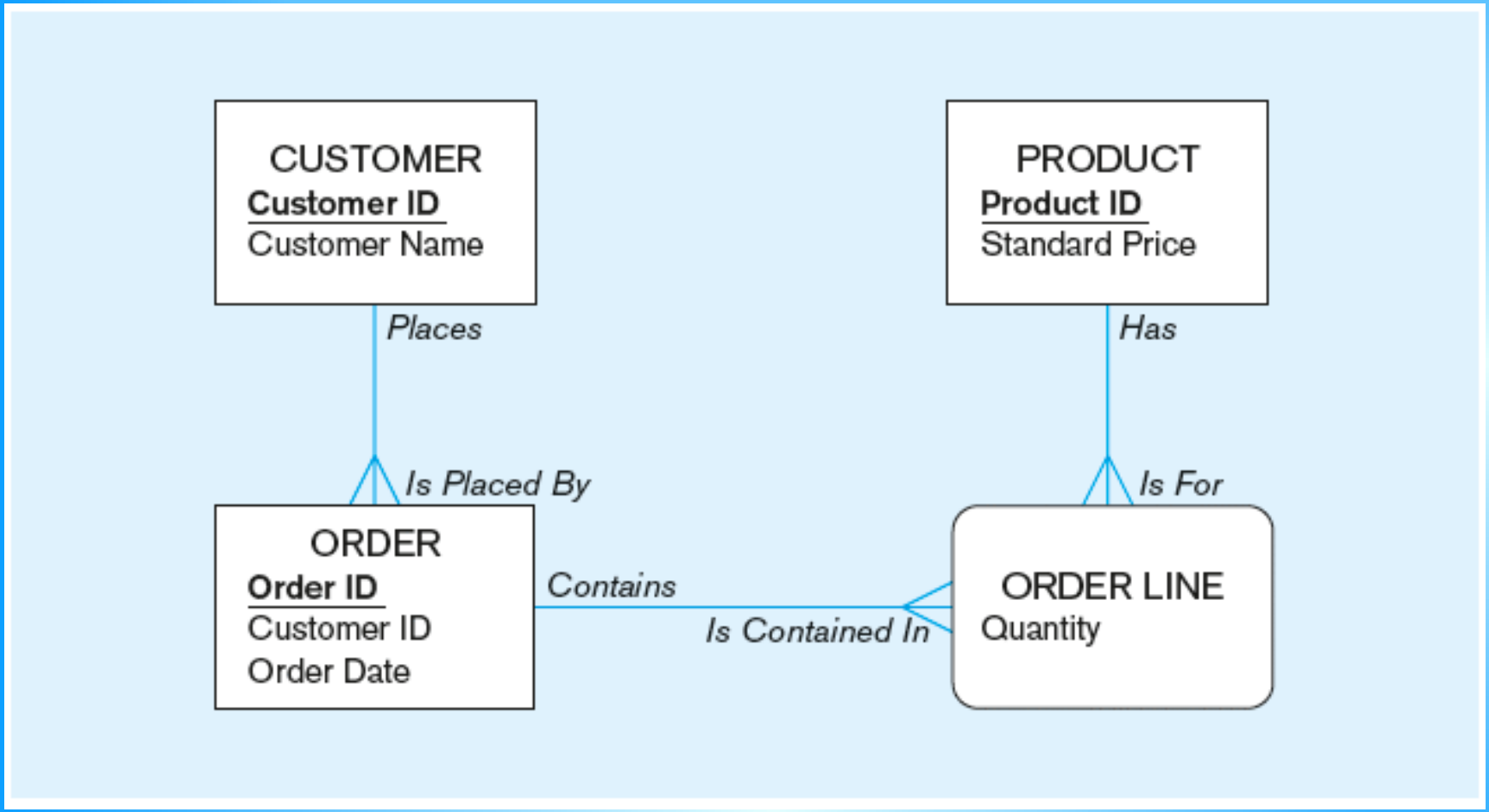


Relevant only if you are creating a temporary table

Each SQL statement ends w/ a semicolon

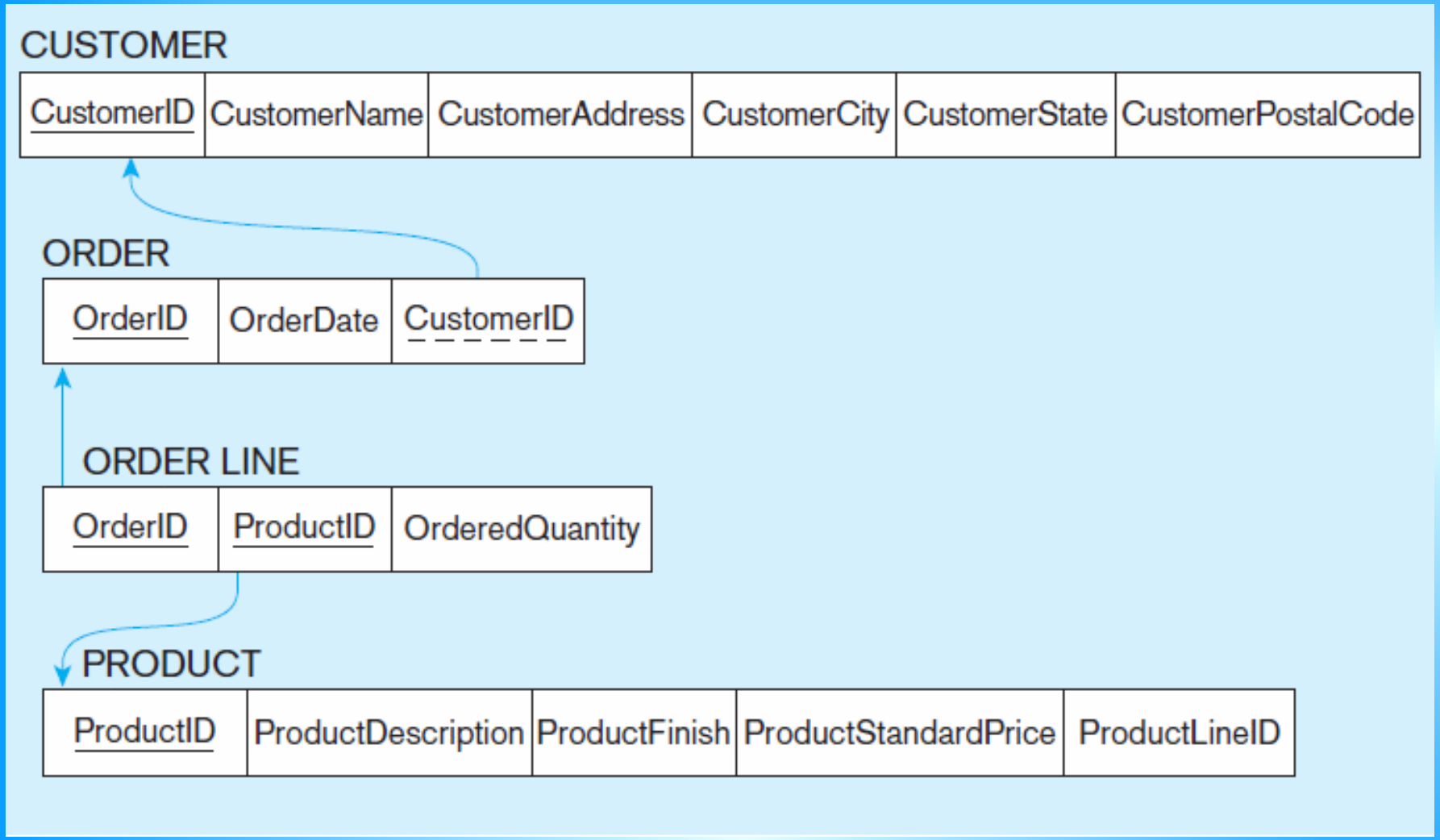
Specify PRESERVE ROWS for a session-specific temporary table

The following slides create tables for this project data model



(from Chapter 1, Figure 1-3b)

Normalized relations showing referential integrity constraints



Ready for database implementation

SQL database definition commands for Pine Valley Furniture Company (Oracle 11g)

```
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));
```

```
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));
```

Overall table definitions

Figure 6-6

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 specification

```
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));
```

Primary keys can never have NULL values

Identifying the primary key

Non-nullable specification

```
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));
```

Some primary keys are composite– composed of multiple attributes

Controlling the values in attributes

Default value

```
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));
```

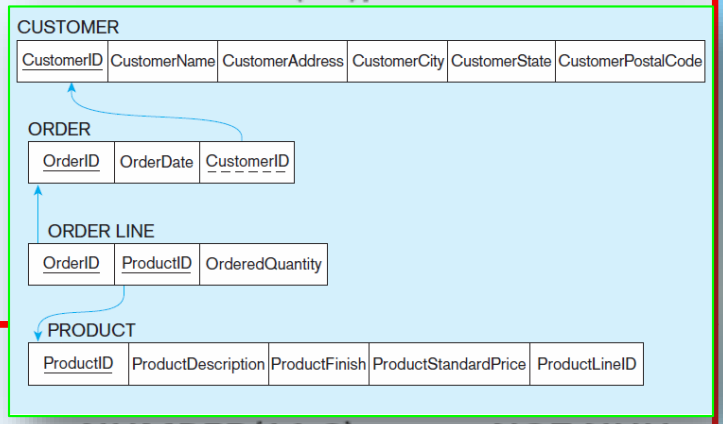
Domain constraint: a CHECK constraint

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
         CustomerState
         CustomerPostalCode
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



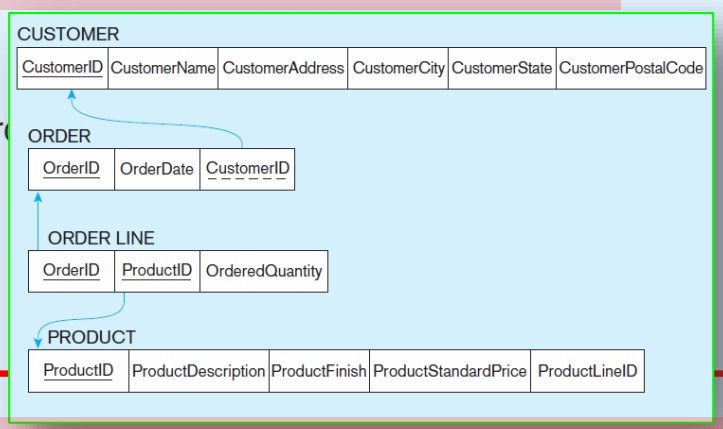
Foreign key of the dependent table

Identifying foreign keys and establishing relationships

```
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
                 ProductFinish
                                    CHECK (Pr
                 ProductStandardPrice
                 ProductLineID
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));
```



Foreign keys of the dependent table

More on Data Integrity Controls

Referential integrity: a constraint that ensures that foreign key values of a table must match primary key values of a related table in 1:M relationships (restricting the insertion of dependent records); however, this constraint does not prevent changes that may happen to the primary key, e.g., update primary records and delete primary records.

Primary Key referential integrity on update

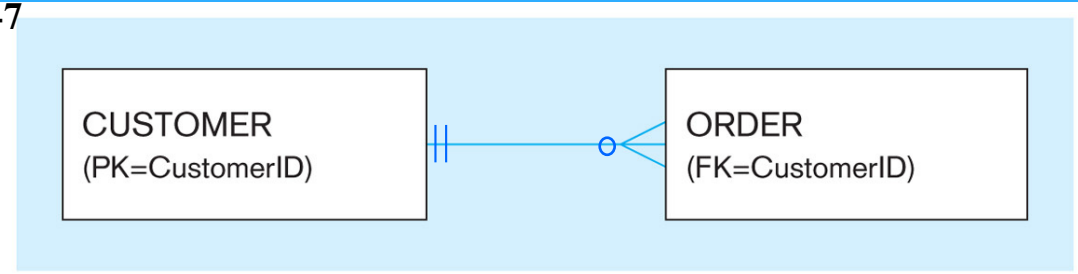
- **ON UPDATE RESTRICT** clause: Any updates that would delete or change a primary key value will be rejected unless no foreign key references that value in any child table.
- **ON UPDATE CASCADE** clause: A change in a primary key value will flow through (cascade) to the child table by also updating the value in that child table. This is the most flexible option.
- **ON UPDATE SET NULL** clause: An update in a primary key value will set FK value in the child table to null, losing the connection b/n the two instances.
- **ON UPDATE SET DEFAULT** clause: An update in a primary key value will set FK value in the child table to a predefined default value.

Primary Key referential integrity on deletion

- **ON DELETE RESTRICT** clause: A record in the parent table cannot be deleted unless no foreign key in any child table references the primary key value of that record. This option probably makes the most sense.
- **ON DELETE CASCADE** clause: Deleting a record the parent table will remove all the associated records in the child table(s). Use with caution: it may produce undesirable results.
- **ON DELETE SET NULL** clause: Before the record in the parent table is deleted, the associated FK value in the child table is set to null.
- **ON DELETE SET DEFAULT** clause: Before the record in the parent table is deleted, the associated FK value in the child table is set to a predefined default value.

Ensuring data integrity through updates

Figure 6-7



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);
```

Note: Different DBMS may have different syntaxes.

Oracle Supports the Following Three

- **ON DELETE CASCADE:** when a referenced parent table row is removed all the children are removed automatically)
- **ON DELETE SET NULL:** it allows data that references the parent key to be deleted, but not updated. When referenced data in the parent key is deleted, all rows in the child table that depend on those parent key values have their foreign keys set to null.
- **ON DELETE NO ACTION:** it is the default, preventing deleting a parent when there are children.

(Other referential actions not supported by PK/FK integrity constraints of Oracle can be enforced using database triggers.)

Example: Oracle ON DELETE CASCADE

```
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) ON DELETE CASCADE);
```

Create a table with "CREATE TABLE…LIKE"

This is used to create a table based on the definition of an existing table.

```
CREATE TABLE newTbl LIKE existing_table;
```

To verify the result:

```
DESCRIBE newTbl
```

(This statement is available in SAS PROC SQL and MySQL but not in Oracle; however, it is easy to achieve the same result in Oracle. We will do this in a lab session.)

Changing Table Definitions

- **ALTER TABLE** statement allows you to change column specifications:

  ALTER TABLE *table_name* alter_table_action;

- Alter_table_actions:

  ADD [COLUMN] column_definition
  ALTER [COLUMN] column_name **SET DEFAULT** default_value
  ALTER [COLUMN] column_name **DROP DEFAULT**
  DROP [COLUMN] column_name **[RESTRICT] [CASCADE]**
  MODIFY [COLUMN] column_name column_definition
  ADD table_constraint

- Example (adding and dropping a new column in Oracle):

ALTER TABLE *Customer_T*
ADD CustomerType VARCHAR(12);

No word "Column"

ALTER TABLE *Customer_T*
Drop Column CustomerType;

With word "Column"

Removing Tables

The **DROP TABLE** statement allows you to remove tables from your schema. It removes the whole table, including its definition, contents, constrains, etc.

```
DROP TABLE Table_name;
```

Example:

```
DROP TABLE Customer_T;
```

Removing all the data from a table

Different from the DROP TABLE statement, the **TRUNCATE TABLE** statement just removes all the **data** from a specified table but the table structure is still retained.

```
TRUNCATE TABLE Table_name;
```

Example:

```
TRUNCATE TABLE Customer_T;
```

Insert Statement

To add a row(s) of data to a table, use **INSERT INTO**

- Inserting into a table (note the order of the values)

```
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);
```

- Inserting from another table

```
INSERT INTO CaCustomer_T
SELECT * FROM Customer_T
    WHERE CustomerState = 'CA';
```

The two tables must have the same structure.

Batch Input Methods

- The SQL*Loader program in Oracle
- The BULK INSERT command in Transact-SQL of SQL Server

Creating Tables with Identity Columns

```
CREATE TABLE Customer_T
(CustomerID INTEGER GENERATED ALWAYS AS IDENTITY
     (START WITH 1
     INCREMENT BY 1
     MINVALUE 1
     MAXVALUE 10000
     NO CYCLE),
CustomerName          VARCHAR2(25) NOT NULL,
CustomerAddress       VARCHAR2(30),
CustomerCity          VARCHAR2(20),
CustomerState         CHAR(2),
CustomerPostalCode    VARCHAR2(9),
CONSTRAINT Customer_PK PRIMARY KEY (CustomerID);
```

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);
```

Note: This was introduced in SQL:200n and is available in SQL Server and MS Access but not in Oracle; however, it can be easily achieved with Oracle programming by creating a sequence and a trigger.)

The DELETE Statement
(Removes rows of data from a table)

- Delete certain rows

  DELETE FROM *Customer_T*

  WHERE CustomerState = 'HI';

- Delete all rows

  DELETE FROM *Customer_T;*

  - It does the same as TRUNCATE TABLE.
  - Use with caution.

The UPDATE Statement

(Modifies **data contents** in existing rows)

```
UPDATE Product_T
SET ProductStandardPrice = 775
WHERE ProductID =7;
```

The MERGE Statement

Source of data

Target table

```
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  SET
    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);
```

Condition

True

False

SET command is missing in the book

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

Useful for updating master tables with new data.

Query Tables

- Now we have stored a big amount of good data in the database tables, how to retrieve (or make used of) the data from these tables?
- The tool to use is the **SELECT** statement.
- The syntax of the SELECT statement is relatively easy but it is powerful and can be used to produce complex data processing procedures.
- Always test on a small table before you query a big data table and make sure your query produces the desired result.
- The following slides formally introduce the SELECT statement although we have used it many times already.

The SELECT Statement

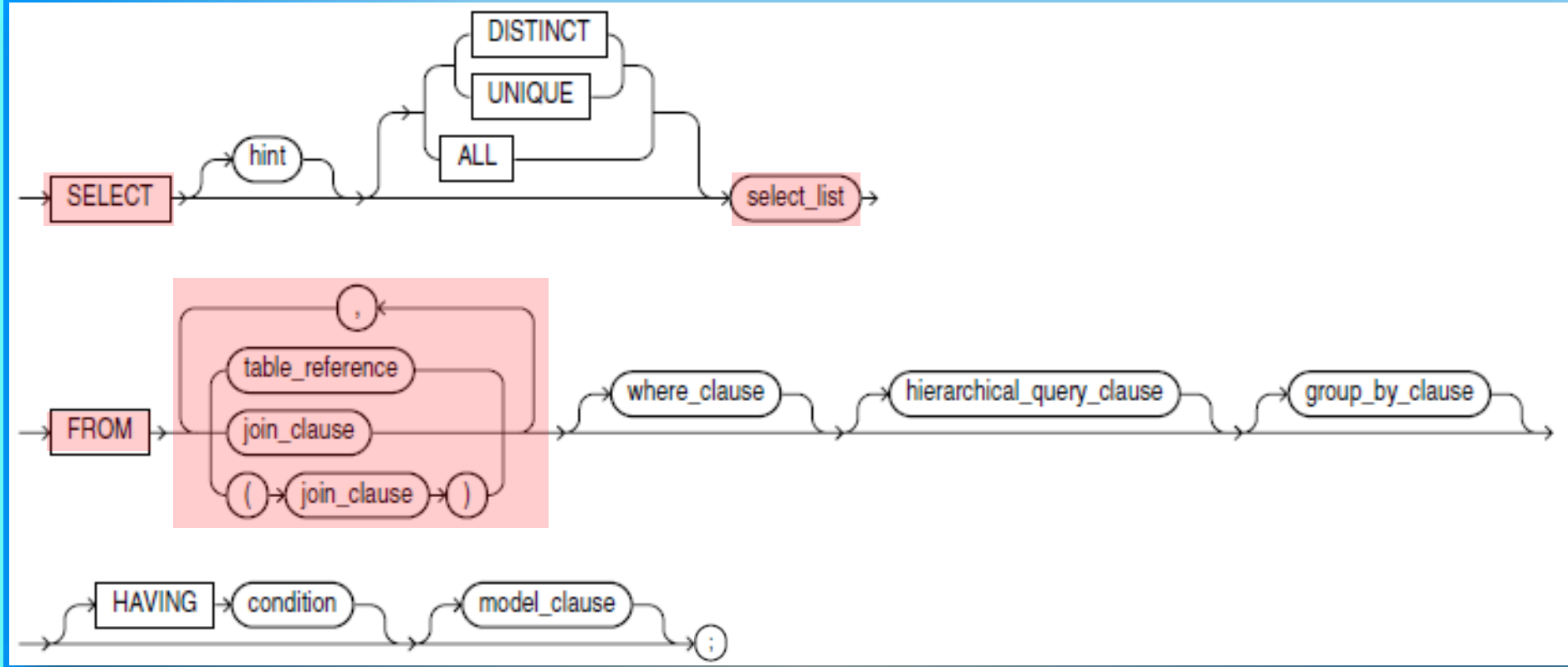
(Used for queries on single or multiple tables)

Clauses of the SELECT statement:

- **SELECT** ← A required clause
  - Lists the columns (and/or expressions) that should be returned from the query.
- **FROM** ← A required clause
  - Identifies the table(s) or view(s) from which data will be obtained.
- WHERE
  - Indicates the condition(s) under which a row will be included in the result from the tables (or views) that will be joined.
- GROUP BY
  - Indicates categorization of results.
- HAVING
  - Indicates the condition(s) under which a category (group) will be included.
- ORDER BY
  - Sorts the result according to specified criteria.

The Syntax Diagram of the SELECT Statement

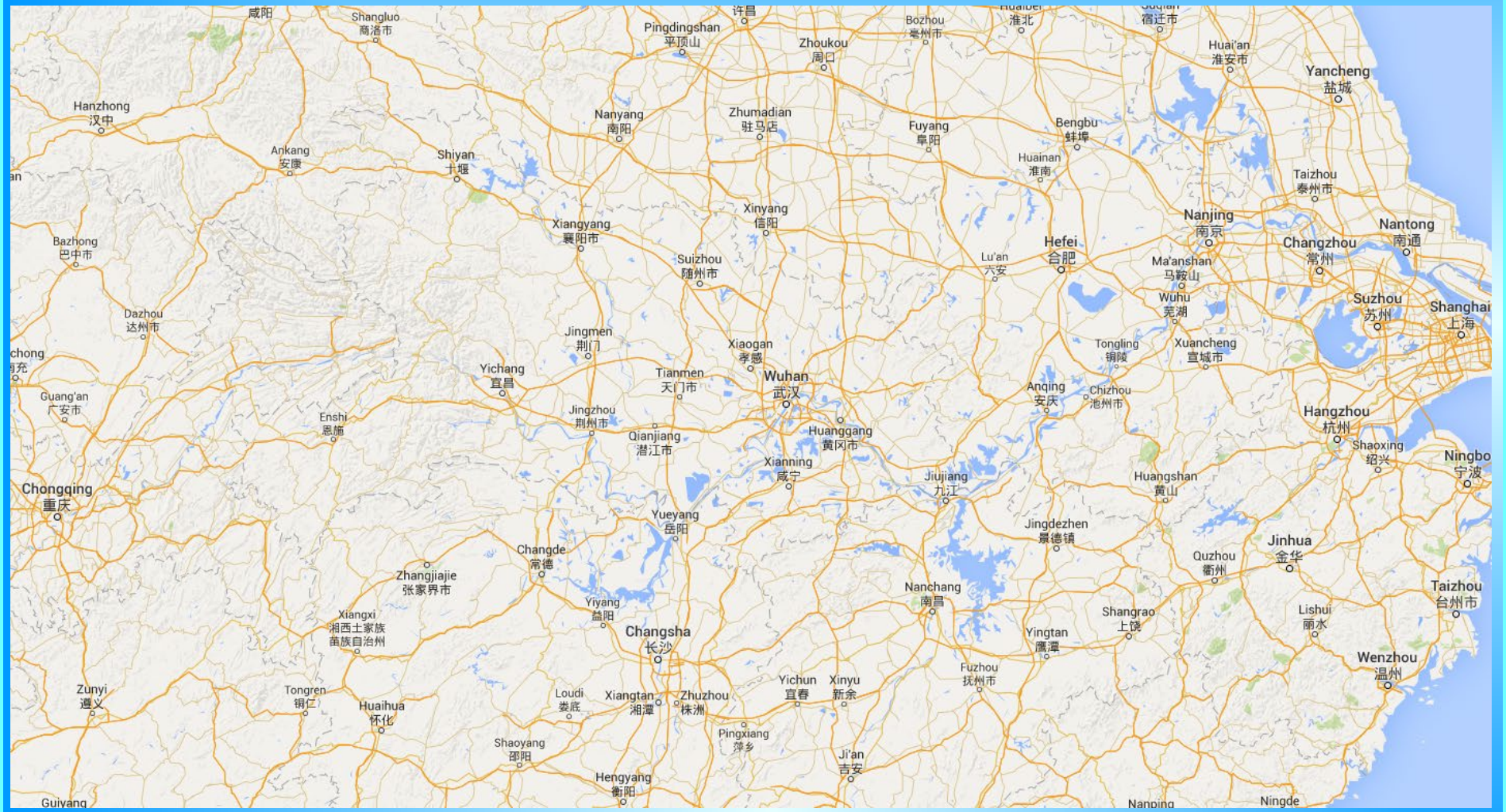
The order of syntax used in the SELECT statement can be illustrated by a syntax diagram. The following diagram shows that only the SELECT and FROM clauses are mandatory.



A syntax diagram can be used to check if your query is correct or not.

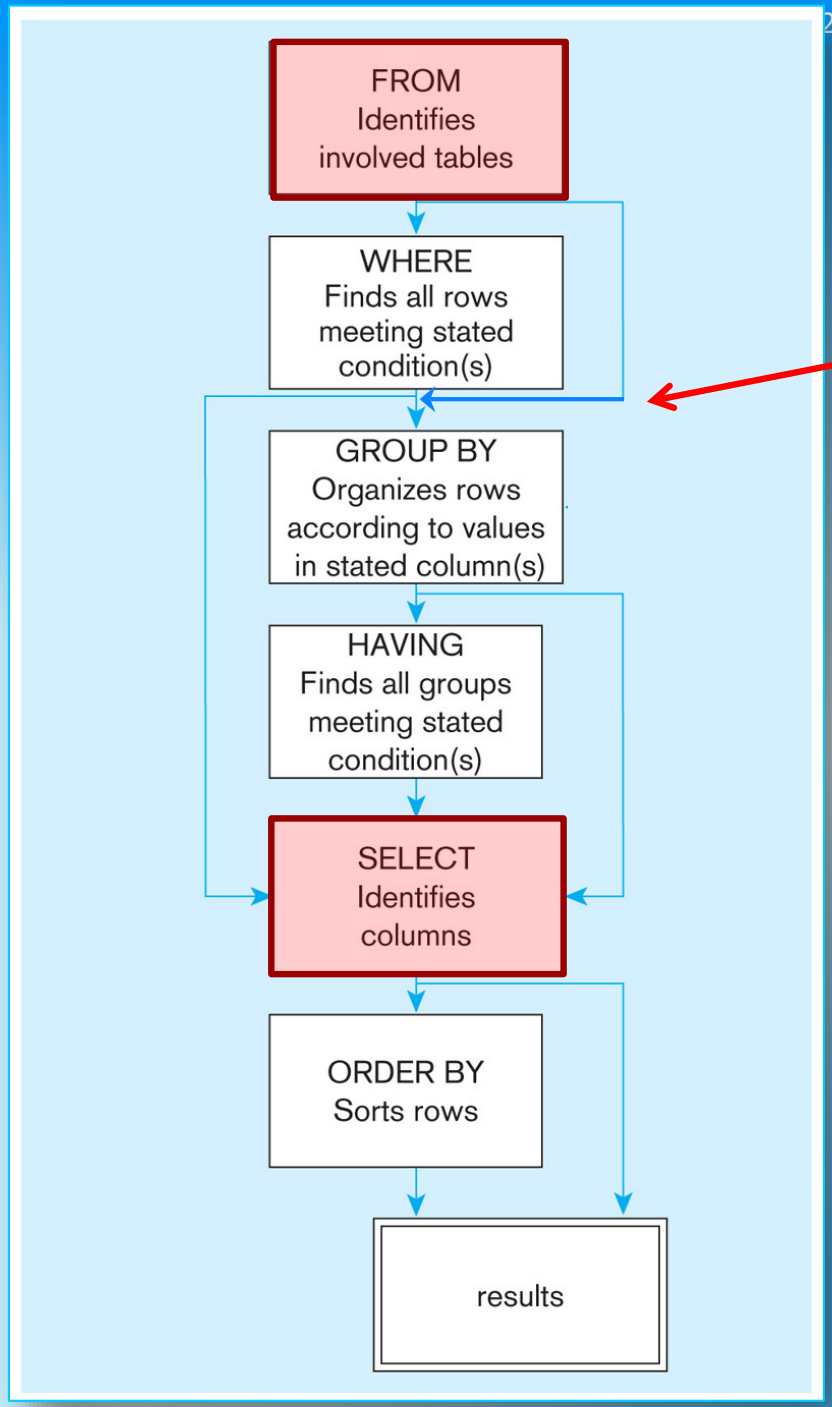
Analogy: A Cruise Trip From Shanghai to Chongqing:

Shanghai → Nanjing → Wuhan → Jingzhong → Yichong → Chongqing;



SELECT statement processing order

Note that the processing order is different from the order of syntax used in the SELECT statement and that only the SELECT and FROM clauses are mandatory.



Note: an error in the book

A Simple SELECT statement Example

Find products with a standard price less than $275

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

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

Using Wildcards in a SELECT Statement

- **Asterisk (*):** to represent all columns

  SELECT * FROM *Table_T;*

  (Returns all the columns of all the rows from Table_T)

- **Percent sign (%)** paired with key word **LIKE**, e.g., LIKE 'X%X' in the WHERE clause: to represent a collection of any number of characters.

  SELECT * FROM Suppliers_T

  WHERE supplier_name LIKE 'Hill%';

  (Returns all the rows from Suppliers_T whoes column supplier's names start with "Hill")

- **Underscore (_)** paired with key word **LIKE**, e.g., LIKE 'X_X' in the WHERE clause: to represent exactly one character.

  SELECT * FROM Suppliers_T

  WHERE supplier_name LIKE 'Sm_th';

  (Returns suppliers whose names are 5-character long, where the first two characters are 'Sm' and the last two characters are 'th.' For example, it could return suppliers whose name is 'Smith' or 'Smyth.')

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';
```

Optional in Oracle

Oracle does not support this "AS"

Oracle does not support column alias in the rest of the SELECT statement, except in a HAVING clause

The correct code in Oracle

```
SELECT CUST.CUSTOMERNAME AS NAME,
    CUST.CUSTOMERADDRESS
FROM CUSTOMER_T CUST
    WHERE CUSTOMERNAME = 'Home Furnishings';
```

```
SELECT CUST.CUSTOMERNAME NAME,
    CUST.CUSTOMERADDRESS
FROM CUSTOMER_T CUST
    WHERE CUSTOMERNAME = 'Home Furnishings';
```

Result:

NAME	CUSTOMERADDRESS
Home Furnishings	1900 Allard Ave.

Using Expressions in the SELECT Statement

Mathematical manipulations (+,-,*,/) can be used to manipulate any numeric columns.

```
SELECT ProductID, ProductStandardPrice,
ProductStandardPrice * 1.1 AS Plus10Percent
FROM Product_T;
```

PRODUCTID	PRODUCTSTANDARDPRICE	PLUS10PERCENT
2	200.0000	220.00000
3	375.0000	421.50000
…	…	…

Some Functions in the SELECT Statement

- AVG calculates average
- COUNT counts # of row returned by the query
- MIN returns the minimum value of an expression
- MAX returns the maximum value of an expression
- MOD returns the remainder of a division operation
- SUM calculates a sum
- ROUND rounds up a number to specific # of decimal places
- TOP finds the top of n values in a set
- LOWER changes to all lower case
- UPPER changes to all upper case
- CONCAT concatenates strings
- SUBSTR isolates a substring
- NEXT_DAY computes the next day in sequence
- MONTHS_BETWEEN calculates the number of months between two dates

Example: Using a Function in SELECT Statement

Using the **COUNT** *aggregate function* to find totals numbers:

```
SELECT COUNT(*) FROM Orderline_T
    WHERE OrderID = 1004;
```

Note:

- COUNT(*) counts all rows selected by a query regardless of whether any rows contain NULL values.
- COUNT(columnName) tallies only the rows that contain values.

Caution: with aggregate functions you cannot have single-valued columns included in the SELECT clause, unless they are included in the GROUP BY clause.

Wrong!

```
SELECT ProductID, COUNT (*)
    FROM OrderLine_T
        WHERE OrderID=1004;
```

However, in Oracle, you are allowed to do so when you use aggregate functions in a subquery, e.g.,

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

Checking Missing Values
(Understand your data before you take action)

- **IS NULL,** e.g.

```
SELECT * FROM Customer_T
    WHERE CoustomerPostalCode IS NULL;
```

- **IS NOT NULL,** e.g.

```
SELECT * FROM Customer_T
  WHERE CoustomerPostalCode IS NOT NULL;
```

SELECT Example–Boolean Operators

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

AND	Join two ore more conditions and returns results only when all conditions are true.
OR	Join two ore more conditions and returns results when any one condition is true.
NOT	Negates an expression.

The precedence of the operators:

NOT → AND → OR

But using of parentheses can make a difference.

SELECT Example–Boolean Operators

Example 1 (normal precedence of Boolean operators)

You try to find out all the desks or tables that are more expensive than $300 with the following query.

```
SELECT ProductDescription, ProductFinish, ProductStandardPrice
    FROM Product_T
        WHERE ProductDescription LIKE '%Desk'
            OR ProductDescription LIKE '%Table'
            AND ProductStandardPrice > 300;
```

Result:

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

But you did not exactly get what you wanted.

SELECT Example–Boolean Operators

Example 2 (parentheses override the normal precedence of Boolean operators)

Find out all the desks or tables that are more expensive than $300.

```
SELECT ProductDescription, ProductFinish, ProductStandardPrice
        FROM Product_T
                WHERE (ProductDescription LIKE '%Desk'
                        OR ProductDescription LIKE '%Table')
                        AND ProductStandardPrice > 300;
```

Result:

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

Ranges for Qualification in SELECT Statements

- **BETWEEN … AND**, e.g.

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

- **NOT BETWEEN … AND**, e.g.

```
SELECT ProductDescription, ProductStandardPrice
  FROM Product_T
   WHERE ProductStandardPrice NOT BETWEEN 200 AND 300
```

Using DISTINCT or ALL Keywords in SELECT Statements

- **DISTINCT**, for removing duplicate results, e.g.

```
SELECT DISTINCT OrderID
    FROM OrderLine_T;
```

- **ALL**, for returning all results regardless of redundancy, e.g.

```
SELECT ALL OrderID
    FROM OrderLine_T;
```

Using IN or NOT IN Clauses in SELECT Statements

- **IN**, for judging the membership of a list, e.g.

```
SELECT CustomerName, CustomerCity, CustomerState
    FROM Customer_T
      WHERE CustomerState IN ('FL', 'TX', 'CA', 'HI');
```

- **NOT IN**, for judging the non-membership of a list, e.g.

```
SELECT CustomerName, CustomerCity, CustomerState
    FROM Customer_T
      WHERE CustomerState NOT IN ('FL', 'TX', 'CA', 'HI');
```

Sorting Results with the ORDER BY Clause in a SELECT Statement

```
SELECT CustomerName, CustomerCity, CustomerState
    FROM Customer_T
        WHERE CustomerState IN ('FL', 'TX', 'CA', 'HI')
            ORDER BY CustomerState, CustomerName;
```

Sorted first — Sorted second

Result:

CUSTOMERNAME	CUSTOMERCITY	CUSTOMERSTATE
California Classics	Santa Clara	CA
Impressions	Sacramento	CA
Contemporary Casual	Gainesville	FL
...	...	...

You can sort the result in ascending order (the default) or descending order, e.g.,

```
... ORDER BY CustomerState DESC;
... ORDER BY CustomerState ASC;
```

Using Column Position(s) in the ORDER BY Clause to Represent the Column Names

The following two SELECT statements are equivalent:

```
SELECT CustomerName, CustomerCity, CustomerState
        FROM Customer_T
          WHERE CustomerState IN ('FL', 'TX', 'CA', 'HI')
                ORDER BY CustomerState, CustomerName;
```

```
SELECT CustomerName, CustomerCity, CustomerState
        FROM Customer_T
          WHERE CustomerState IN ('FL', 'TX', 'CA', 'HI')
                ORDER BY 3, 1;
```

Categorizing Results Using the GROUP BY Clause in a SELECT Statement

It is useful when the GROUP BY clause is paired with aggregate functions (AVG, SUM, COUNT, …) to provide summary information for the group:

- *Scalar aggregate*: single value returned from a SQL query with an aggregate function
- *Vector aggregates:* multiple values returned from a SQL query with an aggregate function (via GROUP BY)

```
SELECT CustomerState, COUNT(CustomerState)
 FROM Customer_T
  GROUP BY CustomerState;
```

You can also nest a group within a group:

```
SELECT CustomerState, CustomerCity, COUNT(CustomerCity)
 FROM Customer_T
  GROUP BY CustomerState, CustomerCity;
```

Caution: When you use the GROUP BY

- The columns that are allowed to be specified in the SELECT clause are limited.
- Each column referenced in the SELECT clause must also be referenced in the GROUP BY clause, unless the column is an argument for an aggregate function in the SELECT clause.

Using the HAVING Clause with the GROUP BY Clause in a SELECT Statement

For qualifying Results by Categories

```
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. In this example, only those groups with total numbers of CustomerState greater than 1 will be included in the final result.

A More Complicated Query Using the HAVING, GROUP BY and ORDER BY Clauses

Find the product finish of Cherry, Natural Ash, Natural Maple, or White Ash and the average price of the products of that finish is less than $750.

```
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;
```

Note: the clauses must be used in the listed order (refer to the SQL processing order covered earlier)

Views

- **Base Table**: a table containing the raw data, correspondent to a relation in the logical DB design.
- **View (Dynamic View, Virtual Table)**
  - A "virtual table" created dynamically upon request by a user.
  - No data actually is stored, instead data is drown from a base table(s) made available to user.
  - Based on SQL SELECT statement on base table(s) or even a view(s).
- **Materialized View**
  - Stored copy of the result of the SQL query of the view definition.
  - Exists as a table.
  - Must be refreshed periodically to match the corresponding base table(s) and/or view(s).

CREATE VIEW or MATERIALIZED VIEW

```
CREATE VIEW Invoice_V AS
  SELECT c.CustomerID, CustomerAddress, o.OrderID, p.ProductID,
  ProductStrandardPrice, OrderedQuantity
      FROM Customer_T c, Order_T o, OrderLine_T ol, Product_T p
      WHERE c.CustomerID=o.CustomerID
            AND o.OrderID=ol.OrderID
            AND p.ProductID=ol.ProductID
```

```
CREATE MATERIALIZED VIEW Sales_MV
          BUILD IMMEDIATE REFRESH FAST ON COMMIT
          AS SELECT t.calendar_year, p.prod_id,
          SUM(s.amount_sold) AS sum_sales
          FROM times t, products p, sales s
          WHERE t.time_id = s.time_id AND p.prod_id = s.prod_id
          GROUP BY t.calendar_year, p.prod_id;
```

Advantages of Views

- Simplify query commands.
- Assist with data security (but don't just rely on views for security; there are more important security measures) and confidentiality.
- 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. If the view definition contains any of the following, the view cannot be used to update the data:
  - The DISTINCT keyword is used in the SELECT clause.
  - The SELECT clause contains an expression or a function.
  - More than one table is reference in the query.
  - A non-updatable view is referenced.
  - The GROUP BY or HAVING clause is used.

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.