INFORMATION SYSTEM DEPARTMENT

INFORMATION TECHNOLOGY FACULTY– HCM UNIVERSITY SCIENCE

INTRODUCTION TO DATABASE

Chapter 03

Structured Query Language (SQL)

Lecturer- PhD. NGUYEN TRAN MINH THU

KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Outline

- SQL Overview
- Basic SQL
  - SQL Data Definition and Data Types
  - Specifying Constraints in SQL
  - Insert/Update/Delete Statement in SQL
  - Basic Retrieval Queries in SQL
- Advanced SQL
  - Complex SQL Retrieval Queries
  - View
  - Ensuring Transaction Integrity
    - Specifying Constraints as Assertions & Action Trigger
- Tips for Developing Queries

Outline

- SQL Overview
- Basic SQL
  - SQL Data Definition and Data Types
  - Specifying Constraints in SQL
  - Insert/Update/Delete Statement in SQL
  - Basic Retrieval Queries in SQL
- Advanced SQL
  - Complex SQL Retrieval Queries
  - View
  - Ensuring Transaction Integrity
    - Specifying Constraints as Assertions & Action Trigger
- Tips for Developing Queries

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. F. 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
- 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
- 1989, 1992, 1999, 2003, 2006, 2008–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 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)

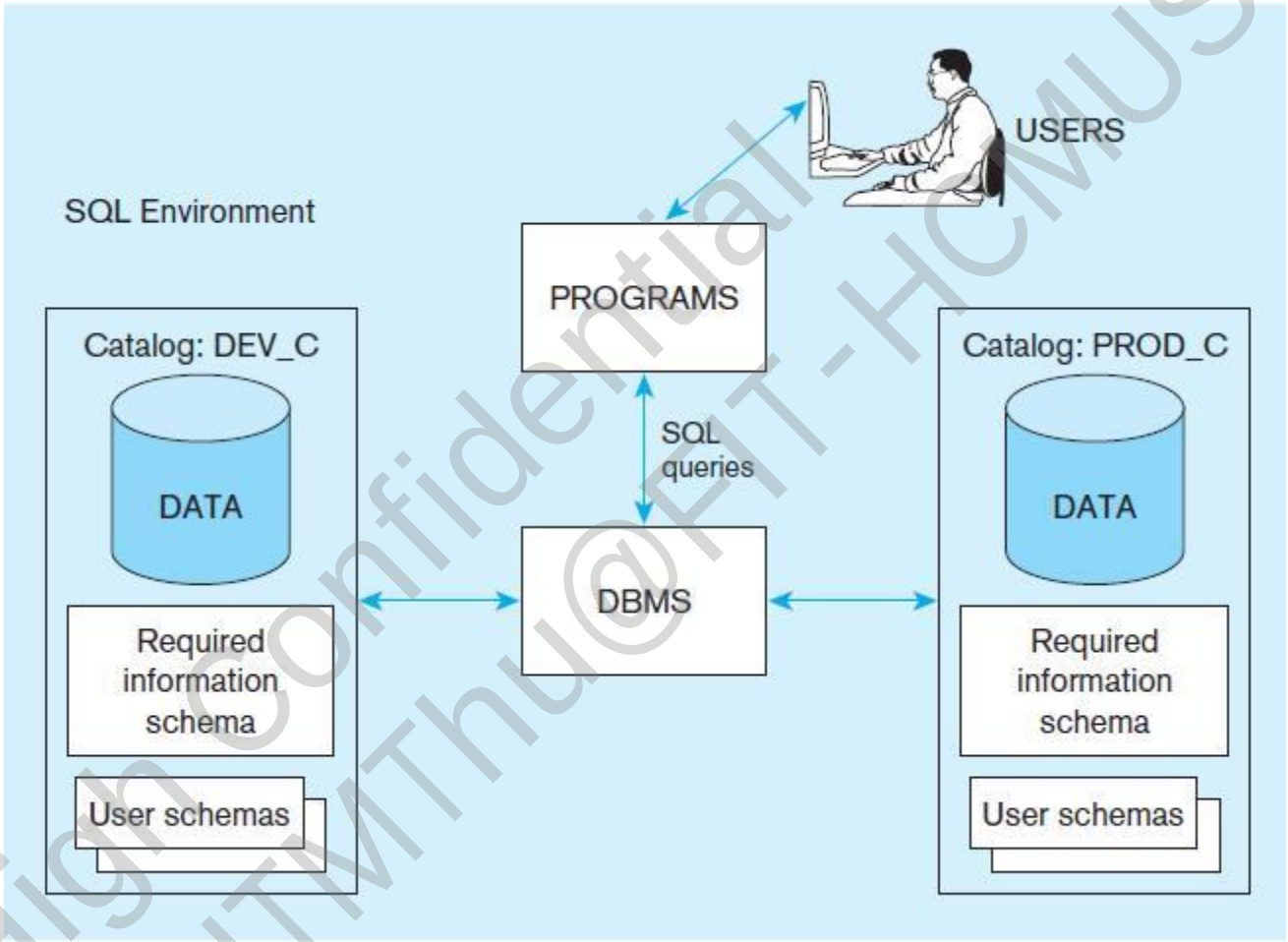
Benefits of a Standardized Relational Language

- 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

A simplified schematic of a typical SQL environment, as described by the SQL: 2008 standard



Outline

- SQL Overview
- Basic SQL
  - SQL Data Definition and Data Types
  - Specifying Constraints in SQL
  - Insert/Update/Delete Statement in SQL
  - Basic Retrieval Queries in SQL
- Advanced SQL
  - Complex SQL Retrieval Queries
  - View
  - Ensuring Transaction Integrity
    - Specifying Constraints as Assertions & Action Trigger
- Tips for Developing Queries

SQL Data Types

❑ Number

- INTEGER
- SMALLINT
- NUMERIC, NUMERIC(p), NUMERIC(p,s)
- DECIMAL, DECIMAL(p), DECIMAL(p,s)
- REAL
- DOUBLE PRECISION
- FLOAT, FLOAT(p)

❑ Character String

- CHARACTER or CHAR
- CHARACTER(n) or CHAR (n)
- CHARACTER VARYING(n) or VARCHAR(n)
- NATIONAL CHARACTER (n) orNCHAR(n)

SQL Data Types

❑ Bit string

- BIT, BIT(x)
- BIT VARYING(x)

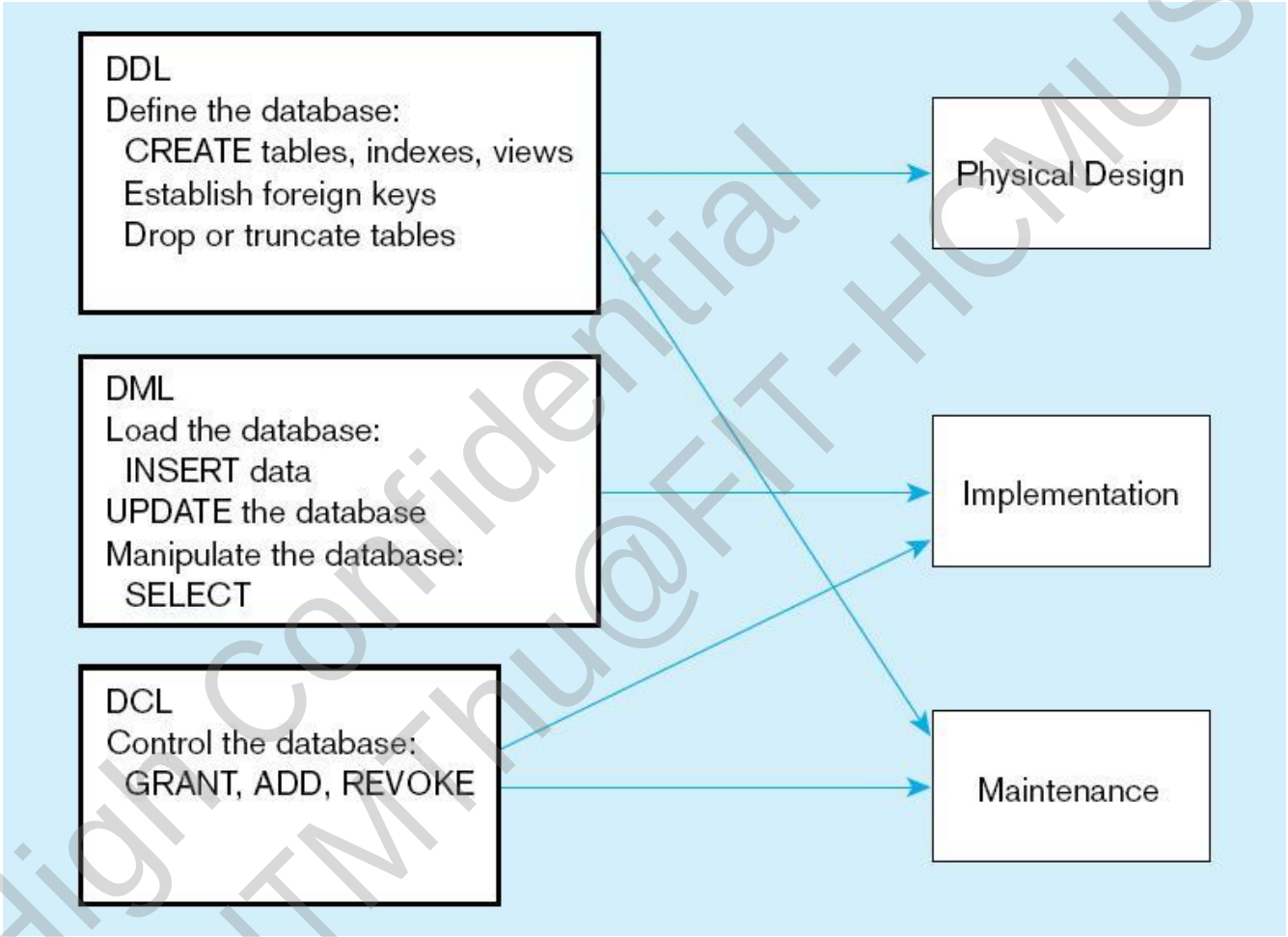
❑ Datetime

- DATE (day, month, year)
- TIME (hour, minute, second)
- TIMESTAMP (Date & Time)
- DATETIME: DATE and TIME (only in SQLof SQL Server)

❑ Boolean

- Store truth values: TRUE, FALSE & UNKNOW

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

Create Statements

- SYNTAX FOR CREATE TABLE

```
CREATE TABLE table_name (
    column1 <datatype> [<column_constraint],
    column2 <datatype> [<column_constraint],
    column3 <datatype> [<column_constraint],
   ....
);
```


Introduction to Database

©2020 Information System Department

16

Outline

- SQL Overview
- Basic SQL
  - SQL Data Definition and Data Types
  - Specifying Constraints in SQL
  - Insert/Update/Delete Statement in SQL
  - Basic Retrieval Queries in SQL
- Advanced SQL
  - Complex SQL Retrieval Queries
  - View
  - Ensuring Transaction Integrity
    - Specifying Constraints as Assertions & Action Trigger
- Tips for Developing Queries

Specifying Constraints in SQL

- **Column Constraints**
  - NOT NULL
  - NULL
  - UNIQUE
  - DEFAULT
  - PRIMARY KEY
  - FOREIGN KEY / REFERENCES
  - CHECK
- **Naming for column constraint**

```
CONSTRAINT <Name_Constraint>  <constraint>
```

Create Statements

```
CREATE TABLE EMPLOYEE
        ( Fname          VARCHAR(15)      NOT NULL,
          Minit          CHAR,
          Lname          VARCHAR(15)      NOT NULL,
          Ssn            CHAR(9)          NOT NULL,
          Bdate          DATE,
          Address        VARCHAR(30),
          Sex            CHAR,
          Salary         DECIMAL(10,2),
          Super_ssn      CHAR(9),
          Dno            INT              NOT NULL,
        PRIMARY KEY (Ssn),
```

```
CREATE TABLE DEPARTMENT
        ( Dname          VARCHAR(15)      NOT NULL,
          Dnumber        INT              NOT NULL,
          Mgr_ssn        CHAR(9)          NOT NULL,
          Mgr_start_date DATE,
        PRIMARY KEY (Dnumber),
        UNIQUE (Dname),
        FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn) );
```

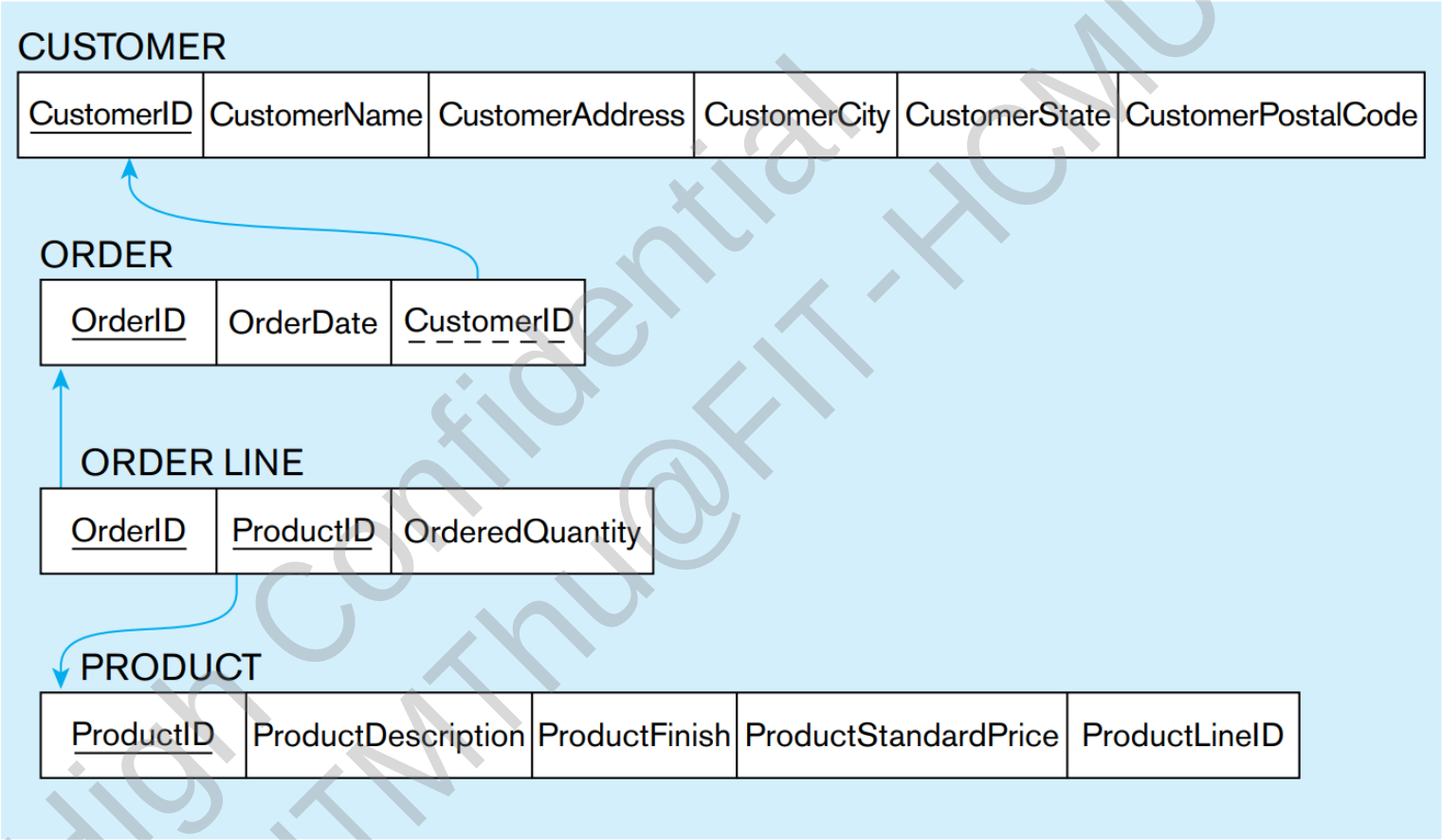
```
CREATE TABLE DEPT_LOCATIONS
        ( Dnumber        INT              NOT NULL,
          Dlocation      VARCHAR(15)      NOT NULL,
        PRIMARY KEY (Dnumber, Dlocation),
        FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber) );
```

```
CREATE TABLE PROJECT
        ( Pname          VARCHAR(15)      NOT NULL,
          Pnumber        INT              NOT NULL,
          Plocation      VARCHAR(15),
          Dnum           INT              NOT NULL,
        PRIMARY KEY (Pnumber),
        UNIQUE (Pname),
        FOREIGN KEY (Dnum) REFERENCES DEPARTMENT(Dnumber) );
```

```
CREATE TABLE DEPENDENT
        ( Essn           CHAR(9)          NOT NULL,
          Dependent_name VARCHAR(15)      NOT NULL,
          Sex            CHAR,
          Bdate          DATE,
          Relationship   VARCHAR(8),
        PRIMARY KEY (Essn, Dependent_name),
        FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn) );
```

```
CREATE TABLE WORKS_ON
        ( Essn           CHAR(9)          NOT NULL,
          Pno            INT              NOT NULL,
          Hours          DECIMAL(3,1)     NOT NULL,
        PRIMARY KEY (Essn, Pno),
        FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn),
        FOREIGN KEY (Pno) REFERENCES PROJECT(Pnumber) );
```

The following slides create tables for this enterprise data model



SQL Overall table definitions

```
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));
```

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 primary key

Non-nullable specifications

```
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));
```

Primary key

Some primary keys are composite– composed of multiple attributes

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));
```

Default value

Domain 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        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));
```

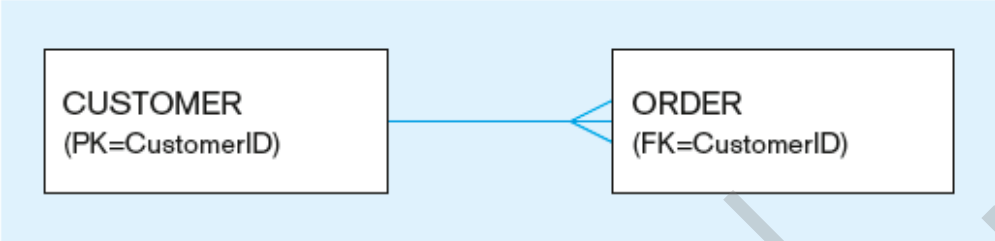
Primary key of parent table

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

Ensuring data integrity through updates



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);
```

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

Specifying Constraints in SQL

```
CREATE TABLE EMPLOYEE (
Fname  VARCHAR(15)   NOT NULL,
  …  ,
Dno INT NOT NULL DEFAULT 1,
CONSTRAINT EMPPK PRIMARY KEY (Ssn),
CONSTRAINT EMPSUPERFK
      FOREIGN KEY (Super_ssn) REFERENCES EMPLOYEE(Ssn)
      ON DELETE SET NULL ON UPDATE CASCADE,
CONSTRAINT EMPDEPTFK
      FOREIGN KEY(Dno) REFERENCES DEPARTMENT(Dnumber)
      ON DELETE SET DEFAULT ON UPDATE CASCADE)
```

Specifying Key and Referential Integrity Constraints

Schema Change Statements in SQL

- **DROP Command**
  - **DROP SCHEMA**: drop a whole schema
  - **DROP TABLE:** remove tables from schema
  - There are two drop behavior options: CASCADE and RESTRICT

```
DROP SCHEMA COMPANY CASCADE;

DROP TABLE DEPENDENT CASCADE;;
```

Schema Change Statements in SQL

- **The ALTER Command**
  - **Alter table actions** include adding or dropping a column (attribute), changing a column definition, and adding or dropping table constraints

```
ALTER TABLE COMPANY.EMPLOYEE ADD COLUMN Job VARCHAR(12);
```

```
ALTER TABLE COMPANY.EMPLOYEE DROP COLUMN Address CASCADE;
```

```
ALTER TABLE COMPANY.DEPARTMENT ALTER COLUMN Mgr_ssn
DROP DEFAULT;
```

```
ALTER TABLE COMPANY.DEPARTMENT ALTER COLUMN Mgr_ssn
SET DEFAULT '333445555';
```

```
ALTER TABLE COMPANY.EMPLOYEE
DROP CONSTRAINT EMPSUPERFK CASCADE;
```

Outline

- SQL Overview
- Basic SQL
  - SQL Data Definition and Data Types
  - Specifying Constraints in SQL
  - Insert/Update/Delete Statement in SQL
  - Basic Retrieval Queries in SQL
- Advanced SQL
  - Complex SQL Retrieval Queries
  - View
  - Ensuring Transaction Integrity
    - Specifying Constraints as Assertions & Action Trigger
- Tips for Developing Queries

INSERT, DELETE, and UPDATE Statements in SQL

- The INSERT Command: Adds one or more rows to a table
  - Inserting into a table

```
INSERT INTO EMPLOYEE
VALUES ( 'Richard', 'K', 'Marini', '653298653',
         '1962-12-30', '98 Oak Forest, Katy, TX',
         'M',37000, '653298653', 4 );
```

Inserting a record that has some null attributes requires identifying the fields that actually get data

```
INSERT INTO EMPLOYEE (Fname, Lname, Dno, Ssn)
VALUES ('Richard', 'Marini', 4, '653298653');
```

INSERT, DELETE, and UPDATE Statements in SQL

- The INSERT Command: Adds one or more rows to a table
  - Inserting from another table

```
CREATE TABLE WORKS_ON_INFO
( Emp_name VARCHAR(15),
  Proj_name VARCHAR(15),
  Hours_per_week DECIMAL(3,1));

INSERT INTO WORKS_ON_INFO
SELECT E.Lname, P.Pname, W.Hours
FROM PROJECT P, WORKS_ON W, EMPLOYEE E
WHERE P.Pnumber = W.Pno AND W.Essn = E.Ssn;
```

INSERT, DELETE, and UPDATE Statements in SQL

- The DELETE Command: Removes rows from a table
  - Delete certain rows

```
DELETE FROM EMPLOYEE
WHERE Lname = 'Brown';

DELETE FROM EMPLOYEE
WHERE Ssn = '123456789';

DELETE FROM EMPLOYEE
WHERE Dno = 5;
```

  - Delete all rows

```
DELETE FROM EMPLOYEE
```

INSERT, DELETE, and UPDATE Statements in SQL

- Update Statement: Modifies data in existing rows

```
UPDATE PROJECT
SET Plocation = 'Bellaire', Dnum = 5
WHERE Pnumber = 10;

UPDATE EMPLOYEE
SET Salary = Salary * 1.1
WHERE Dno = 5;
```

Outline

- SQL Overview
- Basic SQL
  - SQL Data Definition and Data Types
  - Specifying Constraints in SQL
  - Insert/Update/Delete Statement in SQL
  - Basic Retrieval Queries in SQL
- Advanced SQL
  - Complex SQL Retrieval Queries
  - View
  - Ensuring Transaction Integrity
    - Specifying Constraints as Assertions & Action Trigger

Basic Retrieval Queries in SQL

- The SELECT-FROM-WHERE Structure of Basic SQL Queries

```
SELECT <attribute list>
FROM <table list>
WHERE <condition>;
```

where

- <attribute list> is a list of attribute names whose values are to be retrieved by the query.
- <table list> is a list of the relation names required to process the query.
- <condition> is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query

Logical comparison operators: =, <, <=, >, >=, and <>

Basic Retrieval Queries in SQL

- Query 0. Retrieve the birth date and address of the employee(s) whose name is 'John B. Smith'

```
SELECT Bdate, Address
FROM EMPLOYEE
WHERE Fname = 'John' AND Minit = 'B' AND
Lname ='Smith';
```

Bdate	Address
1965-01-09	731Fondren, Houston, TX

Basic Retrieval Queries in SQL

- Query 1. Retrieve the name and address of all employees who work for the 'Research' department

```
SELECT Fname, Lname, Address
FROM EMPLOYEE, DEPARTMENT
WHERE Dname = 'Research'
AND Dnumber = Dno;
```

Fname	Lname	Address
John	Smith	731 Fondren, Houston, TX
Franklin	Wong	638 Voss, Houston, TX
Ramesh	Narayan	975 Fire Oak, Humble, TX
Joyce	English	5631 Rice, Houston, TX

Basic Retrieval Queries in SQL

- Query 2. For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birth date

```
SELECT Pnumber, Dnum, Lname, Address, Bdate
FROM PROJECT, DEPARTMENT, EMPLOYEE
WHERE Dnum = Dnumber AND Mgr_ssn = Ssn AND
Plocation = 'Stafford
```

Pnumber	Dnum	Lname	Address	Bdate
10	4	Wallace	291Berry, Bellaire, TX	1941-06-20
30	4	Wallace	291Berry, Bellaire, TX	1941-06-20

Basic Retrieval Queries in SQL

- Ambiguous Attribute Names, Aliasing, Renaming, and Tuple Variables.

```
SELECT Fname, Name, Address
FROM EMPLOYEE, DEPARTMENT
WHERE Name = 'Research' AND Dnumber = Dnumber;

SELECT E.Fname, E.Name, E.Address
FROM EMPLOYEE E, DEPARTMENT D
WHERE D.Name = 'Research' AND D.Dnumber = E.Dnumber;

SELECT E.Fname, E.Lname, S.Fname, S.Lname
FROM EMPLOYEE AS E, EMPLOYEE AS S
WHERE E.Super_ssn = S.Ssn;
```

Basic Retrieval Queries in SQL

❑ Unspecified WHERE Clause and Use of the Asterisk

■ A missing WHERE clause indicates no condition on tuple selection; hence, all tuples of the relation specified in the FROM clause qualify and are selected for the query result.

```
SELECT Ssn
FROM EMPLOYEE;
```

■ If more than one relation is specified in the FROM clause and there is no WHERE clause, then the CROSS PRODUCT—all possible tuple combinations—of these relations is selected.

```
SELECT Ssn, Dname
FROM EMPLOYEE, DEPARTMENT;
```

Basic Retrieval Queries in SQL

❑ Tables as Sets in SQL

■ Duplicate elimination is an expensive operation. One way to implement it is to sort the tuples first and then eliminate duplicates.

■ The user may want to see duplicate tuples in the result of a query.

■ When an aggregate function is applied to tuples, in most cases we do not want to eliminate duplicates.

```
SELECT Salary
FROM EMPLOYEE;
```

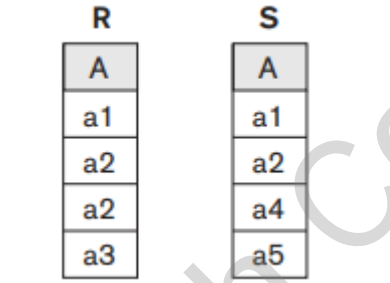
Salary
30000
40000
25000
43000
38000
25000
25000
55000

```
SELECT DISTINCT Salary
FROM EMPLOYEE;
```

Salary
30000
40000
25000
43000
38000
55000

Basic Retrieval Queries in SQL

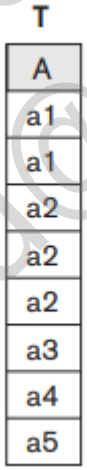
- Set union (UNION), set difference (EXCEPT),and set intersection (INTERSECT) operations
  - SQL also has corresponding multiset operations, which are followed by the keyword ALL (UNION ALL, EXCEPT ALL, INTERSECT ALL). Their results are multisets (duplicates are not eliminated)



R
A
a1
a2
a2
a3

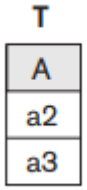
S
A
a1
a2
a4
a5

Two tables: R(A) and S(A)



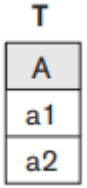
T
A
a1
a1
a2
a2
a2
a3
a4
a5

R(A)UNION ALL S(A)



T
A
a2
a3

R(A) EXCEPT ALL S(A)



T
A
a1
a2

R(A) INTERSECT ALL S(A)

Basic Retrieval Queries in SQL

- Query 4. Make a list of all project numbers for projects that involve an employee whose last name is 'Smith', either as a worker or as a manager of the department that controls the project

```
(SELECT DISTINCT Pnumber
 FROM PROJECT, DEPARTMENT, EMPLOYEE
 WHERE Dnum = Dnumber AND Mgr_ssn = Ssn
 AND Lname = 'Smith')
UNION
(SELECT DISTINCT Pnumber
 FROM PROJECT, WORKS_ON, EMPLOYEE
 WHERE Pnumber = Pno AND Essn = Ssn
 AND Lname = 'Smith');
```

Basic Retrieval Queries in SQL

- Substring Pattern Matching and Arithmetic Operators
  - **LIKE** comparison operator (string pattern matching):
    - % replaces an arbitrary number of zero or more characters
    - underscore (_) replaces a single character
    - ESCAPE (\): escape character
      - For example: 'AB_CD\%EF' ESCAPE '\' represents the literal string 'AB_CD%EF' because \ is specified as the escape character

Query: Retrieve all employees whose address is in Houston, Texas

```
SELECT Fname, Lname
FROM EMPLOYEE
WHERE Address LIKE '%Houston,TX%';
```

Query: retrieve all employees who were born during the 1970s

```
SELECT Fname, Lname
FROM EMPLOYEE
WHERE Bdate LIKE '_ _ 7 _ _ _ _ _ _ _ _';
```

Basic Retrieval Queries in SQL

- Another feature allows the use of arithmetic in queries.
  - The standard arithmetic operators for addition (+), subtraction (−), multiplication (*), and division (/) can be applied to numeric values or attributes with umeric domains
  - we can rename an attribute in the query result using AS in the SELECT clause

Query: Show the resulting salaries if every employee working on the 'ProductX' project is given a 10% raise

```
SELECT E.Fname, E.Lname, 1.1 * E.Salary AS Increased_sal
FROM EMPLOYEE AS E, WORKS_ON AS W, PROJECT AS P
WHERE E.Ssn = W.Essn AND W.Pno = P.Pnumber AND
P.Pname = 'ProductX';
```

Basic Retrieval Queries in SQL

- **Boolean Operators**: AND, OR, and NOT Operators for customizing conditions in WHERE clause
  - Boolean query a without use of parentheses

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

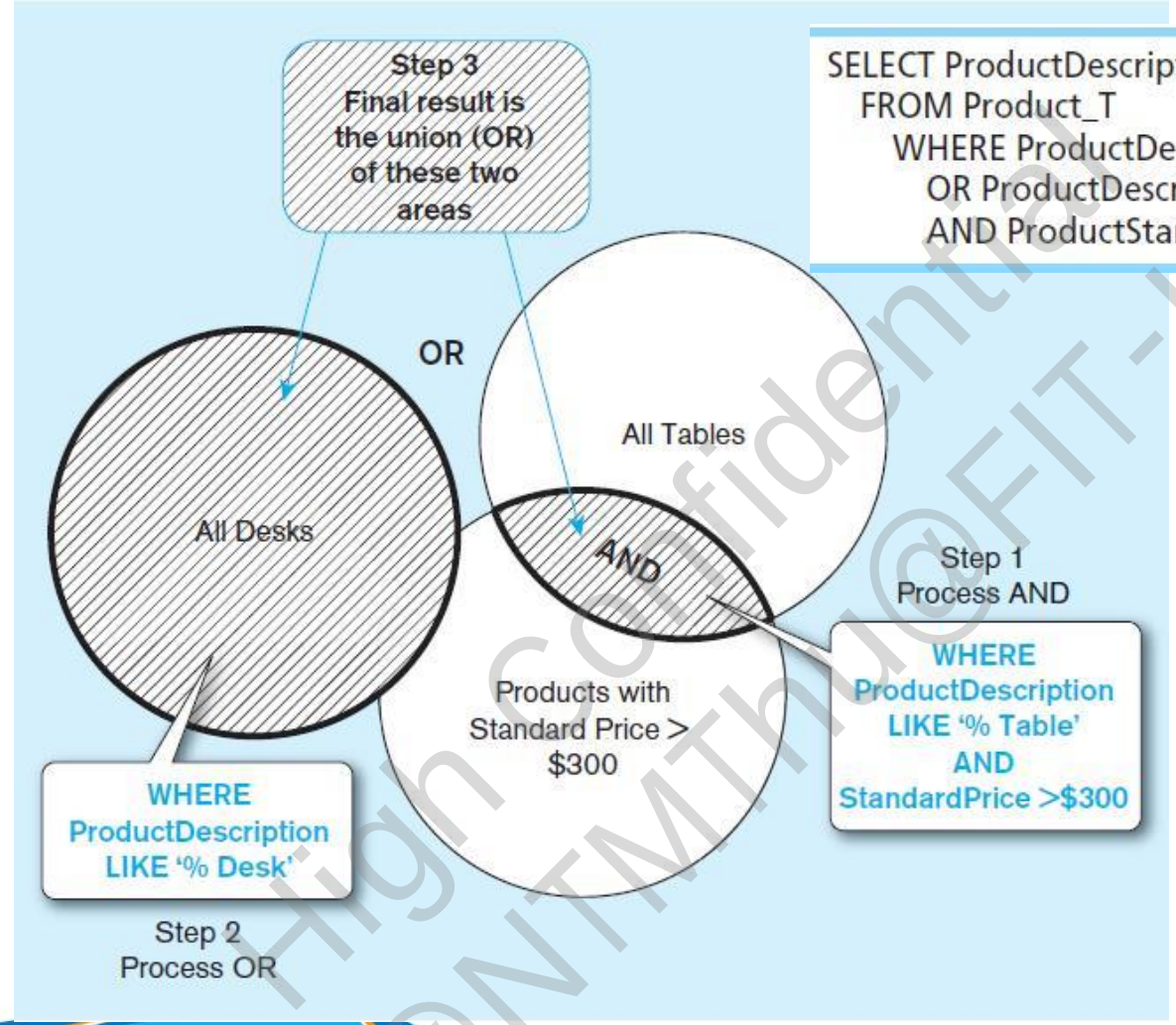
By default, processing order of Boolean operators is NOT, then AND, then OR

  - With parentheses:

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

These override the normal precedence of Boolean operators. this case parentheses make the OR take place before the AND

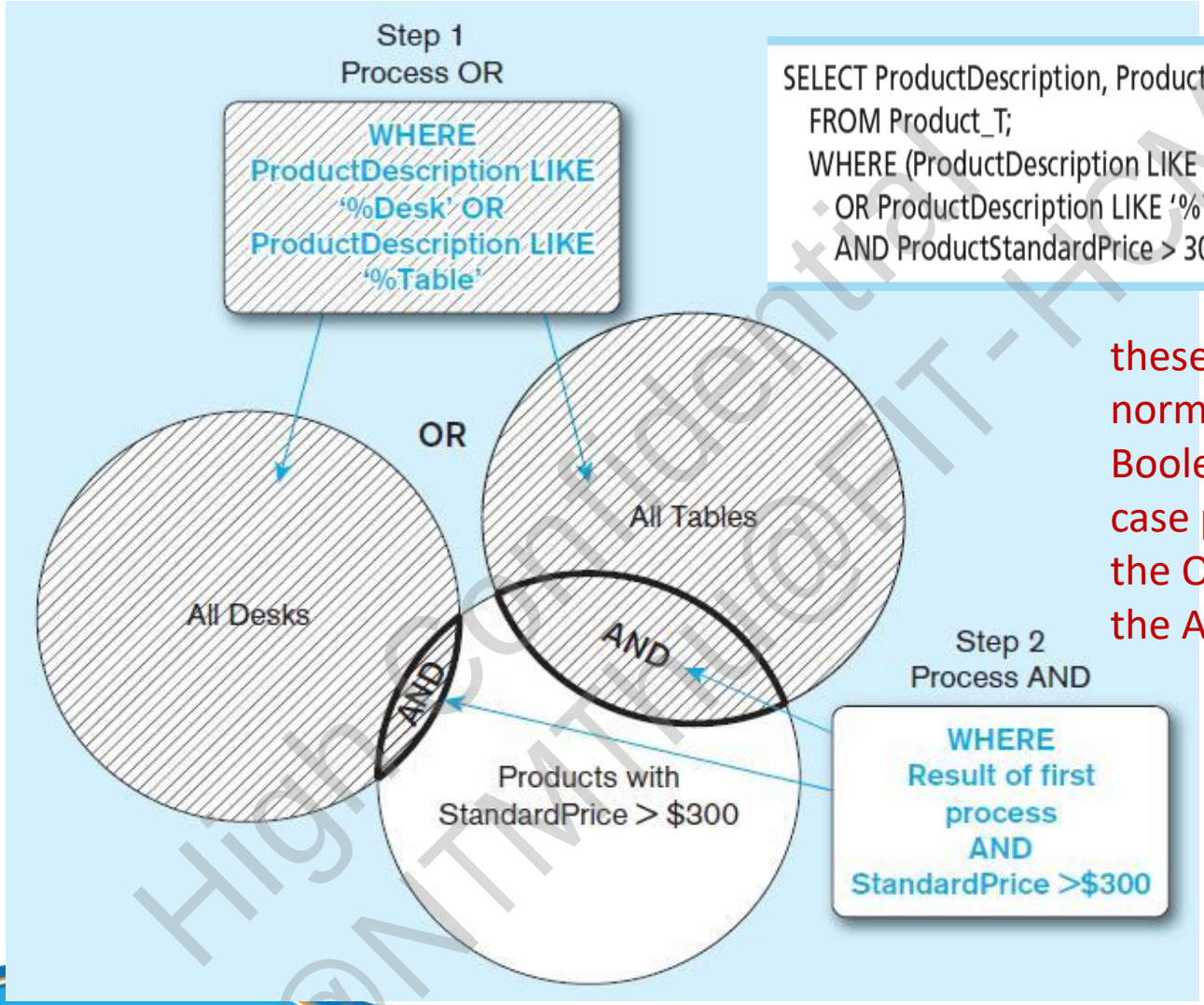
Boolean query A without use of parentheses




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

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

Boolean query B with use of parentheses




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

these override the normal precedence of Boolean operators. this case parentheses make the OR take place before the AND

Basic Retrieval Queries in SQL

- Comparison operator: **BETWEEN, NOT BETWEEN**

Query: Retrieve all employees in department 5 whose salary is between $30,000 and $40,000.

```
SELECT *
FROM EMPLOYEE
WHERE (Salary BETWEEN 30000 AND 40000) AND Dno = 5;
```

- Ordering of Query Results: **ORDER BY**

Query: Retrieve a list of employees and the projects they are working on, ordered by department and, within each department, ordered alphabetically by last name, then first name

```
SELECT D.Dname, E.Lname, E.Fname, P.Pname
FROM   DEPARTMENT AS D, EMPLOYEE AS E,
       WORKS_ON AS W,PROJECT AS P
WHERE  D.Dnumber = E.Dno AND E.Ssn = W.Essn
       AND W.Pno =P.Pnumber
ORDER BY D.Dname, E.Lname, E.Fname;
```

DESC:descending order
ASC: ascending order

Summary of Basic SQL Retrieval Queries

```
SELECT <attribute list>
FROM <table list>
[WHERE <condition> ]
[ORDER BY <attribute list> ];
```

Outline

- SQL Overview
- Basic SQL
  - SQL Data Definition and Data Types
  - Specifying Constraints in SQL
  - Insert/Update/Delete Statement in SQL
  - Basic Retrieval Queries in SQL
- Advanced SQL
  - Complex SQL Retrieval Queries
  - View
  - Ensuring Transaction Integrity
    - Specifying Constraints as Assertions & Action Trigger
- Tips for Developing Queries

More Complex SQL Retrieval Queries

- Comparisons Involving NULL

```
SELECT Fname, Lname
FROM EMPLOYEE
WHERE Super_ssn IS NULL;
```

```
SELECT Fname, Lname
FROM EMPLOYEE
WHERE Super_ssn IS NOT NULL;
```

More Complex SQL Retrieval Queries

- Multiset Comparisons:
  - IN/NOT IN: to match a list of values, consider using IN
  - EXISTS/NOT EXISTS
  - ALL
  - ANY/SOME

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (SELECT STATEMENT);
```

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS (SELECT STATEMENT);
```

More Complex SQL Retrieval Queries

- Set/Multiset Comparisons:

```
SELECT DISTINCT Pnumber
FROM PROJECT
WHERE Pnumber IN
      (SELECT Pnumber
       FROM PROJECT, DEPARTMENT, EMPLOYEE
       WHERE Dnum = Dnumber AND Mgr_ssn = Ssn AND
             Lname = 'Smith' )
     OR
     Pnumber IN
     (SELECT Pno
      FROM WORKS_ON, EMPLOYEE
      WHERE Essn = Ssn AND Lname = 'Smith');
```

More Complex SQL Retrieval Queries

- Set/Multiset Comparisons:

```
SELECT DISTINCT Essn
FROM WORKS_ON
WHERE Pno IN ( 1,2,3)
```

```
SELECT E.Fname, E.Name, E.Address
FROM   EMPLOYEE E
WHERE EXISTS
                (SELECT *
                 FORM DEPARTMENT D
                 WHERE E.Dno= D.Dnumber
                       AND D.Name = 'Research');
```

```
SELECT Lname, Fname
FROM EMPLOYEE
WHERE Salary > ALL ( SELECT Salary
                     FROM EMPLOYEE
                     WHERE Dno = 5 );
```

More Complex SQL Retrieval Queries

- Nested Queries:
  - **Noncorrelated**–executed once for the entire outer query
  - **Correlated**–executed once for each row returned by the outer query

(Outer query)

```
SELECT <danh sách các cột>
FROM <danh sách các bảng>
WHERE <so sánh tập hợp> (
        SELECT <danh sách các cột>
        FROM <danh sách các bảng>
        WHERE <điều kiện>)
```

(Subquery)

Correlated vs. Noncorrelated Subqueries

- **Noncorrelated subqueries:**
  - Do not depend on data from the outer query
  - Execute once for the entire outer query
- **Correlated subqueries:**
  - Make use of data from the outer query
  - Execute once for each row of the outer query
  - Can use the EXISTS operator

More Complex SQL Retrieval Queries

- Nested Queries:
  - **Noncorrelated**–executed once for the entire outer query

```
SELECT E.Fname, E.Name, E.Address
FROM   EMPLOYEE E
WHERE  E.Dno IN
            (SELECT D.Dnumber
             FORM DEPARTMENT D
             WHERE D.Name = 'Research');
```

The IN operator will test to see if the E.Dno value of a row is included in the list returned from the subquery

Subquery is embedded in parentheses. In this case it returns a list that will be used in the WHERE clause of the outer query

More Complex SQL Retrieval Queries

- Nested Queries:
  - **Noncorrelated**–executed once for the entire outer query

```
SELECT E.Fname, E.Name, E.Address
FROM   EMPLOYEE E
WHERE  E.SSN IN
            (SELECT W.ESSN
             FORM WORK_ON W);
```

The IN operator will test to see if the E.SSN value of a row is included in the list returned from the subquery

Subquery is embedded in parentheses. In this case it returns a list that will be used in the WHERE clause of the outer query

More Complex SQL Retrieval Queries

- Nested Queries:
  - **Correlated**–executed once for each row returned by the outer query

The EXISTS operator will return a TRUE value if the subquery resulted in a non-empty set, otherwise it returns a FALSE

```
SELECT E.Fname, E.Name, E.Address
FROM   EMPLOYEE E
WHERE EXISTS
            (SELECT *
             FROM WORK_ON W
             WHERE W.ESSN = E.SSN );
```

➔ A correlated subquery always refers to an attribute from a table referenced in the outer query

The subquery is testing for a value that comes from the outer query

More Complex SQL Retrieval Queries

- Nested Queries:
  - **Correlated**–executed once for each row returned by the outer query

```
SELECT E.Fname, E.Lname
FROM EMPLOYEE AS E
WHERE E.Ssn IN
            (SELECT D.Essn
             FROM DEPENDENT AS D
             WHERE E.Fname = D.Dependent_name
             AND E.Sex = D.Sex );
```

More Complex SQL Retrieval Queries

- Nested Queries:
  - **Correlated**–executed once for each row returned by the outer query

```
SELECT E.Fname, E.Lname
FROM EMPLOYEE AS E
WHERE EXISTS
                (SELECT *
                 FROM DEPENDENT AS D
                 WHERE E.Ssn = D.Essn
                 AND E.Sex = D.Sex
                 AND E.Fname = D.Dependent_name)
```

More Complex SQL Retrieval Queries

- Nested Queries:
  - **Correlated**–executed once for each row returned by the outer query

```
SELECT E.Fname, E.Lname
FROM EMPLOYEE AS E
WHERE NOT EXISTS
                (SELECT *
                 FROM DEPENDENT AS D
                 WHERE E.Ssn = D.Essn
                 AND E.Sex = D.Sex
                 AND E.Fname = D.Dependent_name)
```

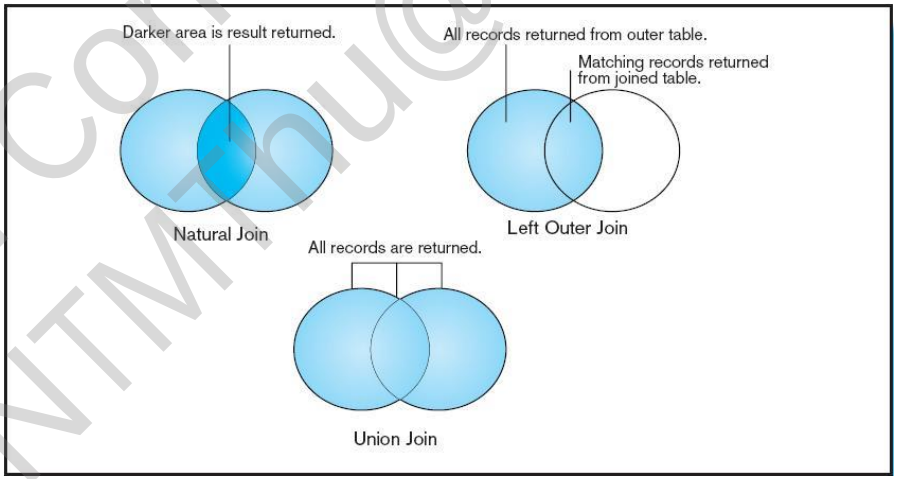
More Complex SQL Retrieval Queries

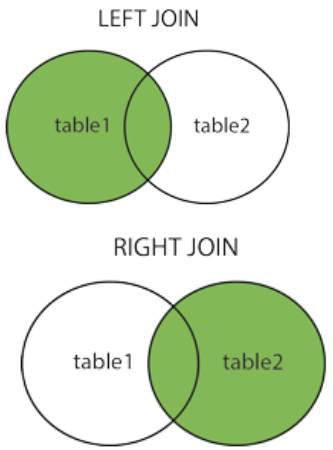
- Joined Tables in SQL and Outer Joins
  - **Join–**a relational operation that causes two or more tables with a common domain to be combined into a single table or view
  - **Equi-join**–a join in which the joining condition is based on equality between values in the common columns; common columns appear redundantly in the result table
  - **Natural join**–an equi-join in which one of the duplicate columns is eliminated in the result table

The common columns in joined tables are usually the primary key of the dominant table and the foreign key of the dependent table in 1:M relationships.

More Complex SQL Retrieval Queries

- Joined Tables in SQL and Outer Joins
  - **Outer join (Left/Right Join)**–a join in which rows that do not have matching values in common columns are nonetheless included in the result table (as opposed to inner join, in which rows must have matching values in order to appear in the result table)
  - **Union join/Cross join**–includes all columns from each table in the join, and an instance for each row of each table





More Complex SQL Retrieval Queries

- Joined Tables in SQL and Outer Joins

```
SELECT Fname, Lname, Address
FROM (EMPLOYEE [inner] JOIN DEPARTMENT
      ON Dno = Dnumber)
WHERE Dname = 'Research';
```

```
SELECT Fname, Lname, Address
FROM (EMPLOYEE NATURAL JOIN
(DEPARTMENT AS DEPT (Dname, Dno, Mssn, Msdate)))
WHERE Dname = 'Research';
```

More Complex SQL Retrieval Queries

- Joined Tables in SQL and Outer Joins

```
SELECT Pnumber, Dnum, Lname, Address, Bdate
FROM ((PROJECT JOIN DEPARTMENT ON Dnum = Dnumber)
      JOIN EMPLOYEE ON Mgr_ssn = Ssn)
WHERE Plocation = 'Stafford';
```

```
SELECT E.Lname AS Employee_name,
       S.Lname AS Supervisor_name
FROM (EMPLOYEE AS E LEFT OUTER JOIN EMPLOYEE AS S
      ON E.Super_ssn = S.Ssn);
```

More Complex SQL Retrieval Queries

Aggregate Functions in SQL

Query 19. Find the sum of the salaries of all employees, the maximum salary, the minimum salary, and the average salary

```
SELECT SUM (Salary), MAX (Salary),
       MIN (Salary), AVG (Salary)
FROM EMPLOYEE;
```

```
SELECT SUM (Salary) AS Total_Sal,
       MAX (Salary) AS Highest_Sal,
       MIN (Salary) AS Lowest_Sal,
       AVG (Salary) AS Average_Sal
FROM EMPLOYEE;
```

More Complex SQL Retrieval Queries

❑ Aggregate Functions in SQL

Query 20: Find the sum of the salaries of all employees of the 'Research' department, as well as the maximum salary, the minimum salary, and the average salary in this department

```
SELECT SUM (Salary), MAX (Salary), MIN (Salary),
AVG (Salary)
FROM (EMPLOYEE JOIN DEPARTMENT ON Dno = Dnumber)
WHERE Dname = 'Research';
```

Query 21: Retrieve the total number of employees in the company

```
SELECT COUNT (*)
FROM EMPLOYEE;
```

Query 22: The number of employees in the 'Research' department

```
SELECT COUNT (*)
FROM EMPLOYEE, DEPARTMENT
WHERE DNO = DNUMBER
      AND DNAME = 'Research';
```

More Complex SQL Retrieval Queries

❑ Aggregate Functions in SQL

Query 23. Count the number of distinct salary values in the database

```
SELECT COUNT (DISTINCT Salary)
FROM EMPLOYEE;
```

Query 24: Retrieve the names of all employees who have two or more dependents

```
SELECT Lname, Fname
FROM EMPLOYEE
WHERE ( SELECT COUNT (*)
        FROM DEPENDENT
        WHERE Ssn = Essn ) > = 2;
```

More Complex SQL Retrieval Queries

❑ Grouping: The GROUP BY and HAVING Clauses

Query 25. For each department, retrieve the department number, the number of employees in the department, and their average salary.

```
SELECT Dno, COUNT (*), AVG (Salary)
FROM EMPLOYEE
GROUP BY Dno;
```

Query 26. For each project, retrieve the project number, the project name, and the number of employees who work on that project.

```
SELECT Pnumber, Pname, COUNT (*)
FROM PROJECT, WORKS_ON
WHERE Pnumber = Pno
GROUP BY Pnumber, Pname;
```

More Complex SQL Retrieval Queries

❑ Grouping: The GROUP BY and HAVING Clauses

Query 27. For each project on which more than two employees work, retrieve the project number, the project name, and the number of employees who work on.

```
SELECT Pnumber, Pname, COUNT (*)
FROM PROJECT, WORKS_ON
WHERE Pnumber = Pno
GROUP BY Pnumber, Pname
HAVING COUNT (*) > 2;
```

DIVISION Operation in SQL

- R÷S

R	A	B	C	D	E
	α	a	α	a	1
	α	a	γ	a	1
	α	a	γ	b	1
	β	a	γ	a	1
	β	a	γ	b	3
	γ	a	γ	a	1
	γ	a	γ	b	1
	γ	a	β	b	1

S	D	E
b_i	a	1
	b	1

R÷S	A	B	C
a_i	α	a	γ
	γ	a	γ

DIVISION Operation in SQL

- **Using EXCEPT**

```
SELECT R1.A, R1.B, R1.C
FROM R R1
WHERE NOT EXISTS (
                    ( SELECT S.D, S.E  FROM S)
                    EXCEPT
                    ( SELECT R2.D, R2.E
                    FROM R R2
                    WHERE R1.A=R2.A AND R1.B=R2.B
                    AND R1.C=R2.C )
                )
```

DIVISION Operation in SQL

- **Using NOT EXISTS**

```
SELECT R1.A, R1.B, R1.C
FROM R R1
WHERE NOT EXISTS (
        SELECT *
        FROM S
        WHERE NOT EXISTS (
                SELECT *
                FROM R R2
                WHERE R2.D=S.D AND R2.E=S.E
                AND R1.A=R2.A AND R1.B=R2.B AND R1.C=R2.C ))
```

DIVISION Operation in SQL

- **Using Count Function**

```
SELECT R.A
FROM R
[WHERE R.B IN (SELECT S.B FROM S   [WHERE <ĐK>]]
GROUP BY R.A
HAVING COUNT(DISTINCT R.B) = ( SELECT COUNT(S.B)
                               FROM S
                               [WHERE <ĐK>])
```

DIVISION Operation in SQL

- **Using EXCEPT**

Query: Find the names of employees who work on all the projects controlled by department number 5

```
SELECT E.Fname, E.Lname
FROM EMPLOYEE E ,WORK_ON W1,
WHERE E.Ssn = W1.Essn
        AND NOT EXISTS (
                (SELECT P.Pnumber
                 FROM PROJECT   P
                 WHERE  P.Dnum = 5)
                 EXCEPT
                ( SELECT W2.Pno
                  FROM WORK_ON W2
                  WHERE W1.SSN=W2.Essn)
                )
```

DIVISION Operation in SQL

- **Using EXISTS**

Query: Find the names of employees who work on all the projects controlled by department number 5

```
SELECT E.Fname, E.Lname
FROM EMPLOYEE E ,WORK_ON W1,
WHERE E.Ssn = W1.Essn
        AND NOT EXISTS ( SELECT *
                         FROM PROJECT   P
                         WHERE  P.Dnum = 5  AND
                          NOT EXISTS   ( SELECT *
                                         FROM WORK_ON W2
                                         WHERE W1.ESsn=W2.Essn
                                          AND P.Dnum = W2.Pno ))
```

DIVISION Operation in SQL

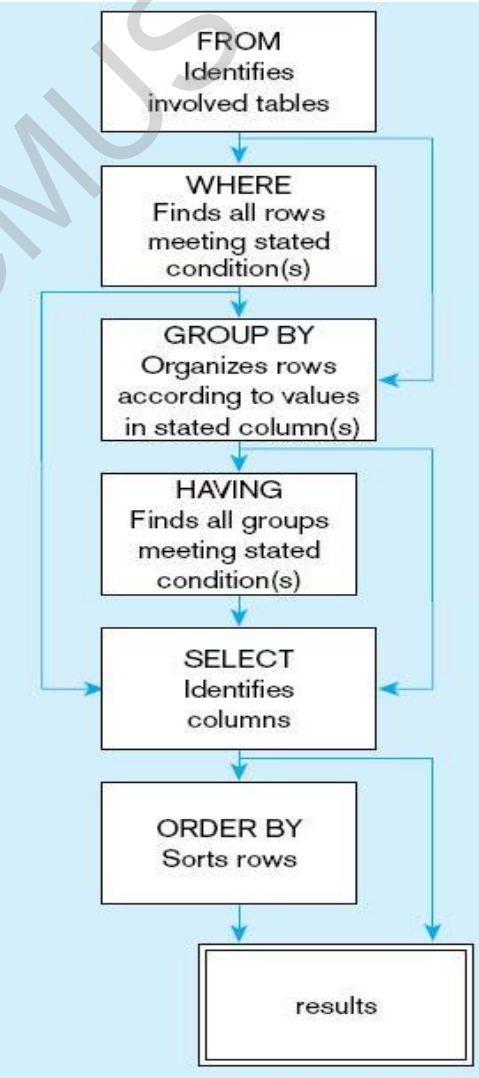
- **Using Count Function**

  Query: Find the names of employees who work on all the projects controlled by department number 5

```
SELECT E.Fname, E.Lname
FROM EMPLOYEE E, WORK_ON W
WHERE E.Ssn = W.Essn
        AND W.Pno IN (SELECT P.Pnumber
                      FROM PROJECT  P
                      WHERE  P.Dnum = 5)
GROUP BY W.Essn
HAVING COUNT(DISTINCT W.Pno) = ( SELECT COUNT(P.Pnumber)
                                 FROM PROJECT P
                                 WHERE P.Dnum = 5)
```

Summary of SQL Queries

```
SELECT <attribute and function list>
FROM <table list>
[ WHERE <condition> ]
[ GROUP BY <grouping attribute(s)> ]
[ HAVING <group condition> ]
[ ORDER BY <attribute list> ];
```



SQL statement processing order
(based on van der Lans, 2006 p.100)

Outline

- SQL Overview
- Basic SQL
  - SQL Data Definition and Data Types
  - Specifying Constraints in SQL
  - Insert/Update/Delete Statement in SQL
  - Basic Retrieval Queries in SQL
- Advanced SQL
  - Complex SQL Retrieval Queries
  - View
  - Ensuring Transaction Integrity
    - Specifying Constraints as Assertions & Action Trigger
- Tips for Developing Queries

Views (Virtual Tables) in SQL

Concept of a View in SQL

- 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 corresponding base tables

Views (Virtual Tables) in SQL

Specification of Views in SQL

- CREATE VIEW
- DROP VIEW
- We can now specify SQL queries on a view—or virtual table—in the same way we specify queries involving base tables

```
CREATE VIEW WORKS_ON1
AS SELECT Fname, Lname, Pname, Hours
   FROM EMPLOYEE, PROJECT, WORKS_ON
   WHERE Ssn = Essn AND Pno = Pnumber;
```

WORKS_ON1

Fname	Lname	Pname	Hours

```
CREATE VIEW DEPT_INFO(Dept_name, No_of_emps,Total_sal)
AS SELECT Dname, COUNT (*), SUM (Salary)
   FROM DEPARTMENT, EMPLOYEE
   WHERE Dnumber = Dno
   GROUP BY Dname;
```

DEPT_INFO

Dept_name	No_of_emps	Total_sal

Views (Virtual Tables) in SQL

Specification of Views in SQL

- We can now specify SQL queries on a view—or virtual table
- DROP VIEW

```
SELECT Fname, Lname
FROM WORKS_ON1
WHERE Pname = 'ProductX';
```

```
DROP VIEW WORKS_ON1;
```

Views (Virtual Tables) in SQL

- 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

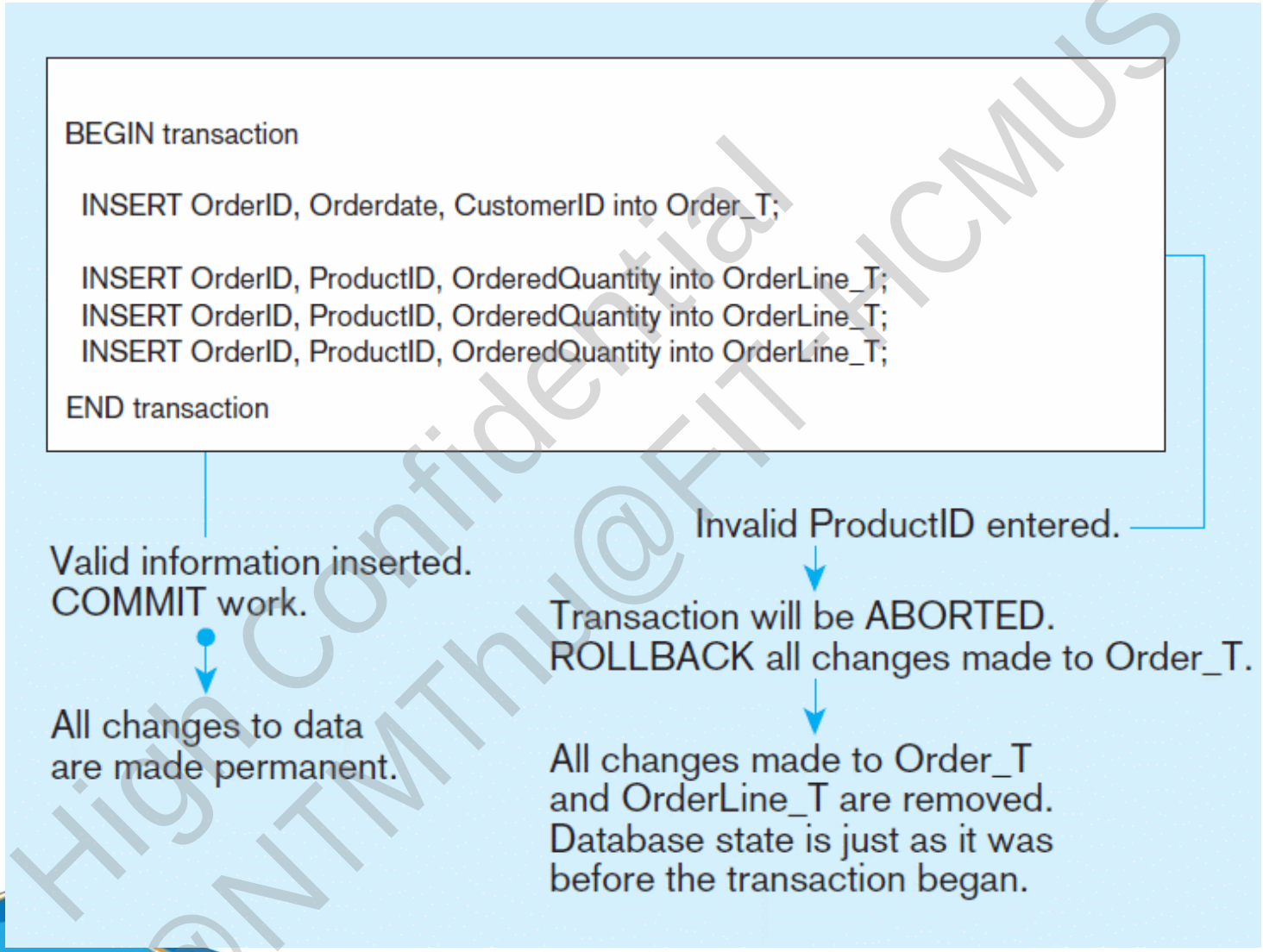
Outline

- SQL Overview
- Basic SQL
  - SQL Data Definition and Data Types
  - Specifying Constraints in SQL
  - Insert/Update/Delete Statement in SQL
  - Basic Retrieval Queries in SQL
- Advanced SQL
  - Complex SQL Retrieval Queries
  - View
  - Ensuring Transaction Integrity
    - Specifying Constraints as Assertions & Action Trigger
- Tips for Developing Queries

Ensuring Transaction Integrity

- Transaction = A discrete unit of work that must be completely processed or not processed at all
  - May involve multiple updates
  - If any update fails, then all other updates must be cancelled
- SQL commands for transactions
  - **BEGIN TRANSACTION/END TRANSACTION**
    - Marks boundaries of a transaction
  - **COMMIT**
    - Makes all updates permanent
  - **ROLLBACK**
    - Cancels updates since the last COMMIT

An SQL Transaction sequence (in pseudocode)

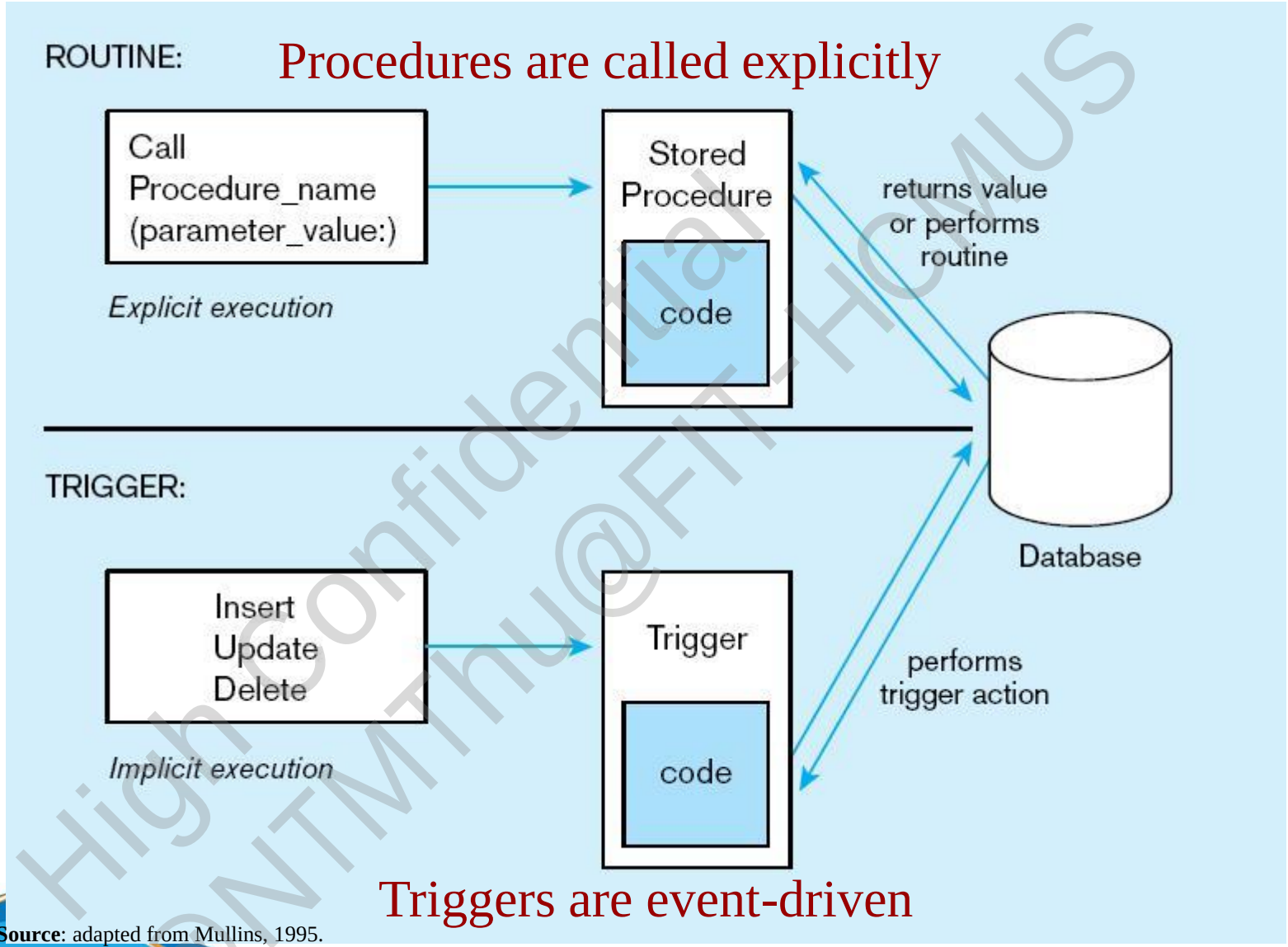


Routines and Triggers

Routines and Triggers

- **Routines**
  - Program modules that execute on demand
- **Functions–**routines that return values and take input parameters
- **Procedures–**routines that do not return values and can take input or output parameters
- **Triggers–**routines that execute in response to a database event (INSERT, UPDATE, or DELETE)

Triggers contrasted with stored procedures (based on Mullins 1995)



Source: adapted from Mullins, 1995.

Specifying Constraints as Assertions and Actions as Triggers

- Specifying General Constraints as Assertions in SQL
  - CREATE ASSERTION statement
  - Each assertion is given a constraint name and is specified via a condition similar to the WHERE clause of an SQL query

For example, to specify the constraint that the salary of an employee must not be greater than the salary of the manager of the department that the employee works for in SQL, we can write the following assertion:

```
CREATE ASSERTION SALARY_CONSTRAINT
CHECK ( NOT EXISTS ( SELECT *
                     FROM EMPLOYEE E, EMPLOYEE M,DEPARTMENT D
                     WHERE E.Salary>M.Salary
                     AND E.Dno = D.Dnumber
                     AND D.Mgr_ssn = M.Ssn));
```

Specifying Constraints as Assertions and Actions as Triggers

- Introduction to Triggers in SQL
  - CREATE TRIGGER statement
  - Triggers are used when you need to perform, under specified conditions, a certain action as the result of some database event (e.g., the execution of a DML statement such as INSERT, UPDATE, or DELETE or the DDL statement ALTER TABLE)
  - Thus, a trigger has three parts—**the event, the condition,** and **the action**—and these parts are reflected in the coding structure for triggers

Events are: inserting a new employee record, changing an employee's salary, or changing an employee's supervisor

```
CREATE TRIGGER SALARY_VIOLATION
BEFORE INSERT OR UPDATE OF SALARY, SUPERVISOR_SSN ON EMPLOYEE
FOR EACH ROW
WHEN ( NEW.SALARY > ( SELECT SALARY FROM EMPLOYEE
                      WHERE SSN = NEW.SUPERVISOR_SSN))
    INFORM_SUPERVISOR(NEW.Supervisor_ssn,NEW.Ssn);
```

Condition that determines whether the rule action should be executed

The action to be taken:
- The action is usually a sequence of SQL statements
- The action is to execute the stored procedure

Outline

- SQL Overview
- Basic SQL
  - SQL Data Definition and Data Types
  - Specifying Constraints in SQL
  - Insert/Update/Delete Statement in SQL
  - Basic Retrieval Queries in SQL
- Advanced SQL
  - Complex SQL Retrieval Queries
  - View
  - Ensuring Transaction Integrity
    - Specifying Constraints as Assertions & Action Trigger
- Tips for Developing Queries

Tips for Developing Queries

- Be familiar with the data model (entities and relationships)
- Understand the desired results
- Know the attributes desired in results
- Identify the entities that contain desired attributes
- Review ERD
- Construct a WHERE equality for each link
- Fine tune with GROUP BY and HAVING clauses if needed
- Consider the effect on unusual data

Query Efficiency Considerations

- Instead of SELECT *, identify the specific attributes in the SELECT clause; this helps reduce network traffic of result set
- Limit the number of subqueries; try to make everything done in a single query if possible
- If data is to be used many times, make a separate query and store it as a view

Guidelines for Better Query Design

- Understand how indexes are used in query processing
- Keep optimizer statistics up-to-date
- Use compatible data types for fields and literals
- Write simple queries
- Break complex queries into multiple simple parts
- Don’t nest one query inside another query
- Don’t combine a query with itself (if possible avoid self-joins)

Guidelines for Better Query Design (cont.)

- Create temporary tables for groups of queries
- Combine update operations
- Retrieve only the data you need
- Don’t have the DBMS sort without an index
- Learn!
- Consider the total query processing time for ad hoc queries