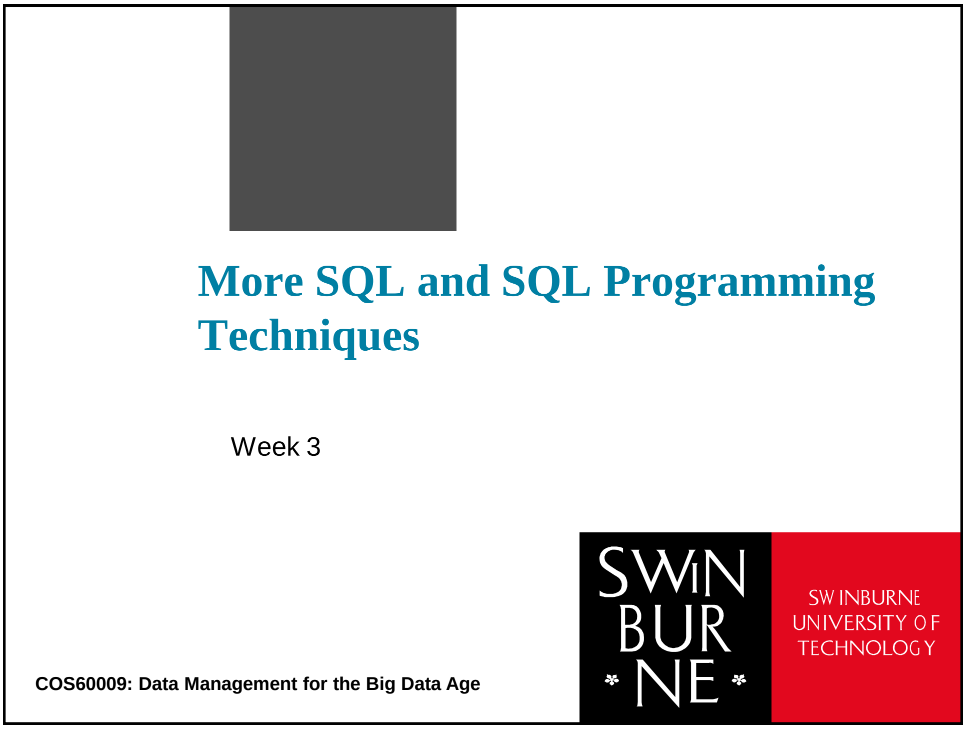


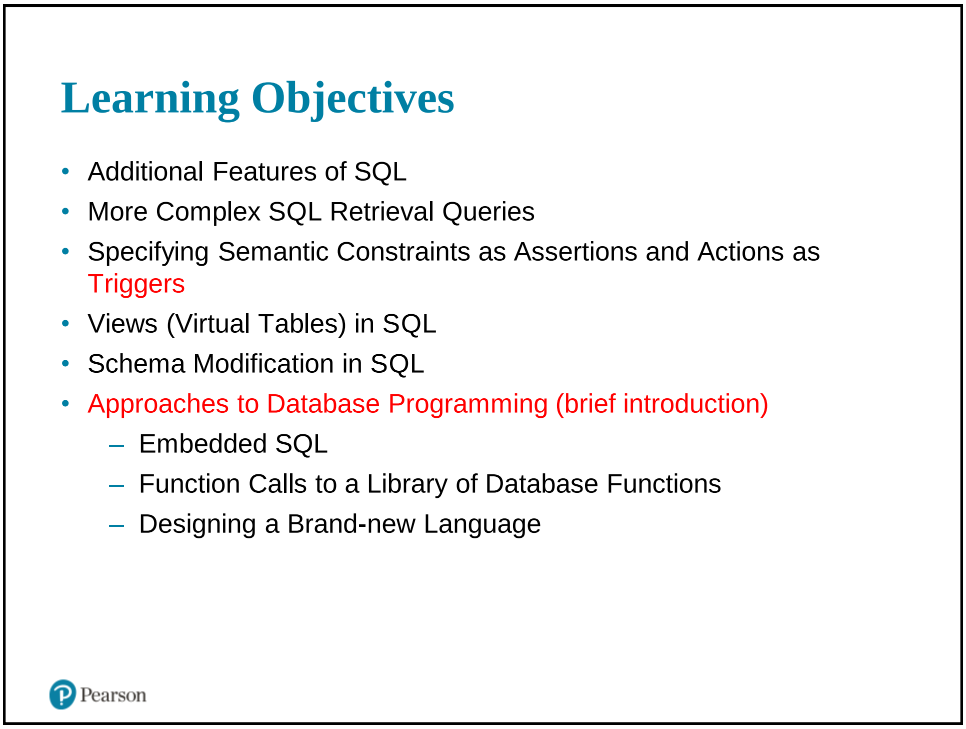
More SQL and SQL Programming Techniques

Week 3

COS60009: Data Management for the Big Data Age

SWINBURNE UNIVERSITY OF TECHNOLOGY

1



Learning Objectives

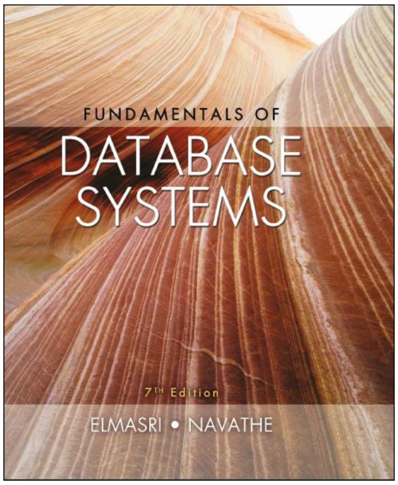
- Additional Features of SQL
- More Complex SQL Retrieval Queries
- Specifying Semantic Constraints as Assertions and Actions as Triggers
- Views (Virtual Tables) in SQL
- Schema Modification in SQL
- Approaches to Database Programming (brief introduction)
  - Embedded SQL
  - Function Calls to a Library of Database Functions
  - Designing a Brand-new Language

Pearson

2

Fundamentals of Database Systems

Seventh Edition



Chapter 7

More SQL: Complex Queries, Triggers, Views, and Schema Modification

Chapter 10

Introduction to SQL Programming Techniques

Tables as Sets in SQL (1 of 2)

- SQL does not automatically eliminate duplicate tuples in query results
- For aggregate operations duplicates must be accounted for
- Use the keyword **DISTINCT** in the SELECT clause
  - Only distinct tuples should remain in the result

Query 11. Retrieve the salary of every employee (Q11) and all distinct salary values (Q11A).

```
Q11:    SELECT    ALL Salary
        FROM      EMPLOYEE;

Q11A:   SELECT    DISTINCT Salary
        FROM      EMPLOYEE;
```

Tables as Sets in SQL (2 of 2)

- Set operations
  - **UNION**, **EXCEPT** (difference), **INTERSECT**
  - Corresponding multiset operations: UNION ALL, EXCEPT ALL, INTERSECT ALL)
  - Type compatibility is needed for these operations to be valid

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.

```
Q4A:  (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' );
```

5

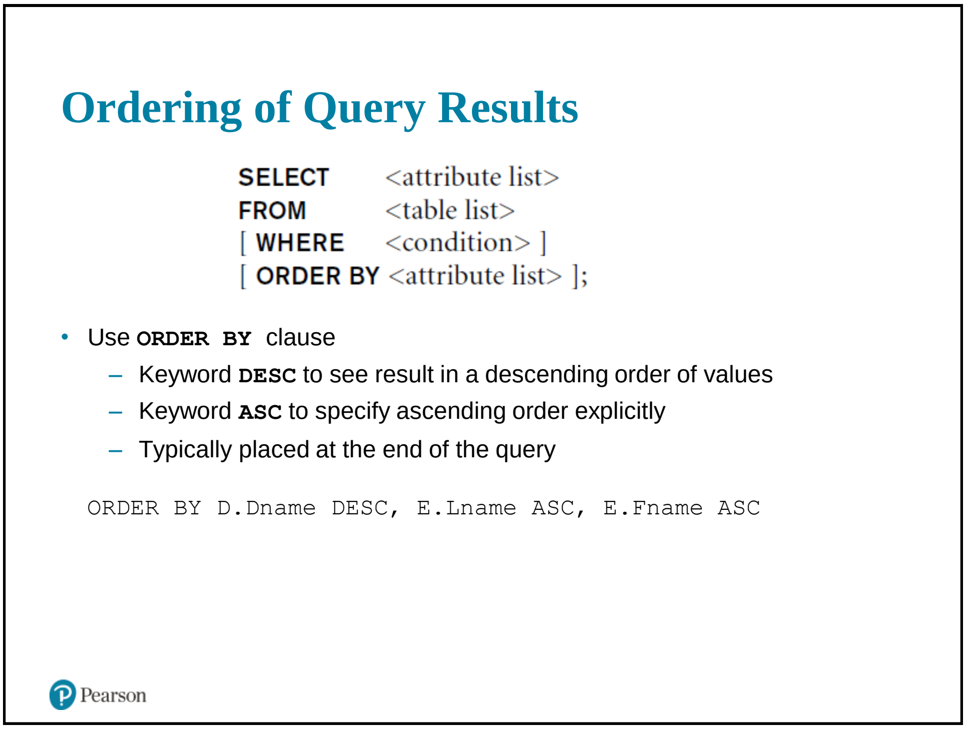
Substring Pattern Matching and Arithmetic Operators

- **LIKE** comparison operator
  - Used for string **pattern matching**: % replaces an arbitrary number of zero or more characters, underscore (_) replaces a single character
  - Examples:
    - **WHERE** Address **LIKE** '%Houston,TX%';
    - **WHERE** Ssn **LIKE** '_ _ 1_ _ 8901';
- **BETWEEN** comparison operator

  WHERE(Salary **BETWEEN** 30000 **AND** 40000) **AND** Dno = 5;
- Standard arithmetic operators:
  - Addition (+), subtraction (–), multiplication (*), and division (/) may be included as a part of **SELECT**
- **Query 13.** Show the resulting salaries if every employee working on the 'ProductX' project is given a 10 percent raise.

```
Q13:   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';
```

6



Ordering of Query Results

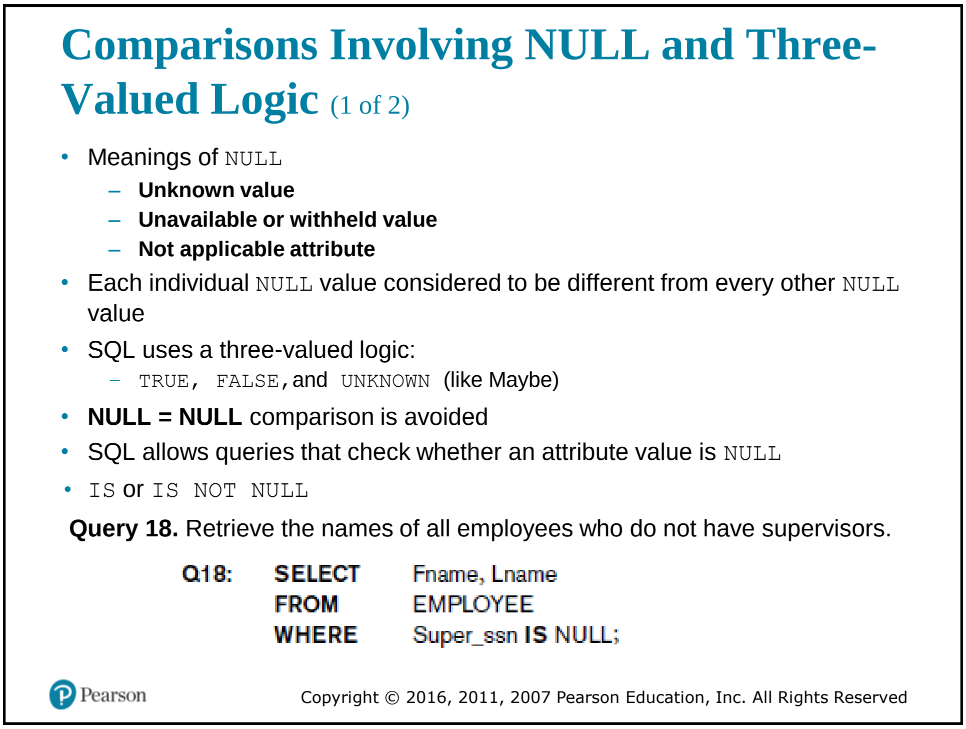
```
SELECT       <attribute list>
FROM         <table list>
[ WHERE      <condition> ]
[ ORDER BY <attribute list> ];
```

- Use **ORDER BY** clause
  - Keyword **DESC** to see result in a descending order of values
  - Keyword **ASC** to specify ascending order explicitly
  - Typically placed at the end of the query

```
ORDER BY D.Dname DESC, E.Lname ASC, E.Fname ASC
```

Pearson

7



Comparisons Involving NULL and Three-Valued Logic (1 of 2)

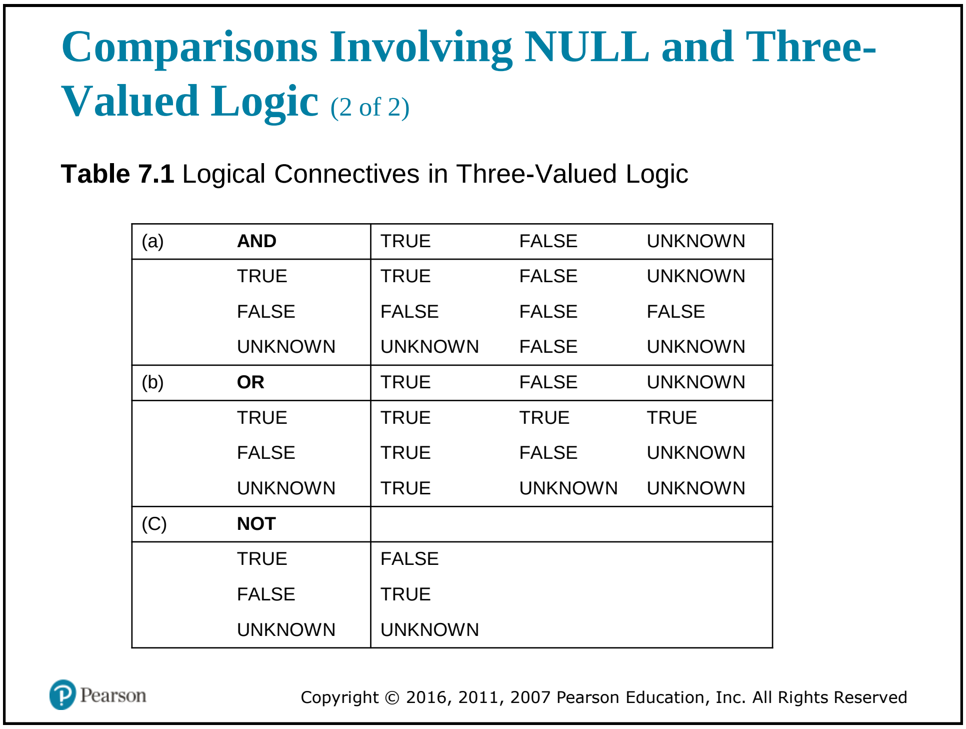
- Meanings of NULL
  - **Unknown value**
  - **Unavailable or withheld value**
  - **Not applicable attribute**
- Each individual NULL value considered to be different from every other NULL value
- SQL uses a three-valued logic:
  - TRUE, FALSE, and UNKNOWN (like Maybe)
- **NULL = NULL** comparison is avoided
- SQL allows queries that check whether an attribute value is NULL
- IS or IS NOT NULL

Query 18. Retrieve the names of all employees who do not have supervisors.

```
Q18:    SELECT    Fname, Lname
        FROM      EMPLOYEE
        WHERE     Super_ssn IS NULL;
```

Pearson

8

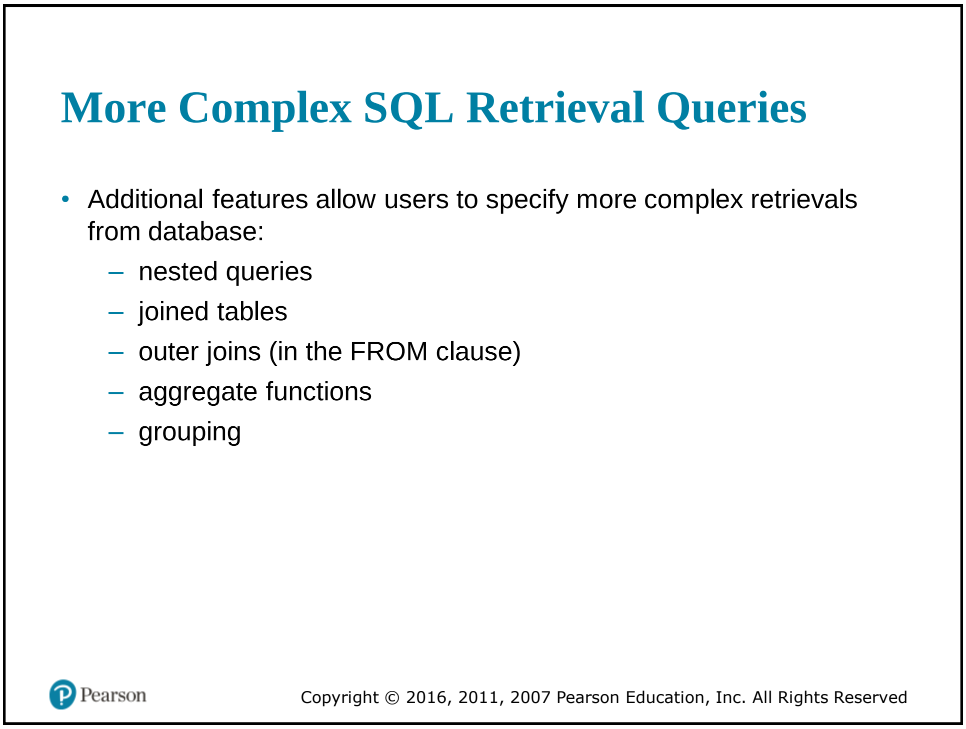


Comparisons Involving NULL and Three-Valued Logic (2 of 2)

Table 7.1 Logical Connectives in Three-Valued Logic

(a)	**AND**	TRUE	FALSE	UNKNOWN
	TRUE	TRUE	FALSE	UNKNOWN
	FALSE	FALSE	FALSE	FALSE
	UNKNOWN	UNKNOWN	FALSE	UNKNOWN
(b)	**OR**	TRUE	FALSE	UNKNOWN
	TRUE	TRUE	TRUE	TRUE
	FALSE	TRUE	FALSE	UNKNOWN
	UNKNOWN	TRUE	UNKNOWN	UNKNOWN
(C)	**NOT**			
	TRUE	FALSE		
	FALSE	TRUE		
	UNKNOWN	UNKNOWN		

Pearson



More Complex SQL Retrieval Queries

- Additional features allow users to specify more complex retrievals from database:
  - nested queries
  - joined tables
  - outer joins (in the FROM clause)
  - aggregate functions
  - grouping

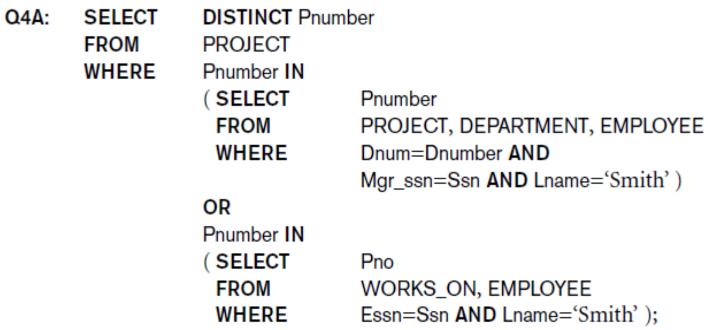
Pearson

Nested Queries, Tuples, and Set/Multiset Comparisons

- **Nested queries**
  - Complete select-from-where blocks within WHERE clause of another query
  - **Outer query and nested subqueries**
- Comparison operator `IN`
  - Compares value *v* with a set (or multiset) of values *V*
  - Evaluates to `TRUE` if *v* is one of the elements in *V*

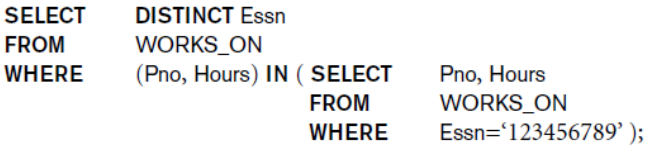
Pearson

Nested Queries (1 of 2)



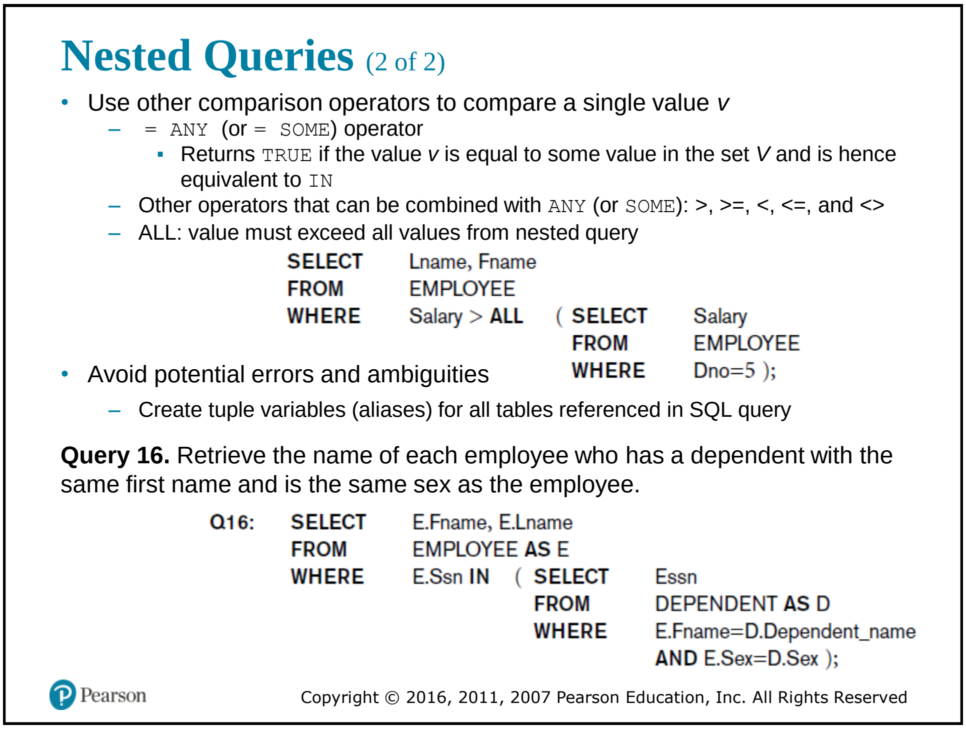
```
Q4A:  SELECT   DISTINCT Pnumber
      FROM     PROJECT
      WHERE    Pnumber IN
               ( SELECT    Pnumber
                 FROM      PROJECT, DEPARTMENT, EMPLOYEE
                 WHERE     Dnum=Dnumber AND
                           Mgr_ssn=Ssn AND Lname='Smith' )
               OR
               Pnumber IN
               ( SELECT    Pno
                 FROM      WORKS_ON, EMPLOYEE
                 WHERE     Essn=Ssn AND Lname='Smith' );
```

- Use tuples of values in comparisons and place them within parentheses



```
SELECT   DISTINCT Essn
FROM     WORKS_ON
WHERE    (Pno, Hours) IN ( SELECT   Pno, Hours
                           FROM     WORKS_ON
                           WHERE    Essn='123456789' );
```

Pearson



Nested Queries (2 of 2)

- Use other comparison operators to compare a single value *v*
  - = ANY (or = SOME) operator
    - Returns TRUE if the value *v* is equal to some value in the set *V* and is hence equivalent to IN
  - Other operators that can be combined with ANY (or SOME): >, >=, <, <=, and <>
  - ALL: value must exceed all values from nested query

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

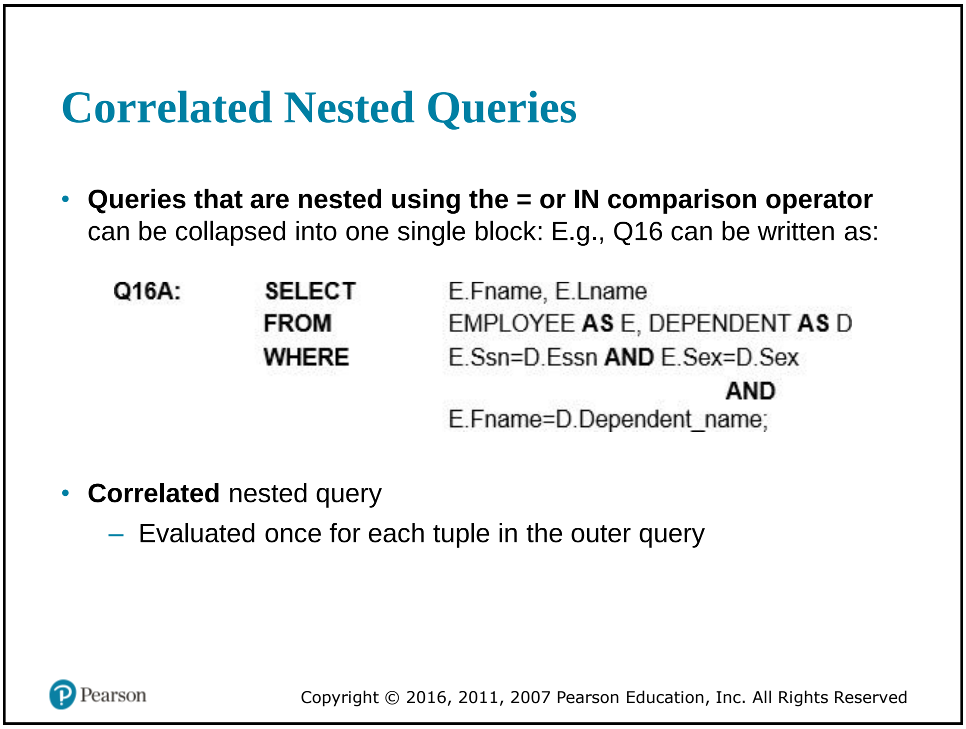
- Avoid potential errors and ambiguities
  - Create tuple variables (aliases) for all tables referenced in SQL query

Query 16. Retrieve the name of each employee who has a dependent with the same first name and is the same sex as the employee.

```
Q16:   SELECT    E.Fname, E.Lname
       FROM      EMPLOYEE AS E
       WHERE     E.Ssn IN   ( SELECT    Essn
                              FROM      DEPENDENT AS D
                              WHERE     E.Fname=D.Dependent_name
                                        AND E.Sex=D.Sex );
```

Pearson

13



Correlated Nested Queries

- **Queries that are nested using the = or IN comparison operator** can be collapsed into one single block: E.g., Q16 can be written as:

```
Q16A:     SELECT     E.Fname, E.Lname
          FROM       EMPLOYEE AS E, DEPENDENT AS D
          WHERE      E.Ssn=D.Essn AND E.Sex=D.Sex
                                                 AND
                     E.Fname=D.Dependent_name;
```

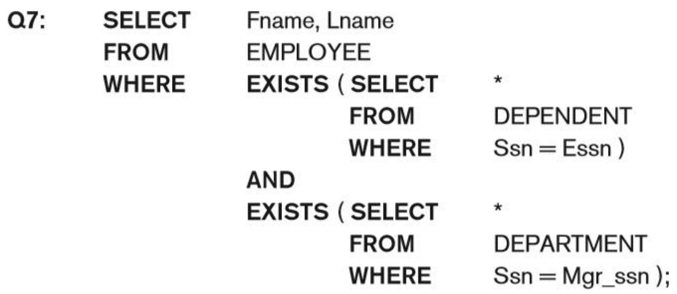
- **Correlated** nested query
  - Evaluated once for each tuple in the outer query

Pearson

14

USE of EXISTS

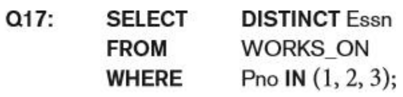
- EXISTS function
  - Check whether the result of a correlated nested query is empty or not. They are Boolean functions that return a TRUE or FALSE result.



```
Q7:    SELECT    Fname, Lname
       FROM      EMPLOYEE
       WHERE     EXISTS ( SELECT    *
                          FROM      DEPENDENT
                          WHERE     Ssn = Essn )
                 AND
                 EXISTS ( SELECT    *
                          FROM      DEPARTMENT
                          WHERE     Ssn = Mgr_ssn );
```

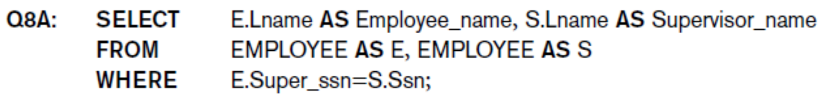
Explicit Sets and Renaming of Attributes in SQL

- Can use explicit set of values in WHERE clause

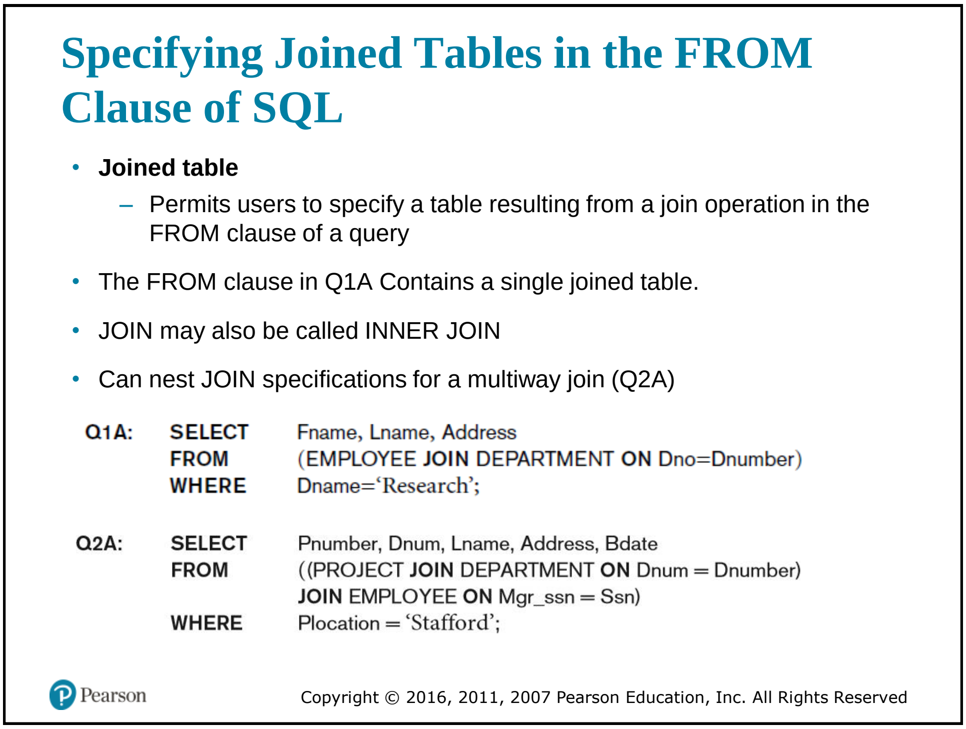


```
Q17:   SELECT    DISTINCT Essn
       FROM      WORKS_ON
       WHERE     Pno IN (1, 2, 3);
```

- Use qualifier AS followed by desired new name
  - Rename any attribute that appears in the result of a query



```
Q8A:   SELECT    E.Lname AS Employee_name, S.Lname AS Supervisor_name
       FROM      EMPLOYEE AS E, EMPLOYEE AS S
       WHERE     E.Super_ssn=S.Ssn;
```



Specifying Joined Tables in the FROM Clause of SQL

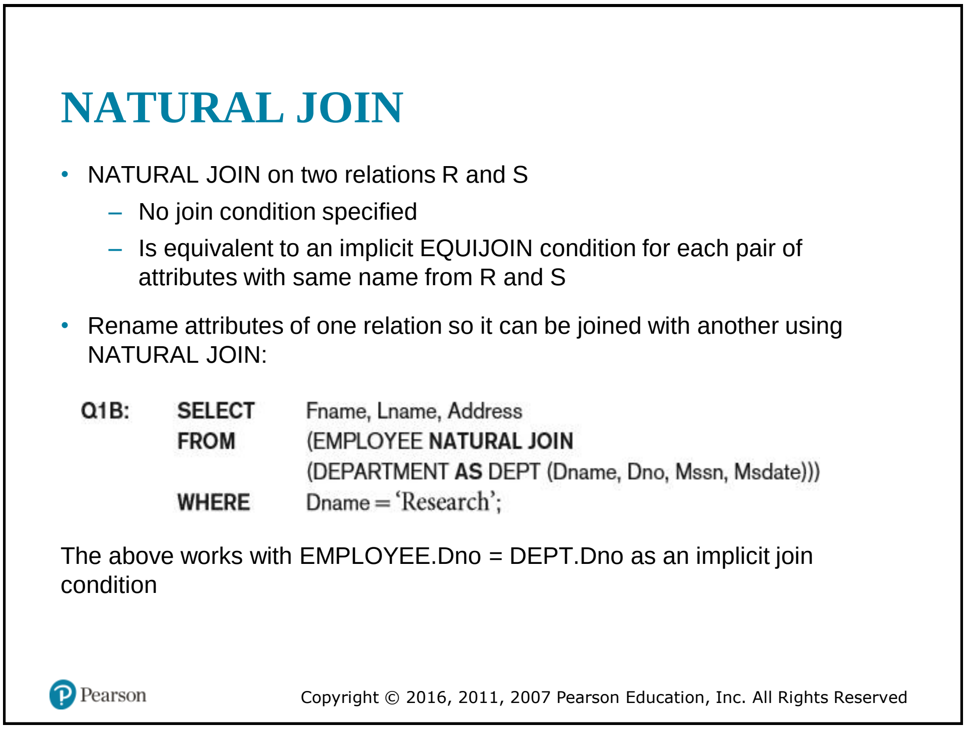
- **Joined table**
  - Permits users to specify a table resulting from a join operation in the FROM clause of a query
- The FROM clause in Q1A Contains a single joined table.
- JOIN may also be called INNER JOIN
- Can nest JOIN specifications for a multiway join (Q2A)

```
Q1A:    SELECT    Fname, Lname, Address
        FROM      (EMPLOYEE JOIN DEPARTMENT ON Dno=Dnumber)
        WHERE     Dname='Research';

Q2A:    SELECT    Pnumber, Dnum, Lname, Address, Bdate
        FROM      ((PROJECT JOIN DEPARTMENT ON Dnum = Dnumber)
                  JOIN EMPLOYEE ON Mgr_ssn = Ssn)
        WHERE     Plocation = 'Stafford';
```

Pearson

17



NATURAL JOIN

- NATURAL JOIN on two relations R and S
  - No join condition specified
  - Is equivalent to an implicit EQUIJOIN condition for each pair of attributes with same name from R and S
- Rename attributes of one relation so it can be joined with another using NATURAL JOIN:

```
Q1B:    SELECT    Fname, Lname, Address
        FROM      (EMPLOYEE NATURAL JOIN
                  (DEPARTMENT AS DEPT (Dname, Dno, Mssn, Msdate)))
        WHERE     Dname = 'Research';
```

The above works with EMPLOYEE.Dno = DEPT.Dno as an implicit join condition

Pearson

18

INNER and OUTER Joins

- INNER JOIN (**versus** OUTER JOIN)
  - Default type of join in a joined table
  - Tuple is included in the result only if a matching tuple exists in the other relation
- LEFT OUTER JOIN
  - Every tuple in left table must appear in result
  - If no matching tuple
    - Padded with NULL values for attributes of right table
- RIGHT OUTER JOIN
  - Every tuple in right table must appear in result
  - If no matching tuple
    - Padded with NULL values for attributes of left table
- FULL OUTER JOIN – combines results of LEFT and RIGHT OUTER JOIN

Pearson

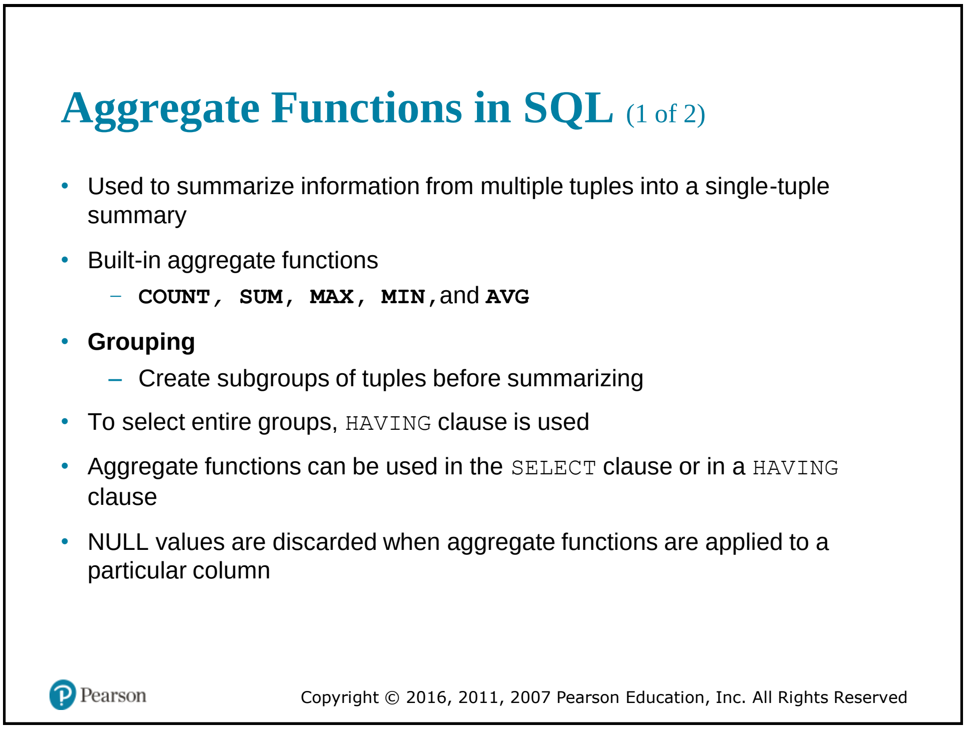
Example: LEFT OUTER JOIN

```
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);
```

Alternate Syntax:

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

Pearson

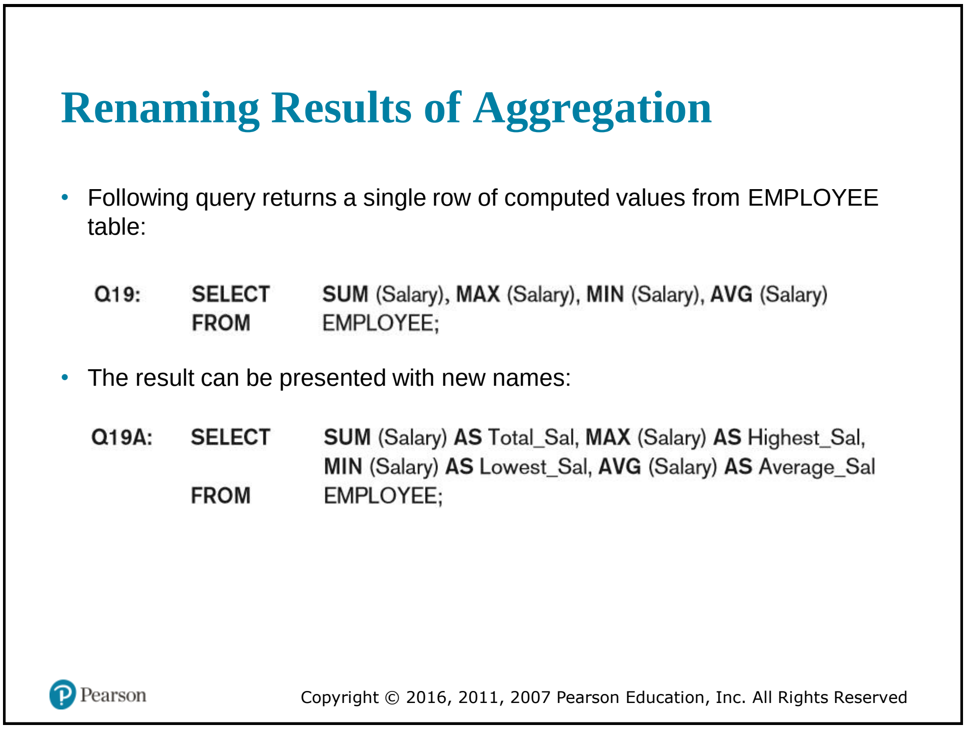


Aggregate Functions in SQL (1 of 2)

- Used to summarize information from multiple tuples into a single-tuple summary
- Built-in aggregate functions
  - **`COUNT, SUM, MAX, MIN,`** and **`AVG`**
- **Grouping**
  - Create subgroups of tuples before summarizing
- To select entire groups, `HAVING` clause is used
- Aggregate functions can be used in the `SELECT` clause or in a `HAVING` clause
- NULL values are discarded when aggregate functions are applied to a particular column

Pearson

21



Renaming Results of Aggregation

- Following query returns a single row of computed values from EMPLOYEE table:

```
Q19:    SELECT    SUM (Salary), MAX (Salary), MIN (Salary), AVG (Salary)
        FROM      EMPLOYEE;
```

- The result can be presented with new names:

```
Q19A:   SELECT    SUM (Salary) AS Total_Sal, MAX (Salary) AS Highest_Sal,
                  MIN (Salary) AS Lowest_Sal, AVG (Salary) AS Average_Sal
        FROM      EMPLOYEE;
```

Pearson

22

Aggregate Functions in SQL (2 of 2)

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.

```
Q20:   SELECT    SUM (Salary), MAX (Salary), MIN (Salary), AVG (Salary)
       FROM      (EMPLOYEE JOIN DEPARTMENT ON Dno=Dnumber)
       WHERE     Dname='Research';
```

Queries 21 and 22. Retrieve the total number of employees in the company (Q21) and the number of employees in the 'Research' department (Q22).

```
Q21:   SELECT    COUNT (*)
       FROM      EMPLOYEE;

Q22:   SELECT    COUNT (*)
       FROM      EMPLOYEE, DEPARTMENT
       WHERE     DNO=DNUMBER AND DNAME='Research';
```

Pearson

23

Grouping: The GROUP BY Clause

- **Partition** relation into subsets of tuples based on **grouping attribute(s)** and apply function to each such group independently
- `GROUP BY` clause specifies grouping attributes, grouping attributes must appear in the SELECT clause
- COUNT (*) counts the number of rows in the group

```
Q24:   SELECT      Dno, COUNT (*), AVG (Salary)
       FROM        EMPLOYEE
       GROUP BY    Dno;
```

- If the grouping attribute has NULL as a possible value, then a separate group is created for the null value (e.g., null Dno in the above query)
- GROUP BY may be applied to the result of a JOIN:

```
Q25:   SELECT      Pnumber, Pname, COUNT (*)
       FROM        PROJECT, WORKS_ON
       WHERE       Pnumber = Pno
       GROUP BY    Pnumber, Pname;
```

Pearson

24

Grouping: The GROUP BY and HAVING Clauses

- `HAVING` clause provides a condition to select or reject an entire group:
- **Query 26.** 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 the project.

```
Q26:    SELECT      Pnumber, Pname, COUNT (*)
        FROM        PROJECT, WORKS_ON
        WHERE       Pnumber = Pno
        GROUP BY    Pnumber, Pname
        HAVING      COUNT (*) > 2;
```

Pearson

Combining the WHERE and the HAVING Clauses

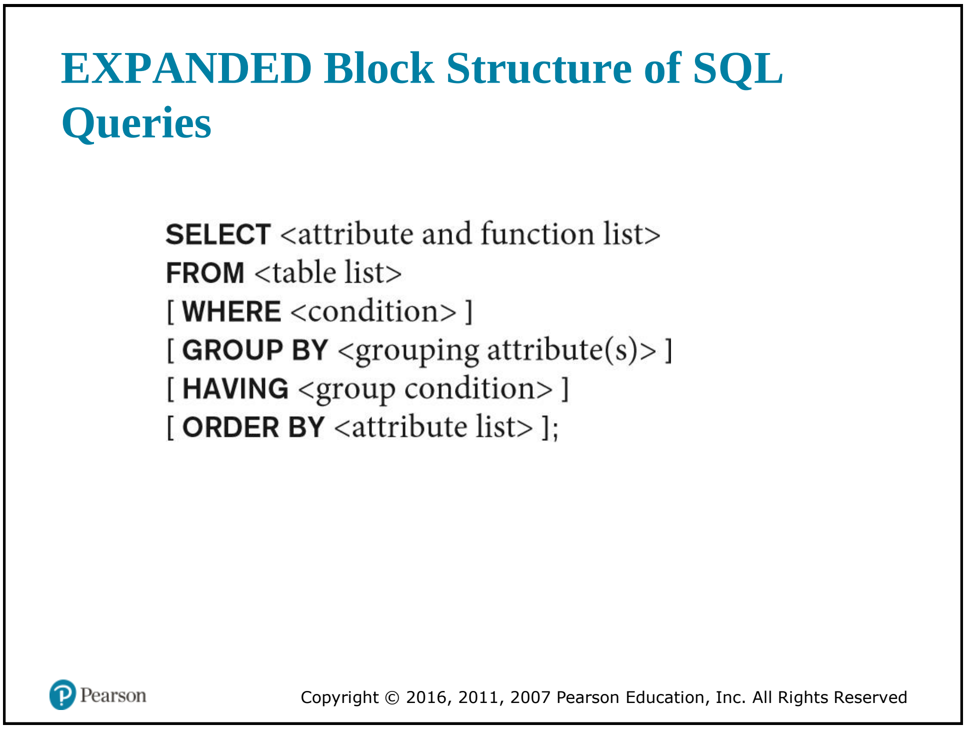
- Consider the query: we want to count the **total** number of employees whose salaries exceed $40,000 in each department, but only for departments where more than five employees work.
- Incorrect Query:
- Correct Specification of the Query: the WHERE clause applies tuple by tuple whereas HAVING applies to entire group of tuples

```
SELECT      Dno, COUNT (*)
FROM        EMPLOYEE
WHERE       Salary>40000
GROUP BY    Dno
HAVING      COUNT (*) > 5;
```

Query 28. For each department that has more than five employees, retrieve the department number and the number of its employees who are making more than $40,000.

```
Q28   SELECT    Dnumber, COUNT(*)
      FROM      DEPARTMENT, EMPLOYEE
      WHERE     Dnumber=Dno AND Salary>40000 AND
                Dno in
                (SELECT      Dno
                  FROM        EMPLOYEE
                  GROUP BY    Dno
                  HAVING      COUNT(*) > 5)
     GROUP BY Dnumber;
```

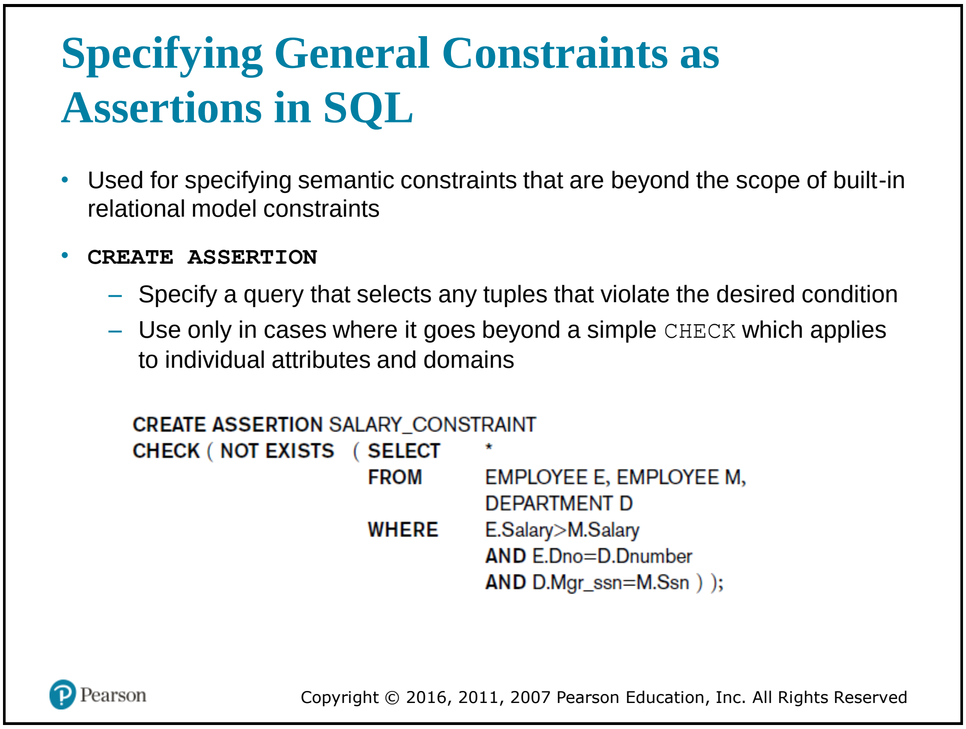
Pearson



EXPANDED Block Structure 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> ];
```

27



Specifying General Constraints as Assertions in SQL

- Used for specifying semantic constraints that are beyond the scope of built-in relational model constraints
- `CREATE ASSERTION`
  - Specify a query that selects any tuples that violate the desired condition
  - Use only in cases where it goes beyond a simple `CHECK` which applies to individual attributes and domains

```
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 ) );
```

28

Introduction to Triggers in SQL

- **CREATE TRIGGER**
  - Specify automatic actions that database system will perform when certain events and conditions occur
  - Used to monitor the database
- Typical trigger has three components (ECA) which make it a rule for an “active database”: **Event(s), Condition, Action**
- An EXAMPLE with standard Syntax

```
R5:
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)
```

Pearson

Views (Virtual Tables) in SQL

- Concept of a view in SQL
  - Single table derived from other tables called the **defining tables**
  - Considered to be a virtual table that is not necessarily populated
  - View is always up-to-date
- **CREATE VIEW** command: give table name, list of attribute names, and a query to specify the contents of the view (In V1, attributes retain the names from base tables. In V2, attributes are assigned names)
- Once a View is defined, SQL queries can use it in the FROM clause
- **DROP VIEW** command: dispose of a view

```
V1:   CREATE VIEW   WORKS_ON1
      AS SELECT     Fname, Lname, Pname, Hours
         FROM       EMPLOYEE, PROJECT, WORKS_ON
         WHERE      Ssn=Essn AND Pno=Pnumber;

V2:   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;
```

Pearson

Schema Change Statements in SQL

- **Schema evolution commands**
  - DBA may want to change the schema while the database is operational
  - Does not require recompilation of the database schema
- **DROP** command: drop named schema elements, such as tables, domains, or constraint Drop behavior options:
  - Drop behavior options: `CASCADE` and `RESTRICT`
  - Example: `DROP SCHEMA COMPANY CASCADE;`
    This removes the schema and all its elements including tables, views, constraints, etc.
- **Alter table** command: add or drop a column (attribute), change a column definition, add or drop table constraints
  - Example:

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

Schema Change Statements in SQL (Cont'd)

- Change constraints specified on a table: add or drop a named constraint

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

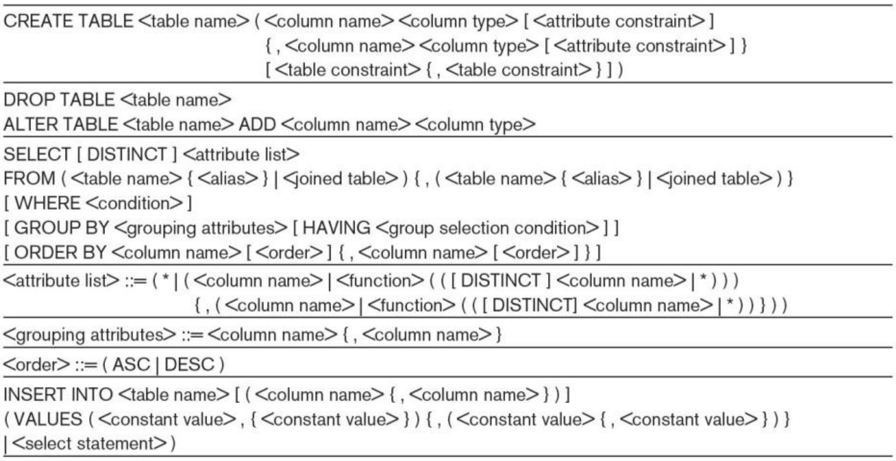
- To drop a column
  - Choose either `CASCADE` or `RESTRICT`
  - `CASCADE` would drop the column from views etc. `RESTRICT` is possible if no views refer to it.

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

- Default values can be dropped and altered :

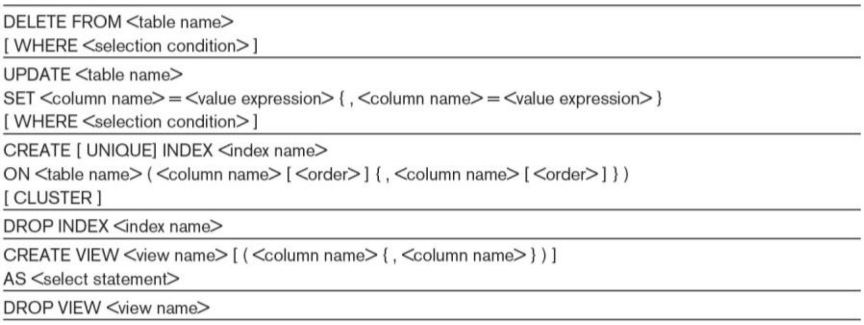
```
ALTER TABLE  COMPANY.DEPARTMENT  ALTER COLUMN  Mgr_ssn
    DROP DEFAULT;
ALTER TABLE  COMPANY.DEPARTMENT  ALTER COLUMN  Mgr_ssn
    SET DEFAULT  '333445555';
```

Table 7.2 Summary of SQL Syntax (1 of 2)



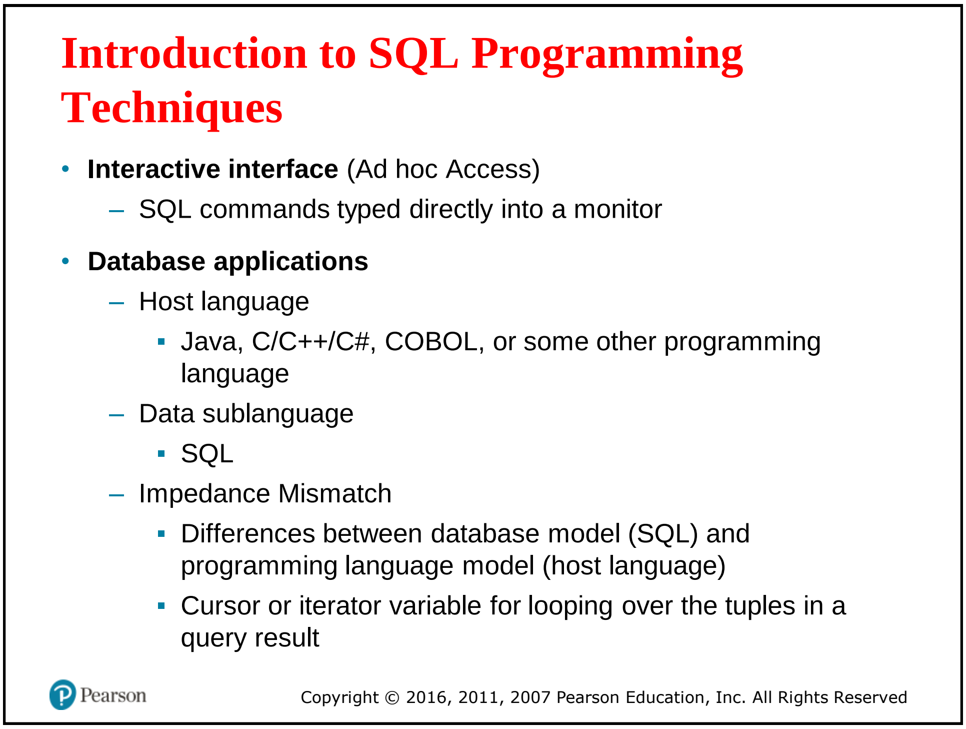
```
CREATE TABLE <table name> ( <column name> <column type> [ <attribute constraint> ]
                            { , <column name> <column type> [ <attribute constraint> ] }
                            [ <table constraint> { , <table constraint> } ] )
--------------------------------------------------------------------------------
DROP TABLE <table name>
ALTER TABLE <table name> ADD <column name> <column type>
--------------------------------------------------------------------------------
SELECT [ DISTINCT ] <attribute list>
FROM ( <table name> { <alias> } | <joined table> ) { , ( <table name> { <alias> } | <joined table> ) }
[ WHERE <condition> ]
[ GROUP BY <grouping attributes> [ HAVING <group selection condition> ] ]
[ ORDER BY <column name> [ <order> ] { , <column name> [ <order> ] } ]
--------------------------------------------------------------------------------
<attribute list> ::= ( * | ( <column name> | <function> ( ( [ DISTINCT ] <column name> | * ) ) )
                     { , ( <column name> | <function> ( ( [ DISTINCT] <column name> | * ) ) } ) )
--------------------------------------------------------------------------------
<grouping attributes> ::= <column name> { , <column name> }
--------------------------------------------------------------------------------
<order> ::= ( ASC | DESC )
--------------------------------------------------------------------------------
INSERT INTO <table name> [ ( <column name> { , <column name> } ) ]
( VALUES ( <constant value> , { <constant value> } ) { , ( <constant value> { , <constant value> } ) }
| <select statement> )
```

Table 7.2 Summary of SQL Syntax (2 of 2)



```
DELETE FROM <table name>
[ WHERE <selection condition> ]
--------------------------------------------------------------------------------
UPDATE <table name>
SET <column name> = <value expression> { , <column name> = <value expression> }
[ WHERE <selection condition> ]
--------------------------------------------------------------------------------
CREATE [ UNIQUE] INDEX <index name>
ON <table name> ( <column name> [ <order> ] { , <column name> [ <order> ] } )
[ CLUSTER ]
--------------------------------------------------------------------------------
DROP INDEX <index name>
--------------------------------------------------------------------------------
CREATE VIEW <view name> [ ( <column name> { , <column name> } ) ]
AS <select statement>
--------------------------------------------------------------------------------
DROP VIEW <view name>
```

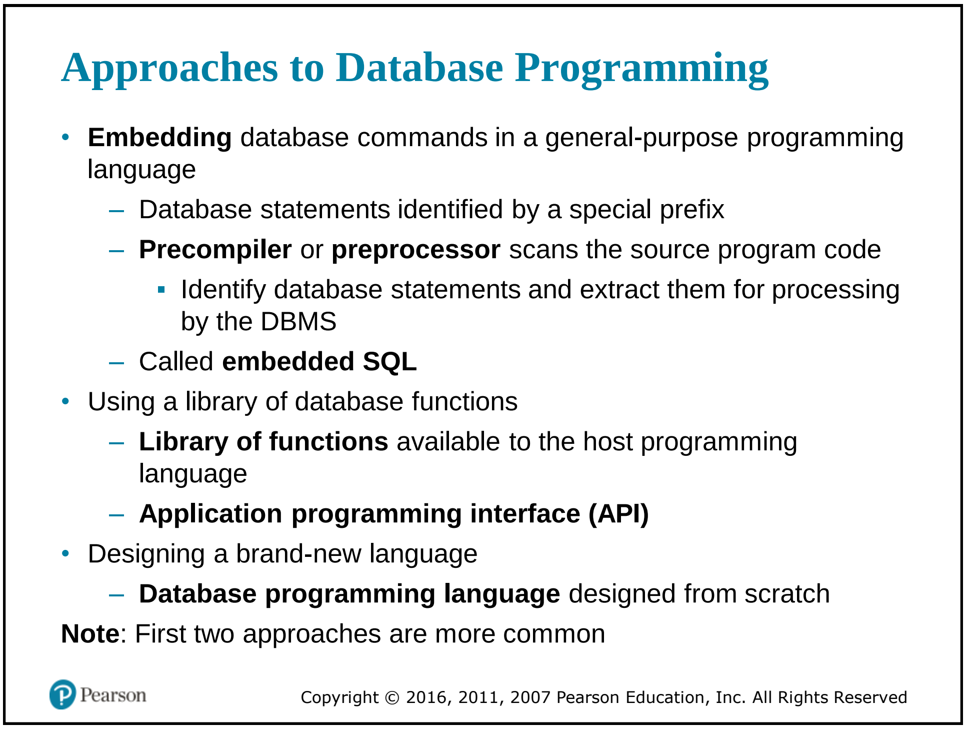
Note: The commands for creating and dropping indexes are not part of standard SQL.



Introduction to SQL Programming Techniques

- **Interactive interface** (Ad hoc Access)
  - SQL commands typed directly into a monitor
- **Database applications**
  - Host language
    - Java, C/C++/C#, COBOL, or some other programming language
  - Data sublanguage
    - SQL
  - Impedance Mismatch
    - Differences between database model (SQL) and programming language model (host language)
    - Cursor or iterator variable for looping over the tuples in a query result

35

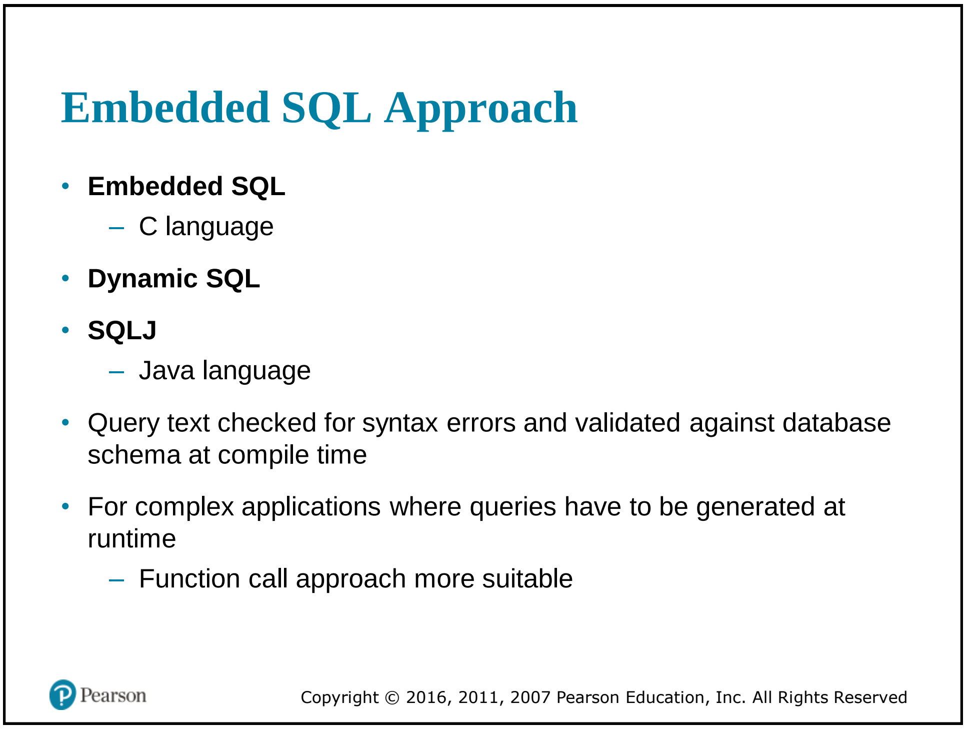


Approaches to Database Programming

- **Embedding** database commands in a general-purpose programming language
  - Database statements identified by a special prefix
  - **Precompiler** or **preprocessor** scans the source program code
    - Identify database statements and extract them for processing by the DBMS
  - Called **embedded SQL**
- Using a library of database functions
  - **Library of functions** available to the host programming language
  - **Application programming interface (API)**
- Designing a brand-new language
  - **Database programming language** designed from scratch

Note: First two approaches are more common

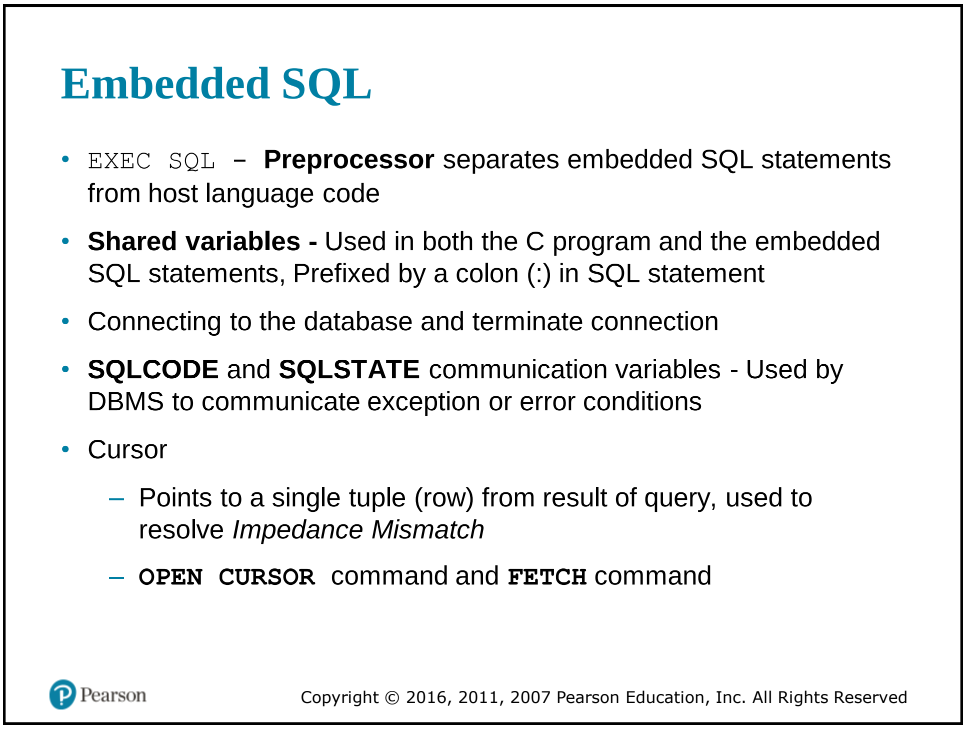
36



Embedded SQL Approach

- **Embedded SQL**
  - C language
- **Dynamic SQL**
- **SQLJ**
  - Java language
- Query text checked for syntax errors and validated against database schema at compile time
- For complex applications where queries have to be generated at runtime
  - Function call approach more suitable

37

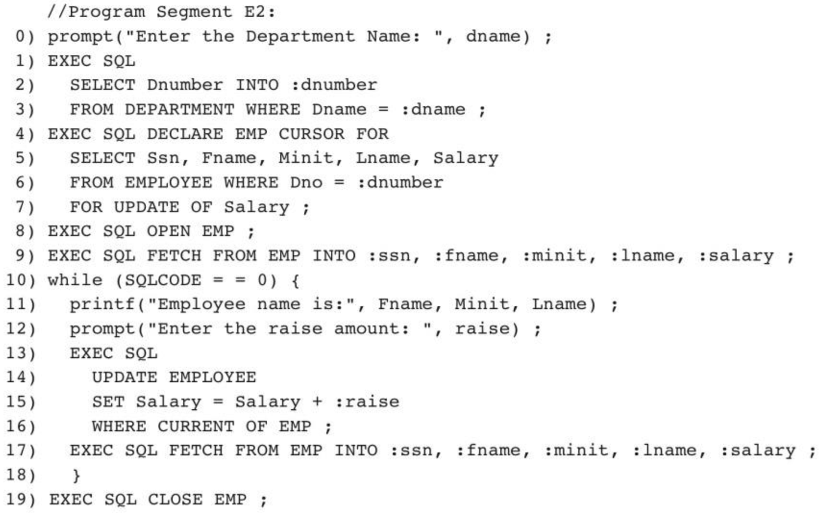


Embedded SQL

- `EXEC SQL` – **Preprocessor** separates embedded SQL statements from host language code
- **Shared variables -** Used in both the C program and the embedded SQL statements, Prefixed by a colon (:) in SQL statement
- Connecting to the database and terminate connection
- **SQLCODE** and **SQLSTATE** communication variables - Used by DBMS to communicate exception or error conditions
- Cursor
  - Points to a single tuple (row) from result of query, used to resolve *Impedance Mismatch*
  - **`OPEN CURSOR`** command and **`FETCH`** command

38

Figure 10.3 Program Segment E2, a C Program Segment That Uses Cursors with Embedded SQL for Update Purposes



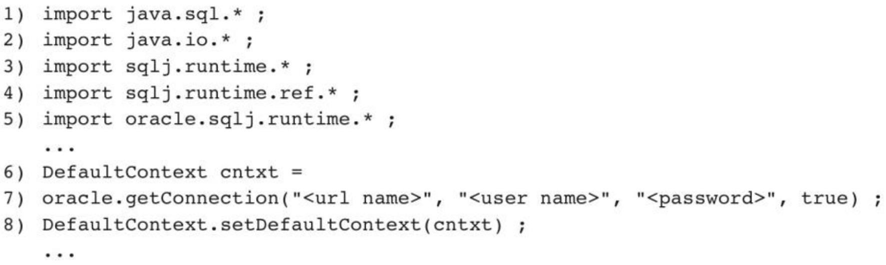
```
    //Program Segment E2:
    prompt("Enter the Department Name: ", dname) ;
    EXEC SQL
      SELECT Dnumber INTO :dnumber
      FROM DEPARTMENT WHERE Dname = :dname ;
    EXEC SQL DECLARE EMP CURSOR FOR
      SELECT Ssn, Fname, Minit, Lname, Salary
      FROM EMPLOYEE WHERE Dno = :dnumber
      FOR UPDATE OF Salary ;
    EXEC SQL OPEN EMP ;
    EXEC SQL FETCH FROM EMP INTO :ssn, :fname, :minit, :lname, :salary ;
    while (SQLCODE = = 0) {
      printf("Employee name is:", Fname, Minit, Lname) ;
      prompt("Enter the raise amount: ", raise) ;
      EXEC SQL
        UPDATE EMPLOYEE
        SET Salary = Salary + :raise
        WHERE CURRENT OF EMP ;
      EXEC SQL FETCH FROM EMP INTO :ssn, :fname, :minit, :lname, :salary ;
      }
    EXEC SQL CLOSE EMP ;
```

Figure 10.4 Program Segment E3, a C Program Segment That Uses Dynamic SQL for Updating a Table

```
    //Program Segment E3:
    EXEC SQL BEGIN DECLARE SECTION ;
    varchar sqlupdatestring [256] ;
    EXEC SQL END DECLARE SECTION ;
    . . .
    prompt("Enter the Update Command: ", sqlupdatestring) ;
    EXEC SQL PREPARE sqlcommand FROM :sqlupdatestring ;
    EXEC SQL EXECUTE sqlcommand ;
    . . .
```

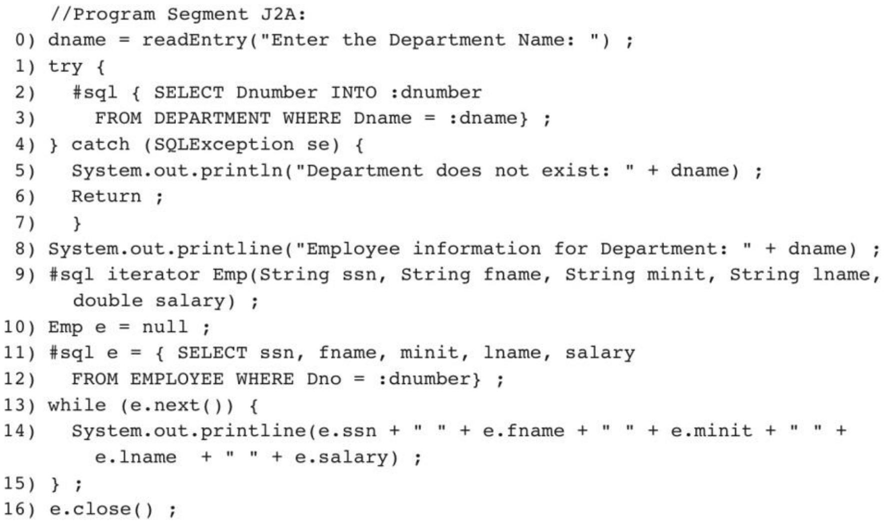
SQLJ: Embedding SQL Commands in Java

- Standard adopted by several vendors for embedding SQL in Java
- Import several class libraries
- **Default context**
- Uses **exceptions** for error handling
  - `SQL Exception` is used to return errors or exception conditions
- Example: Importing Classes Needed for Including SQLJ in Java Programs in Oracle, and Establishing a Connection and Default Context



```
import java.sql.* ;
import java.io.* ;
import sqlj.runtime.* ;
import sqlj.runtime.ref.* ;
import oracle.sqlj.runtime.* ;
...
DefaultContext cntxt =
oracle.getConnection("<url name>", "<user name>", "<password>", true) ;
DefaultContext.setDefaultContext(cntxt) ;
...
```

Figure 10.8 Program Segment J2A, a Java Program Segment That Uses a Named Iterator to Print Employee Information in a Particular Department



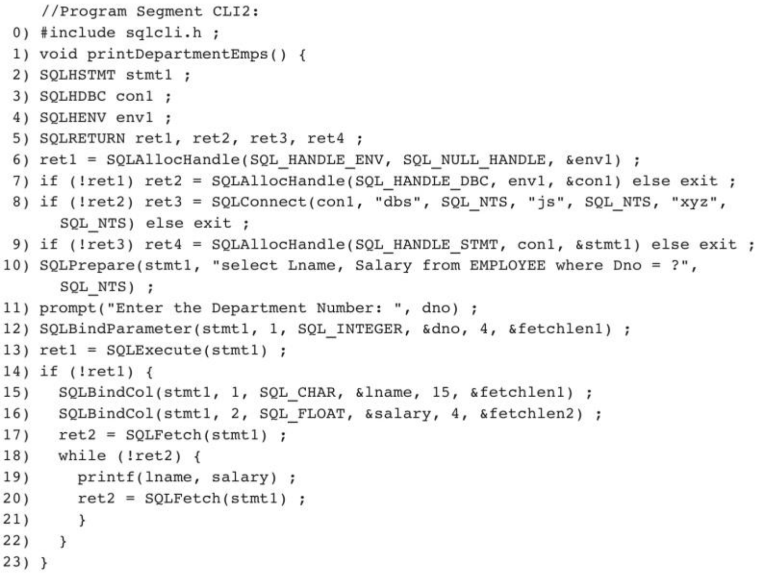
```
//Program Segment J2A:
dname = readEntry("Enter the Department Name: ") ;
try {
  #sql { SELECT Dnumber INTO :dnumber
    FROM DEPARTMENT WHERE Dname = :dname} ;
} catch (SQLException se) {
  System.out.println("Department does not exist: " + dname) ;
  Return ;
  }
System.out.printline("Employee information for Department: " + dname) ;
#sql iterator Emp(String ssn, String fname, String minit, String lname,
  double salary) ;
Emp e = null ;
#sql e = { SELECT ssn, fname, minit, lname, salary
  FROM EMPLOYEE WHERE Dno = :dnumber} ;
while (e.next()) {
  System.out.printline(e.ssn + " " + e.fname + " " + e.minit + " " +
    e.lname  + " " + e.salary) ;
} ;
e.close() ;
```

Library of Function Calls Approach

- SQL/CLI & JDBC
- Use of function calls
  - **Dynamic** approach for database programming
- Library of functions
  - Also known as **application programming interface (API)**
  - Used to access database
- **SQL Call Level Interface (SQL/CLI)** - Part of SQL standard
- More flexibility
- More complex programming
- No checking of syntax done at compile time

Pearson

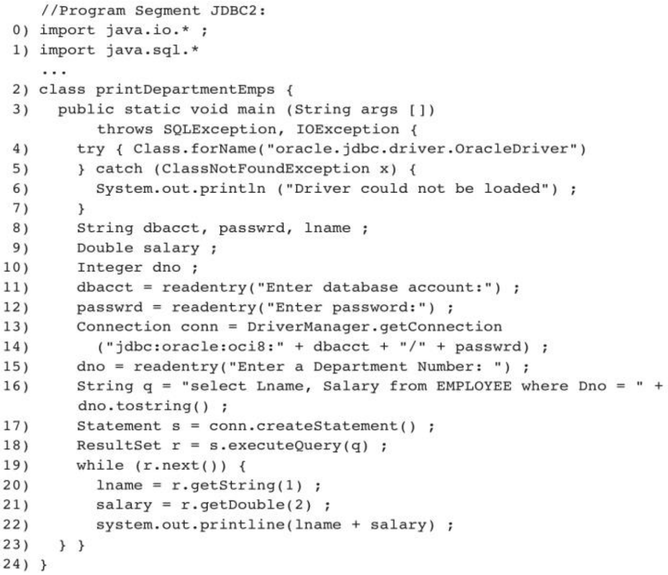
Figure 10.11 Program Segment CLI2, a C Program Segment That Uses SQL/CLI for a Query with a Collection of Tuples in Its Result



```
//Program Segment CLI2:
#include sqlcli.h ;
void printDepartmentEmps() {
SQLHSTMT stmt1 ;
SQLHDBC con1 ;
SQLHENV env1 ;
SQLRETURN ret1, ret2, ret3, ret4 ;
ret1 = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env1) ;
if (!ret1) ret2 = SQLAllocHandle(SQL_HANDLE_DBC, env1, &con1) else exit ;
if (!ret2) ret3 = SQLConnect(con1, "dbs", SQL_NTS, "js", SQL_NTS, "xyz",
  SQL_NTS) else exit ;
if (!ret3) ret4 = SQLAllocHandle(SQL_HANDLE_STMT, con1, &stmt1) else exit ;
SQLPrepare(stmt1, "select Lname, Salary from EMPLOYEE where Dno = ?",
  SQL_NTS) ;
prompt("Enter the Department Number: ", dno) ;
SQLBindParameter(stmt1, 1, SQL_INTEGER, &dno, 4, &fetchlen1) ;
ret1 = SQLExecute(stmt1) ;
if (!ret1) {
  SQLBindCol(stmt1, 1, SQL_CHAR, &lname, 15, &fetchlen1) ;
  SQLBindCol(stmt1, 2, SQL_FLOAT, &salary, 4, &fetchlen2) ;
  ret2 = SQLFetch(stmt1) ;
  while (!ret2) {
    printf(lname, salary) ;
    ret2 = SQLFetch(stmt1) ;
    }
  }
}
```

Pearson

Figure 10.13 Program Segment JDBC2, a Java Program Segment That Uses JDBC for a Query with a Collection of Tuples in Its Result



```
//Program Segment JDBC2:
import java.io.* ;
import java.sql.*
...
class printDepartmentEmps {
  public static void main (String args [])
      throws SQLException, IOException {
    try { Class.forName("oracle.jdbc.driver.OracleDriver")
    } catch (ClassNotFoundException x) {
      System.out.println ("Driver could not be loaded") ;
    }
    String dbacct, passwrd, lname ;
    Double salary ;
    Integer dno ;
    dbacct = readentry("Enter database account:") ;
    passwrd = readentry("Enter password:") ;
    Connection conn = DriverManager.getConnection
      ("jdbc:oracle:oci8:" + dbacct + "/" + passwrd) ;
    dno = readentry("Enter a Department Number: ") ;
    String q = "select Lname, Salary from EMPLOYEE where Dno = " +
    dno.tostring() ;
    Statement s = conn.createStatement() ;
    ResultSet r = s.executeQuery(q) ;
    while (r.next()) {
      lname = r.getString(1) ;
      salary = r.getDouble(2) ;
      system.out.printline(lname + salary) ;
  } }
}
```

Database Programming Language Approach

- **Stored procedures**
  - Program modules stored by the DBMS at the database server
  - Can be functions or procedures
- SQL/PSM (**SQL/Persistent Stored Modules**)
  - Extensions to SQL
  - Include general-purpose programming constructs in SQL
- Does not suffer from the impedance mismatch problem
- Programmers must learn a new language

SQL/PSM: Extending SQL for Specifying Persistent Stored Modules

- Conditional branching statement:

```
IF <condition> THEN <statement list>
ELSEIF <condition> THEN <statement list>
...
ELSEIF <condition> THEN <statement list>
ELSE <statement list>
END IF ;
```

- Constructs for looping

```
WHILE <condition> DO
        <statement list>
END WHILE ;
REPEAT
        <statement list>
UNTIL <condition>
END REPEAT ;

FOR <loop name> AS <cursor name> CURSOR FOR <query> DO
        <statement list>
END FOR ;
```

Pearson

Copyright

Pearson