Ders # 7

Introduction to SQL Programming Techniques

From Elmasri/Navathe textbook Ch9,26
Sciore textbook, Ch 9-10
Outline:

- Database Programming Approaches
- Embedded SQL
- JDBC
- Stored Procedures, SQL/PSM
- PHP
- Summary

Objective:

Various techniques for accessing and manipulating a database via programs in general-purpose languages s.a. Java, C, etc.

- Modern application programming architectures and techniques s.a. JAVA EE.
Database Programming

- **Objective:**
  - To access a database from an application program (as opposed to interactive interfaces)

- **Why?**
  - An interactive interface is convenient but not sufficient
    - A majority of database operations are made thru application programs (increasingly thru web applications)
Database Programming Approaches

1. **Embedded commands:**
   - Database commands are embedded in a general-purpose programming language

2. **Library of database functions:**
   - Available to the host language for database calls; known as an API
     - API standards for Application Program Interface

3. **A brand new, full-fledged language**
   - Minimizes impedance mismatch
   - **impedance mismatch:** Incompatibilities between a host programming language and the database model, e.g.,
     - type mismatch and incompatibilities; requires a new binding for each language
     - set vs. record-at-a-time processing
       - need special iterators to loop over query results and manipulate individual values
Basic Steps in Database Programming

I. Client program *opens a connection* to the database server

II. Client program *submits queries to and/or updates* the database

III. When database access is no longer needed, client program *closes (terminates) the connection*
1-) Embedded SQL

- Most SQL statements can be embedded in a general-purpose host programming language such as ADA, COBOL, C, Java
- An embedded SQL statement is distinguished from the host language statements by enclosing it between `EXEC SQL` or `EXEC SQL BEGIN` and a matching `END-EXEC` or `EXEC SQL END` (or semicolon)
  - Syntax may vary with language
  - *Shared variables* (used in both languages) usually prefixed with a colon (:) in SQL; used without (:) in the host program.
Variable Declaration:

- Variables inside `DECLARE` are shared and can appear (while prefixed by a colon) in SQL statements.
- `SQLCODE` is used to communicate errors/exceptions between the database and the program.

```sql
int loop;
EXEC SQL BEGIN DECLARE SECTION;
    varchar dname[16], fname[16], lname[16], ...;
    char ssn[10], bdate[11], ...;
    float salary, raise;
    int dno, dnumber, SQLCODE, ...;
EXEC SQL END DECLARE SECTION;
```

Connecting to a Database:

- Connection (multiple connections are possible but only one is active)
  ```sql
  CONNECT TO server-name AS connection-name
  AUTHORIZATION user-account-info;
  ```
- Change from an active connection to another one
  ```sql
  SET CONNECTION connection-name;
  ```
- Disconnection
  ```sql
  DISCONNECT connection-name;
  ```
Example 1: retrieving single tuple

```c
loop = 1;
while (loop) {
    prompt (“Enter SSN: “, ssn);
    EXEC SQL
    select FNAME, LNAME, ADDRESS, SALARY
    into :fname, :lname, :address, :salary
    from EMPLOYEE where SSN == :ssn;
    if (SQLCODE == 0) printf(fname, ...);
    else printf(“SSN does not exist: “, ssn);
    prompt(“More SSN? (1=yes, 0=no): “, loop);
    EXEC END-EXEC
}
```
Example 2: Retrieving multiple tuples

- A cursor (iterator) is needed to process multiple tuples
- FETCH commands move the cursor to the next tuple
- CLOSE CURSOR indicates that the processing of query results has been completed

```
// Program Segment E2:
0) prompt("Enter the Department Name: " dname)
1) EXEC SQL
2) select DNUMBER into :dnumber
3) from DEPARTMENT where DNAME = :dname ;
4) EXEC SQL DECLARE EMP CURSOR FOR
5) select SSN, FNAME, MINIT, LNAME, SALARY
6) from EMPLOYEE where DNO = :dnumber
7) FOR UPDATE OF SALARY ;
8) EXEC SQL OPEN EMP ;
9) EXEC SQL FETCH from EMP into :ssn, :fname, :minit, :lname, :salary
10) while (SQLCODE == 0) {
11)   printf("Employee name is: ", fname, minit, lname)
12)   prompt("Enter the raise amount: ")
13)   EXEC SQL
14)   update EMPLOYEE
15)   set SALARY = SALARY + :raise
16)   where CURRENT OF EMP ;
17)   EXEC SQL FETCH from EMP into :ssn, :fname, :minit, :lname, :salary
18) }
19) EXEC SQL CLOSE EMP ;
```
Dynamic SQL

- Objective:
  - Composing and executing new (not previously compiled) SQL statements at run-time
    - a program accepts SQL statements from the keyboard at run-time
    - a point-and-click operation translates to certain SQL query
  - Dynamic update is relatively simple; dynamic query can be complex
    - because the type and number of retrieved attributes are unknown at compile time

- Example:

```sql
EXEC SQL BEGIN DECLARE SECTION;
  varchar sqlupdatestring[256];
EXEC SQL END DECLARE SECTION;

... prompt ("Enter update command:", sqlupdatestring);
EXEC SQL PREPARE sqlcommand FROM :sqlupdatestring;
EXEC SQL EXECUTE sqlcommand;
```

- No syntax check or other types of checks are possible at compile time.
- Unable to know the type or number of attributes to be retrieved by the SQL query at compile time.
- PREPARE is useful in case the dynamic SQL is to be executed in the code repeatedly.
Embedded SQL in Java: SQLJ

- SQLJ: a **standard** for embedding SQL in Java
- An SQLJ translator converts SQL statements into Java
  - These are executed thru the *JDBC* interface
- Certain classes have to be imported. e.g., `java.sql`
- Example: *Establishing a connection*
  1) import java.sql.* ;
  2) import java.io.* ;
  3) import sqlj.runtime.*
  4) import sqlj.runtime.ref.*
  5) import oracle.sqlj.runtime.*
  6) DefaultContext cntxt =
  7) oracle.getConnection("<url name>", "<user name>", "<password>", true)
  8) DefaultContext.setDefaultContext(cntxt);
Example 1: *retrieving single tuple*

```java
string dname, ssn, fname, fn, lname, ln, bdate, address
char minit, mi;
double salary, sal;
integer dna, dnumber;
ssn = readEntry("Enter a Social Security Number: ")
try {
    #sql{select FNAME, MINIT, LNAME, ADDRESS, SALARY
            into :fname, :minit, :lname, :address, :salary
            from EMPLOYEE where SSN = :ssn} ;
} catch (SQLException se) {
    System.out.println("Social Security Number does not exist: "+ ssn)
    Return ;
}
System.out.println(fname + " " + minit + " " + lname + " " + address + " " + salary)
```
Example 2: retrieving multiple tuples with named iterator

SQLJ supports two types of iterators:

- **named iterator**: associated with a query result
- **positional iterator**: lists only attribute types in a query result

A **FETCH** operation retrieves the next tuple in a query result:

```sql
fetch iterator variable into program-variable
```

```java
dname = readEntry("Enter the Department Name: ")
try {
    #sql{select DNUMBER into :dnumber
        from DEPARTMENT where DNAME = :dname}
} catch CSQLException se) {
    System.out.println("Department does not exist: " + dname)
    Return ;
}
System.out.println("Employee information for Department: " + dname)
#sql iterator Emp(String ssn, String fname, String mlnlt, String lname ,double salary)
Emp e = null ;
#sql e = {select ssn, fname, mlnlt, lname, salary
    from EMPLOYEE where DNO :dnumber}
while (e.next()) {
    System.out.println(e.ssn + " " + e.fname + " " + e.minit + " " + e.lname + " " + e.salary)
};
e.close() ;
```
Example 3: *retrieving multiple tuples with positional iterator*

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.println("Employee information for Department: " + dname)
#sql iterator Emppos(String, String, String, String, String, double)
Emppos e = null ;
#sql e ={select ssn, fname, minit, lname, salary
    from EMPLOYEE where DNO = : dnumber} ;
#sql {fetch :e into :ssn, :fn, :mi, :ln, :sal}
while (!e.endFetch()) {
    System.out.println(ssn + " " + fn + " " + mi + " " + ln + " " + sal)
    #sql {fetch :e into :ssn, :fn, :mi, :ln, :sal}
};
e.close();
2-) Database Programming with Functional Calls

- **Embedded SQL** provides static database programming

- **API**: Dynamic database programming with a library of functions
  - **Advantage:**
    - No preprocessor needed (thus more flexible)
  - **Disadvantage:**
    - SQL syntax checks to be done at run-time
    - requires more complex programming to access query results because the types and numbers of attributes in a query result may not be known in advance.

- **Example:**
  - **SQL/CLI**
    - A part of the SQL standard
    - Provides easy access to several databases within the same program
    - Certain libraries (e.g., `sqlcli.h` for C) have to be installed and available
    - SQL statements are dynamically created and passed as string parameters in the calls
  - **JDBC**
    - SQL connection function calls for Java programming
    - A Java program with JDBC functions can access any relational DBMS that has a JDBC driver
    - JDBC allows a program to connect to several databases (known as data sources)
Steps in JDBC Database Access:

Driver \(\rightarrow\) Connection \(\rightarrow\) Statement \(\rightarrow\) ResultSet

1. Import JDBC library \((\text{java.sql}.*\)) and Load JDBC driver:
   - \text{Class.forName(“oracle.jdbc.driver.OracleDriver”)}
   - in the command line:
     - \text{-Djdbc.drivers = oracle.jdbc.driver}

2. Define appropriate variables and Create a connect object (via \text{getConnection})

3. Create a statement object from the \text{Statement} class:
   - \text{PreparedStatement}
     - Identify statement parameters (designated by question marks)
     - Bound parameters to program variables
     - Execute SQL statement (referenced by an object) via JDBC’s \text{executeQuery}
   - \text{CallableStatement}

6. Process query results (returned in an object of type ResultSet)
   - \text{ResultSet} is a 2-dimenional table
Example 1 (retrieving single tuple)

```java
import java.io.*;
import java.sql.*;
class getEmplInfo {
    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, ssn, lname;
        Double salary;

        dbacct = readentry("Enter database account:");
        passwrd = readentry("Enter password:");
        String stmt1 = "select LNAME, SALARY from EMPLOYEE where SSN = ?";
        PreparedStatement p = conn.prepareStatement(stmt1);
        ssn = readentry("Enter a Social Security Number:");
        p.clearParameters();
        p.setString(1, ssn);
        ResultSet r = p.executeQuery();
        while (r.next()) {
            lname = r.getString(1);
            salary = r.getDouble(2);
            System.out.println(lname + salary);
        }
    }
}
```
Example2 (retrieving multiple tuples)

```java
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()) {
            name = r.getString(1);
            salary = r.getDouble(2);
            System.out.println(lname + salary);
        }
    }
}
```
3-Database Stored Procedures

- Persistent procedures/functions (modules) are stored locally and executed by the database server
  - As opposed to execution by clients

Advantages:
- If the procedure is needed by many applications, it can be invoked by any of them (thus reduce duplications)
- Execution by the server reduces communication costs
- Enhance the modeling power of views

Disadvantages:
- Every DBMS has its own syntax and this can make the system less portable
Stored Procedure Constructs

- **SQL/PSM:**
  - Part of the SQL standard for writing persistent stored modules
- **SQL + stored procedures/functions + additional programming constructs**
  - E.g., branching and looping statements
  - Enhance the power of SQL
- **A stored procedure**
  ```sql
  CREATE PROCEDURE procedure-name (params)
  local-declarations
  procedure-body;
  ```
- **A stored function**
  ```sql
  CREATE FUNCTION fun-name (params) RETURNS return-type
  local-declarations
  function-body;
  ```
- **Calling a procedure or function**
  ```sql
  CALL procedure-name/fun-name (arguments);
  ```
CREATE FUNCTION DEPT_SIZE (IN deptno INTEGER)
RETURNS VARCHAR[7]
DECLARE TOT_EMPS INTEGER;

SELECT COUNT (*) INTO TOT_EMPS
    FROM SELECT EMPLOYEE WHERE DNO = deptno;
IF TOT_EMPS > 100 THEN RETURN “HUGE”
ELSEIF TOT_EMPS > 50 THEN RETURN “LARGE”
ELSEIF TOT_EMPS > 30 THEN RETURN “MEDIUM”
ELSE RETURN “SMALL”
ENDIF;
EXAMPLE#2: Stock management

Here is the part of stock tracking db for this example:

```sql
create table item
(
    item_id serial,
    description varchar(64) not null,
    cost_price numeric(7,2),
    sell_price numeric(7,2),
    CONSTRAINT item_pk PRIMARY KEY(item_id)
); 
create table stock
(
    item_id integer not null,
    quantity integer not null,
    CONSTRAINT stock_pk PRIMARY KEY(item_id),
    CONSTRAINT stock_item_id_fk FOREIGN KEY(item_id)
    REFERENCES item(item_id)
); 
```
Check stock condition w/ a function

create table reorders
(
    item_id integer,
    message text
);

-- reorders
-- scan the stock table to raise re orders of item low on stock

create function reorders(min_stock int4) returns integer as $$
declare
    reorder_item integer;
    reorder_count integer;
    stock_row stock%rowtype;
    msg text;
begin
    select count(*) into reorder_count from stock
    where quantity <= min_stock;
    for stock_row in select * from stock
    where quantity <= min_stock
    loop
        declare
            item_row item%rowtype;
        begin
            select * into item_row from item
            where item_id = stock_row.item_id;
            msg = 'order more ' ||
            item_row.description || 's at ' ||
            to_char(item_row.cost_price,'99.99');
            insert into reorders
            values (stock_row.item_id, msg);
        end;
    end loop;
    return reorder_count;
end;
$$ language plpgsql;

Check stock condition w/ a trigger

create function reorder_trigger() returns trigger AS $$
declare
    mq integer;
    item_record record;
begin
    mq := tg_argv[0];
    raise notice 'in trigger, mq is %', mq;
    if new.quantity <= mq
    then
        select * into item_record from item
        where item_id = new.item_id;
        insert into reorders
        values (new.item_id, item_record.description);
    end if;
    return NULL;
end;
$$ language plpgsql;

create trigger trig_reorder
after insert or update ON stock
for each row
execute procedure reorder_trigger(3);
...additional tables are

create table orderinfo
(
    orderinfo_id serial,
    customer_id integer not null,
    date_placed date not null,
    date_shipped date,
    shipping numeric(7,2),
    CONSTRAINT orderinfo_pk PRIMARY KEY(orderinfo_id),
    CONSTRAINT orderinfo_customer_id_fk FOREIGN KEY(customer_id) REFERENCES customer(customer_id)
);

create table orderline
(
    orderinfo_id integer not null,
    item_id integer not null,
    quantity integer not null,
    CONSTRAINT orderline_pk PRIMARY KEY(orderinfo_id, item_id),
    CONSTRAINT orderline_orderinfo_id_fk FOREIGN KEY(orderinfo_id) REFERENCES orderinfo(orderinfo_id),
    CONSTRAINT orderline_item_id_fk FOREIGN KEY(item_id) REFERENCES item(item_id)
);
Example: (state what the following trigger does..)
create function customer_trigger() returns trigger AS $$
declare
    order_record record;
begin
select * into order_record from orderinfo
    where customer_id = old.customer_id
    and date_shipped is NULL;
if not found
then
    raise notice 'deletion allowed: no outstanding orders';
    raise notice 'old.customer_id is %', old.customer_id;
    return NULL;
for order_record in select * from orderinfo
    where customer_id = old.customer_id
loop
    delete from orderline
    where orderinfo_id = order_record.orderinfo_id;
end loop;
delete from orderinfo
    where customer_id = old.customer_id;

    return old;
else
    raise notice 'deletion aborted: outstanding orders present';
    return NULL;
end if;
end;
$$ language plpgsql;

create trigger trig_customer before delete on customer
for each row execute procedure customer_trigger();
Web Programming w/ PHP

- Overview
- Structured, semi-structured, and unstructured data
- PHP
- Example of PHP
- Basic features of PHP
- Overview of PHP Database programming
Overview

- Hypertext documents
  - Common method of specifying contents
  - Various languages
    - HTML (HyperText Markup Language)
      - Used for generating static web pages
    - XML (eXtensible Markup Language)
      - Standard for exchanging data over the web
    - PHP (PHP Hypertext Preprocessor {recursive acronym})
      - Dynamic web pages
Structured, semi-structured, and unstructured data

- **Structured data**
  - Strict format (predefined schema)
  - Disadv: In real world, not all data collected is structured
  - Ex: Information stored in DB

- **Semi-structured data**
  - Data may have certain structure but not all information collected has identical structure
  - No exact pre-defined schema but
    - Semi-structured data *(names of attributes, relationships, and classes)* is mixed in with its schema (self-describing data)
    - Can be displayed as a graph
  - Some attributes may exist in some of the entities of a particular type but not in others
  - Ex: XML

- **Unstructured data**
  - Very limited indication of data type
  - No schema information
    - E.g., a simple text document
    - HTML
Unstructured data:

- Limited indication of data types
  - E.g., web pages in html contain some unstructured data
- Figure shows part of HTML document representing unstructured data
- Difficult to interpret by computer programs BECAUSE no schema (type of data) information is known.
  - XML, conversely, provides easier interpretation and exchange Web documents b/w computers.

```
<html>
<head>
  ...
</head>
<body>
  <h1>List of company projects and the employees in each project</h1>
  <h2>The ProductX project</h2>
  <table width="100%" border=0 cellspacing=0>
    <tr>
      <td width="50%"><font size="2" face="Arial">John Smith</font></td>
      <td>32.5 hours per week</td>
    </tr>
    <tr>
      <td width="50%"><font size="2" face="Arial">Joyce English</font></td>
      <td>20.0 hours per week</td>
    </tr>
    <tr>
      <td width="50%"><font size="2" face="Arial">Franklin Wong</font></td>
      <td>10.0 hours per week</td>
    </tr>
  </table>
</body>
</html>
```

Figure 26.2
Part of an HTML document representing unstructured data.
PHP

- Open source
- General purpose scripting language
- Interpreter engine in C
  - Can be used on nearly all computer types
- Particularly suited for manipulation of text pages
- Manipulates (dynamic html) at the Web server
  - Conversely, JavaScript is downloaded and executed on the client
- dynamic html: Webs pages, where part of the info is extracted from databases are called dynamic web pages.
- Has libraries of functions for accessing databases
A simple PHP Example

Suppose the file containing program segment P1 is stored at www.myserver.com/example/greeting.php

```php
//Program Segment P1:
0) <?php
1) // Printing a welcome message if the user submitted their name
   // through the HTML form
2) if ($_POST['user_name']) {
3)     print("Welcome, ");
4)     print($_POST['user_name']);
5) }
6) else {
7)     // Printing the form to enter the user name since no name has
        // been entered yet
8)     print <<< _HTML_
9)     <FORM method="post" action="$_SERVER["PHP_SELF"]">
10)    Enter your name: <input type="text" name="user_name">
11)    <BR/>
12)    <INPUT type="submit" value="SUBMIT NAME">
13)    </FORM>
14) _HTML_;
15) }
16) ?>
```

![Figure 26.3](Slide 9-31)
(a) PHP program segment for entering a greeting,
(b) Initial form displayed by PHP program segment,
(c) User enters name John Smith,
(d) Form prints welcome message for John Smith.
Overview of basic features of PHP

- PHP variables, data types, and programming constructs
  - Variable names start with $ and can include characters, letters, numbers, and _.
    - No other special characters are permitted
    - Are case sensitive
    - Can’t start with a number
  - Variables are not types
    - Values assigned to variables determine their type
    - Assignments can change the type
  - Variable assignments are made by =
Overview of basic features of PHP

- PHP variables, data types, and programming constructs (contd.)
  - Main ways to express strings
    - Single-quoted strings (lines 0, 1, 2)
      - '\' represents a quote in a string
    - Double-quoted strings (line 7)
      - Variable names can be interpolated
    - Here documents (line 8-11)
      - Enclose a part of a document between `<<<DONMANE` and end it with a single line containing the document name DONAME
  - Single and double quotes
    - The quotes should be straight quotes ('') not (') or (')

```php
0) print 'Welcome to my Web site.';
1) print 'I said to him, "Welcome Home"';
2) print 'We\'ll now visit the next Web site';
3) printf('The cost is $%.2f and the tax is $%.2f', $cost, $tax);
4) print strtolower('AbCdE');
5) print ucwords(strtolower('JOHN smith'));
6) print 'abc' . 'efg'
7) print "send your email reply to: $email_address"
8) print <<<FORM_HTML
9) <FORM method="post" action="$_SERVER['PHP_SELF']">
10) Enter your name: <input type="text" name="user_name">
11) FORM_HTML
```

Figure 26.4 Illustrating basic PHP string and text values.
Overview of basic features of PHP

- PHP variables, data types, and programming constructs (contd.)
  - String operations
    - (.) is concatenate as in Line 6
    - (strtolower()) converts string into lower case
    - Others as needed
  - Numeric data types follows C rules

```php
0) print 'Welcome to my Web site.';
1) print 'I said to him, "Welcome Home"';
2) print 'We\'ll now visit the next Web site';
3) printf('The cost is %.2f and the tax is %.2f', $cost, $tax);
4) print strtolower('AbCdE');
5) print ucwords(strtolower('JOHN smith'));
6) print 'abc' . 'efg'
7) print "send your email reply to: $email_address"
8) print <<<FORM_HTML
9) <FORM method="post" action="$_SERVER[\'PHP_SELF\']"/>
10) Enter your name: <input type="text" name="user_name">
11) FORM_HTML
```

Figure 26.4

Illustrating basic PHP string and text values.
Overview of basic features of PHP

- PHP variables, data types, and programming constructs (contd.)
  - Other programming constructs similar to C language constructs
    - for-loops
    - while-loops
    - if-statements
  - Boolean logic
    - True/false is equivalent no non-zero/zero
    - Comparison operators
      - ==, !=, >, >=, <, <=

- PHP Arrays
  - Allow a list of elements
  - Can be 1-dimensional or multi-dimensional
  - Can be **numeric** or **associative**
    - Numeric array is based on a numeric index
    - Associative array is based on a key => value relationship
  - Line 0: $teaching is a associative array
    - Line 1 shows how the array can be updated/accessed
  - Line 5: $courses is a numeric array
    - No key is provided => numeric array
  - There are several ways of looping through arrays
    - Line 3 and 4 show “for each” construct for looping through each and every element in the array
    - Line 7 and 10 show a traditional “for loop” construct for iterating through an array
Overview of basic features of PHP

- PHP Functions
  - Code segment P1 in Figure 26.6 has two functions
    - `display_welcome()`
    - `display_empty_form()`
  - Line 14-19 show how these functions can be called

```php
//Program Segment P1:
0) function display_welcome() {
1)   print("Welcome, ");
2)   print($_POST['user_name']);
3) }
4)
5) function display_empty_form(); {
6)   print <<< HTML
7)   <FORM method="post" action="$_SERVER['PHP_SELF']"><BR/>
8)   Enter your name: <INPUT type="text" name="user_name">
9) }<BR/>
10) <INPUT type="submit" value="Submit name" HTML;>
11) 
12) if ($_POST['user_name']) {
13)   display_welcome();
14} }
15) } else {
16)   display_empty_form();
17) }
0) function course_instructor ($course, $teaching_assignments) {
1)   if (array_key_exists($course, $teaching_assignments)) {
2)       $instructor = $teaching_assignments[$course];
3)       RETURN "$instructor is teaching $course";
4)   } else {
5)       RETURN "there is no $course course";
6)   }
7)   $teaching = array('Database' => 'Smith', 'OS' => 'Carrick',
8)       'Graphics' => 'Kam');
9)   $teaching['Graphics'] = 'Benson'; $teaching['Data Mining'] = 'Kam';
10)   $x = course_instructor('Database', $teaching);
11)   print($x);
12)   $x = course_instructor('Computer Architecture', $teaching);
13)   print($x);
14)   print($x);
```
Overview of basic features of PHP

- PHP Server Variables and Forms
  - There are a number of built-in entries in PHP function. Some examples are:
    - $_SERVER['SERVER_NAME']
      - This provides the Website name of the server computer where PHP interpreter is running
    - $_SERVER['REMOTE_ADDRESS']
      - IP address of client user computer that is accessing the server
    - $_SERVER['REMOTE_HOST']
      - Website name of the client user computer
    - $_SERVER['PATH_INFO']
      - The part of the URL address that comes after backslash (/) at the end of the URL
    - $_SERVER['QUERY_STRING']
      - The string that holds the parameters in the IRL after ?.
    - $_SERVER['DOCUMENT_ROOT']
      - The root directory that holds the files on the Web server
Overview of PHP Database Programming

- Connecting to the database
  - Must load PEAR DB library module DB.php
  - DB library functions are called using DB::<function_name>
  - The format for the connect string is:
    - <DBMS>://<userid>:<password>@<DBserver>
    - For example:
      - $d=DB::connect('oci8://ac1:pass12@www.abc.com/db1')
  - Line 10-12 shows how information collected via forms can be stored in the database.

```php
0) require 'DB.php';
1) $d = DB::connect('oci8://ac1:pass12@www.host.com/db1');
2) if (DB::isError($d)) { die("cannot connect — ". $d->getMessage()); }
   ...
3) $q = $d->query("CREATE TABLE EMPLOYEE
4)   (Emp_id INT,
5)   Name VARCHAR(15),
6)   Job VARCHAR(10),
7)   Dno INT") ;
8) if (DB::isError($q)) { die("table creation not successful — ". $q->getMessage()); }
   ...
9) $d->setErrorHandling(PEAR_ERROR_DIE);
   ...
Some code here to collect data from a form like P’ in previous slide, 47

10) $eid = $d->nextID('EMPLOYEE');
11) $q = $d->query("INSERT INTO EMPLOYEE VALUES
12)   ($eid, $_POST['emp_name'], $_POST['emp_job'], $_POST['emp_dno']) ");
   ...
13) $eid = $d->nextID('EMPLOYEE');
14) $q = $d->query("INSERT INTO EMPLOYEE VALUES (?, ?, ?, ?, ?)",
15) array($eid, $_POST['emp_name'], $_POST['emp_job'], $_POST['emp_dno']));
```

A way to prevent SQL injection.
Overview of PHP Database Programming

- Retrieval queries and Database tables
  - Lines 4-7 retrieves name and department number of all employee records
  - Lines 8-13 is a dynamic query (conditions based on user selection)
    - Values for these are entered through forms
  - Lines 14-17 is an alternative way of specifying a query and looping over its records
    - Function $d->getAll holds all the records in $allresult

```php
0) require 'DB.php';
1) $d = DB::connect('oci8://acct1:pass12@www.host.com/dbname');
2) if (DB::isError($d)) { die("cannot connect - ". $d->getMessage()); }
3) $d->setErrorHandler(PEAR_ERROR_DIE);
   ...
4) $q = $d->query('SELECT Name, Dno FROM EMPLOYEE');
5) while ($r = $q->fetchRow()) {
6)   print "employee $r[0] works for department $r[1] \n";
7) }
   ...
8) $q = $d->query('SELECT Name FROM EMPLOYEE WHERE Job = ? AND Dno = ?',
9)   array($_POST['emp_job'], $_POST['emp_dno']) );
10) print "employees in dept $_POST['emp_dno'] whose job is
    $_POST['emp_job']: \n"
11) while ($r = $q->fetchRow()) {
12)   print "employee $r[0] \n";
13) }
   ...
14) $allresult = $d->getAll('SELECT Name, Job, Dno FROM EMPLOYEE');
15) foreach ($allresult as $r) {
16)   print "employee $r[0] has job $r[1] and works for department $r[2] \n";
17) }
   ...
```

Figure 26.9
Illustrating database retrieval queries.
Summary

- Assertions provide a means to specify additional constraints.
- Triggers are assertions that define actions to be automatically taken when certain conditions occur.
- A database may be accessed in an interactive mode; Most often, however, data in a database is manipulate via application programs.
- Several methods of database programming:
  - Embedded SQL
  - Dynamic SQL
  - JDBC
  - Stored procedure and functions
  - Web Programming with PHP