8

Web Databases

Prerequisites for This Section

- **Readings:**
  - **Required:** Connolly and Begg, sections 29.2.5 - 29.2.8 and 29.3 - 29.6 (in third edition, sections 28.3 - 28.7 except 28.7.1).
  - **Required:** Connolly and Begg, sections 29.3.2 and 29.8 (in third edition, sections 28.4.2 and 28.9).
  - **Required:** Connolly and Begg, section 29.7 (in third edition, section 28.8).
  - **Required:** Connolly and Begg, section 8.2.5 – 8.2.7

Section Objectives

In this section you will learn:

① Several Client-Server architectures and Java-Based approaches.

② JDBC is the most prominent and mature approach for accessing relational DBMS from Java appears.

③ JDBC package defines a database access API that includes DriverManager, Connection, Statement, PreparedStatement, CallableStatement, ResultSet, DatabaseMetaData, ResultSetMetaData, SQLException and SQLWarning classes.

④ PL/SQL is Oracle's procedural extension to SQL. It has concepts similar to modern programming language, such as variable and constant declarations, control structure, and modularization.

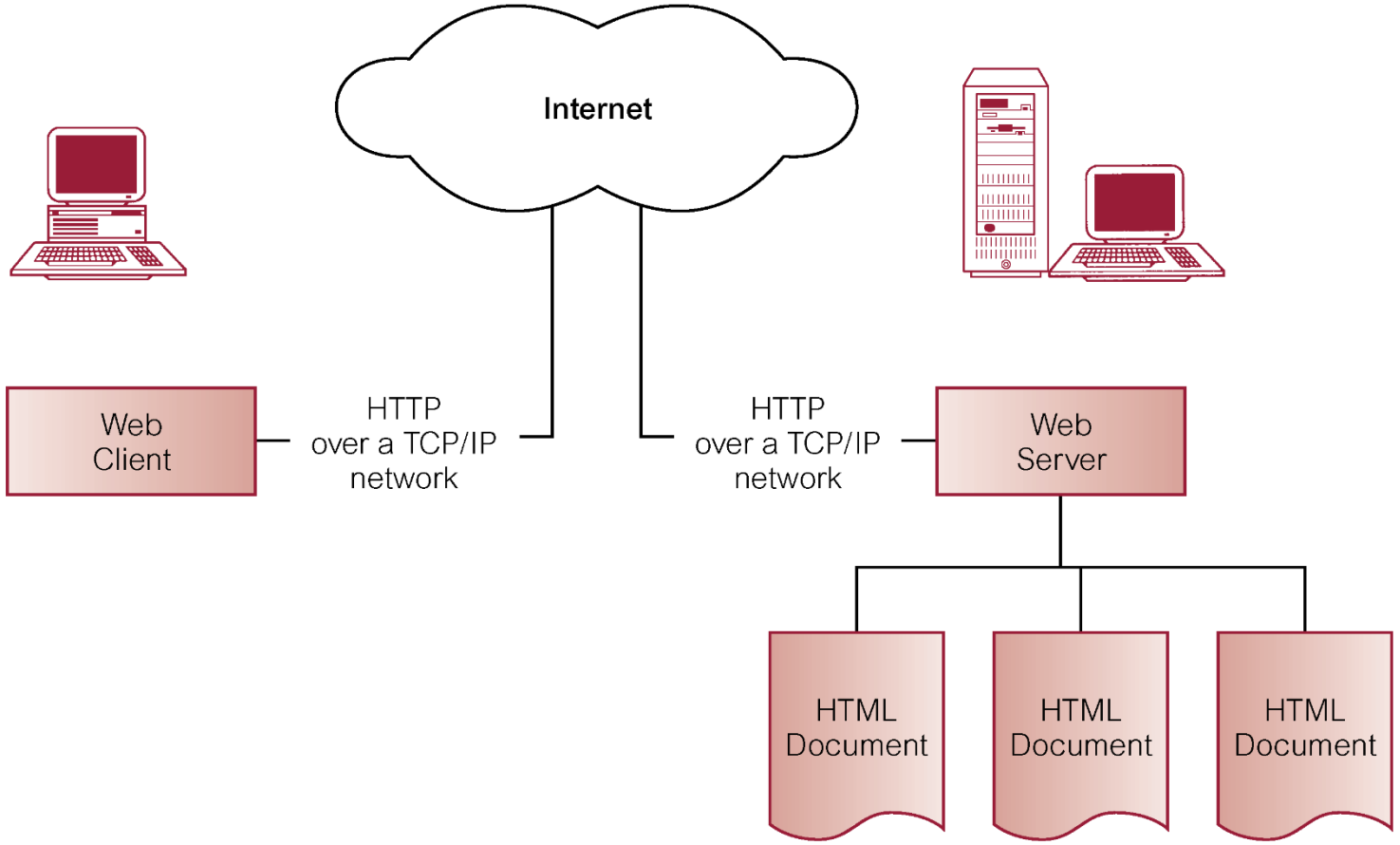
Agenda

1. Web Database Applications
2. JDBC (Java Database Connectivity)
3. PL/SQL

Web Environment

- The World Wide Web (Web for short) is a hypermedia-based system that provides a means of browsing information on the Internet in a non-sequential way using hyperlinks.
- Information on the Web is presented on Web pages.
- The Web consists of a network of computers that can act in two roles:
  - As servers, providing Web pages
  - As clients, usually referred to as browsers, requesting Web pages

Basic Components of Web Environment



Web Application

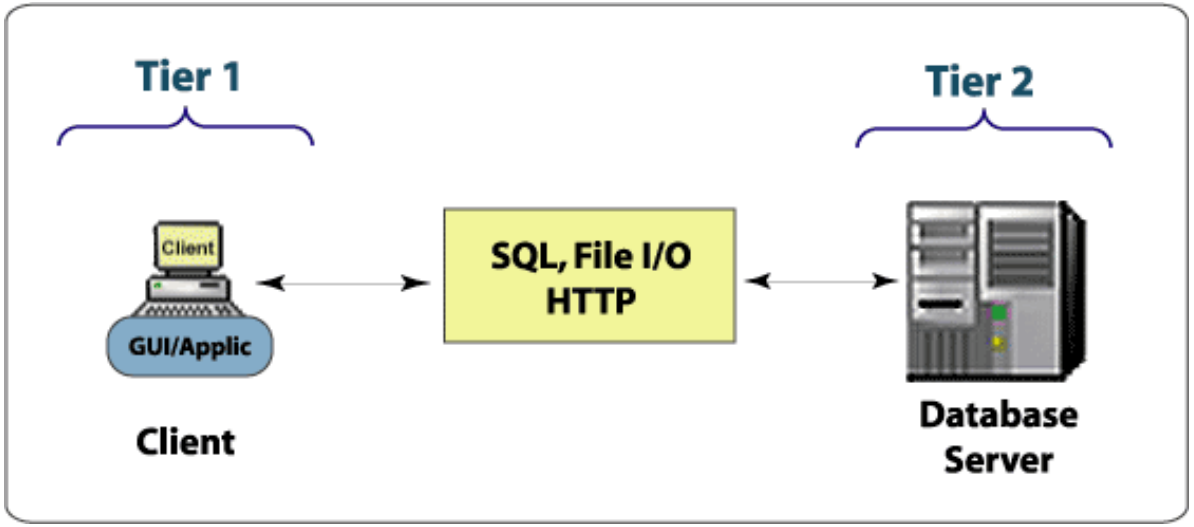
- As architecture of Web was designed to be platform-independent, can significantly lower deployment and training costs.
- Many Web sites today are file-based where each Web document is stored in separate files.
  - Significant management problems for large sites.
  - Many Web sites now contain more dynamic information.
- Accessing database directly from Web would be a better approach.

Database Application

- Data-intensive business applications consist of four components:
  - the database (DB)
  - the transaction logic
  - the application logic
  - the user interface
- In the mainframe environment, these four components were all in one place.
- To accommodate an increasingly decentralized business environment, the client-server system was developed.

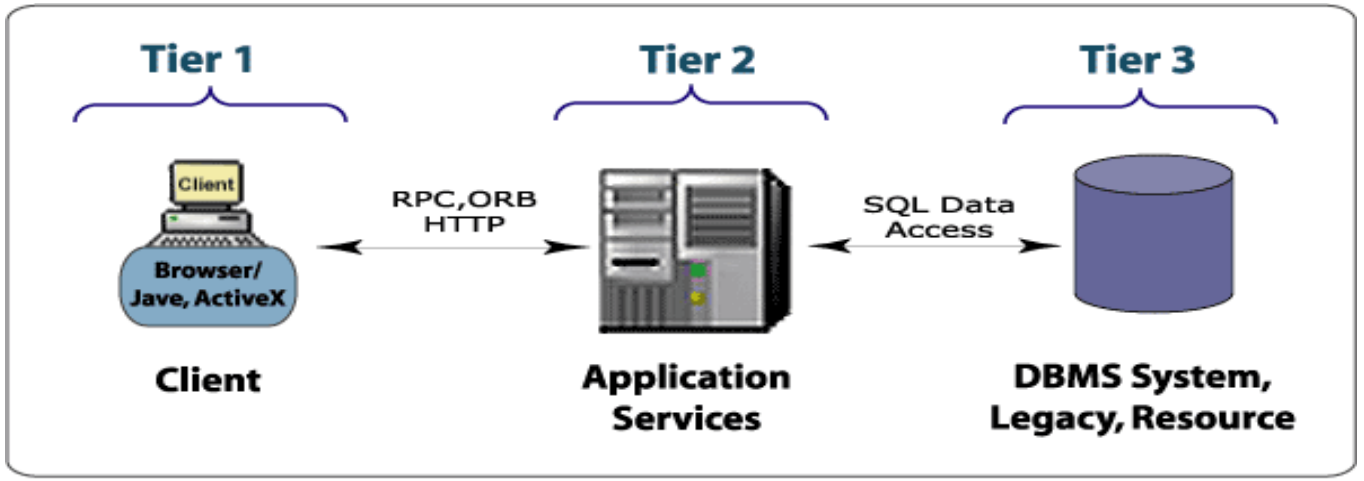
Two-Tier Client-Server Architecture

- In this two-tier architecture:
  - **Clients** are primarily responsible for executing the basic application programs that issue data requests to the server, process the data, and handle the presentation of data. (interface, application logic)
  - **Server** is a centralized provider for data and security services to the clients. (DB + transaction logic)

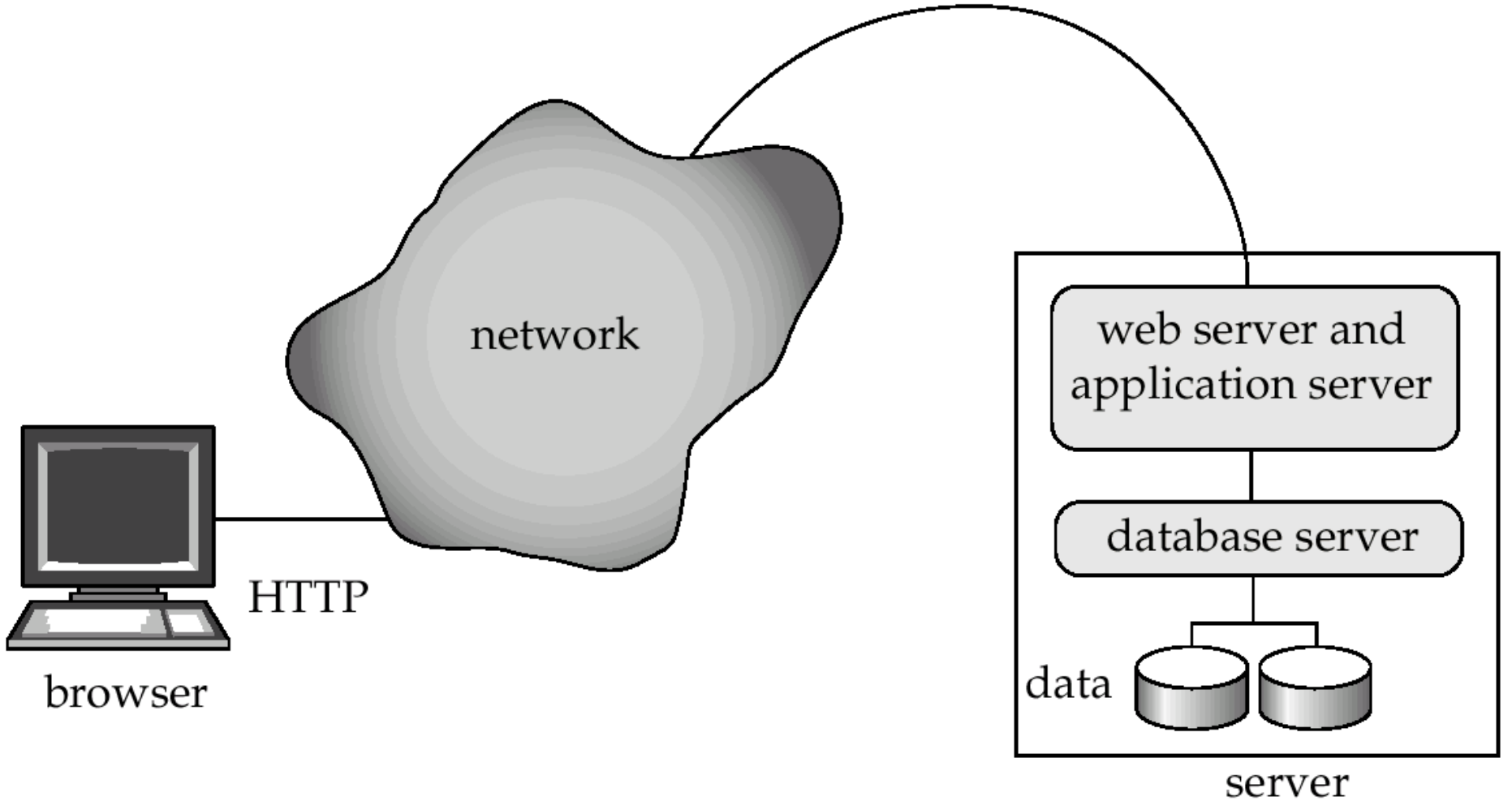


Three-Tier Client-Server Architecture

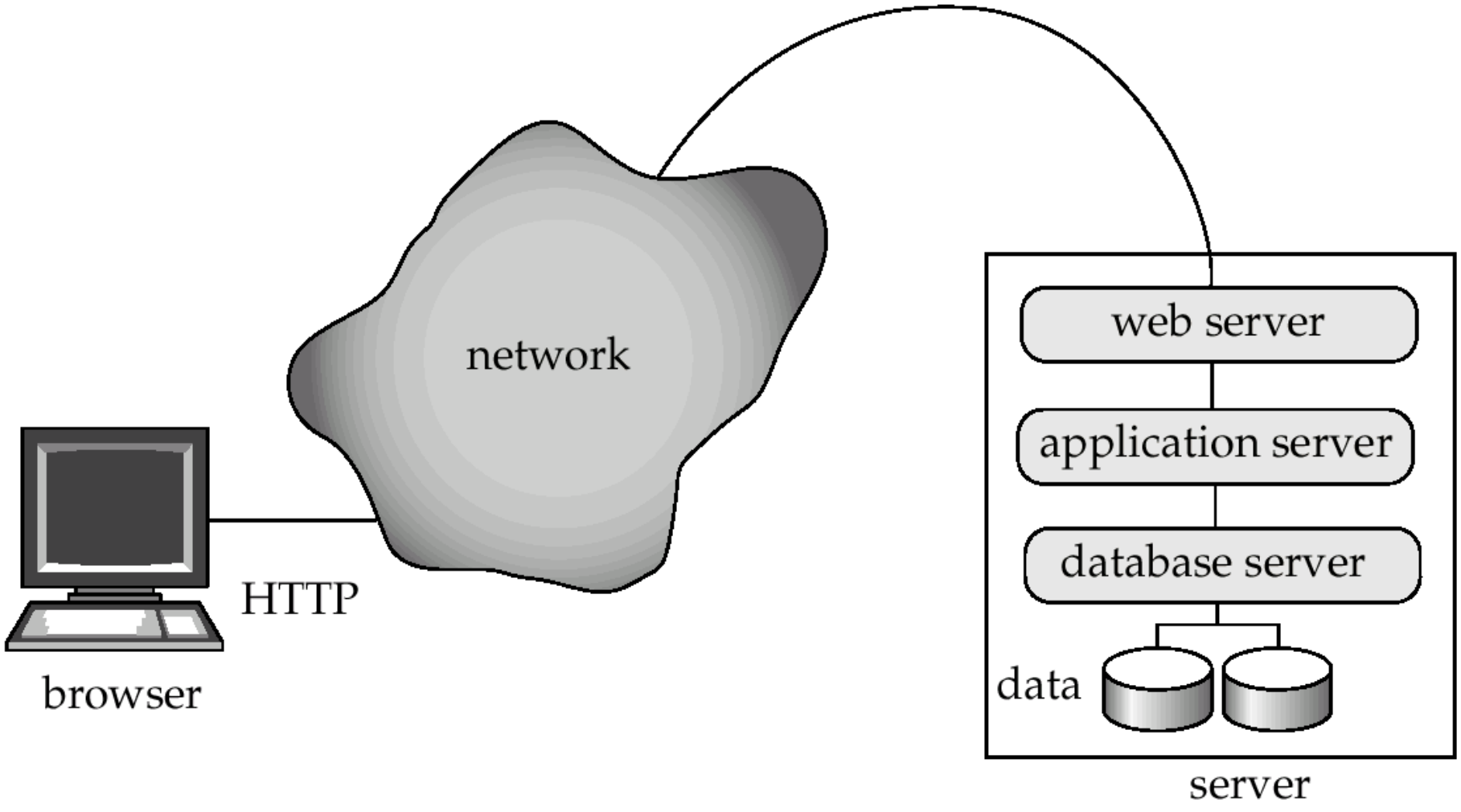
- The three tiers are:
  - a "thin" client who runs only the user interface on the end-user computer. (interface)
  - an application server that runs the business logic and processes the data. (application logic)
  - a database server. (DB, transaction logic)



Three-Tier Web Database Architecture



Four-Tier Web Database Architecture



Technologies of Web Database Application

① Scripting languages such as JavaScript and VBScript

② Common Gateway Interface (CGI)

③ Microsoft Web Platform: .NET, Active Server Pages (ASP), and ActiveX Data Objects (ADO)

④ JDBC

⑤ JSP+Java Servlet +Java Beans

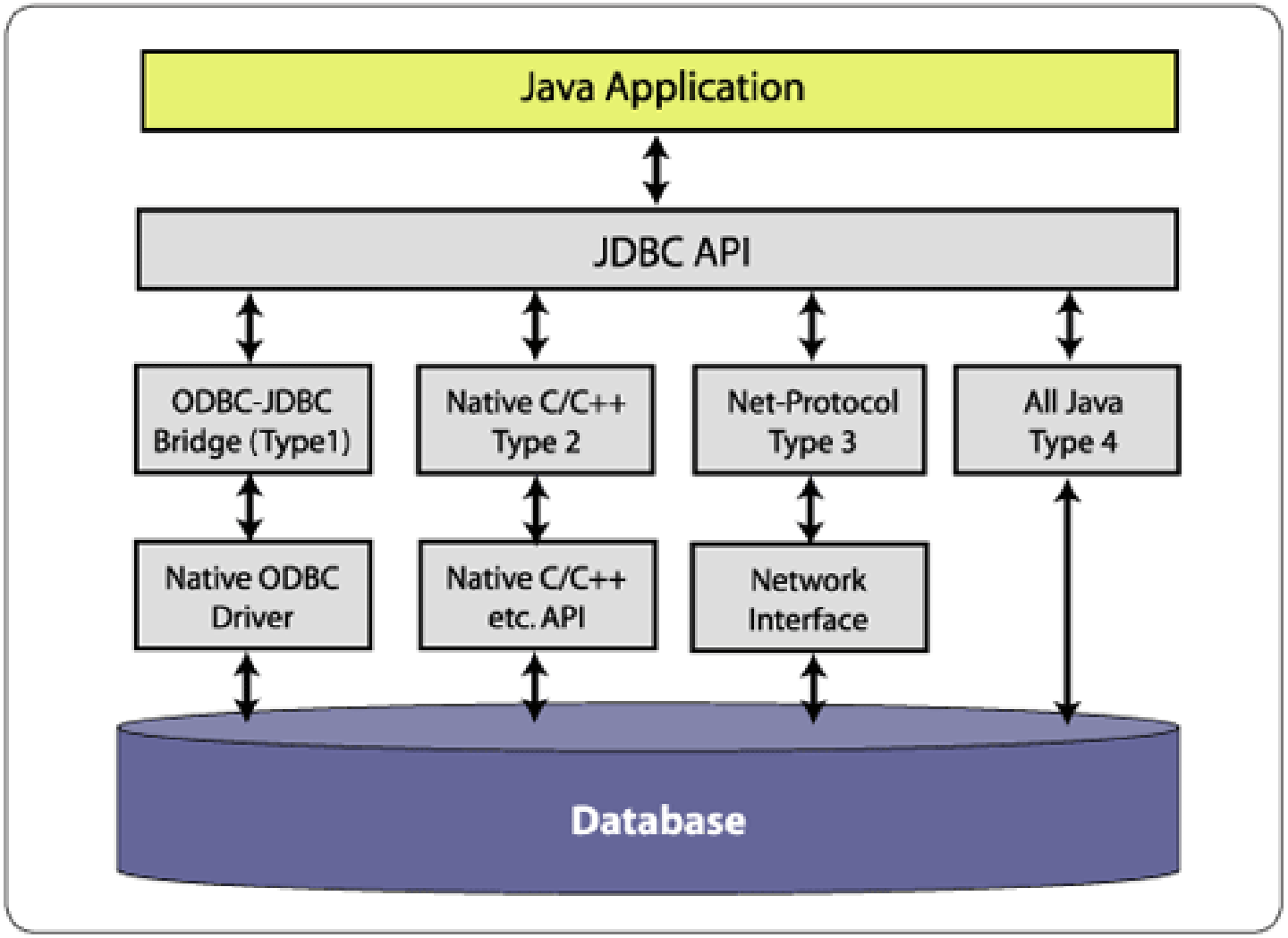
Agenda

1. Web Database Applications
2. JDBC (Java Database Connectivity)
3. PL/SQL

Java and JDBC

- JDBC (Java DataBase Connectivity) is an API that is part of the Java languages designed to access multiple DBMSs over the Web at different URLs.
- Modeled after ODBC, JDBC API supports basic SQL functionality.
- With JDBC, Java can be used as host language for writing database applications.
- JDBC resembles dynamic SQL, in which SQL statements are passed in the form of strings.

JDBC Drivers



JDBC Drivers

- Four different implementation types of DBMS-specific driver:

① **Type 1: JDBC-ODBC bridge**

This driver translates JDBC calls into ODBC calls.

② **Type 2: JDBC-Gateway**

This pure Java driver connects to a database middleware server that in turn interconnects multiple databases and performs any necessary translations.

③ **Type 3: Java JDBC Native Code**

This partial Java driver converts JDBC calls into client API for the DBMS.

④ **Type 4: Pure Java JDBC**

This driver connects directly to the DBMS.

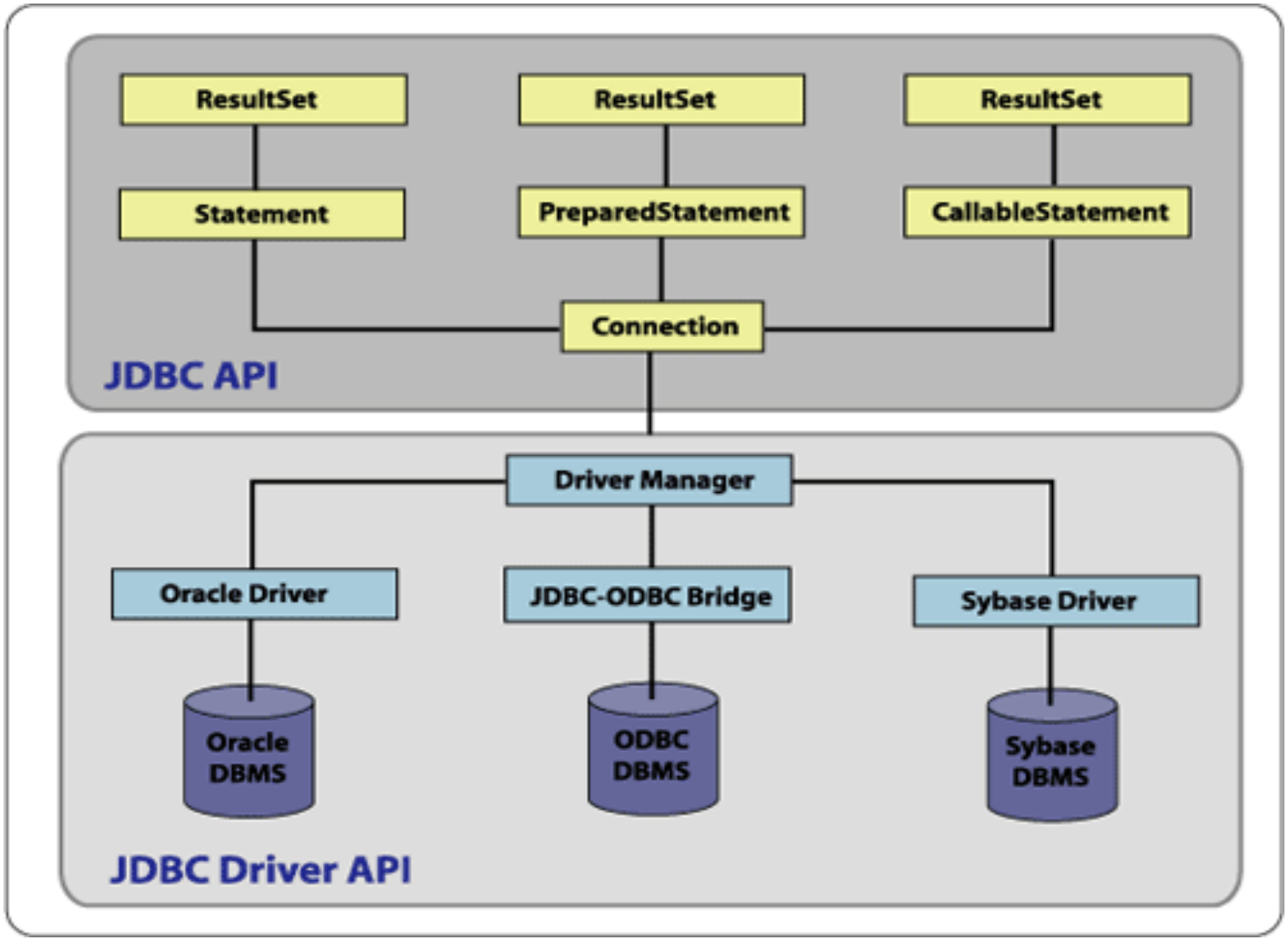
Load a JDBC Driver

- In each case, a Java program can access any database using the JDBC API as long as a number of conditions are met.
  - An appropriate, DBMS-specific driver must exist.
  - This driver must be loaded.
  - This driver must be registered with the driver manager.
- A driver can be loaded using the **Class.forName()**. But, before that, the program must import the Java.sql package that defines JDBC SQL dialect and methods.

```
import java.sql.*;

Class.forName("jdbc.driver_name");
```

JDBC Functions



Basic JDBC Functions

- Some basic classes that JDBC API defines:

① **DriverManager**

provides methods that manage a set of available JDBC drivers.

① **Connection**

represents the connection with the database.

① **Statement**

contains methods for executing a SQL statement.

① **ResultSet**

contains methods for accessing the results of an executed SQL statement.

Example of Java + JDBC

```
Connection con =
DriverManager.getConnection("jdbc:odbc:wombat","login", "password");

Statement stmt = con.createStatement();

ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Table1");

while (rs.next()) {
        int x = rs.getInt("a");
        string s = rs.getString("b");
        float f = rs.getFloat("c");
        }
```

JDBC Functions

- JDBC API defines classes:
  - ① **DriverManager**
  - ② **Connection**
  - ③ **Statement**
    - a) **Statement**: contains methods for executing a SQL statement.
    - b) **PreparedStatement**: represents an SQL statement that has been precompiled and stored for subsequent execution.
    - c) **CallableStatement**: contains methods to execute SQL stored procedures.
  - ④ **ResultSet**
  - ⑤ **DatabaseMetaData**： provides information about the database.
  - ⑥ **ResultSetMetaData**： contains details of a ResultSet.
  - ⑦ **SQLException** and **SQLWarning**: encapsulate database access errors and warnings.

Connection class -- Connecting to a DB

- **Connection class** can be used to establish a connection with a database at a specific URL, with a specific user ID. and password.

Connection dbcon
DriverManager.getConnection("URL","user_id","password");

- When getConnection is called, the driver manager will select the appropriate driver from the previously loaded JDBC drivers and establish the database connection. If the connection is successful, it returns a **Connection Object** that is assigned to dbcon.

- A database connection can be closed by invoking the close method:

dbcon.close().

Statement Class

- **Function: Statement class** is used to execute complete SQL statements without parameters.
- **Created by**: A **Statement object** is created by invoking the createStatement method on a previously established Connection object.

  Statement st = dbcon.createStatement();

- **Methods** of Statement object:

  ① **ExecuteQuery:** executes a SELECT statemen

  ② **ExecuteUpdate:** execute an UPDATE, DELETE, INSERT, or DDL statement. (These are equivalent to the EXECUTE IMMEDIATE dynamic SQL statements.)

Statement Class

- In general, an SQL statement can be stored in a **string variable** that can be used as the argument in the executeUpdate and executeQuery methods.
- The result of a query is stored in a **ResultSet** object that is declared when the query is executed.

Statement Class – executeUpdate()

- Consider an update:

  insert a new row into Branch.

int nrows = st.executeUpdate(“INSERT INTO Branch VALUES ('B009', NULL, 'London', 'SW1 4EH')");

- This method returns the number of rows affected.
- In our example, if 1 is returned, it means that the new row was inserted. If the update returns a 0, it means that the insert failed.

Statement Class – executeQuery()

- Consider the query that lists the first and last names of staff based on their first names. We will construct the query as a string using the concatenation operator + and store it in query1.

```
string fname = readString("Enter First Name: ");

string query1 = "SELECT fName, lName FROM Staff
    WHERE fName = '  " + fname + " '";

ResultSet res1 = st.executeQuery(query1);
```

- The rows in the ResultSet can be retrieved one at a time using a cursor.

PreparedStatement Class

- If the same SQL statement is executed many times, the **PreparedStatement** class can be used to pre-compile the SQL statement.
- This is equivalent to the PREPARE, EXECUTE, and USING statements in dynamic SQL.
- For example:

```
preparedStatement st2 = dbcon.prepareStatement ("SELECT
    fName, lName FROM Staff WHERE fName = ?");

string fname = readString("Enter First Name:");

st2.setString(1, fname);

ResultSet res2 = st2.executeQuery();
```

ResultSet Class and Cursor

- **Cursors** are allowed to retrieve one row at a time from a resulting table in a **ResultSet** object.
- JDBC provides the **getXXX** methods for retrieving column values of the row to which the cursor is currently pointing. Similar to **setXXX**, XXX is a valid type.
- For example:

```
string rfname, rlname;
while (res2.next()) {
        rfname = res2.getString(fName);
        rlname = res2.getString(2);
        system.out.print(rfname+" "+rlname);
};
```

Databases and Result Metadata

- Unlike embedded SQL, JDBC allows application programs to request **schema information** about a database (such as its tables) or about retrieved tables (such as field names, types, etc.).
- These two kinds of information are obtained using the **getMetaData** method, which stores them in a DatabaseMetaData object or a ResultSetMetaData object, respectively.
- For example,

```
ResultSet res3 = st.executeQuery("SELECT * FROM Branch");

ResultSetMetaData resmd = res3.getMetaData();
```

Executing Transactions

- Each JDBC statement is treated as a separate transaction that is automatically committed when it completes successfully. In order to allow two or more statements to execute as a single transaction, we need to disable the *autocommit* mode on the Connection object:

  dbcon.setautocommit(false);

- A new transaction automatically starts when the previous one is either committed or rolled back. For instance:

  dbcon.commit; or dbcon.rollback;

SQLException Class - Error Handling

- JDBC provides the **SQLException** class to deal with errors.
- For example:

```
try {       ResultSet res3 = st.executeQuery("SELECT * FROM Branch");
}
catch (SQLException e1) {
        System.out.println("SQL Error");
       while (e1 != null) {
       System.out.println("Message = "+ e1.getMessage());
       System.out.println("SQLState = "+ e1.getSQLstate());
       System.out.println("ErrorCode = "+ e1.getErrorCode());
       e1 = e1.getNextException();
      };
};
```

Agenda

1. Web Database Applications
2. JDBC (Java Database Connectivity)
3. PL/SQL

The Main Features of PL/SQL

- PL/SQL is Oracle's procedural extension to SQL.
- PL/SQL is part of the Oracle server.
- PL/SQL has concepts similar to modern programming languages, such as:
  - Variable and constant declaration
  - Control structure
  - Exception handling
  - modularization

General Structure of a PL/SQL Block

- PL/SQL is a block-structured language.
- Blocks can be entirely separate or nested within one another.
- PL/SQL block has three parts:

① **Declaration Part (optional)**

in which variables, constants, cursors, and exceptions are defined and possibly initialized.

② **Executable Part (mandatory)**

in which the variables are manipulated.

③ **Exception Part (optional)**

to handle any exceptions raised during execution.

General Structure of a PL/SQL Block

[DECLARE — **Optional**

-- declarations**]**

BEGIN — **Mandatory**

-- executable statement

[EXCEPTION — **Optional**

-- exception handlers**]**

END — **Mandatory**

Example of Block Structure

```
DECLARE
        vpCount    NUMBER;
        vStaffNo VARCHAR2(5) := 'SG14';
BEGIN
        SELECT COUNT(*) INTO vpCount
        FROM PropertyForRent
        WHERE staffNo = vStaffNo;
END;
```

① Declarations

a) Variables and constant variables

vStaffNo **VARCHAR2**(5) := 'SG14';

MAX_PROPERTIES **CONSTANT NUMBER** := 100;

b) Declare a variable to be of the same type as a column

vStaffNo Staff.staffNo%**TYPE**;

vStaffNo1 vStaffNo %**TYPE**;

c) Declare a variable to be of the same type as an entire row of a table

vStaffRec Staff%**ROWTYPE**

②Executable Part – Assignments

- Variables can be assigned in two ways:

① Using the normal assignment statement (**:=**)

vStaffNo := ‘SG14’;

vRent := 500;

② As the result of an SQL **SELECT** or **FETCH** statement

SELECT COUNT (*)

INTO x

FROM PropertyForRent

WHERE staffNo = vStaffNo;

②Executable Part – Control Statements

- PL/SQL supports the usually conditional, iterative, and sequential flow-of-control mechanisms:
  - **IF–THEN–ELSE–END IF;**
  - **LOOP-EXIT WHEN-END LOOP;**

    WHILE-END LOOP;
  - **GOTO.**

③Exceptions

- An **exception** is an identifier in PL/SQL raised during the execution of a block, which terminates its main body of actions.
- To handle raised exceptions, separate routines called **exception handlers** are specified.
- Exception handling:

① A user-defined **exception** is defined in the **declarative part** of a PL/SQL block.

② In the **executable part** a **check** is made for the exception condition, if found, the exception is raised.

③ The **exception handler** itself is defined at the end of the PL/SQL block.

③Exceptions – An Example

```
DECLARE
   vpCount   NUMBER;
   vStaffNo VARCHAR2(5) := 'SG14';
   e_too_many_properties EXCEPTION;
BEGIN
   SELECT COUNT(*) INTO vpCount
   FROM PropertyForRent
   WHERE staffNo = vStaffNo;
   IF vpCount = 100
        RAISE e_too_many_properties;
    END IF
EXCEPTION
   WHEN e_too_many_properties THEN
   dbms_output.put_line ('Member of staff'||staffNo||already managing 100 properties');
END;
```

Cursors

- PL/SQL uses **cursors** to allow the rows of a query result to be accessed one at a time.

① **DECLARE**: cursor is defined in the DECLARE section.

CURSOR cursorName **IS** selectStatement;

① **OPEN**: cursor is first opened in the statement section.

② **FETCH**: the rows of the query result can be retrieved one at a time using a FETCH statement.

③ **CLOSE**: cursor is closed on completion of the fetches.

Example of Using Cursors

```
DECLARE
   vPropertyNo  PropertyForRent.propertyNo%TYPE;
   vPostcode      PropertyForRent.postcode%TYPE;
   CURSOR  propertyCursor IS
          SELECT propertyNo, postcode
          FROM    PropertyForRent
          WHERE staffNo = 'SG14';
BEGIN
   OPEN propertyCursor;
   LOOP
      FETCH propertyCursor  INTO  vPropertyNo, vPostcode;
          ……
   END LOOP;
   IF propertyCursor%ISOPEN THEN CLOSE propertyCursor; END IF;
EXCEPTION
          ……
END;
```

Subprograms, Stored Procedures, Functions, and Packages

- **Subprograms** are named PL/SQL blocks that can take parameter and be invoked.
- PL/SQL has two types of subprograms:

  ① **(stored) Procedures**

  ② **Functions**

- Procedures and functions are very similar to those found in most high-level programming language.
- A **package** is a collection of procedures, functions, variables, and statement that are grouped together and stored as a single program unit.

Example of Procedures

- A parameter has a specified name and data type but can also be designated as:
  - **IN** parameter is used as an input value only.
  - **OUT** parameter is used as an output value only.
  - **IN OUT** parameter is used as both an input and an output value.
- By adding the following lines to change a anonymous block into a procedure:

CREATE OR REPLACE PROCEDURE PropertiesForStaff
(**IN** vStaffNo **VARCHAR2**)
AS …

- The procedure could be executed in SQL*Plus as:

SQL> **SET SERVEROUTPUT ON;**
SQL> **EXECUTE** PropertiesForStaff (‘SG14’);

Example of Packages

- A package has two parts:
  - **a specification**: declares all public constructs
  - **a body**: defines all constructs
- Create a package specification as:

```
CREATE OR REPLACE PACKAGE StaffPropertiesPackage AS
    procedure PropertiesForStaff (vSatffNo VARCHAR2);
END StaffPropertiesPackage;
```

- **Create the package body as:**

```
CREATE OR REPLACE PACKAGE BODY StaffPropertiesPackage  AS
    …
END StaffPropertiesPackage;
```

- Call the PropertiesForStaff procedure as:

```
StaffPropertiesPackage.PropertiesForStaff('SG14')
```

Triggers

- A **trigger** defines an action that the database should take when some event occurs in the application.
- The code within a trigger, called the **trigger body**, is made up of a PL/SQL block, Java program, or ‘C’ callout.
- Triggers are based on the **Event-Condition-Action** (ECA) model.

① **Event**

② **Condition**

③ **Action**

Triggers

- Syntax of defining trigger:

CREATE [OR REPLACE] TRIGGER trigger_name {**BEFORE** | **AFTER** | **INSTEAD OF**}

{**INSERT** | **DELETE** | **UPDATE [OF** columnname{,columnname…}]}

ON tablename [**REFERENCING** corr_name_def {, corr_name_def…}]

[**FOR EACH ROW** | **FOR EACH STATEMENT**]

[**WHEN** (search_condition)]

BEGIN statement; {statement; …} **END** ;

corr_name_def is defined as:

{OLD old_row_corr_name

| NEW new_row_corr_name}

Example of Triggers

- Trigger to enforce the constraint that a member of staff cannot manage more than 100 properties at any one time

```
CREATE TRIGGER StaffNotHandlingTooMuch
BEFORE INSERT OR UPDATE ON PropertyForRent
FOR EACH ROW
DECLARE
  vpCount    NUMBER
BEGIN
   SELECT COUNT(*) INTO vpCount
   FROM PropertyForRent
   WHERE staffNo = :new.staffNo;
    IF vpCount = 100
     dbms_output.put_line ('Member'|| :new.staffNo ||'already managing 100 properties');
    END IF;
END;
```

Section Objectives

In this section you will learn:

① Several Client-Server architectures and Java-Based approaches.

② JDBC is the most prominent and mature approach for accessing relational DBMS from Java appears.

③ JDBC package defines a database access API that includes DriverManager, Connection, Statement, PreparedStatement, CallableStatement, ResultSet, DatabaseMetaData, ResultSetMetaData, SQLException and SQLWarning classes.

④ PL/SQL is Oracle's procedural extension to SQL. It has concepts similar to modern programming language, such as variable and constant declarations, control structure, and modularization.

Questions?

Assignments

Prerequisites for Next Section

- **Readings:**
  - **Required:** Connolly and Begg, sections 2.1, 9.1–9.6, 9.8–9.13, 15.1 (in third edition, sections 2.1, 9.1–9.6, 9.8–9.13, 14.1).
  - **Elective:** Connolly and Begg, sections 2.3 and 9.7