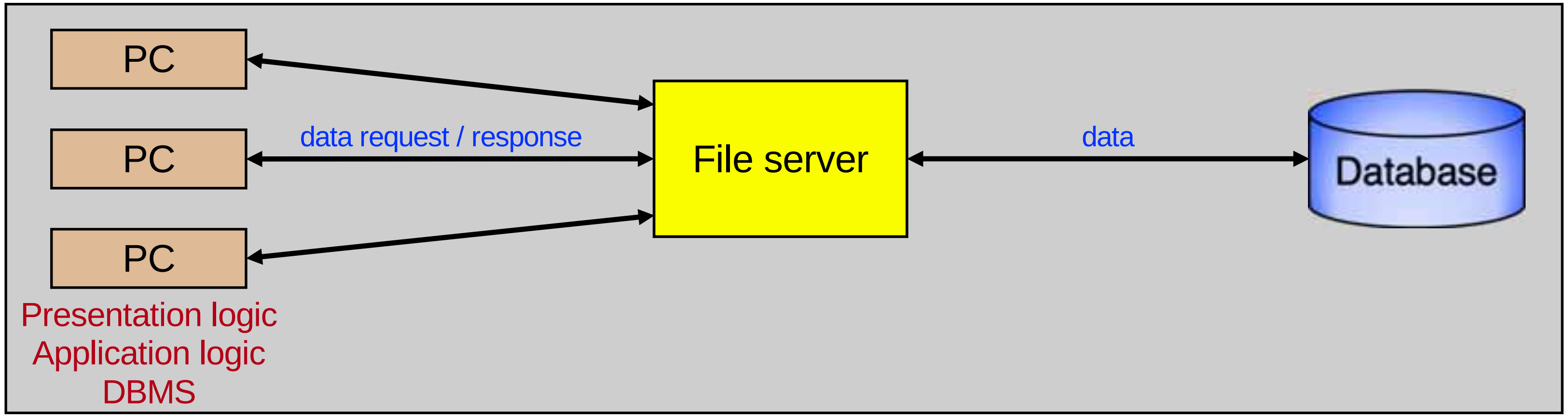
COMP 3311
DATABASE MANAGEMENT SYSTEMS

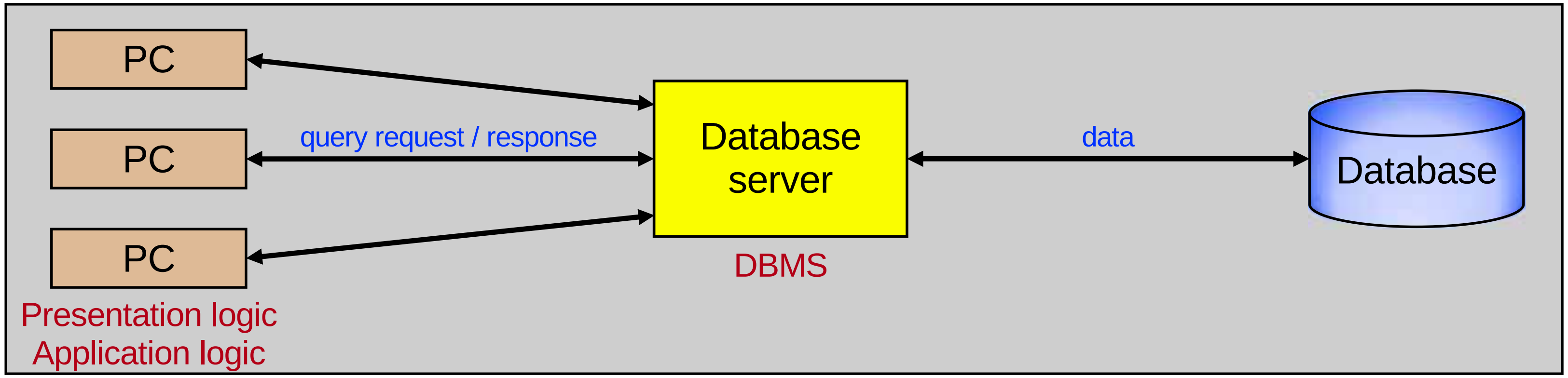
LECTURE 10
STRUCTURED QUERY LANGUAGE (SQL)

DATABASE SYSTEM ARCHITECTURES

Centralized

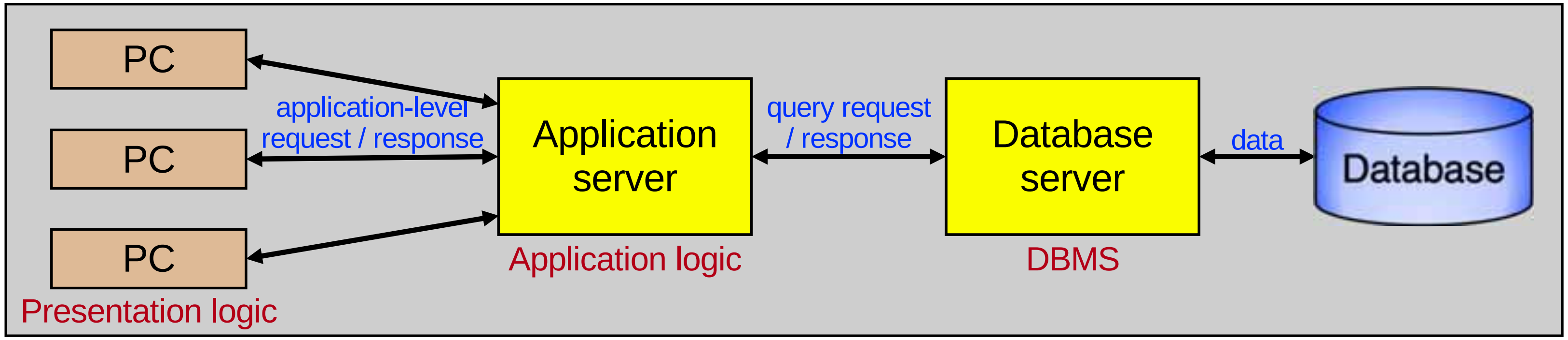


Two-tier (client-server)

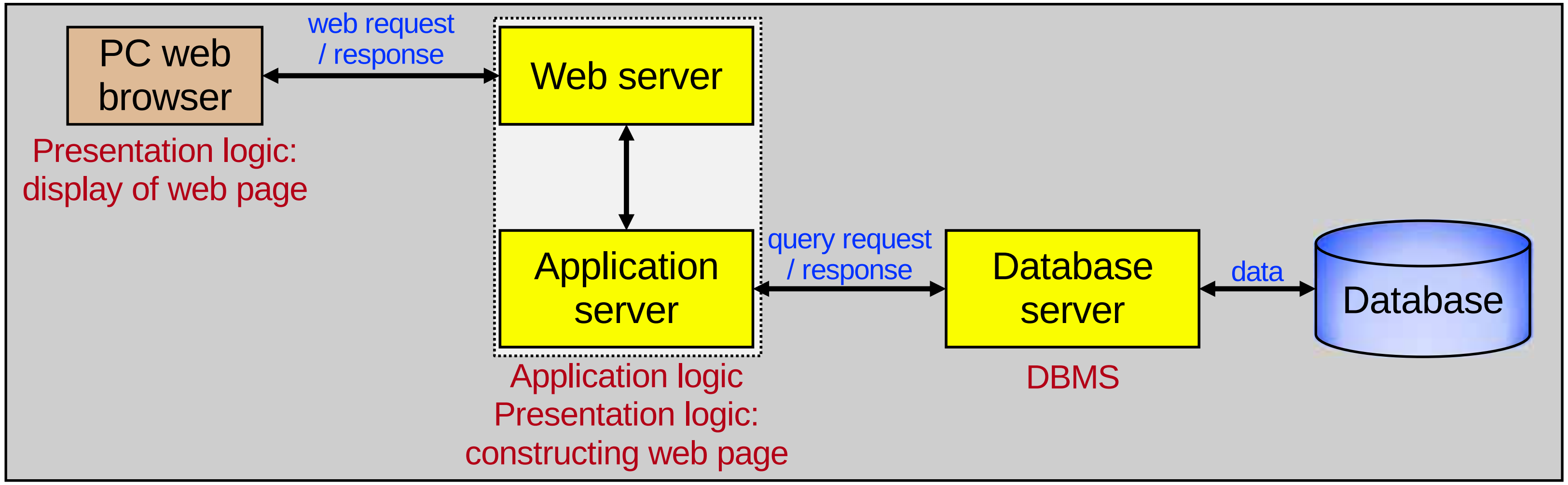


DATABASE SYSTEM ARCHITECTURES (CONT'D)

Three-tier



n-tier (e.g., web database connectivity)



DATABASE SYSTEM ARCHITECTURES (CONT'D)

Tiered system architectures

– The aim is to decouple the centralized architecture by combining a central computer's powerful computing capabilities with the flexibility of PCs.

Fat client: The presentation logic and application logic are handled by the client (i.e., the PC).

➢ Common in cases where it makes sense to couple an application's workflow (e.g., opening of windows, screens, and forms) with its look and feel (i.e., its front-end).

Thin client: Only the presentation logic is handled by the client.

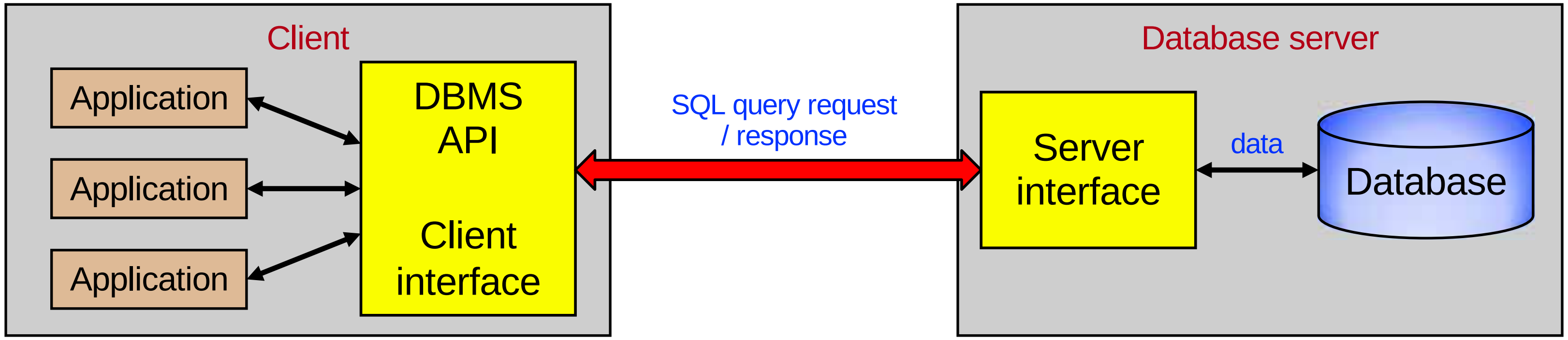
➢ Common in cases where the application logic and database logic are tightly coupled (fat server/thin client architecture).

– It is also possible to decouple the application logic from the DBMS and place this in a separate layer (i.e., an application server) and to also decouple some of the presentation logic from the PCs (e.g., a web server).

API BASICS

- To utilize DBMS services, client applications use a specific application programming interface (API) provided by the DBMS.
  - Facebook, Google, Instagram, etc. have such APIs.

- The DBMS API exposes an interface through which the services provided by the DBMS can be accessed.
  - The client and server interfaces often are implemented in the form of network sockets that use a specific port number on the server (e.g., port 1521 for the course Oracle Database server).



PROPRIETARY VS UNIVERSAL API

Proprietary, DBMS-specific API

- Provided by most vendors, but requires client applications to:
  - be aware of the DBMS that will be utilized on the server side.
  - be modified to interact with a new DBMS API.

Generic, vendor-agnostic universal API

- Allows easy porting of applications to multiple DBMSs.
- Does not allow access to some vendor-specific optimizations.

Examples:

- ODBC (Open Database Connectivity)
- JDBC (Java Database Connectivity)
- ADO.NET (ActiveX Data Objects for Microsoft’s .NET framework)

EMBEDDED VS CALL-LEVEL API

Embedded API

- SQL statements are part of the host programming language source code.
- An SQL pre-compiler parses and checks the SQL instructions *before* the program is compiled and replaces these with source code instructions native to the host programming language used.

Call-level API

- Passes SQL instructions to the DBMS by direct calls to a series of procedures, functions or methods provided by the API.
- The calls perform actions such as setting up a database connection, sending queries and iterating over the query result.

EARLY VS LATE BINDING

- SQL binding is the translation of SQL statements in a programming language into a form that can be executed by the DBMS.
  - Involves performing tasks such as validating table and attribute names, checking whether the user or client has sufficient access rights and generating an efficient query plan to access the data.

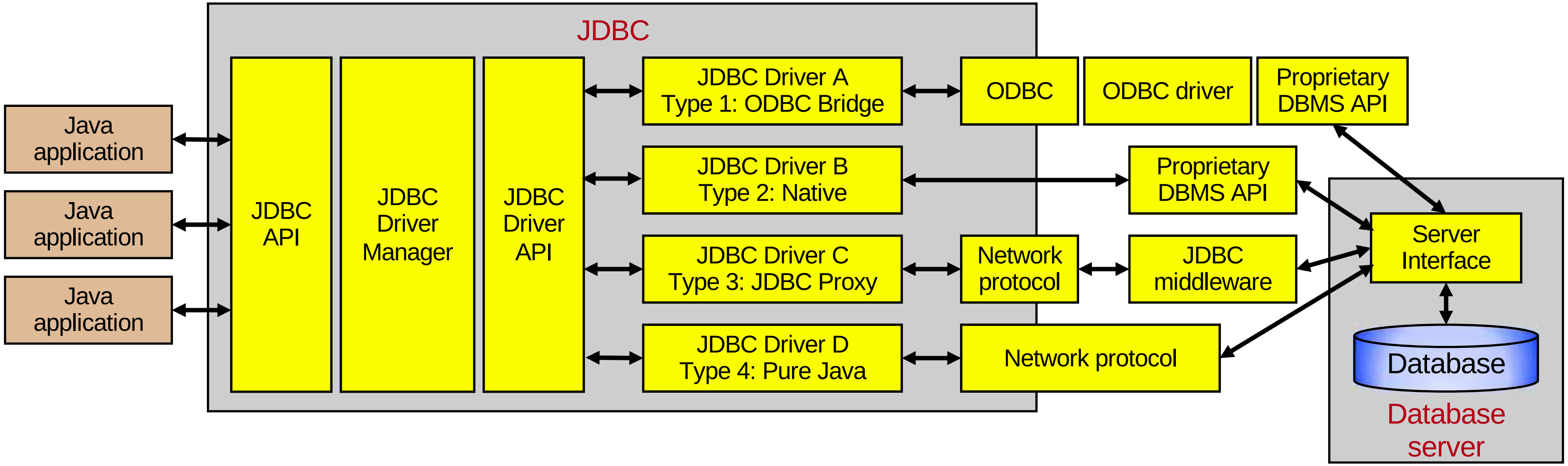
- Early binding performs these tasks *only once* *before program execution* (i.e., using a pre-compiler with an embedded API).

- Late binding performs these tasks *every time* *at runtime* (i.e., when using a call-level API).

☞ **It is still possible to do early binding using call-level APIs by using stored procedures in the DBMS.**

JAVA DATABASE CONNECTIVITY (JDBC)

- A call-level API for Java that is highly portable and object-oriented.
  - Database connections, drivers, queries and results are all expressed as objects, based on uniform interfaces.
  - Exposes a uniform set of methods, no matter which DBMS is used.

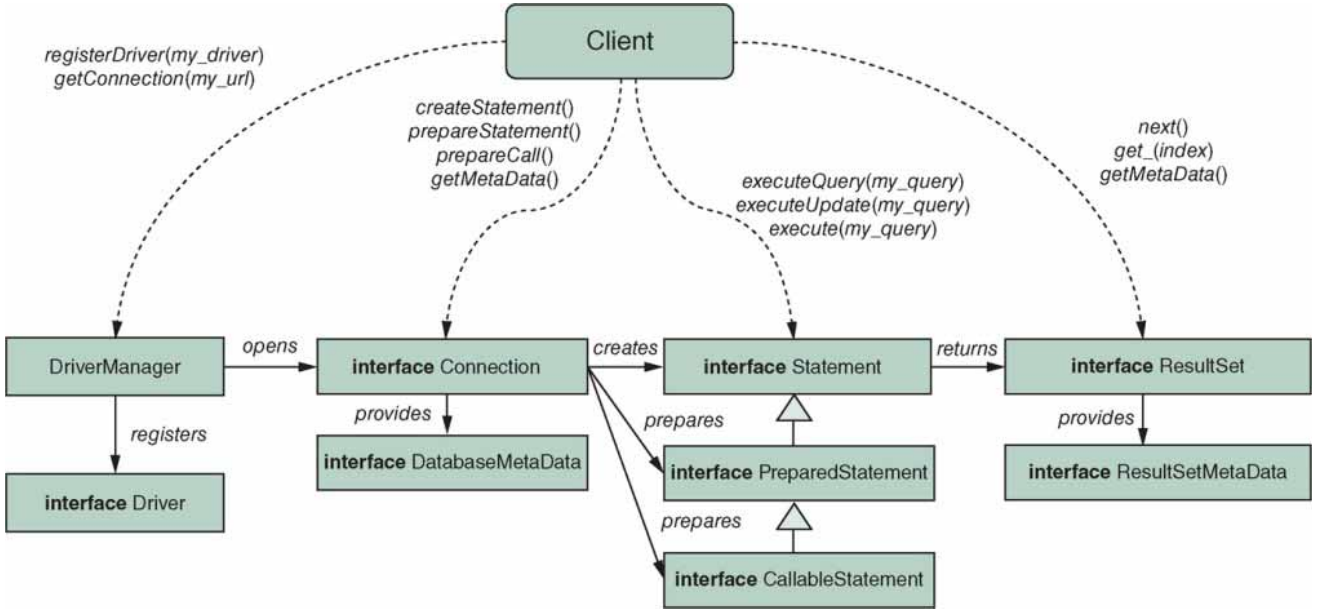


JAVA DATABASE CONNECTIVITY (JDBC) (cont')

- DriverManager is a singleton object which acts as the basic service to register and manage JDBC drivers.
- Driver objects implement the Driver interface and enable the communication between the DriverManager and the DBMS using one of four types of drivers.
  - Type-1: ODBC Bridge drivers do not communicate with a DBMS directly, but instead translate JDBC calls to corresponding ODBC calls.
  - Type-2: Native drivers are written in Java, but will communicate to a DBMS using its native database API.
  - Type-3: JDBC Proxy drivers are written in Java. The JDBC client uses standard networking sockets to communicate with an application server, which converts the calls into a native database API call or utilizes a different JDBC type-1, 2, or 4 driver on its end.
  - Type-4: Pure Java drivers are written in Java and use networking functionality to connect directly with the database server.
- The getConnection method creates a database connection using one of the registered drivers.

JAVA DATABASE CONNECTIVITY (JDBC) (cont')

```
DriverManager.registerDriver(new org.sqlite.JDBC());
String dbURL = "jdbc:sqlite:my_database";
Connection conn = DriverManager.getConnection(dbURL);
if (conn != null) {
   System.out.println("Connected to the database");
   DatabaseMetaData dm = conn.getMetaData();
   System.out.println("Driver name: " + dm.getDriverName);
   conn.close();
}
```



JDBC: EXECUTING SQL STATEMENTS

```
Statement selectStatement = conn.createStatement("select * from Book");
ResultSet selectResult = selectStatement.executeQuery();
                                   .
                                   .
                                   .
```

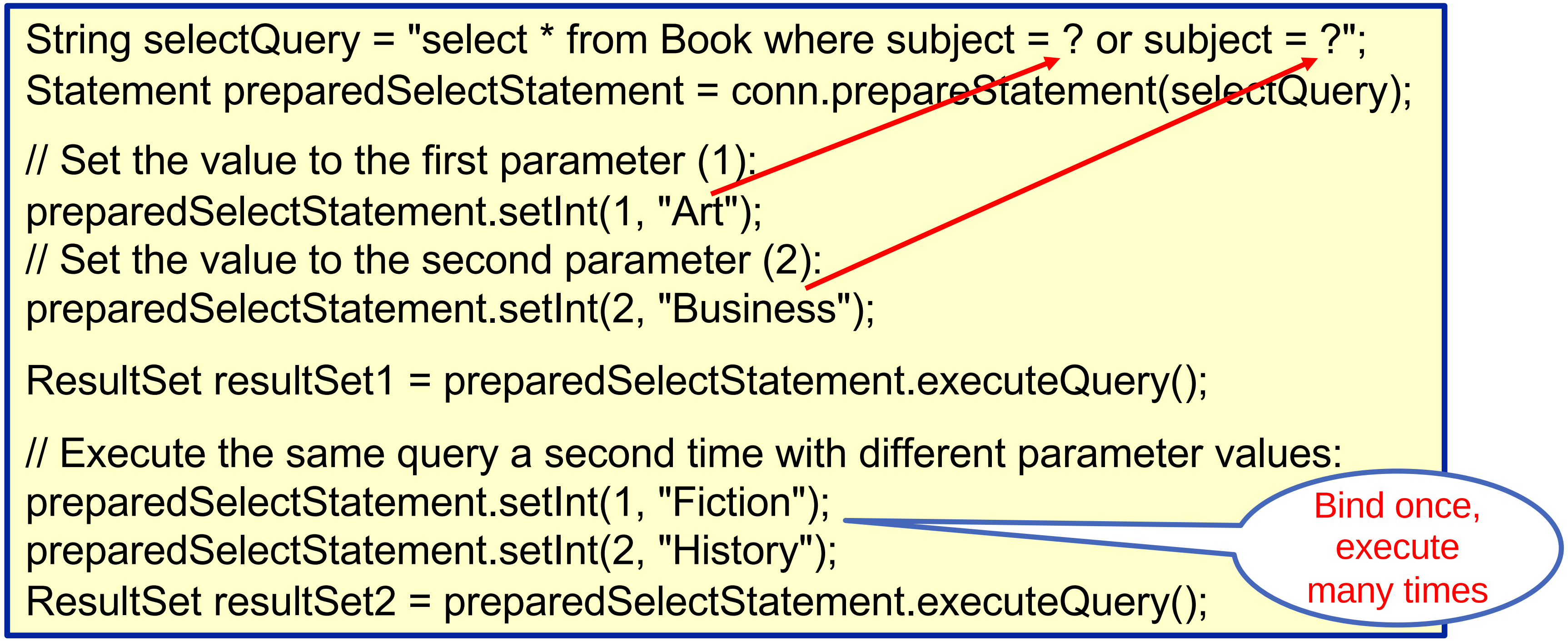
- SQL statements are executed and results returned within the context of a database connection.
- A Statement object represents an SQL instruction.
- An SQL statement is created with the createStatement method.
- The executeQuery method is used to execute an SQL select statement and return a ResultSet representing the returned data.
- The executeUpdate method is used to execute insert, update and delete statements.

JDBC: CURSORS

```
Statement selectStatement = conn.createStatement("select * from Book");
ResultSet selectResult = selectStatement.executeQuery();
while (selectResult.next()) {
   String bookTitle = selectResult.getString("title"); // or: .getString(1);
   int bookQuantity = selectResult.getInt("quantityInStock"); // or: .getInt(2);
   System.out.println(bookTitle + " has " + bookQuantity + " books in stock.");
}
```

- Since SQL is a set-oriented language, the query result (the ResultSet object) will generally contain multiple tuples.
- Host languages, such as Java, are essentially record-oriented.
  - They cannot work on more than one record/tuple at a time.
- To overcome this impedance mismatch, JDBC uses a cursor mechanism to step through result sets.
  - A cursor is a programmatic control structure that enables one-by-one traversal over the records in a query result set.

JDBC: STORED PROCEDURES



```
String selectQuery = "select * from Book where subject = ? or subject = ?";
Statement preparedSelectStatement = conn.prepareStatement(selectQuery);

// Set the value to the first parameter (1):
preparedSelectStatement.setInt(1, "Art");
// Set the value to the second parameter (2):
preparedSelectStatement.setInt(2, "Business");

ResultSet resultSet1 = preparedSelectStatement.executeQuery();

// Execute the same query a second time with different parameter values:
preparedSelectStatement.setInt(1, "Fiction");
preparedSelectStatement.setInt(2, "History");
ResultSet resultSet2 = preparedSelectStatement.executeQuery();
```

Bind once, execute many times

- The prepareStatement and prepareCall methods are used to create objects representing prepared (often parameterized) statements and stored procedure calls, respectively.

JDBC: STORED PROCEDURES

- The PreparedStatement interface extends Statement with functionalities to bind a query once and then execute it multiple times in an efficient manner.

- Prepared statements also provide support for parameterized queries by passing query parameters, which are instantiated using setter methods such as setInt, setString, etc.

- Question marks (?) are used inside an SQL query to indicate that this represents a parameter value that will be bound later.

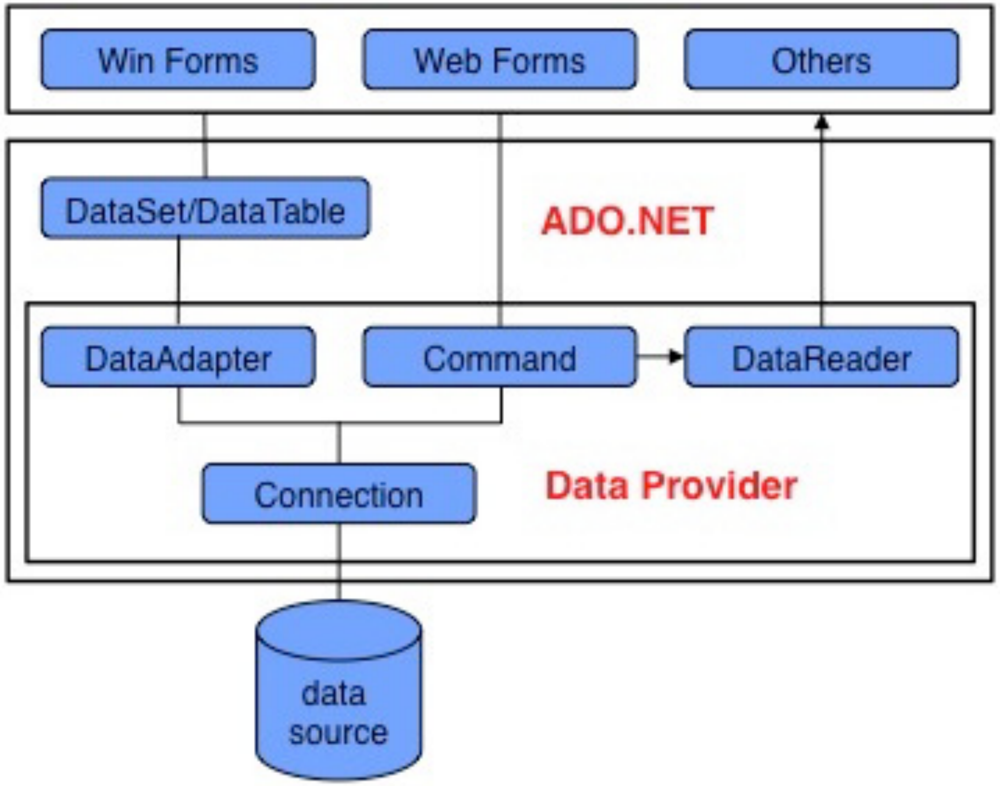
- CallableStatement extends PreparedStatement and offers support to execute stored procedures.

ADO.NET: ARCHITECTURE

- ADO.NET offers a collection of data providers, which consist of objects that handle creation of database connections, sending queries and reading results.

- A DataSet/DataTable data structure provides a disconnected way to hold the data retrieved from a database.

- A DataSet can hold several tables, while a DataTable can hold only one table.

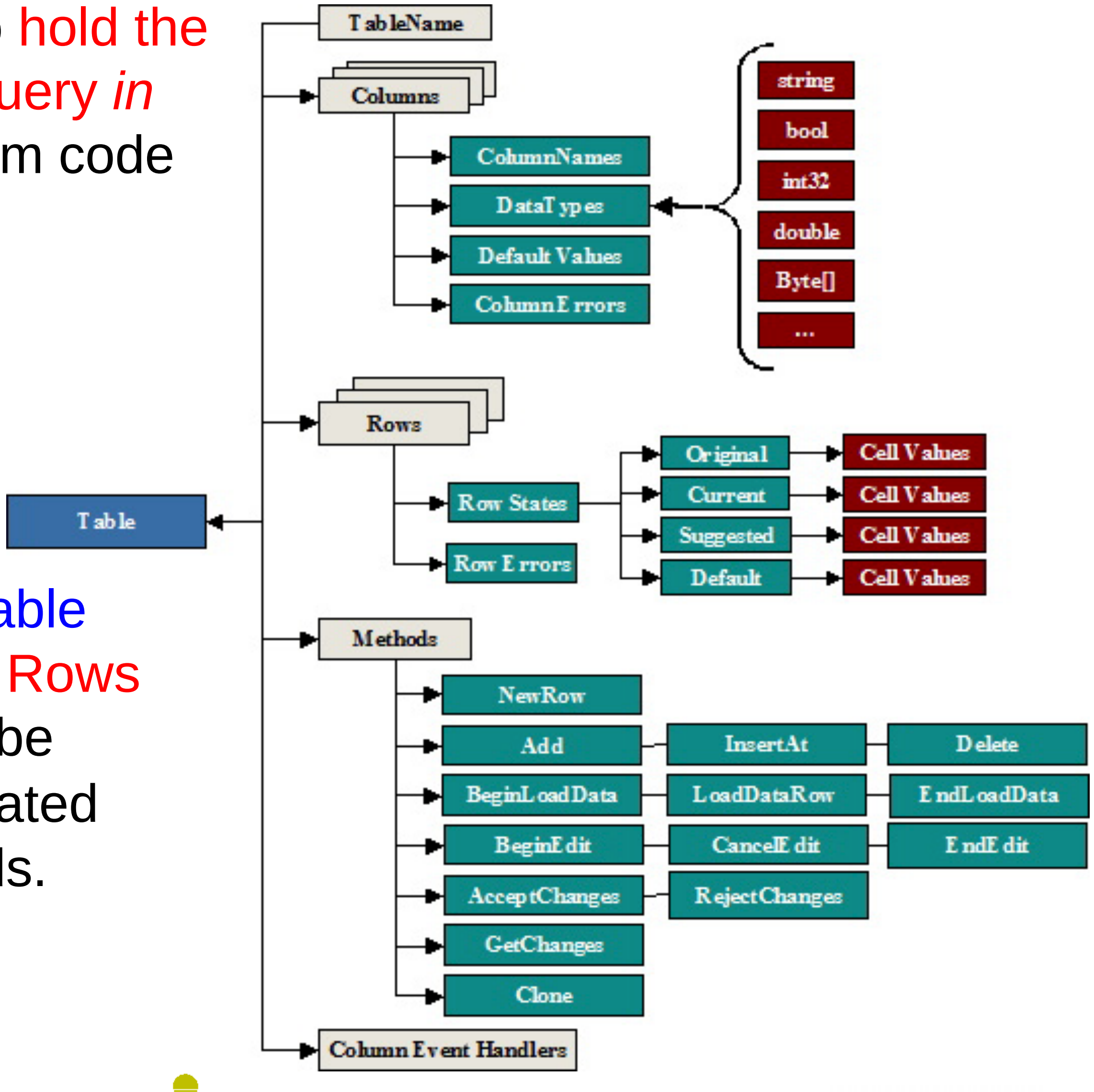


ADO.NET: DATATABLE

- A DataTable is used to hold the data resulting from a query *in memory* so that program code can manipulate it.

- A DataTable can hold at most one table (i.e., a query result).

- A table within a DataTable contains Columns and Rows collections, which can be accessed and manipulated using standard methods.



ADO.NET: EXAMPLE DATABASE ACCESS

```
String connectionString = "Data Source= ...";
OracleConnection conn = new OracleConnection(connectionString);
conn.Open();
string title;
string author;
DataTable dataTable = new DataTable();
OracleCommand cmd1 = conn.CreateCommand();
OracleCommand cmd2 = conn.CreateCommand();
cmd1.CommandText = "select trunc(avg(price),2) from Book";
cmd2.CommandText = "select title, firstName, lastName from Author natural join Book where subject='Art'”;
decimal averagePrice = Convert.ToDecimal(cmd1.ExecuteScalar());
OracleDataAdapter da = new OracleDataAdapter(cmd2.CommandText, conn);
da.Fill(dataTable);
foreach (DataRow row in dataTable.Rows) {
   title = row["TITLE"].ToString();
   author = row["LASTNAME"].ToString() + ", " + row["FIRSTNAME"].ToString();
   Console.Write(title + " by " + author + "\n");
}
Console.Write("The average book price is " + averagePrice + ".");
conn.Close();
```

OracleConnection
Set up a connection to the database.

OracleCommand
Create the queries in the context of the connection.

Set the two queries.

Execute the two queries.

Loop over the result set.

This C# code fragment shows the Connection, Command and DataTable objects using the .NET Framework Data Provider for Oracle Database.

ORACLE PL/SQL

- PL/SQL (Procedural Language/SQL) allows SQL statements to be embedded into a procedural programming language.

- It combines the data manipulation power of SQL with the data processing power of procedural languages.

- A block, which is delimited by begin…end and which can be nested, is the basic processing unit in which statements:
  - are case insensitive.
  - use C style comments /*…*/.
  - use := operator to assign values to a variable.
  - use = operator for comparison.

Allowed SQL statements: select, insert, update, delete (i.e., DML)

Not allowed SQL statements: create, drop, alter, rename (i.e., DDL)

ORACLE PL/SQL (CONT'D)

- A PL/SQL program is stored as a database object and can be
  - a procedure, which does not return a value.
  - a function, which returns a value using the return keyword.

- Both types of programs can accept parameters which can be one of

in	a read-only variable for giving input (the default)
out	a read-write variable for getting output
in out	a read-write variable for giving input and getting output

- A procedure is invoked using the exec keyword.

- A function is invoked by assigning its result to a variable or using it in a select statement.

BASIC STRUCTURE & DATA TYPES

create or replace procedure *procedure_name* [as | is]

Declaration section: contains declaration of variables, types, and local subprograms.

begin **Executable section:** contains procedural and SQL statements. This is the only section of a block that is required.

exception **Exception handling section:** contains error handling statements.

end;

Variable Data Types

- A data type used to define the attributes of a table (i.e., number, int, char, varchar2, date, etc.).
- The same as an attribute (*table_name.attribute_name%*type) or a row (*table_name%*rowtype).

FLOW OF CONTROL STATEMENTS

- **Sequential control**

  goto – branch to a label unconditionally

  null – pass control to the next statement

  return – returns control to the calling block and may return a value.

- **Conditional control**

  if-then, if-then-else, if-then-elsif – conditional processing

  case – selects one sequence of statements to execute

- **Iterative control**

  loop *statements* end loop;

  while *condition* loop *statements* end loop;

  for *loop_variable* in [reverse] *lower_bound..upper_bound*
    loop *statements* end loop;

  exit / exit when *condition* – exit the current loop possibly conditionally

  continue / continue when *condition* – exit current loop iteration

PL/SQL PROCEDURE EXAMPLE

Increment the rating of a sailor if the rating is less than 5.

```
create or replace procedure L10Example1 (sid in int) as
   -- sailorName is the same type as sName in the Sailor table
   sailorName   Sailor.sName%type;
   -- sailorRating is the same type as rating in the Sailor table
   sailorRating  Sailor.rating%type;
begin
   -- Fetch the sailor’s name and rating into the variables sailorName and sailorRating
   select sName, rating into sailorName, sailorRating from Sailor where sailorId=sid;
   if sailorRating<5 then
      update Sailor set rating=sailorRating+1 where sailorId=sid;
      -- Write record updated message to the Script Output pane
      dbms_output.put_line('Sailor ' || sailorName || '(' || sid || ') rating updated from ' ||
         sailorRating || ' to ' || (sailorRating+1) || '.');
   else
      -- Write record NOT updated message to the Script Output pane
      dbms_output.put_line('Sailor ' || sailorName || '(' || sid || ') rating ' || sailorRating || ' NOT updated.');
   end if;
end L10Example1;
```

SELECT INTO STATEMENT

select *attribute_name* into *variable_name* from *table_name* [where *condition*];

- Retrieves a value from a table in the database and assigns it to *variable_name*.
- The select ... into statement should retrieve only one record as a variable can hold only one value.
- If the select ... into statement returns more than one or no value, an exception will be raised => handle in the exception section.
- The number of columns and their data type in the select clause must match with the number of variables and their data types in the into clause.
- The values are retrieved and populated in the same order as specified in the select clause.

CURSORS

- If a select statement returns more than one record, a cursor is normally used to process the records one-at-a-time.

- A cursor is like a pointer that points to a single record and allows access to the attribute values of that record.

- A cursor is defined in the declare section using the syntax:

  cursor *cursor_name* is *select_statement*;

- A cursor can be used and managed *explicitly* using the open, fetch and close commands and by checking cursor status.

- It can also be used and managed *implicitly* using the for...loop statement where the *cursor_name* replaces the range limit so the loop ranges from the first record of the cursor to the last record of the cursor.

CURSOR STATUS

- The possible values of a cursor status are:

*cursor_name%***found** Returns TRUE if the fetch operation succeeded; else returns FALSE.

*cursor_name%***notfound** Returns TRUE if the fetch operation failed; else returns FALSE.

*cursor_name%***isopen** Returns TRUE if the cursor is still open; else returns FALSE.

*cursor_name%***rowcount** Returns the number of records fetched.

PL/SQL CURSOR EXAMPLE

```
create or replace procedure L10Example2 as
   currentSailorId   Sailor.sailorId%type;
   -- Declare the cursors for the sailor and reserves tables
   cursor sailorCursor is select * from Sailor order by sName;
   cursor reservesCursor is select count(boatid) reservations from reserves where sailorId=currentSailorId;
begin
   -- Fetch the sailorCursor records one-by-one
   for sailorRecord in sailorCursor loop
      -- Assign the sailor id for the current sailor record
      currentSailorId:=sailorRecord.sailorId;
      -- Fetch the reservesCursor records one-by-one
      for reservesRecord in reservesCursor loop
         -- Insert into appropriate table
         if reservesRecord.reservations=0 then
            insert into NoReservations values (sailorRecord.sailorId, sailorRecord.sName);
         else
            insert into YesReservations values (sailorRecord.sailorId, sailorRecord.sName);
         end if;
      end loop;
   end loop;
end L10Example2;
```

PL/SQL EXCEPTIONS

- Predefined exceptions are raised implicitly by PL/SQL if the exception occurs.

- User-defined exceptions are declared in the declaration section,

  ```
  exception_name exception;
  ```

  raised explicitly within a **begin**...**end** block

  ```
  if condition then
     raise exception_name;
  end if;
  ```

  and handled in the **exception** section within the **begin**...**end** block.

  ```
  exception
     when exception_name then
                 ⋮
  ```

Predefined Exceptions

ACCESS_INTO_NULL	ORA-06530
CASE_NOT_FOUND	ORA-06592
COLLECTION_IS_NULL	ORA-06531
CURSOR_ALREADY_OPEN	ORA-06511
DUP_VAL_ON_INDEX	ORA-00001
INVALID_CURSOR	ORA-01001
INVALID_NUMBER	ORA-01722
LOGIN_DENIED	ORA-01017
NO_DATA_FOUND	ORA-01403
NOT_LOGGED_ON	ORA-01012
PROGRAM_ERROR	ORA-06501
ROWTYPE_MISMATCH	ORA-06504
SELF_IS_NULL	ORA-30625
STORAGE_ERROR	ORA-06500
SUBSCRIPT_BEYOND_COUNT	ORA-06533
SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532
SYS_INVALID_ROWID	ORA-01410
TIMEOUT_ON_RESOURCE	ORA-00051
TOO_MANY_ROWS	ORA-01422
VALUE_ERROR	ORA-06502
ZERO_DIVIDE	ORA-01476

PL/SQL EXCEPTIONS EXAMPLE

```
create or replace procedure L10Example3 (sid in int) as
  -- sailorName is the same type as sName in the Sailor table
  sailorName   Sailor.sName%type;
  -- sailorRating is the same type as rating in the Sailor table
  sailorRating  Sailor.rating%type;
begin
  -- Fetch the sailor’s name and rating into the variables sailorName and sailorRating
  select sName, rating into sailorName, sailorRating from Sailor where sailorId=sid;
  if sailorRating<5 then
    update Sailor set rating=sailorRating+1 where sailorId=sid;
    -- Write record updated message to the Script Output pane
    dbms_output.put_line('Sailor ' || sailorName || '(' || sid || ') rating updated from ' ||
      sailorRating || ' to ' || (sailorRating+1) || '.');
  else
    -- Write record NOT updated message to the Script Output pane
    dbms_output.put_line('Sailor ' || sailorName || '(' || sid || ') rating ' || sailorRating || ' NOT updated.');
  end if;
exception
  when no_data_found then
    -- Write exception message to the Script Output pane
    dbms_output.put_line('There is no sailor with id ' || sid || '.');
end L10Example3;
```

STRUCTURED QUERY LANGUAGE (SQL): SUMMARY

- Structured Query Language (SQL) is a relational query language that provides facilities to

Query Relations

- Select-From-Where Statement
- Set Operations (Union, Intersect, Except)
- Nested Subqueries (to test for set membership, comparison, cardinality)
- Aggregate Functions (avg, min, max, sum, count)
- Group By with Having clause

Create and Modify Relations

- Create, Alter, Drop Tables
- Specify integrity constraints: domain, key, foreign key, general
- Specify views
- Insert, Delete, Update Tuples

Access a Database from a Programming Language