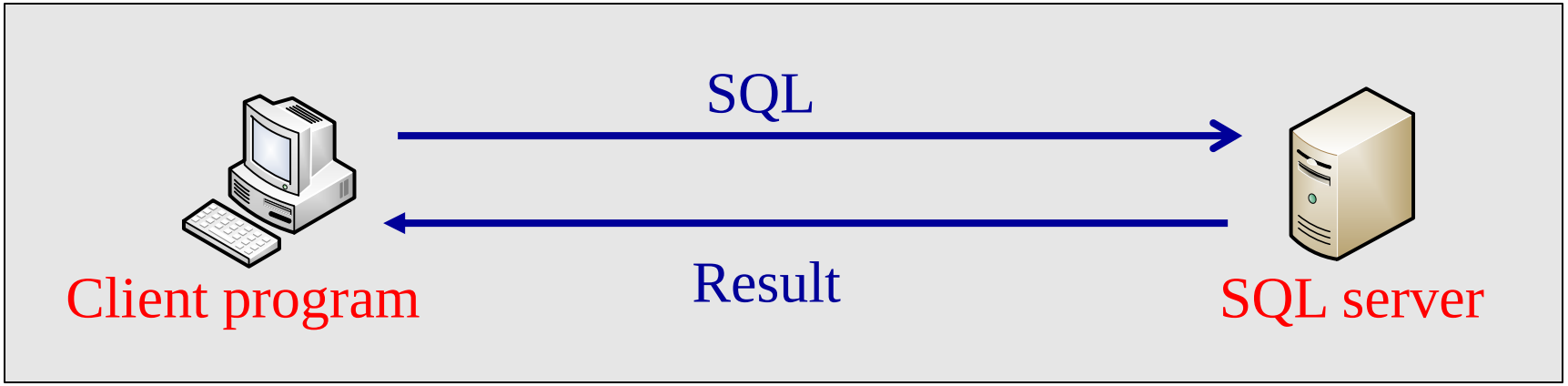
Database Application Development

Client-server Model



- The program running on the client machine sends SQL statements to the database server.
- The results of the SQL statements will be returned to the client.

SQL in Application Code

- SQL commands can be called from within a host language (e.g., C++ or Java) program.
  - Must include a statement to *connect* to the right database.
  - SQL statements can refer to host variables (including special variables used to return status).
- Two main integration approaches:
  - Embed SQL in the host language (e.g. Embedded SQL, SQLJ)
  - Create special API to call SQL commands (e.g. JDBC)

Impedance mismatch:

- SQL relations are (multi-) sets of records, with no *a priori* bound on the number of records. No such data structure exist traditionally in procedural programming languages such as C++.
  - SQL supports a mechanism called a *cursor* to handle this.

Embedded SQL

- Approach: Embed SQL in the host language.
  - A preprocessor converts the SQL statements into special API calls.
  - Then a regular compiler is used to compile the code.
- Language constructs:
  - Connecting to a database:
    EXEC SQL CONNECT
  - Declaring variables:
    EXEC SQL BEGIN (END) DECLARE SECTION
  - Statements:
    EXEC SQL Statement;

Note all SQL commands must be prefixed by **EXEC SQL**

Variables must be declared in a special form

Example:

```
EXEC SQL BEGIN DECLARE SECTION
char c_sname[20];
long c_sid;
short c_rating;
float c_age;
EXEC SQL END DECLARE SECTION
```

Variables must be declared in a special form

Example:

```
EXEC SQL BEGIN DECLARE SECTION
char c_sname[20];
long c_sid;
short c_rating;
float c_age;
EXEC SQL END DECLARE SECTION
```

Declare some C variables as usual.

Variables must be declared in a special form

Example:

```
EXEC SQL BEGIN DECLARE SECTION
char c_sname[20];
long c_sid;
short c_rating;
float c_age;
EXEC SQL END DECLARE SECTION
```

All variables declared between these two statements can also be used within SQL statements

C types are mapped into SQL types:

```
char c_sname[20] ≡ c_sname CHARACTER( 20 )
long c_sid ≡ c_sid INTEGER
short c_rating ≡ c_rating SMALL INT
float c_age ≡ c_age REAL
```

When used inside SQL statements, C variables must be prefixed by a colon

```
EXEC SQL INSERT INTO Sailors VALUES
                    (:c_sname, :c_sid, :c_rating, :c_age);
```

Observe that a semicolon terminates the command, as per the convention for terminating a statement in C.

Observe the colons

Cursors

- SQL relations are sets of records of arbitrary cardinality,

  How do we access records?

  Use **Cursor**
- A cursor is a **pointer to** a row in the relation for which it is defined
- Can declare a cursor on a query statement

```
EXEC SQL DECLARE sinfo CURSOR FOR
        SELECT S.sname, S.age
        FROM Sailors S, Boats B, Reserves R
        WHERE S.sid=R.sid AND R.bid=B.bid AND B.color='red'
        ORDER BY S.sname;
```

- One can *open* a cursor, and repeatedly *fetch* a tuple to move the cursor, until all tuples have been retrieved
  - **OPEN** sinfo

    When cursor sinfo is opened, it is positioned before the first row

  - **FETCH** sinfo **INTO** :c_sname, :c_age

    When the **FETCH** is executed, sinfo is pointing to the next row (1st row when the **FETCH** is executed for the first time)

  - When we are done, we should *close* the cursor

    CLOSE sinfo

- Use a special clause, **ORDER BY**, to control the order in which tuples are returned

  note: Fields in **ORDER BY** clause must also appear in **SELECT** clause

- See p.191 of your textbook for some more properties of cursors.

Embedding SQL in C: An Example

```
char SQLSTATE[6];
EXEC SQL BEGIN DECLARE SECTION
char c_sname[20]; short c_minrating; float c_age;
EXEC SQL END DECLARE SECTION
c_minrating = random();
EXEC SQL DECLARE sinfo CURSOR FOR
    SELECT S.sname, S.age
    FROM Sailors S
    WHERE S.rating > :c_minrating
    ORDER BY S.sname;
EXEC SQL OPEN sinfo;
do {
    EXEC SQL FETCH sinfo INTO :c_sname, :c_age;
    printf("%s is %d years old\n", c_sname, c_age);
} while (SQLSTATE != '02000');
EXEC SQL CLOSE sinfo;
```

Embedding SQL in C: An Example

```
char SQLSTATE[6];
EXEC SQL BEGIN DECLARE SECTION
char c_sname[20]; short c_minrating; float c_age;
EXEC SQL END DECLARE SECTION
c_minrating = random();
EXEC SQL DECLARE sinfo CURSOR FOR
    SELECT S.sname, S.age
    FROM Sailors S
    WHERE S.rating > :c_minrating
    ORDER BY S.sname;
EXEC SQL OPEN sinfo;
do {
    EXEC SQL FETCH sinfo INTO :c_sname, :c_age;
    printf("%s is %d years old\n", c_sname, c_age);
} while (SQLSTATE != '02000');
EXEC SQL CLOSE sinfo;
```

Variable used by SQL to signify an error

Embedding SQL in C: An Example

```
char SQLSTATE[6];
EXEC SQL BEGIN DECLARE SECTION
char c_sname[20]; short c_minrating; float c_age;
EXEC SQL END DECLARE SECTION
c_minrating = random();
EXEC SQL DECLARE sinfo CURSOR FOR
    SELECT S.sname, S.age
    FROM Sailors S
    WHERE S.rating > :c_minrating
    ORDER BY S.sname;
EXEC SQL OPEN sinfo;
do {
    EXEC SQL FETCH sinfo INTO :c_sname, :c_age;
    printf("%s is %d years old\n", c_sname, c_age);
} while (SQLSTATE != '02000');
EXEC SQL CLOSE sinfo;
```

The variables declared here can be used inside SQL statements

Embedding SQL in C: An Example

```
char SQLSTATE[6];
EXEC SQL BEGIN DECLARE SECTION
char c_sname[20]; short c_minrating; float c_age;
EXEC SQL END DECLARE SECTION
c_minrating = random();
EXEC SQL DECLARE sinfo CURSOR FOR
    SELECT S.sname, S.age
    FROM Sailors S
    WHERE S.rating > :c_minrating
    ORDER BY S.sname;
EXEC SQL OPEN sinfo;
do {
    EXEC SQL FETCH sinfo INTO :c_sname, :c_age;
    printf("%s is %d years old\n", c_sname, c_age);
} while (SQLSTATE != '02000');
EXEC SQL CLOSE sinfo;
```

A random value is assigned into the variable c_minrating. (Assume we have a random function which can generate a random rating value.)

Embedding SQL in C: An Example

```
char SQLSTATE[6];
EXEC SQL BEGIN DECLARE SECTION
char c_sname[20]; short c_minrating; float c_age;
EXEC SQL END DECLARE SECTION
c_minrating = random();
EXEC SQL DECLARE sinfo CURSOR FOR
    SELECT S.sname, S.age
    FROM Sailors S
    WHERE S.rating > :c_minrating
    ORDER BY S.sname;
EXEC SQL OPEN sinfo;
do {
    EXEC SQL FETCH sinfo INTO :c_sname, :c_age;
    printf("%s is %d years old\n", c_sname, c_age);
} while (SQLSTATE != '02000');
EXEC SQL CLOSE sinfo;
```

sinfo is declared as a cursor, which points to the result of the SQL statement.

Embedding SQL in C: An Example

```
char SQLSTATE[6];
EXEC SQL BEGIN DECLARE SECTION
char c_sname[20]; short c_minrating; float c_age;
EXEC SQL END DECLARE SECTION
c_minrating = random();
EXEC SQL DECLARE sinfo CURSOR FOR
    SELECT S.sname, S.age
    FROM Sailors S
    WHERE S.rating > :c_minrating
    ORDER BY S.sname;
EXEC SQL OPEN sinfo;
do {
    EXEC SQL FETCH sinfo INTO :c_sname, :c_age;
    printf("%s is %d years old\n", c_sname, c_age);
} while (SQLSTATE != '02000');
EXEC SQL CLOSE sinfo;
```

The cursor will be positioned before the first row of the results.

Embedding SQL in C: An Example

```
char SQLSTATE[6];
EXEC SQL BEGIN DECLARE SECTION
char c_sname[20]; short c_minrating; float c_age;
EXEC SQL END DECLARE SECTION
c_minrating = random();
EXEC SQL DECLARE sinfo CURSOR FOR
    SELECT S.sname, S.age
    FROM Sailors S
    WHERE S.rating > :c_minrating
    ORDER BY S.sname;
EXEC SQL OPEN sinfo;
do {
    EXEC SQL FETCH sinfo INTO :c_sname, :c_age;
    printf("%s is %d years old\n", c_sname, c_age);
} while (SQLSTATE != '02000');
EXEC SQL CLOSE sinfo;
```

The cursor will be advanced to the next row. Then the value of the 1st attribute will be put into c_sname and the value of the 2nd attribute will be put into c_age.

Embedding SQL in C: An Example

```
char SQLSTATE[6];
EXEC SQL BEGIN DECLARE SECTION
char c_sname[20]; short c_minrating; float c_age;
EXEC SQL END DECLARE SECTION
c_minrating = random();
EXEC SQL DECLARE sinfo CURSOR FOR
    SELECT S.sname, S.age
    FROM Sailors S
    WHERE S.rating > :c_minrating
    ORDER BY S.sname;
EXEC SQL OPEN sinfo;
do {
    EXEC SQL FETCH sinfo INTO :c_sname, :c_age;
    printf("%s is %d years old\n", c_sname, c_age);
} while (SQLSTATE != '02000');
EXEC SQL CLOSE sinfo;
```

Print out c_sname and c_age.
Note that
- %s is the descriptor for strings.
- %d is the descriptor for integers.
- \n is newline.

Embedding SQL in C: An Example

```
char SQLSTATE[6];
EXEC SQL BEGIN DECLARE SECTION
char c_sname[20]; short c_minrating; float c_age;
EXEC SQL END DECLARE SECTION
c_minrating = random();
EXEC SQL DECLARE sinfo CURSOR FOR
    SELECT S.sname, S.age
    FROM Sailors S
    WHERE S.rating > :c_minrating
    ORDER BY S.sname;
EXEC SQL OPEN sinfo;
do {
    EXEC SQL FETCH sinfo INTO :c_sname, :c_age;
    printf("%s is %d years old\n", c_sname, c_age);
} while (SQLSTATE != '02000');
EXEC SQL CLOSE sinfo;
```

02000 = No more data

Embedding SQL in C: An Example

```
char SQLSTATE[6];
EXEC SQL BEGIN DECLARE SECTION
char c_sname[20]; short c_minrating; float c_age;
EXEC SQL END DECLARE SECTION
c_minrating = random();
EXEC SQL DECLARE sinfo CURSOR FOR
    SELECT S.sname, S.age
    FROM Sailors S
    WHERE S.rating > :c_minrating
    ORDER BY S.sname;
EXEC SQL OPEN sinfo;
do {
    EXEC SQL FETCH sinfo INTO :c_sname, :c_age;
    printf("%s is %d years old\n", c_sname, c_age);
} while (SQLSTATE != '02000');
EXEC SQL CLOSE sinfo;
```

Close the cursor

Dynamic SQL

- SQL query strings are not always known at compile time (e.g., spreadsheet, graphical DBMS frontend): Allow construction of SQL statements on-the-fly

- Example:

```
char c_sqlstring[]=
{“DELETE FROM Sailors WHERE rating>5”};
EXEC SQL PREPARE readytogo FROM :c_sqlstring;
EXEC SQL EXECUTE readytogo;
```

Database API: Alternative to embedding

- Rather than modify compiler, add library with database calls (API)
- Special standardized interface: procedures/objects.
- Pass SQL strings from language, presents result sets in a language-friendly way.
- Microsoft's ODBC (Open Database Connectivity) becoming C/C++ standard on Windows.
- Sun's JDBC (Java Database Connectivity): a Java equivalent.
- DBMS-independent at both source code and executable level.

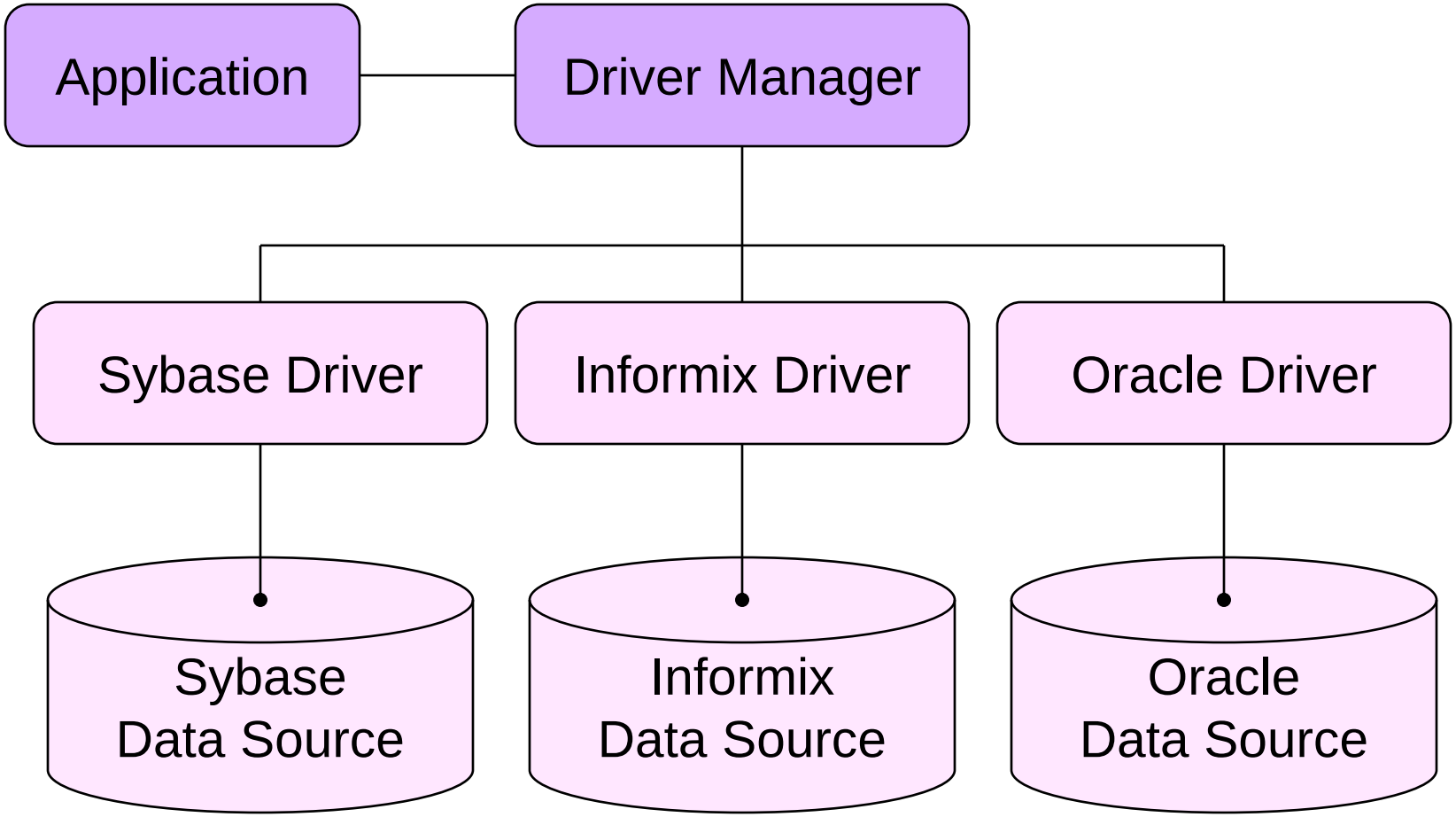
- Provides a collection of object classes that allows SQL access of databases
- Advantages of this approach

  a: executables can be DBMS-independent (while for embedded SQL, only source code can be DBMS-independent)

  b: SQL statement can be constructed at run time: ideal for dynamic SQL
- Where is the magic?

  Driver: a level of indirection.

JDBC Architecture



- JDBC Architectural components and their main functionalities

  a. **Application**: submits SQL statements (which will be translated into function calls), and get the results back

  b. **Driver manager**: load driver and pass function calls from the application to drivers

  c. **Driver(s)**: set up connection with the data sources, passes requests and returns results, and translation of data and error format

  d. **Data source**: process commands from and returns results to the corresponding driver

To access data source (i.e., database) we must

- load a driver

  API call:

  DriverManager.registerDriver(“ <driver name>”)

- set up a connection

  API call:

```
Connection conn =
    DriverManager.getConnection( url, uid, pwd );
```

  url contains information about the driver, and the database to be connected. uid and pwd are userid and password of a user.

JDBC: An Example

```
DriverManager.registerDriver
    (“oracle.jdbc.driver.OracleDriver” );
Connection conn =
    DriverManager.getConnection(“jdbc:oracle:oci8:@db00.cse.cuhk.edu.hk”,
    “scott”, “tiger” );
Statement stmt = conn.createstatement();
String query = “SELECT name, rating FROM Sailors”;
ResultSet rs = stmt.executeQuery( query );
// loop through result tuples (rs is a cursor)
while ( rs.next() ) {
    String s = rs.getString( “name” );
    Int n = rs.getInt( “rating” );
    System.out.println( s + “ “ + n );
}
```

JDBC: An Example

```
DriverManager.registerDriver
    (“oracle.jdbc.driver.OracleDriver” );
```

Loading driver

```
Connection conn =
    DriverManager.getConnection(“jdbc:oracle:oci8:@db00.cse.cuhk.edu.hk”,
    “scott”, “tiger” );
Statement stmt = conn.createstatement();
String query = “SELECT name, rating FROM Sailors”;
ResultSet rs = stmt.executeQuery( query );
// loop through result tuples (rs is a cursor)
while ( rs.next() ) {
    String s = rs.getString( “name” );
    Int n = rs.getInt( “rating” );
    System.out.println( s + “ “ + n );
}
```

JDBC: An Example

```
DriverManager.registerDriver
    ("oracle.jdbc.driver.OracleDriver" );
Connection conn =
    DriverManager.getConnection("jdbc:oracle:oci8:@db00.cse.cuhk.edu.hk",
                                "scott", "tiger" );
Statement stmt = conn.createstatement();
String query = "SELECT name, rating FROM Sailors";
ResultSet rs = stmt.executeQuery( query );
// loop through result tuples (rs is a cursor)
while ( rs.next() ) {
    String s = rs.getString( "name" );
    Int n = rs.getInt( "rating" );
    System.out.println( s + " " + n );
}
```

Connect to the SQL server.
A connection object (named conn) will be created, if the login is valid.

url of the SQL server

Username and password

JDBC: An Example

```
DriverManager.registerDriver
    (“oracle.jdbc.driver.OracleDriver” );
Connection conn =
    DriverManager.getConnection(“jdbc:oracle:oci8:@db00.cse.cuhk.edu.hk”,
    “scott”, “tiger” );
Statement stmt = conn.createstatement();
String query = “SELECT name, rating FROM Sailors”;
ResultSet rs = stmt.executeQuery( query );
// loop through result tuples (rs is a cursor)
while ( rs.next() ) {
    String s = rs.getString( “name” );
    Int n = rs.getInt( “rating” );
    System.out.println( s + “ “ + n );
}
```

Use the createstatement method of conn to create a statement object.

JDBC: An Example

```
DriverManager.registerDriver
    (“oracle.jdbc.driver.OracleDriver” );
Connection conn =
    DriverManager.getConnection(“jdbc:oracle:oci8:@db00.cse.cuhk.edu.hk”,
    “scott”, “tiger” );
Statement stmt = conn.createstatement();
String query = “SELECT name, rating FROM Sailors”;
ResultSet rs = stmt.executeQuery( query );
// loop through result tuples (rs is a cursor)
while ( rs.next() ) {
    String s = rs.getString( “name” );
    Int n = rs.getInt( “rating” );
    System.out.println( s + “ “ + n );
}
```

Create a string object “query” to store the SQL statement.

JDBC: An Example

```
DriverManager.registerDriver
    (“oracle.jdbc.driver.OracleDriver” );
Connection conn =
    DriverManager.getConnection(“jdbc:oracle:oci8:@db00.cse.cuhk.edu.hk”,
    “scott”, “tiger” );
Statement stmt = conn.createstatement();
String query = “SELECT name, rating FROM Sailors”;
ResultSet rs = stmt.executeQuery( query );
// loop through result tuples (rs is a cursor)
while ( rs.next() ) {
    String s = rs.getString( “name” );
    Int n = rs.getInt( “rating” );
    System.out.println( s + “ “ + n );
}
```

Invoke the “executeQuery” method, which passes the SQL stored in “query” to the server. The result of the query will be stored in the ResultSet (cursor) object “rs”.

JDBC: An Example

```
DriverManager.registerDriver
    (“oracle.jdbc.driver.OracleDriver” );
Connection conn =
    DriverManager.getConnection(“jdbc:oracle:oci8:@db00.cse.cuhk.edu.hk”,
    “scott”, “tiger” );
Statement stmt = conn.createstatement();
String query = “SELECT name, rating FROM Sailors”;
ResultSet rs = stmt.executeQuery( query );
// loop through result tuples (rs is a cursor)
while ( rs.next() ) {
    String s = rs.getString( “name” );
    Int n = rs.getInt( “rating” );
    System.out.println( s + “ “ + n );
}
```

rs works like a cursor, but you don’t need to open it. Each time when this method is called, the pointer will be advanced to the next record in the result set and return true. It will return false when the cursor is positioned after the last record.

JDBC: An Example

```
DriverManager.registerDriver
    ("oracle.jdbc.driver.OracleDriver" );
Connection conn =
    DriverManager.getConnection("jdbc:oracle:oci8:@db00.cse.cuhk.edu.hk",
    "scott", "tiger" );
Statement stmt = conn.createstatement();
String query = "SELECT name, rating FROM Sailors";
ResultSet rs = stmt.executeQuery( query );
// loop through result tuples (rs is a cursor)
while ( rs.next() ) {
    String s = rs.getString( "name" );
    Int n = rs.getInt( "rating" );
    System.out.println( s + " " + n );
}
```

Retrieve the column values.
getString is used to get a string value.
getInt is used to get an integer value.
Similarly, there are getBoolean, getLong, and getFloat etc.
The argument can be a field name or a position e.g. rs.getString(1) and rs.getInt(2).

Print out s and n

JDBC: Database Update

```
Connection conn =
DriverManager.getConnection( “jdbc:oracle:oci8:@db00.cse.cuhk.edu.hk”, “scott”,
    “tiger” );
Statement stmt = conn.createstatement();

// create tables
stmt.executeUpdate( “create table Sailors” +
            “(sid integer, sname varchar(32),” +
            “rating integer, age float)” );
// insert values into the tables
stmt.executeUpdate ( “insert into Sailors values ( 22, ‘Dustin’, 7, 45.0 )” );
stmt.executeUpdate ( “insert into Sailors values ( 25, ‘Smith’, 8, 50.0 )” );
stmt.executeUpdate ( “insert into Sailors values ( 30, ‘Wang’, 9, 25.0 )” );
```

JDBC: Database Update

```
Connection conn =
DriverManager.getConnection( "jdbc:oracle:oci8:@db00.cse.cuhk.edu.hk", "scott",
    "tiger" );
Statement stmt = conn.createstatement();

// create tables
stmt.executeUpdate( "create table Sailors" +
            "(sid integer, sname varchar(32)," +
            "rating integer, age float)" );
// insert values into the tables
stmt.executeUpdate ( "insert into Sailors values ( 22, 'Dustin', 7, 45.0 )" );
stmt.executeUpdate ( "insert into Sailors values ( 25, 'Smith', 8, 50.0 )" );
stmt.executeUpdate ( "insert into Sailors values ( 30, 'Wang', 9, 25.0 )" );
```

executeUpdate is a method to execute SQL insert, update, and delete statements, or DDL statements.

JDBC: Commit execution

By default, any update is committed after it is finished.
But we can turn off the default:

```
conn.setAutoCommit(false)
```

Then we need to do the following:

```
<statement 1>
....
....
<statement n>
conn.commit()
```

All these statements will be considered as an atomic unit (transaction).

Use conn.rollback() to roll back (undo) all the updates.

JDBC: Prepared Statement

Allows one to pass the parameters just before execution.

Example:

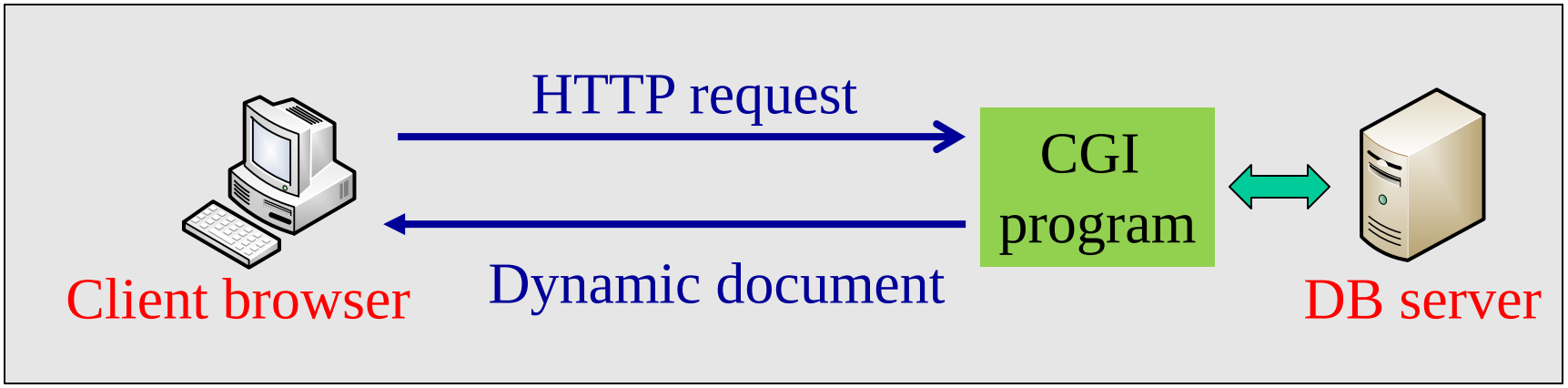
This method prepares a precomplied SQL statement and stores it in the PreparedStatement object.The parameters can be filled later.

```
PreparedStatement pstmt =
    conn.prepareStatement( " insert into Sailors values (?,?,?,?)" );
pstmt.setInt ( 1, 22 );
pstmt.setString ( 2, "Dustin" );
pstmt.setInt ( 3, 7 );
pstmt.setFloat ( 4, 45.0 );
pstmt.execute();
```

The "?" placeholders can be used anywhere in SQL statements, where they can be replaced with a value. They can also appear in where clause, e.g. "where age=?"

We may execute SQL multiple times, but we build the query only once i.e. more efficient. In addition it can prevent SQL injection.

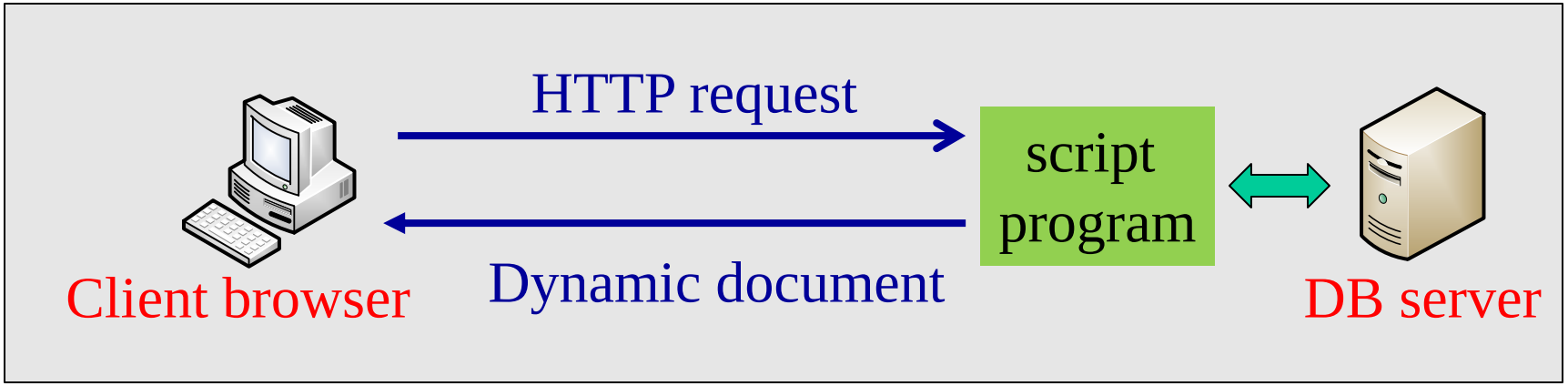
Web Applications (CGI)



- A client can send HTTP requests to the server.

  e.g. http://abc.com/cgi-bin/prog.pl?23
- The idea of Common Gateway Interface (CGI) is to execute a CGI program at the server site and send the output to the client's browser.
- CGI allows programmers to use any of several languages, e.g. C, C++, Bourne Shell, C Shell, Tcl or Perl.

Web Applications (script)



- Embed a script (e.g. PHP script) in an HTML document.
- The script will be run on the server site.
- It will be more efficient than CGI, if only a small part of the document is dynamic.
- Note, CGI will generate the entire document.

Appendix: SQL Injection

- SQL injection is a technique where malicious users can inject SQL commands into an entry field for execution.
- Suppose a program reads a username from a console or a web form, and then uses the input to construct a query as follows:

```
statement = “SELECT * FROM users WHERE name = ‘ ” + username + “ ’ ;”
```

- The statement is intended to retrieve the record for a particular username.

- However, the "username" variable can be crafted in a specific way by a malicious user.

  e.g. ' OR '1'='1

- The statement will become

```
SELECT * FROM users WHERE name = ‘ ’ OR ‘1’=‘1’;
```

- Therefore the malicious user can get all the information from the table.
- What will happen, if the following string is inserted?

```
a’; DROP TABLE users;  SELECT * FROM userinfo WHERE  ‘t’ = ‘t
```

Note: This example is taken from wiki (https://en.wikipedia.org/wiki/SQL_injection)