# 15. SQL Host-Language Interface

**API**：使用 API，我们可以通过写宿主语言程序来访问数据库，通过字符串将 SQL 指令传入函数中；

- 与嵌入式 SQL 相比，提高了 SQL 与宿主语言系统的独立性；
- API 的例子：
  - C + ODBC （SQL/CLI 标准）
  - Java + JDBC
  - PHP + PEAR/DB

## 15.1 SQL/CLI

**ODBC**：一个提供 SQL 语句执行的调用层级接口的 API；

**SQL/CLI**：ODBC 的改编版本，由 SQL 2016 提出，通过包含头文件 `"sqlcli.h"` 来调用。

**SQL/CLI 数据结构**：使用 CLI 的程序必须创建并处理四类结构：Environment 环境，Connection 连接，Statement 命令，Description 描述；这些结构形成了层次式结构。

**为数据结构创建句柄**

```
rt = SQLAllocHandle(hType, hIn, hOut)
```

- `hType` 表示句柄的类型，包括
  - `SQL_HANDLE_ENV`：创建 SQLHENV 句柄；
  - `SQL_HANDLE_DBC`：创建 SQLHDBC 类型的句柄；
  - `SQL_HANDLE_STMT`：创建 SQLHSTMT 类型的句柄；
- `hIn` 需要传入更高级别的句柄；
- `hOut` 返回创建句柄的地址；
- `rt` 为返回值，`0` 表示正常，其余表示异常。

**环境句柄**：在为与 DB 服务器的连接准备时创建。

```
SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &myEnv);
```

**连接句柄**：用来连接至 DB 服务器，必须在环境中创建。

```
SQLAllocHandle(SQL_HANDLE_DBC, myEnv, &myCon);
```

通过如下命令连接至 DB 服务器：

```
SQLConnect(myCon, sqlserver, SQL_NTS, userid, SQL_NTS, pword, SQL_NTS);
```

**命令句柄**：创建并发送一个 SQL 命令至 DB 服务器，必须在连接中创建。

```
SQLAllocHandle(SQL_HANDLE_STMT, myCon, &myStmt);
```

**命令的执行**：包括准备、执行两部分（或结合为直接执行）。

- **准备**：准备一条 SQL 指令的执行，生成查询计划。

  ```
  SQLPrepare(sh, st, sl);
  ```

  - `sh` 是命令句柄；
  - `st` 是一个指向 SQL 指令的指针（字符串）；
  - `sl` 表示 `st` 指向的字符串长度，如果不知道具体长度，则可以使用 `SQL_NTS` 表示以空 (NULL) 作为结束符。
- **执行**

  ```
  SQLExecute(sh);
  ```

- **直接执行**

  ```
  SQLExecuteDirect(sh, st, sl);
  ```

**取回数据**：包括绑定变量、取回数据两部分。

- **绑定变量**：将变量绑定至元组属性上

  ```
  SQLBindCol(sh, colNo, colType, pVar, varSize, varInfo);
  ```

  - `sh` 是命令句柄；
  - `colNo` 是元组内属性个数；
  - `colType` 表示元组内的属性的类型；
  - `pVar` 表示指向待储存位置的指针，是一个指针数组；
  - `varSize` 表示 `pVar` 指针数组的长度；
  - `varInfo` 是一个 `int` 类型指针，用来被 `SQLBindCol` 提供关于值的额外信息。
- **取回数据**

  ```
  SQLFetch(sh);
  ```

  - 数据被取回并且存入之前 `pVar` 指向的位置；
  - 返回一个 `SQLRETURN` 类型的值，`0` 表示成功，`SQL_NO_DATA` 表示已取完。

**向查询传递参数**：当 SQL 命令进行准备阶段时，允许命令中出现 `?` 占位符，但是执行前需要将占位符绑定参数。

```
SQLBindParameter(sh, parameter_position, SQL_PARAMETER_MODE, provide_type,
  hostVar_type, column, digit, hostVar, hostVar_len, &indicator);
```

- `sh` 为命令句柄；

- `parameter_position` 为参数的位置（第几个）；
- `SQL_PARAMETER_MODE` 表示参数的类型（输入/输出/输入输出），分别用 `SQL_PARAMETER_MODE_{IN / OUT / INOUT}` 表示。
- `provide_type` 表示 SQL 中提供的值类型；
- `hostVar_type` 表示宿主变量的类型；
- `column` 表示第几列的属性绑定至这个元素；
- `digit` 表示保留小数位数；
- `hostVar` 表示宿主变量；
- `indicator` 表示指示变量。

## 15.2 JDBC

**JDBC**：一个用来访问 SQL 数据库的 JavaAPI，在 Java 中通过如下指令导入：

```
import java.sql.*;
```

- 是 Java 语言的一部分；
- 面向对象特性；
- 通过网络访问数据库。

**驱动**：支持 JDBC 的数据库产品会提供 JDBC 驱动。

- 驱动将 JDBC-SQL 转换为 DBMS-SQL；
- Java 会在运行时夹在驱动；
- 应用程序通过驱动管理器来选择对于访问特定 DBMS 的驱动。

**加载驱动**：使用 `Class.forName()` 加载驱动，比如：

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

- 还可以使用 `com.mysql.jdbc.Driver`，`oracle.jdbc.driver.OracleDriver`，`com.Microsoft.sqlserver.jdbc.SQLServerDriver` 等等。
- 连接至超过一个数据库时，分别调用 `Class.forName` 即可。
- 在新版 Java 中，`getConnection()` 会直接加载相关数据库的驱动，不需要手动加载。

**连接**

- **建立与 DBMS 的连接**

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

  - `url` 表示 DBMS 的网址；
  - `uid` 与 `pwd` 表示用户名与密码。
- **关闭连接**

  ```
  myCon.close();
  ```

**命令执行**

- **创建命令**（与 CLI 创建命令句柄类似）

```
Statement myStmt = myCon.createStatement();
```

- **创建并准备（查询/修改）命令**（与 CLI 创建命令+准备命令类似）

```
PreparedStatement myPStmt = myCon.createStatement(Q);
```

- **准备并执行查询命令**（与 CLI 直接执行类似）

```
ResultSet rs = myStmt.executeQuery(Q);
```

- **准备并执行修改命令**：

```
myStmt.executeUpdate(U);
```

- **执行一个已准备的查询**

```
ResultSet rs = myPStmt.executeQuery();
```

- **执行一个已准备的修改**

```
myPStmt.executeUpdate();
```

**JDBC 中的游标**：`ResultSet` 内置了一个游标。

- 使用 `ResultSet.next()` 即可访问下一组数据（如果没有下一组数据，返回 `false`）；
- 访问第 $i$ 个数据可以使用 `getXXX(i)` 命令取出，其中 `XXX` 表示数据类型，比如：
  - `getString(1), getInt(2), getFloat(3)` 等等。

**参数传递**：`PreparedStatement` 中可以包含 `?` 占位符，在执行前需要进行参数传递。

```
myPStmt.setXXX(i, v);
```

- `i` 表示参数传递的位置（第几个）；
- `v` 表示传递的参数；
- `XXX` 需要使用参数类型替换，如 `setInt, setString` 等等。
- 整个语句的意思：用 `v` 替换第 `i` 个 `?`。

**SQLJ**：由于 JDBC 这种调用级接口在某些情况下效率低，特别是在静态情况下；所以 SQLJ 提供了 Java 的嵌入式 SQL 支持。

- 具体介绍略。

## 15.3 PHP

**PHP**：一个在 HTML 网页中使用的脚本语言，使用如下指令表示：

```
<? php
  PHP code
?>
```

**PHP 变量**：必须以 `$` 开始。

- 可以不定义变量的类型直接赋值。
- **对象变量**从类中建立。

  ```
  $obj = new myClass();
  ```

  那么，可以使用 `$obj -> myVar` 与 `$obj -> myFunc` 调用类中的变量与函数。
- 字符串值：PHP 的字符串值中，双引号表示字符串其中的相应内容应替换为其表示的变量的值，单引号表示不需要替换。

  ```
  $100 = "one hundred dollars.";
  $sue = 'You owe me $100.';
  $joe = "You owe me $100.";
  ```

  则 `$sue` 的值为其本身，`$joe` 的值为 `You owe me one hundred dollars.`。

**PHP 数组**：包括 `numeric` 与 `associative` 两类。

- `numeric` 数组：使用 0、1、2 作为下标；
- `associative` 数组：使用字符串作为下标，将字符串映射到一个值。如果 `x => y` 是 `associative` 数组的一部分，那么 `$a[x]` 的结果是 `y`。

**PEAR 库**：PHP 扩展与应用库，包括 DB 库；DB 库中包含许多与 JDBC 类似的函数，通过如下指令包含 DB 库：

```
include(DB.php);
```

**连接**：使用 `DB::connect` 连接数据库。

```
$myCon = DB::connect(vendor://username:password, hostname/database-name);
```

或使用 `MySQLi` 函数如下所示（访问 `localhost` 数据库，用户名 `root` 密码 `123`，数据库名称 `myDB`）。

```
$myCon = mysqli.connect("localhost", "root", "123", "myDB");
```

**执行 SQL 命令**：使用 `query` 函数即可。

```
$d = 'CS';
$result = $myCon -> query("SELECT name, age FROM S". "WHERE dept=$d;");
```

- 根据 PHP 字符串特性，可以方便地使用参数传递；
- 使用 `.` 连接字符串；
- 如果查询失败，则 `$result` 将会存一个错误代码；否则 `$result` 为游标。
- 特别地，若使用的信息是用户填在 Web 服务器上的，则可以使用 `associative` 数组 `$_POST` 代替，比如：

  ```
  result = $myCon -> query("INSERT INTO S(sno, name) VALUES(" .
                           "$_POST['num'], $_POST['name']");
  ```

**PHP 游标**：是一个 `numeric` 数组，可以通过 `fetchRow` 获得下一个元组或 `0`（如果已经是最后一个了），例如：

```
while($tuple = $result -> fetchRow()) {
  name = $tuple[0];
  age = $tupe[1];
  // process ...
}
```

**PHP 中动态 SQL**：与 JDBC 类似，使用 `?` 占位符并用一个参数数组填充，例如：

```
$pQuery = $myCon -> prepare("INSERT INTO S(sno, name) VALUES(?, ?)");
$args = array('007', 'Bond');
$res = $myCon -> execute($pQuery, $args);
```