《数据库系统原理》课程实验指导

OceanBase 数据库接口

2023年10月

目录

前 言

实验环境说明

本实验环境为任意受支持 Linux 系统上的 OceanBase v4.2.1 数据库， 实验数据采用电商数据库的八张表。

1 数据库接口实验

1.1 实验目的

- 了解数据库应用程序设计采用的 ODBC、JDBC 两种接口，掌握接口及相关软件配置能力。

- 编写数据库应用程序，通过数据库访问接口访问数据库，培养数据库应用程序开发能力。

1.2 实验原理

动态 SQL 与数据库应用编程接口

数据库应用程序设计是数据库应用开发的一个重要方面。数据库系统用户通过两种方式访问数据库:

1）直接使用 SQL 语句交互式访问数据库;

2）通过数据库应用程序，借助嵌入式 SQL 和高级程序设计语言，访问数据库

OceanBase 支持 SQL 语言直接访问数据库，但与高级程序设计语言（例如 C、C++、Java 等）相比，SQL 语言数据处理能力较弱，因此需要将 SQL 与高级程序设计语言结合起来，利用 SQL 访问数据，并将数据传递给高级语言程序进行处理，处理结果再利用 SQL 写回数据库。

这种嵌在高级语言程序中的 SQL 语句称为嵌入式 SQL（或者称为 ESQL），ESQL 随着应用程序执行被调用，完成数据库数据读写等数据管理功能，而高级语言程序则负责对数据库中数据进行统计分析等深层次处理。ESQL 分成两种:

1）静态 ESQL，在程序执行前 SQL 的结构就已经确定，但可以在执行时传递一些数值参数。

数据库系统执行静态 ESQL 时，首先利用预编译分离 C、C++、Java 等宿主程序语言中的 SQL 语句，代之以过程或函数调用，然后对剩余程序正常编译和连接库函数等。分离出来的 SQL 语句则在数据库端进行处理，进行语法检查、安全性检查和优化执行策略，绑定在数据库上形成包（packet）供应用程序调用。

2）动态 ESQL，数据库应用程序执行时才确定所执行的 SQL 语句的结构和参数，通过数据库应用编程接口 ODBC、JDBC 等访问数据库。

数据库系统执行动态 SQL 时，无法事先确实知道是什么样的 SQL 语句，从而无法进行静态绑定。这种绑定过程只能在程序执行过程中生成了确定的要执行的 SQL 语句时才能进行，称为动态绑定。

本次实验面向动态 SQL，应用程序采用 ODBC、JDBC 两种接口访问数据库。

1.3 实验内容

1. 了解通用数据库应用编程接口（例如 JDBC、ODBC 等）的配置方法。
2. 利用 C、C++、Java 等高级程序设计语言编程实现简单的数据库应用程序，掌握基于 ODBC、JDBC 接口的数据库访问的基本原理和方法，访问 LTE 网络数据库，执行查找、增加、删除、更新等操作，掌握基于应用编程接口的数据库访问方法。

1.4 实验要求

1. 基于 JDBC 接口或基于 ODBC 接口的数据库访问实验，二选一完成一个即可；

2. 实验时选取 TD-LTE 数据库作为数据源，参照后文中的示例，自行设计访问 TLE 网络数据库访问操作；

1.5 实验步骤

步骤 1.实验准备

以课堂所学关于 SQL 语言相关内容为基础，课后查阅、自学 ODBC、JDBC 等接口有关内容，包括体系结构、工作原理、数据访问过程、主要 API 接口的语法和使用方法等。

步骤 2. 数据库访问接口环境配置

根据实验所选的应用编程接口 ODBC、JDBC，分别从不同网站下载接口驱动程序，安装配置接口环境，为后续实验做准备。

步骤 3. 数据库连接及访问

针对 TD-LTE 网络配置数据库，编写 C、C++、Java 应用程序，通过 ODBC、JDBC 接口，连接数据库，对数据库内容进行查询、插入、删除、更新等操作，观察记录实验结果。

1.6 实验总结

在实验中有哪些重要问题或者事件？你如何处理的？你的收获是什么？有何建议和意见等等。

2 实验示例

2.1 JDBC 接口访问

Java 数据库连接，（Java Database Connectivity，简称 JDBC）是 Java 语言中用来规范客户端程序如何来访问数据库的应用程序接口，提供了诸如查询和更新数据库中数据的方法。用于执行 SQL 语句的 Java API 可以为多种关系数据库提供统一访问接口，应用程序可基于它操作数据。

2.1.1 安装配置 JDK

对于 Debian/CentOS 等系统，可使用 yum 安装。

```
    yum install java-1.8.0-openjdk-devel
```

如果是其他系统，可参考各自系统的类似`java-1.8.0-openjdk-devel`包的安装方式,或者使用`sdkman`工具进行安装。

2.1.2 MySQL Connector/J

安装 MySQL Connector/J，并配置运行环境。

2.1.2.1 下载并安装 MySQL Connector/J

推荐使用 MySQL Connector/J 5.1.47 版本。详细的下载及安装方法，请参考 Connector/J 下载(https://downloads.mysql.com/archives/c-j/)、Connector/J 安装(https://dev.mysql.com/doc/connector-j/8.1/en/connector-j-installing.html)。

例如，下载 mysql-connector-java-5.1.47.tar.gz 后，将其解压：

```
    tar -xzvf mysql-connector-java-5.1.47.tar.gz
```

2.1.2.2 配置 CLASSPATH

为了让 JVM 能够使用 mysql-connector, 需要配置 CLASSPATH。首先记录 mysql-connector 的下载路径，然后进行配置。示例如下。以使用 bash 为例，对~/.bashrc 进行更改，添加如下内容。

```
    export MYSQL_CONNECTOR_PATH=/root/mysql-connector-java-5.1.47/mysql-connector-java-5.1.47.jar
    export CLASSPATH=$MYSQL_CONNECTOR_PATH:$CLASSPATH
```

其中 MYSQL_CONNECTOR_PATH 为具体的 mysql-connector 的路径。

2.1.3 数据库连接及访问

2.1.3.1 步骤一：获取数据库连接串

回忆 OceanBase 数据库部署时相应的数据库连接串，例如：

```
obclient  -h127.1 -uroot -p12345678 -P2881 -Doceanbase -A
```

数据库连接串包含了访问数据库所需的参数信息，在创建应用程序前，可通过数据库连接串验证登录数据库，保证连接串参数信息正确。

参数说明：

- **-h**：OceanBase 数据库连接 IP，有时候是一个 ODP 地址。
- **-u**：租户的连接用户名，格式为 **用户@租户#集群名称**，集群的默认租户是 'sys'，租户的默认管理员用户是 'root'。直接连接数据库时不填集群名称，通过 ODP 连接时需要填写。
- **-p**：用户密码。
- **-P**：OceanBase 数据库连接端口，也是 ODP 的监听端口。
- **-D**：需要访问的数据库名称。

2.1.3.2 步骤二：编写应用程序

下文以 Linux 中通过 Java 驱动 `Connector/J 5.1.47` 连接数据库为例。

在正确安装 MySQL Connector/J 5.1.47 驱动并配置环境之后，可以通过以下 `OceanBase.java` 文件的示例代码进行数据库连接及使用。

注意

如果是 MySQL Connector/J 8.x 版本，`Class.forName("com.mysql.jdbc.Driver")` 中的 `com.mysql.jdbc.Driver` 需要替换成 `com.mysql.cj.jdbc.Driver`。

```
package OceanBase;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class OceanBase {
    // 创建数据库连接
    public static Connection GetConnection(String username, String password) {
        try {
            Class.forName("com.mysql.jdbc.Driver");
            String sourceURL = "jdbc:mysql://127.0.0.1:2881/tpc";
            Connection conn = DriverManager.getConnection(sourceURL, username,
password);
            System.out.println(conn.getAutoCommit());

            return conn;
```

```
            } catch (Exception e) {
                e.printStackTrace();
                return null;
            }
        }

        // 新建表 customer_t1
        public static void CreateTable(Connection conn, Statement sm) {
            try {
                // 新建表
                sm.executeUpdate(
                        "create table customer_t1 (c_customer_sk integer, c_customer_name
varchar(32))");
            } catch (Exception e) {
                e.printStackTrace();
            }

        }

        // 插入数据
        public static void InsertData(Connection conn, Statement sm) {
            try {
                sm.executeUpdate("insert into customer_t1 values (1,1)");
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        // 更新数据
        public static void UpdateData(Connection conn, Statement sm) {
            try {
                sm.executeUpdate(
                        "update customer_t1 set c_customer_name = 'new data' where
c_customer_sk = 1");
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        // 查询数据，并输出结果
        public static void PrintData(Connection conn, Statement sm) {
```

```
            try {
                ResultSet rs = sm.executeQuery("select * from customer_t1");
                while (rs.next()) {
                    String name = rs.getString("c_customer_name");
                    String id = rs.getString("c_customer_sk");
                    System.out.println("name: " + name + " id: " + id);
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        private static Statement newStatement(Connection conn) {
            try {
                Statement sm = conn.createStatement();

                return sm;
            } catch (Exception e) {
                e.printStackTrace();
                return null;
            }
        }

        public static void main(String[] args) {

            try {
                Connection connection = GetConnection("root", "12345678");

                Statement sm = newStatement(connection);

                // 创建表
                CreateTable(connection, sm);

                // 插入数据
                InsertData(connection, sm);

                // 更新数据
                UpdateData(connection, sm);

                // 打印数据
                PrintData(connection, sm);
```

```
            // 关闭连接
            connection.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

修改代码中的数据库连接参数。参考如下字段及拼接方法，对应的值，则取自步骤一获取的数据库连接串。

```
    connection =
DriverManager.getConnection("jdbc:mysql://{host}:{port}/{dbname}?user={username}&password={*
*****}")


    //示例
    jdbc:mysql://127.0.0.1:2881/test?user=root&password=12345678`
```

host：取自 `-h` 参数，OceanBase 数据库连接地址，有时候是 ODP 地址。

port：取自 `-P` 参数，OceanBase 数据库连接端口，也是 ODP 的监听端口。

dbname：取自 `-D` 参数，需要访问的数据库名称。

username：取自 `-u` 参数，租户的连接用户名，格式为 **用户@租户#集群名称**，集群的默认租户是 'sys'，租户的默认管理员用户是 'root'。直连数据库时不填写集群名称，通过 ODP 连接时需要填写。

password：取自 `-p` 参数，用户密码 。

2.1.4 运行应用程序

代码编辑完成后，可以通过如下命令进行编译：

```
    #配置临时环境配置，根据 mysql-connector-java-5.1.47.jar 实际安装路径填写
    export CLASSPATH=/usr/share/java/mysql-connector-java-5.1.47.jar:$CLASSPATH
    #编译
    javac -encoding UTF-8 OceanBase.java
```

```
[root@a8b67abca06c ~]# javac OceanBase/OceanBase.java -encoding UTF-8
```

编译完成后，通过如下命令运行 `OceanBase`：

```
    java OceanBase
    #输出以下结果说明数据库连接成功，示例语句正确执行
    true
    name: new data id: 1
```

```
[root@a8b67abca06c ~]# java OceanBase.OceanBase
Wed Oct 25 02:35:33 UTC 2023 WARN: Establishing SSL connection without server's identity veri
fication is not recommended. According to MySQL 5.5.45+, 5.6.26+ and 5.7.6+ requirements SSL
connection must be established by default if explicit option isn't set. For compliance with e
xisting applications not using SSL the verifyServerCertificate property is set to 'false'. Yo
u need either to explicitly disable SSL by setting useSSL=false, or set useSSL=true and provi
de truststore for server certificate verification.
true
name: new data id: 1
```

2.2 ODBC 接口访问

ODBC（Open Database Connectivity，开放数据库连接）是由 Microsoft 公司基于 X/OPEN CLI 提出的用于访问数据库的应用程序编程接口，应用程序通过 ODBC 提供的 API 与数据库进行交互，增强了应用程序的可移植性，扩展性和可编程性。ODBC 为异构数据库访问提供统一接口，允许应用程序以 SQL 为数据存取标准，存取不同 DBMS 管理的数据；使应用程序直接操纵 DB 中的数据，免除随 DB 的改变而改变。用 ODBC 可以访问各类计算机上的 DB 文件，甚至访问如 Excel 表和 ASCI I 数据文件这类非数据库对象。

unixODBC 是 ODBC 驱动管理器，用来装载及管理所有 ODBC 驱动，让用户能在 Unix/Linux 系统下使用 ODBC,类似于 JDBC 的 DriverManager。

2.2.1 ODBC 安装配置

本节将以 RHEL 系 Linux 系统为例演示如何安装配置相关工具。其他 Linux Distribution 的安装配置方式类似。

2.2.1.1 前提条件

在安装使用 MySQL Connector/C (libmysqlclient) 前请确保设置了基本的数据库开发环境，要求如下:

- GCC 版本为 3.4.6 及以上，推荐使用 4.8.5 版本。
- CMake 版本为 2.8.12 及以上。

2.2.1.2 步骤一：获取数据库连接串

联系 OceanBase 数据库部署人员或者管理员获取相应的数据库连接串，例如：

```
obclient -h127.1 -uroot -p12345678 -P2881 -Dtpc
```

数据库连接串包含了访问数据库所需的参数信息，在创建应用程序前，可通过数据库连接串验证登录数据库，保证连接串参数信息正确。

参数说明：

- **-h**：OceanBase 数据库连接 IP，有时候是一个 ODP 地址。
- **-u**：租户的连接用户名，格式为 **用户@租户#集群名称**，集群的默认租户是 `sys`，租户的默认管理员用户是 `root`。直接连接数据库时不填集群名称，通过 ODP 连接时需要填写。
- **-p**：用户密码。
- **-P**：OceanBase 数据库连接端口，也是 ODP 的监听端口。
- **-D**：需要访问的数据库名称。

2.2.1.3 步骤二：安装 MySQL Connector/C 驱动

通过 yum 安装 mariadb client

3. 安装 mariadb client。

```
sudo yum install mariadb-devel
```

2.2.2 数据库连接及访问

2.2.2.1 步骤一：编写应用程序

应用程序通过 MySQL Connector/C 与数据库服务器 OBServer 节点交互的基本方式如下：

1. 调用 `mysql_library_init()` 初始化 MySQL 库。

```
mysql_library_init(0, NULL, NULL);
```

2. 调用 `mysql_init()` 初始化一个连接句柄。

```
MYSQL *mysql = mysql_init(NULL);
```

3. 调用 `mysql_real_connect()` 连接到 OBServer 节点。

```
mysql_real_connect (mysql, host_name, user_name, password,
db_name, port_num, socket_name, CLIENT_MULTI_STATEMENTS)
```

4. 调用 `mysql_real_query()` 或 `mysql_query()` 向 OBServer 节点发送 SQL 语句。

```
mysql_query(mysql,"sql_statement");
```

5. 调用 `mysql_store_result()` 或 `mysql_use_result()` 处理其结果。

```
result=mysql_store_result(mysql);
```

6. 调用 `mysql_free_result()` 释放内存。

```
mysql_free_result(result);
```

7. 调用 `mysql_close()` 关闭与 OBServer 节点的连接。

```
mysql_close(mysql);
```

8. 调用 `mysql_library_end()` 结束 MariaDB client 的使用。

```
mysql_library_end();
```

2.2.2.2 示例代码

以 `mysql_test.c` 文件为例，代码如下：

```
#include "mysql.h"
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv) {
    mysql_library_init(0, NULL, NULL);
```

```
    MYSQL *mysql = mysql_init(NULL);
    const char *host_name = "127.0.0.1"; // set your mysql host
    const char *user_name = "root";      // set your user_name
    const char *password = "12345678";   // set your password
    const char *db_name = "test";        // set your databasename
    int port_num = 2883;                 // set your mysql port
    char *socket_name = NULL;
    MYSQL_RES *result;
    MYSQL_FIELD *fields;
    MYSQL_ROW row;
    int status = 0;
    /* connect to server with the CLIENT_MULTI_STATEMENTS option */
    if (mysql_real_connect(mysql, host_name, user_name, password, db_name,
                           port_num, socket_name,
                           CLIENT_MULTI_STATEMENTS) == NULL) {
        printf("mysql_real_connect() failed\n");
        mysql_close(mysql);
        exit(1);
    }

    /* execute multiple statements */
    status = mysql_query(mysql, "DROP TABLE IF EXISTS customer_t1;");

    if (status) {
        printf("Could not execute statement(s)");
        mysql_close(mysql);
        exit(0);
    }

    status = mysql_query(mysql, "CREATE TABLE customer_t1(c_customer_sk "
                                "integer, c_customer_name varchar(32));");
    status = mysql_query(
        mysql, "INSERT INTO customer_t1 VALUES(1,1),(20,'oceanbase');");
    status = mysql_query(
        mysql, "UPDATE customer_t1 SET c_customer_name='new data' WHERE id=1;");
    status = mysql_query(mysql, "SELECT * FROM customer_t1;");

    /* did current statement return data? */
    result = mysql_store_result(mysql);

    if (result) {
        /* yes; process rows and free the result set */
```

```
            // process_result_set(mysql, result);

            int num_fields = mysql_num_fields(result);
            int num_rows = mysql_num_rows(result);

            printf("result: %d rows %d fields\n", num_rows, num_fields);
            printf("----------------------\n");

            fields = mysql_fetch_fields(result);

            for (int i = 0; i < num_fields; ++i) {
                printf("%s\t", fields[i].name);
            }

            printf("\n----------------------\n");

            while ((row = mysql_fetch_row(result))) {
                for (int i = 0; i < num_fields; ++i) {
                    printf("%s\t", row[i] ? row[i] : "NULL");
                }

                printf("\n");
            }

            printf("----------------------\n");

            mysql_free_result(result);
        } else /* no result set or error */
        {
            if (mysql_field_count(mysql) == 0) {
                printf("%lld rows affected\n", mysql_affected_rows(mysql));
            } else /* some error occurred */
            {
                printf("Could not retrieve result set\n");
            }
        }

        status = mysql_query(mysql, "DROP TABLE customer_t1;");

        mysql_close(mysql);
        return 0;
    }
```

修改代码中的数据库连接参数。参考如下字段，对应的值，则取自步骤一获取的数据库连接串。

- **host_name**：取自 `-h` 参数，OceanBase 数据库连接地址，有时候是 ODP 地址。
- **user_name**：取自 `-u` 参数，租户的连接用户名，格式为 **用户@租户#集群名称**，集群的默认租户是 'sys'，租户的默认管理员用户是 `root`。直连数据库时不填写集群名称，通过 ODP 连接时需要填写。
- **password**：取自 `-p` 参数，用户密码。
- **db_name**：取自 `-D` 参数，需要访问的数据库名称。
- **port_num**：取自 `-P` 参数，OceanBase 数据库连接端口，也是 ODP 的监听端口。

2.2.3 运行应用程序

4. 代码编辑完成后，可以通过如下命令进行编译。

```
g++ -I/usr/include/mysql/ -L/usr/lib64/mysql/ -lmysqlclient mysql_test.c -o mysql_test
```

注意

如果没有安装 g++, 需要首先安装 gcc-c++后才能使用 g++命令。

```
yum install gcc-c++
```

选项说明：

- `-I` 选项：指定编译器的搜索路径，以便让编译器能够找到头文件。例如将 `mysql.h` 头文件所在的目录 `/usr/include/mysql` 添加到编译器的搜索路径中，使编译器能够找到 `mysql.h` 头文件。可以使用 `find / -name mysql.h 2>/dev/null` 这个命令查找 `mysql.h` 文件路径。
- `-L` 选项：指定动态链接库的搜索路径。
- `-l` 选项：指定需要链接的库文件。使用 `-l` 选项时，库文件名应该去掉 `lib` 前缀和 `.so` 后缀，例如上面的命令中库文件名为 `libmysqlclient.so`，但是使用 -l 选项时只需要指定 `mysqlclient`。

5. 指定运行路径。

```
export LD_LIBRARY_PATH=/usr/lib64/mysql
```

6. 通过如下命令运行应用程序。

```
./mysql_test
```

输出结果如下，输出如下结果，说明数据库连接成功，示例语句正确执行。

```
[root@a8b67abca06c ~]# ./ob_test
result: 2 rows 2 fields
----------------------
c_customer_sk   c_customer_name
----------------------
1       new data
20      oceanbase
----------------------
```