Differences between Loaders

Loaders 代码简介

1. `openDB`：打开数据库连接
2. `closeDB`：关闭数据库连接
3. `loadData`：加载数据
4. 核心区域：

```
while ((line = infile.readLine()) != null) {
    parts = line.split("\"");
    if (parts.length > 1) {
        studentid = parts[0].replace(",", "");
        name = parts[1];
        loadData(studentid, name);
        cnt++;
    }
}
```

AwfulLoader

AwfulLoader在每次 INSERT 前后都会打开数据库连接和关闭数据库连接，带来了不必要的开销：

```
while ((line = infile.readLine()) != null) {
    parts = line.split("\"");
    if (parts.length > 1) {
        studentid = parts[0].replace(",", "");
        name = parts[1];
        openDB(prop.getProperty("host"),
               prop.getProperty("database"),
               prop.getProperty("user"),
               prop.getProperty("password"));
        loadData(studentid, name);
        closeDB();
        cnt++;
    }
}
```

同时使用字符串拼接来构造 SQL 语句

```
private static void loadData(String studentid, String name) throws SQLException {
    Statement stmt;
    if (con != null) {
        stmt = con.createStatement();
        stmt.execute("insert into students(studentid,name) values('"
                     + studentid + "','" + name.replace("'", "''")
                     + "')");
    }
}
```

VeryBadLoader

相较于AwfulLoader, VeryBadLoader **复用了数据库连接**

```
openDB(prop.getProperty("host"), prop.getProperty("database"),
       prop.getProperty("user"), prop.getProperty("password"));

while ((line = infile.readLine()) != null) {
    parts = line.split("\"");
    if (parts.length > 1) {
        studentid = parts[0].replace(",", "");
        name = parts[1];
        loadData(studentid, name);
        cnt++;
    }
}

closeDB();
```

但仍然使用字符串拼接来构造SQL语句

BadLoader

BadLoader使用了 **PreparedStatement** ，可用于在大量相同查询的情况下提升性能

```
stmt = con.prepareStatement("insert into students(studentid,name)"
                            + " values(?,?)");
```

在 **loadData** 时，使用了一开始预编译的 **PreparedStatement** ，在大量执行的时候可以省去重复parse和planning的时间

```
stmt.setString(1, studentid);
stmt.setString(2, name);
stmt.executeUpdate();
```

See official document: https://jdbc.postgresql.org/documentation/head/server-prepare.html

> Server Prepared Statements
>
> The PostgreSQL™ server allows clients to compile sql statements **that are expected to be reused to avoid the overhead of parsing and planning the statement for every execution**.
>
> This functionality is available at the SQL level via PREPARE and EXECUTE beginning with server version 7.3, and at the protocol level beginning with server version 7.4, but as Java developers we really just want to use the standard PreparedStatement interface.

> Server side prepared statements can improve execution speed as
>
> 1. It sends just statement handle (e.g. `S_1`) instead of full SQL text
> 2. It enables use of binary transfer (e.g. binary int4, binary timestamps, etc); the parameters and results are much faster to parse
> 3. It enables the reuse server-side execution plan
> 4. The client can reuse result set column definition, so it does not have to receive and parse metadata on each execution

AverageLoader

AverageLoader 在打开数据库连接后，关掉了 **AutoCommit**

```
con = DriverManager.getConnection(url, props);
// open connection successfully
con.setAutoCommit(false);
// Disable Auto Commit
```

并在关闭连接前**一次性** commit **所有更改**：

```
openDB(prop.getProperty("host"), prop.getProperty("database"),
       prop.getProperty("user"), prop.getProperty("password"));
while ((line = infile.readLine()) != null) {
    parts = line.split("\"");
    if (parts.length > 1) {
        studentid = parts[0].replace(",", "");
        name = parts[1];
        loadData(studentid, name);
        cnt++;
    }
}
//////////////////////
con.commit();
//////////////////////
closeDB();
```

See official document: https://www.postgresql.org/docs/13/populate.html#DISABLE-AUTOCOMMIT

> ### 14.4.1. Disable Autocommit
>
> When using multiple `INSERT`s, turn off autocommit and just do one commit at the end. (In plain SQL, this means issuing `BEGIN` at the start and `COMMIT` at the end. Some client libraries might do this behind your back, in which case you need to make sure the library does it when you want it done.) **If you allow each insertion to be committed separately, PostgreSQL is doing a lot of work for each row that is added.** An additional benefit of doing all insertions in one transaction is that if the insertion of one row were to fail then the insertion of all rows inserted up to that point would be rolled back, so you won't be stuck with partially loaded data.

a lot of work:

如果不关掉AutoCommit，pgSQL会给每一次 **INSERT** 自动套上一层 **BEGIN ... COMMIT**，从而带来不断开启 **TRANSACTION** 的开销

GoodLoader

相较于 AverageLoader，GoodLoader 使用了**批量提交** (Batch)

```
// loadData
stmt.setString(1, studentid);
stmt.setString(2, name);
stmt.addBatch();
```

```
// Add each row
while ((line = infile.readLine()) != null) {
    parts = line.split("\"");
    if (parts.length > 1) {
        studentid = parts[0].replace(",", "");
        name = parts[1];
        loadData(studentid, name);
        cnt++;
        if (cnt % BATCH_SIZE == 0) {
            stmt.executeBatch();
            stmt.clearBatch();
        }
    }
}
if (cnt % BATCH_SIZE != 0) {
    stmt.executeBatch();
}
con.commit();
```