

《数据库概论》*

实验II 实验报告

* 教师: 张剡.

实验二：用户自定义完整性约束及使用高级程序设计语言访问数据库

张逸凯 171840708

Department of Computer Science and Technology

Nanjing University

zykhelloha@gmail.com

Tel: 18051988316

目录

目录

实验环境

实验过程

创建一个新的数据库

分析并添加约束

添加主键与设置非空

外键约束

用户自定义约束

违反数据约束的操作

实体完整性: 加入一个带有NULL属性的数据

参照完整性: 删除一个部门

参照完整性: 修改后的值超出外键范围

参照完整性: (元组级约束) 加入一个主键重复的元素

用户定义的完整性规则: 加入职工表的成员年龄需要大于18岁

创建触发器

当插入一项工作时, 如果工作时间大于24, 则将其设置为24。

当职工参加一个新的项目时, 年薪增加5%; 如果该职工是某部门的负责人, 则再增加3%。

使用高级程序设计语言 (以Java为例) 访问上述数据库

通过数据库连接API直接访问

使用高级程序设计语言执行输入的SQL查询语句, 输出查询结果 (对 staff 表进行查询操作)

使用高级程序设计语言执行输入的SQL插入、删除、更新语句(采用嵌入式SQL)

测试封装好的代码(不少于五个测试用例, 包含动态SQL)

工业界的动态SQL实现

在高级程序设计语言中通过数据库连接池来管理连接

附加探究: 对不同的连接情况进行测试:

调整DBCP的配置参数, 以获得更快的速度

多线程连接数据库

实验中遇到的困难及解决办法

参考文献及致谢

实验环境

操作系统: Windows 10

软件环境: MySQL workbench 8.0 CE, MySQL 8.0.18

实验过程

创建一个新的数据库

在该数据库下按要求创建如下表格, 完成后加入样本数据

职工（姓名，工号，年龄，年薪，所在部门编号）；

部门（部门名称，部门编号，部门负责人的工号）；

项目（项目名称，项目编号，主管部门编号）；

工作（职工工号，项目编号，工作时间）。

```
1  create table Staff(  
2      name char(20),  
3      id int,  
4      age int,  
5      salary int,  
6      dept_id int  
7  );  
8  
9  create table Department(  
10     dept_name char(20),  
11     dept_id int,  
12     principal_id int  
13 );  
14  
15 create table Project(  
16     proj_name char(20),  
17     proj_id int,  
18     dept_id int  
19 );  
20  
21 create table Job(  
22     staff_id int,  
23     proj_id int,  
24     time int  
25 );  
26  
27 insert into Staff  
28 values  
29 ('zhang', 1, 20, 100, 10),  
30 ('chen', 2, 21, 120, 10),  
31 ('liu', 3, 22, 120, 11),
```

```

32 ('lin', 4, 22, 121, 11),
33 ('liu', 5, 24, 122, 12),
34 ('liu', 6, 21, 123, 12);
35
36 insert into Department
37 values
38 ('technology', 10, 2),
39 ('HR', 11, 3),
40 ('marketing', 12, 6);
41
42 insert into Project
43 values
44 ('app-develop', 20, 10),
45 ('recruit', 21, 11),
46 ('advertise', 22, 12);
47
48 insert into Job
49 values
50 (1, 20, 10),
51 (2, 20, 12),
52 (3, 21, 8),
53 (4, 21, 9),
54 (5, 22, 7),
55 (6, 22, 8);

```

查看各个表情况:

职工表:

```
57 • select * from Staff;
```

58

Result Grid					
Filter Rows:					
	name	id	age	salary	dept_id
▶	zhang	1	20	100	10
	chen	2	21	120	10
	liu	3	22	120	11
	lin	4	22	121	11
	liu	5	24	122	12
	liu	6	21	123	12

部门表:

```
57 • select * from Department;
```

58

Result Grid		
Filter Rows:		
	dept_name	dept_id
▶	technology	10
	HR	11
	marketing	12

项目表:

57 • `select * from Project;`

58

	proj_name	proj_id	dept_id
▶	app-develop	20	10
	recruit	21	11
	advertise	22	12

工作表:

57 • `select * from Job;`

58

	staff_id	proj_id	time
▶	1	20	10
	2	20	12
	3	21	8
	4	21	9
	5	22	7
	6	22	8

编号采用了统一的格式, 比如职工编号没有前缀, 部门编号带有前缀 1, 项目编号带有前缀 2 等.

字段的命名也较通俗易懂.

分析并添加约束

添加主键与设置非空

```

1  #添加主键:
2  alter table Staff add primary key (id);
3  alter table Department add primary key (dept_id);
4  alter table Project add primary key (proj_id);
5  alter table Job add primary key (staff_id, proj_id);
6
7
8  #设置非空:
9  alter table Staff modify id int NOT NULL;
10 alter table Staff modify name char(20) NOT NULL;
11
12 alter table Department modify dept_id int NOT NULL;
13 alter table Department modify dept_name char(20) NOT NULL;
14
15 alter table Project modify proj_id int NOT NULL;
16 alter table Project modify proj_name char(20) NOT NULL;
17
18 alter table Job modify staff_id int NOT NULL;
19 alter table Job modify proj_id int NOT NULL;

```

外键约束

项目表的主管部门编号 引用(或者说外键是)部门表的部门编号

```

1 alter table Project
2 add constraint dept_id
3 foreign key fk_Dept(dept_id)
4 REFERENCES Department(dept_id)
5 ON DELETE NO ACTION      #按照题目要求阻止delete，部门一般不会被删除。
6 ON UPDATE CASCADE;      #按照题目要求同步更新，更新部门编号时项目编号同步更新。

```

工作表的职工工号 引用(或者说外键是)职工表的工号

```

1 alter table Job
2 add foreign key fk_Staff(staff_id)
3 REFERENCES Staff(id)
4 ON DELETE CASCADE
5 ON UPDATE CASCADE;

```

工作表的项目编号 引用(或者说外键是)项目表的项目编号

```

1 alter table Job
2 add foreign key fk_Project(proj_id)
3 REFERENCES Project(proj_id)
4 ON DELETE CASCADE      #项目结束被删除后工作表中同步删除。
5 ON UPDATE CASCADE;

```

部门表的负责人工号 引用(或者说外键是)职工表的工号

```

1 alter table Department
2 add foreign key fk_Staff(principal_id)
3 REFERENCES Staff(id)
4 ON DELETE NO ACTION
5 ON UPDATE CASCADE;

```

职工表的所在部门编号 引用(或者说外键是)部门表的部门编号

```

1 alter table Staff
2 add foreign key fk_Department(dept_id)
3 REFERENCES Department(dept_id)
4 ON DELETE NO ACTION
5 ON UPDATE CASCADE;

```

用户自定义约束

```

1 alter table Staff
2 ADD CONSTRAINT check_staff_age
3 check(age >= 18);

```

设置加入职工表的数据需要年龄大于18岁。

违反数据约束的操作

这里实践了三类数据完整性约束：

实体完整性：加入一个带有NULL属性的数据

但是在一个基表的主关键字中，其属性的取值不能为空值，并且也已设置非空。

```
1 insert into Staff
2 values
3 (NULL, 7, 21, 130, 10);
```

```
117 • insert into Staff
```

```
118 values
```

```
119 (NULL, 7, 21, 130, 10);
```

```
120
```

```
121
```

Output			
Action Output			
#	Time	Action	Message
139	18:55:24	alter table Job add foreign key fk_Staff(staff_id) REFERENCES Staff(id) ON DELETE CASCADE ON UPDATE CASCADE	6 row(s) affected Records: 6 Duplicates: 0 Warnings: 0
140	18:55:24	alter table Job add foreign key fk_Staff(staff_id) REFERENCES Staff(id) ON DELETE CASCADE ON UPDATE CASCADE	6 row(s) affected Records: 6 Duplicates: 0 Warnings: 0
141	18:55:25	alter table Job add foreign key fk_Staff(staff_id) REFERENCES Staff(id) ON DELETE CASCADE ON UPDATE CASCADE	6 row(s) affected Records: 6 Duplicates: 0 Warnings: 0
142	18:59:17	insert into Staff values (NULL, 7, 21, 130, 10)	Error Code: 1048. Column 'name' cannot be null

参照完整性：删除一个部门

MySQL 拒绝删除，因为设置了ON DELETE NO ACTION.

```
1 delete from Department
2 where dept_id = 10;
```

```
113 • delete from Department
```

```
114 where dept_id = 10;
```

```
115
```

```
116
```

Output			
Action Output			
#	Time	Action	Message
133	18:48:15	alter table Job add foreign key fk_Project(proj_id) REFERENCES Project(id) ON DELETE CASCADE ON UPDATE CASCADE	6 row(s) affected Records: 6 Duplicates: 0 Warnings: 0
134	18:49:13	alter table Department add foreign key fk_Staff(principal_id) REFERENCES Staff(id) ON DELETE NO ACTION ON UPDATE CASCADE	3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0
135	18:49:16	alter table Staff add foreign key fk_Department(dept_id) REFERENCES Department(dept_id) ON DELETE NO ACTION ON UPDATE CASCADE	6 row(s) affected Records: 6 Duplicates: 0 Warnings: 0
136	18:49:23	delete from Department where dept_id = 10	Error Code: 1451. Cannot delete or update a parent row: a foreign key constraint fails ('new_schema`.`project`, CONSTRAINT `dept_id` FOREIGN KEY (principal_id) REFERENCES Staff (id) ON DELETE NO ACTION ON UPDATE CASCADE)

参照完整性：修改后的值超出外键范围

```
1 update Staff set dept_id = 999 where name = 'zhang';
```

```
127 • update Staff set dept_id = 999 where name = 'zhang';
```

```
128
```

```
129
```

Output			
Action Output			
#	Time	Action	Message
143	19:04:23	insert into Staff values ('test', 2, 21, 130, 10)	Error Code: 1062. Duplicate entry '2' for key 'PRIMARY'
144	19:13:13	update Staff set dept_id = 999 where name = 'zhang'	Error Code: 1175. You are using safe update mode and you tried to update a table without a WHERE that uses a KEY column. To disable safe mode, please switch to safe update mode by setting the variable 'safe_update' to 0.
145	19:13:49	SET SQL_SAFE_UPDATES = 0	0 row(s) affected
146	19:13:52	update Staff set dept_id = 999 where name = 'zhang'	Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails ('new_schema`.`staff`, CONSTRAINT `staff_dept_id` FOREIGN KEY (dept_id) REFERENCES Department (dept_id) ON DELETE NO ACTION ON UPDATE CASCADE)

参照完整性：（元组级约束）加入一个主键重复的元素

```
1 insert into Staff
2 values
3 ('test', 2, 21, 130, 10);
```

```
121 • insert into Staff
```

```
122 values
```

```
123 ('test', 2, 21, 130, 10);
```

```
124
```

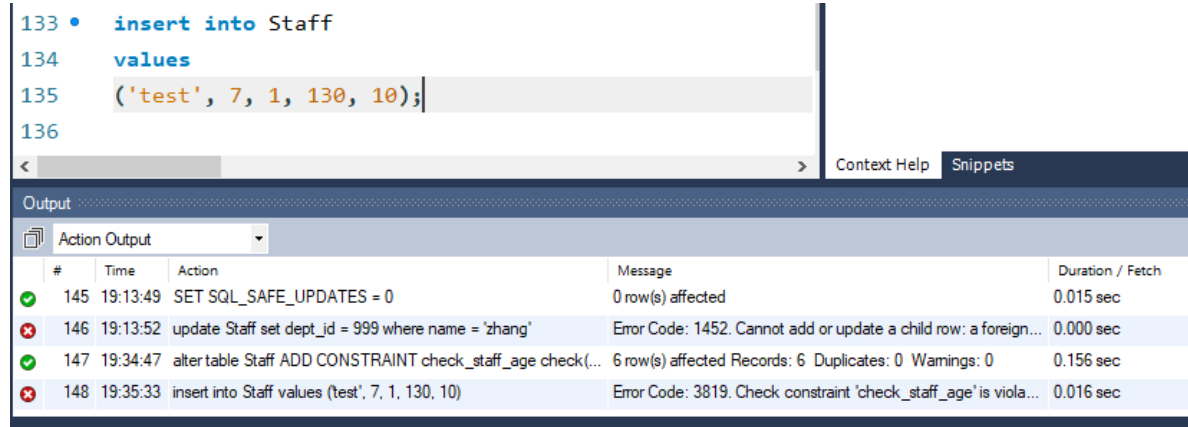
```
125
```

Output			
Action Output			
#	Time	Action	Message
140	18:55:24	alter table Job add foreign key fk_Staff(staff_id) REFERENCES Staff(id) ON DELETE CASCADE ON UPDATE CASCADE	6 row(s) affected Records: 6 Duplicates: 0 Warnings: 0
141	18:55:25	alter table Job add foreign key fk_Staff(staff_id) REFERENCES Staff(id) ON DELETE CASCADE ON UPDATE CASCADE	6 row(s) affected Records: 6 Duplicates: 0 Warnings: 0
142	18:59:17	insert into Staff values (NULL, 7, 21, 130, 10)	Error Code: 1048. Column 'name' cannot be null
143	19:04:23	insert into Staff values ('test', 2, 21, 130, 10)	Error Code: 1062. Duplicate entry '2' for key 'PRIMARY'

可以发现, 在以上违反约束的操作中, 并且破坏了数据的完整性 (即违反完整性约束条件的要求) 时, 系统拒绝执行, 并报警或报错.

用户定义的完整性规则: 加入职工表的成员年龄需要大于**18**岁

```
1  insert into Staff
2  values
3  ('test', 7, 1, 130, 10);
```



133 • insert into Staff
134 values
135 ('test', 7, 1, 130, 10);
136

Output

Action Output

#	Time	Action	Message	Duration / Fetch
✓ 145	19:13:49	SET SQL_SAFE_UPDATES = 0	0 row(s) affected	0.015 sec
✗ 146	19:13:52	update Staff set dept_id = 999 where name = 'zhang'	Error Code: 1452. Cannot add or update a child row: a foreign...	0.000 sec
✓ 147	19:34:47	alter table Staff ADD CONSTRAINT check_staff_age check(...	6 row(s) affected Records: 6 Duplicates: 0 Warnings: 0	0.156 sec
✗ 148	19:35:33	insert into Staff values ('test', 7, 1, 130, 10)	Error Code: 3819. Check constraint 'check_staff_age' is viola...	0.016 sec

创建触发器

注意 `delimiter` 后面要加空格.

当插入一项工作时, 如果工作时间大于**24**, 则将其设置为**24**。

```
1  delimiter
2  create trigger before_update_job_time
3  before insert on Job
4  for each row
5  begin
6      if (new.time > 24)
7          then
8              set new.time = 24;
9          end if;
10 end;
```

```

138 delimiter
139 • create trigger before_update_job_time
140 before insert on Job
141 for each row
142 begin
143     if (new.time > 24)
144     then
145         set new.time = 24;
146     end if;
147 end;
148

```

Output				
Action Output				
#	Time	Action	Message	
✓ 188	20:11:33	create trigger before_update_jo...	0 row(s) affected	
✓ 189	20:11:33	create trigger before_update_jo...	1 row(s) returned	

验证设置工时触发器是否起效

```

139 • insert into Job
140 values
141 (2, 22, 99);
142
143 • select *
144 from Job
145 where staff_id = 2 and proj_id = 22;
146
147 delimiter

```

Result Grid			
staff_id	proj_id	time	
2	22	24	
* NULL	NULL	NULL	

Output				
Action Output				
#	Time	Action	Message	
✓ 195	20:20:05	insert into Job values (2, 22, 99)	1 row(s) affected	
✓ 196	20:20:52	select * from Job where staff_id ...	1 row(s) returned	

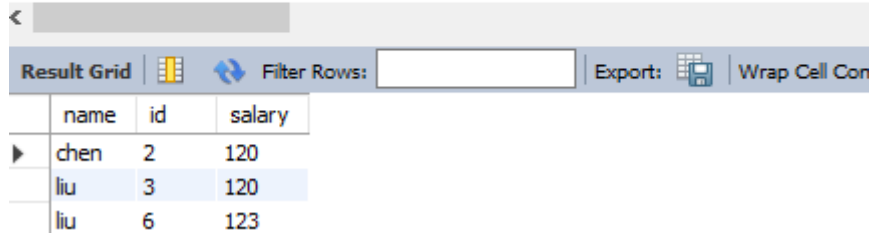
我们发现当想要加入一个工作时间为99的数据时被修改为24;

当职工参加一个新的项目时，年薪增加5%；如果该职工是某部门的负责人，则再增加3%。

```
1  delimiter
2  create trigger before_update_job_join
3  after insert on Job
4  for each row
5  begin
6      if(new.staff_id in
7          (select principal_id from Department)
8      )
9      then update Staff
10         set Staff.salary = Staff.salary * 1.08
11         where Staff.id = new.staff_id;
12     else
13         update Staff
14         set Staff.salary = Staff.salary * 1.05
15         where Staff.id = new.staff_id;
16     end if;
17 end;
```

验证触发器是否有效

```
142  select DISTINCT name, id, salary
143  from Staff, Department
144  where id = Department.principal_id;
145
```



	name	id	salary
▶	chen	2	120
	liu	3	120
	liu	6	123

首先如上图找出负责人. 然后添加一个项目给id为2的职员(负责人):

```

148 • insert into Job
149 values
150 (2, 21, 2);
151
152 • select DISTINCT name, id, salary
153 from Staff, Department
154 where id = 2;
155

```

Result Grid

	name	id	salary
▶	chen	2	129.60000000000002

Result 24 ×

Output

Action Output

	#	Time	Action	Message
✓	216	21:06:43	insert into Job values (2, 21, 2)	1 row(s) affected
✓	217	21:06:48	select DISTINCT name, id, salar...	1 row(s) returned

可以发现触发器起作用, id 为2的职员(负责人)年薪为 $120 \times 1.08 = 129.6$, 因为增长了5%后增长3%, 这里记为增长8%.

使用高级程序设计语言（以Java为例）访问上述数据库

通过数据库连接API直接访问

JDBC (Java Data Base Connectivity), 首先下载 jar 包等进行连接:

```

1 public class Homework {
2     static final String JDBC_DRIVER = "com.mysql.cj.jdbc.Driver";
3     static final String DB_URL = "jdbc:mysql://localhost:3306/new_schema?
allowPublicKeyRetrieval=true&useSSL=false&serverTimezone=UTC";
4
5     // 数据库的用户名与密码
6     static final String USER = "root";
7     static final String PASS = "123456";
8
9     public static void main(String[] args) {
10         Connection conn = null;
11         Statement stmt = null;
12         PreparedStatement pst = null;
13         ResultSet rs = null;
14
15         try {
16             // 注册JDBC接口驱动
17             Class.forName(JDBC_DRIVER);

```

```

18
19 // 打开链接
20 System.out.println("连接数据库...");
21 conn = DriverManager.getConnection(DB_URL, USER, PASS);
22 System.out.println("连接成功");
23
24 pst = conn.prepareStatement("select * from Staff;");
25 // 获取结果集
26 rs = pst.executeQuery();
27 // ... 对结果集进行解析并输出

```

使用高级程序设计语言执行输入的SQL查询语句，输出查询结果（对 `Staff` 表进行查询操作）

```

18
19 try{
20     Class.forName(JDBC_DRIVER);
21
22     // 打开链接
23     System.out.println("连接数据库...");
24     conn = DriverManager.getConnection(DB_URL, USER, PASS);
25     System.out.println("连接成功");
26
27     pst = conn.prepareStatement("select * from Staff;");
28     rs = pst.executeQuery();
29

```

Javadoc Declaration Console X
 <terminated> Homework [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (2019年11月22日 下午7:52:16)

连接数据库...

连接成功

Name	ID	Age	Salary	Department ID
zhang	1	20	100.00	10
chen	2	21	129.60	10
liu	3	22	120.00	11
lin	4	22	121.00	11
liu	5	24	122.00	12
liu	6	21	123.00	12

结果正确.

使用内嵌的 `MySQL` 语句即可:

```

1 pst = conn.prepareStatement("select * from Staff;");
2 rs = pst.executeQuery();

```

使用高级程序设计语言执行输入的SQL插入、删除、更新语句(采用嵌入式SQL)

插入:

```

1 // commands[i] 为 insert into Staff values ('xu', 7, 21, 130, 10);
2 pst = conn.prepareStatement(commands[i]);
3 int resultSet = pst.executeUpdate();
4 if (resultSet > 0) {
5     System.out.println("Insert completed.");
6 }
7 else {
8     System.out.println("Insert failed.");
9 }

```

```

30 commands[0] = "insert into Staff values ('xu', 7, 21, 130, 10);";
31 commands[1] = "update Staff set age = 10 where id = 7;";
32 commands[2] = "delete from Staff where id = 7;";
33
34 for (int i = 0; i < 3; ++i) {
35     pst = conn.prepareStatement(commands[i]);
36
37     int resultSet = pst.executeUpdate();
38     if (resultSet > 0){
39         System.out.println("Insert completed.");
40     }
41     else {
42         System.out.println("Insert failed.");
43     }

```

Javadoc
 Declaration
 Console

<terminated> Homework [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (2019年11月23日 上午9:03:44)

连接数据库...

连接成功

Insert completed.

Name	ID	Age	Salary	Department ID
zhang	1	20	100.00	10
chen	2	21	129.60	10
liu	3	22	120.00	11
lin	4	22	121.00	11
liu	5	24	122.00	12
liu	6	21	123.00	12
xu	7	21	130.00	10

在插入后执行 select 语句, 发现 ('xu', 7, 21, 130, 10); 数据成功被插入。

更新:

```

1 // commands[i] 为 update Staff set name = 'huang' where id = 7;
2 pst = conn.prepareStatement(commands[i]);
3 int resultSet = pst.executeUpdate();
4 if (resultSet > 0) {
5     System.out.println("Insert completed.");
6 }
7 else {
8     System.out.println("Insert failed.");
9 }

```

```
31      commands[1] = "update Staff set name = 'huang' where id = 7;";
```

Javadoc Declaration Console X

<terminated> Homework [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (2019年11月23日 上午9:13:00)

连接数据库...
连接成功
Insert completed.

Name	ID	Age	Salary	Department ID
zhang	1	20	100.00	10
chen	2	21	129.60	10
liu	3	22	120.00	11
lin	4	22	121.00	11
liu	5	24	122.00	12
liu	6	21	123.00	12
huang	7	21	130.00	10

可以发现上图中 新添加的数据, `name` 已经被更改为 "huang".

在其中还出现了一个因为之前有设置 `check` 语句, 所以Update被禁止的情况, 这说明 `java` 连接, 与直接进行数据库操作是完全一致的.

```
31      commands[1] = "update Staff set age = 10 where id = 7;";
```

Javadoc Declaration Console X

<terminated> Homework [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (2019年11月23日 上午9:09:29)

连接数据库...
`java.sql.SQLException: Check constraint 'check_staff_age' is violated.`
at mysql.connector.java@8.0.16/com.mysql.cj.jdbc.exceptions.SQLException.createSQLException(SQLException)
at mysql.connector.java@8.0.16/com.mysql.cj.jdbc.exceptions.SQLException.createSQLException(SQLException)

删除

```
1 // commands[i] 为 delete from Staff where id = 7;  
2 pst = conn.prepareStatement(commands[i]);  
3 int resultSet = pst.executeUpdate();  
4 if (resultSet > 0) {  
5     System.out.println("Insert completed.");  
6 }  
7 else {  
8     System.out.println("Insert failed.");  
9 }
```

```
32      commands[2] = "delete from Staff where id = 7;";  
33
```

Javadoc Declaration Console X

<terminated> Homework [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (2019年11月23日 上午9:14:50)

连接数据库...
连接成功
Insert completed.

Name	ID	Age	Salary	Department ID
zhang	1	20	100.00	10
chen	2	21	129.60	10
liu	3	22	120.00	11
lin	4	22	121.00	11
liu	5	24	122.00	12
liu	6	21	123.00	12

通过 `select` 语句可以发现 `id = 7` 的数据已经被删除

测试封装好的代码(不少于五个测试用例, 包含动态SQL)

静态SQL

```

1  commands[0] = "insert into Staff values ('xu', 7, 21, 130, 10);";
2  commands[1] = "update Staff set name = 'huang' where id = 7;";
3  commands[2] = "delete from Staff where id = 7;";
4  commands[3] = "alter table Department add primary key (dept_id);";
5  commands[4] = "alter table Project modify proj_id int NOT NULL;";
6  // 测试过程见上节。

```

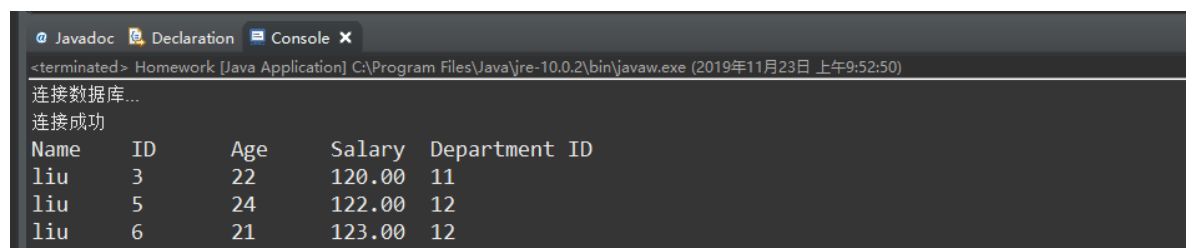
动态SQL: 可以根据用户输入动态确定查询参数个数, 查询类型:

```

1  List<String> params = new ArrayList<>();
2  StringBuilder sb = new StringBuilder();
3  sb.append("SELECT * FROM Staff WHERE 1 = 1");
4
5  if (searchName != null) {
6      sb.append(" AND name = ?");
7      params.add(searchName);
8  }
9  if (searchId != null) {
10     sb.append(" AND id = ?");
11     params.add(searchId);
12 }
13 if (searchDepartment != null) {
14     sb.append(" AND dept_id = ?");
15     params.add(searchDepartment);
16 }
17 PreparedStatement preparedStatement = conn.prepareStatement(sb.toString());
18 for (int i = 1; i <= params.size(); i++) {
19     preparedStatement.setString(i, params.get(i));
20 }
21 ResultSet resultSet = preparedStatement.executeQuery();

```

当测试用例输入为: `name = 'liu'` 时查询结果: (可以动态调整查询对象以及参数个数)



```

<terminated> Homework [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (2019年11月23日 上午9:52:50)
连接数据库...
连接成功
Name    ID    Age    Salary  Department ID
liu     3     22     120.00  11
liu     5     24     122.00  12
liu     6     21     123.00  12

```


可以发现上述**拼接SQL语句**的方法并不是很好, 而且较为麻烦

工业界的动态SQL实现

如下搭建 `springboot` + `MyBatis` 框架实现动态SQL:

`MyBatis` 是基于Java的持久层框架, 可以使用简单的 XML 或注解来配置和映射原生类型,

因为 `IDEA` 集成了需要的包以及相关框架, 所以用以下用 `IDEA` 搭建 `springboot` + `mybatis` 框架, 实现动态SQL:

 New Project

Project Metadata

Group:

Artifact:

Type:

Language:

Packaging:

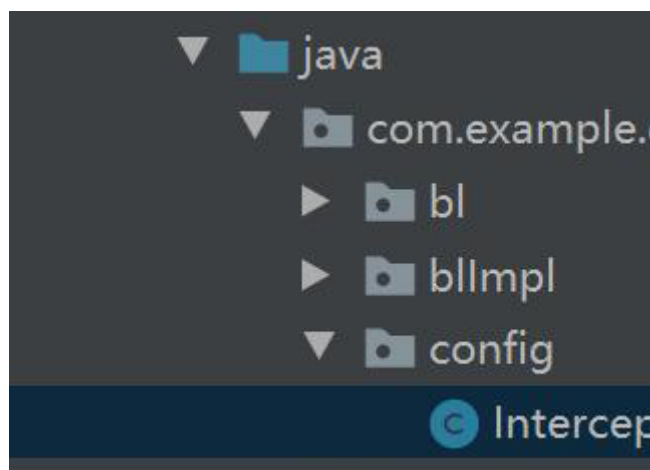
Java Version:

Version:

Name:

Description:

Package:



```

1 <update id="updateDepartment">
2     update Department
3     <set>
4         `name` = #{name},
5         `column` = #{column},
6         `row` = #{row}
7     </set>
8     where dept_id=#{dept_id}
9 </update>

```

如上可以传入参数, 动态更新 Department 表信息;

```

1 <update id="updateJob" parameterType="pers.ZhangYikai.MySQLconnection">
2     update Job
3     <set>
4         <if test="name != null">
5             name = #{username,jdbcType=CHAR},
6         </if>
7         <if test="dept_name != null">
8             `dept_name` = #{password,jdbcType=CHAR},
9         </if>

```

```

10         <if test="salary != null">
11             salary = #{salary,jdbcType=DECIMAL},
12         </if>
13         <if test="isUsed != null">
14             isUsed = #{isUsed,jdbcType=BOOLEAN},
15         </if>
16     </set>
17     where job_id = #{inputID,jdbcType=INTEGER} limit 1
18 </update>
19

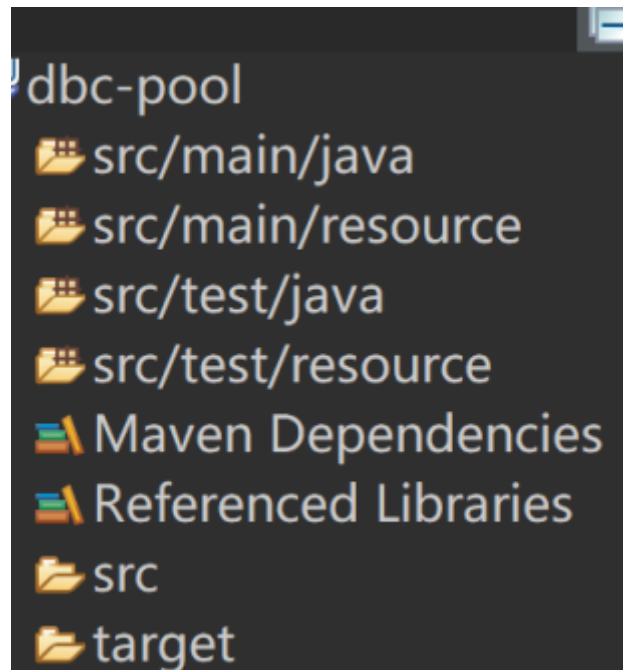
```

上述代码对于多表进行动态查询和设置,可以发现对于SQL正文动态可变,变量个数动态可变,引用对象动态可变.

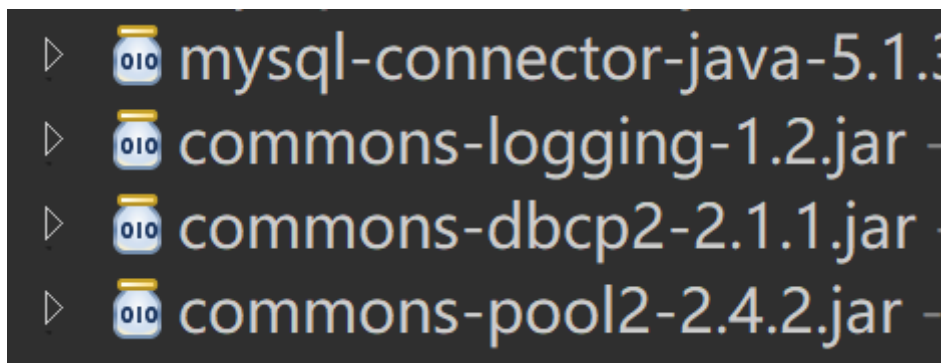
在高级程序设计语言中通过数据库连接池来管理连接

DBCP (DataBase Connection Pool) 是Java数据库连接池的一种,通过数据库连接池,可以让程序自动管理数据库连接的释放和断开。

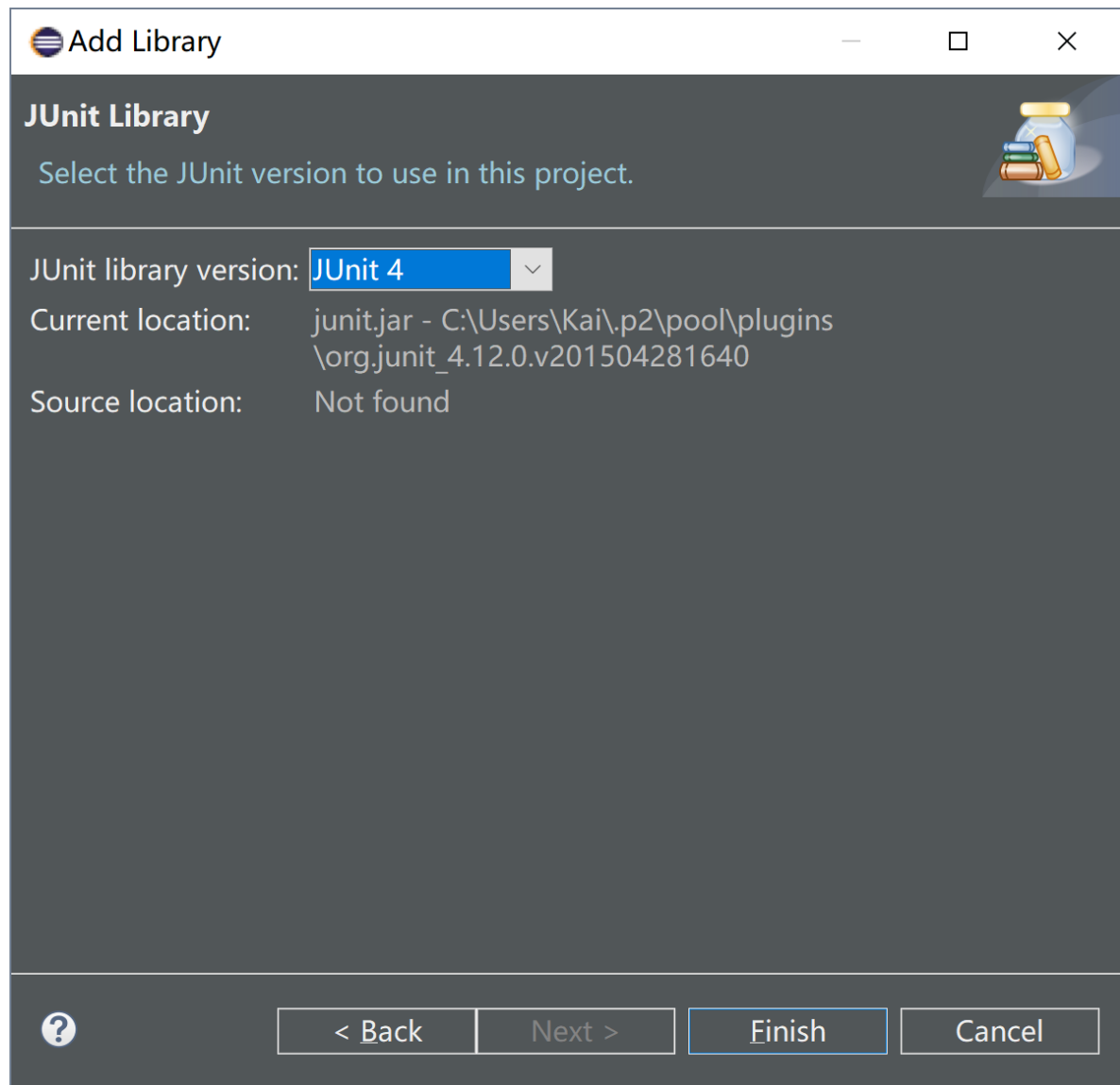
项目架构



所需要的jar包:



加入 junit 进行连接时长测试: (第二小题)



```
1 // DBCP 连接配置代码:
2 public class MyConfig {
3     private static Properties properties = new Properties();
4     private static DataSource dataSource;
5     //加载DBCP配置文件
6     static {
7         try {
8             FileInputStream is = new
9             FileInputStream("config/dbcp.properties");
10            properties.load(is);
11        } catch(IOException e){
12            e.printStackTrace();
13        }
14        try{
15            dataSource =
16            BasicDataSourceFactory.createDataSource(properties);
17        } catch(Exception e){
18            e.printStackTrace();
19        }
20    }
21    // 从连接池中获取一个连接
22    public static Connection getConnection(){
```

```

23     Connection connection = null;
24     try{
25         connection = dataSource.getConnection();
26     }catch(SQLException e){
27         e.printStackTrace();
28     }
29     try {
30         connection.setAutoCommit(false);
31     } catch (SQLException e) {
32         e.printStackTrace();
33     }
34     return connection;
35 }
36
37 public static void main(String[] args) {
38     getConnection();
39 }
40 }

```

```

1  // 获取数据库连接代码:
2  package pers.ZhangYikai.MySQLconnection;
3
4  import java.sql.Connection;
5  import java.sql.DriverManager;
6
7  public class DBConn {
8      private static Connection conn = null;
9
10     // 获取数据库连接
11     public static Connection getConnection() {
12         try {
13             Class.forName("com.mysql.cj.jdbc.Driver");
14             DriverManager.registerDriver(new com.mysql.jdbc.Driver());
15             String dbUrl = "jdbc:mysql://localhost:3306/new_schema?
allowPublicKeyRetrieval=true&useSSL=false&serverTimezone=UTC";
16             conn = DriverManager.getConnection(dbUrl, "root", "123456");
17             System.out.println("=====数据库连接成功=====");
18         } catch (Exception e) {
19             e.printStackTrace();
20             System.out.println("=====数据库连接失败=====");
21             return null;
22         }
23         return conn;
24     }
25 }

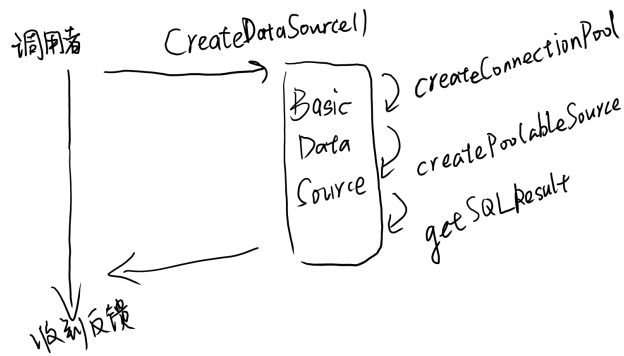
```

```

<terminated> DBCPTest [JUnit] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (2019年11月27日 上午9:18:35)
=====数据库连接成功=====

```

获得连接的过程详述:



1. 创建"物理"连接
2. 创建连接池
3. 创建"池化"的数据区
4. 获得结果

附加探究：对不同的连接情况进行测试：

使用 junit 单元测试(JUnit是一个Java语言的单元测试框架, JUnit 在测试驱动的开发方面有很重要的发展, 是起源于 JUnit 的一个统称为 xUnit 的单元测试框架之一), 进行不同情况的连接效率测试.

```

1 package pers.Zhangyikai.MySQLconnection;
2
3 import java.sql.Connection;
4 import java.sql.SQLException;
5 import java.sql.Statement;
6
7 import org.junit.Test;
8
9 public class MyTest {
10     // 测试类...
  
```

1. 写 n 条数据对应新建 n 个连接

```

1 @Test
2 public void EveryConnection() throws Exception{
3     for(int i = 0; i < 2000; i++) {
4         String sql = "insert into myInsertTestTable values (" + i + ")";
5         Connection conn = DBConn.getConnection();
6         try{
7             Statement stat = conn.createStatement();
8             stat.executeUpdate(sql);
9         }catch(Exception e){
10             e.printStackTrace() ;
11         }finally{
12             try {
13                 conn.close();
14             } catch (SQLException e) {
15                 e.printStackTrace();
16             }
17         }
18     }
19 }
  
```

```

17     }
18     }
19 }

```

可以看到在2000次for语句里, 每次连接都建立了一个新的连接, 然后**通过新的连接进行插入操作**.

2.通过DBCP数据连接池插入, 每次插入之前建立与连接池的连接

```

1     @Test
2     public void DBCP() throws Exception{
3         for(int i = 0; i < 2000; i++){
4             String sql = "insert into myInsertTestTable values (" + i + ")";
5             try {
6                 Connection conn = MyConfig.getConnection();
7                 Statement stat = conn.createStatement();
8                 stat.executeUpdate(sql);
9                 conn.commit();
10                conn.close();
11            } catch (SQLException e) {
12                e.printStackTrace();
13            }
14        }
15    }

```

同样是n个连接对应n个插入操作, **但是通过数据连接池进行连接**;

3. n 个插入操作只通过一条连接

```

1     @Test
2     public void OneConnection() throws Exception{
3         Connection conn = DBConn.getConnection();
4         Statement stat = conn.createStatement();
5         for(int i = 0; i < 2000; i++){
6             String sql = "insert into myInsertTestTable values (" + i + ")";
7             try{
8                 stat.executeUpdate(sql);
9             }catch(Exception e){
10                e.printStackTrace() ;
11            }
12        }
13        conn.close();
14    }

```

从代码中可以看到, **只建立了一个连接, 并进行2000次插入操作**.

4.通过DBCP数据连接池插入, 只建立一个与连接池的连接

```

1     @Test
2     public void DBCPoneConnection() throws Exception{
3         Connection conn = KCYDBCUtil.getConnection();
4         Statement stat = conn.createStatement();

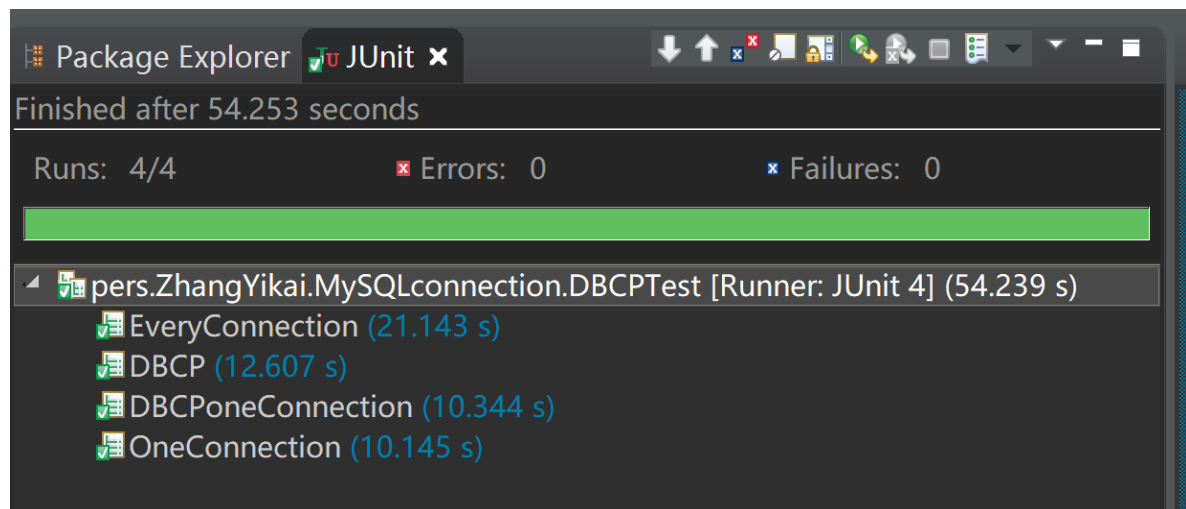
```

```

5         for(int i = 0; i < 2000; i++){
6             String sql = "insert into myInsertTestTable values (" + i + ")";
7             try {
8                 stat.executeUpdate(sql);
9                 conn.commit();
10            } catch (SQLException e) {
11                e.printStackTrace();
12            }
13        }
14        conn.close();
15    }

```

测试结果如下:



(1) 每次插入一条数据前就创建一个连接，该条数据插入完成后关闭该连接。耗时 21.143 秒

(2) 使用DBCP连接池，**每次插入一条数据前，从DBCP连接池中获取一条连接**，该条数据插入完成后**该连接交由DBCP连接池管理**。耗时 12.607 秒

(3) 在插入数据之前创建一条连接，2000个数据全部使用该连接。耗时 10.145 秒

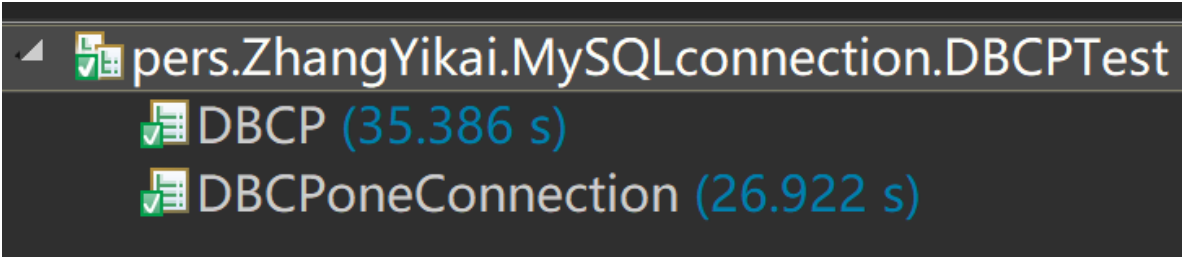
(4) 使用DBCP连接池，**只从DBCP连接池中获取一条连接**，耗时 10.344 秒

调整DBCP的配置参数，以获得更快的速度

参数(阅读手册后得知)

参数	描述
username	传递给JDBC驱动的用于建立连接的用户名
password	传递给JDBC驱动的用于建立连接的密码
url	传递给JDBC驱动的用于建立连接的URL
driverClassName	使用的JDBC驱动的完整有效的java 类名
connectionProperties	当建立新连接时被发送给JDBC驱动的连接参数
defaultAutoCommit	连接池创建的连接的默认的auto-commit状态
defaultReadOnly	连接池创建的连接的默认的read-only状态.
defaultTransactionIsolation	连接池创建的连接的默认的TransactionIsolation状态
initialSize	初始化连接:连接池启动时创建的初始化连接数量
maxActive	最大活动连接:连接池在同一时间能够分配的最大活动连接的数量
maxIdle	最大空闲连接:连接池中容许保持空闲状态的最大连接数量,超过的空闲连接将被释放
minIdle	最小空闲连接:连接池中容许保持空闲状态的最小连接数量,低于这个数量将创建新的连接

之前使用默认参数或者没有配置好参数时, 运行时长很不理想.

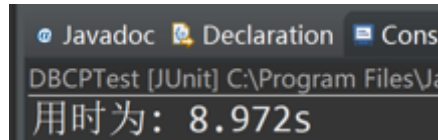


经过多次实验调整参数, 得到了上述较理想的结果.

```

1 // 以下是多线程的核心代码:
2 for(int i = 0; i < threadCount; i++) {
3     Thread thread = new Thread(new Worker(dataBase,latch,i*every,
4     (i+1)*every,flag++,ch++,path,user,pwd,table,url));
5     thread.start();
6 }
7 try {
8     latch.await();
9     long endTimes = System.currentTimeMillis();
10    System.out.println("用时为: " + (endTimes - startTimes)/1000 + "s");
11 } catch (InterruptedException e) {
12     e.printStackTrace();
13 }

```



可以发现多线程连接在多核CPU下速度有一定的提高.

实验中遇到的困难及解决办法

复习老师上课的PPT¹, 以及参考教材²和其他练习题, 完成本次实验还是遇到了很多困难, 比较不容易的部分在动态 SQL 和各种连接方式的差异比较以及多线程连接. 因为之前没有上过 java, 都是从头开始自学, 很难但也收获了不少.

解决办法就是上网学习, 请教同学, 看文档.

参考文献及致谢

[1] 南京大学 2019年春季 数据库概论 课程PPT

[2] 徐洁磐, 柏文阳, 刘奇志. 数据库系统实用教程. 高等教育出版社, 2006

[3] <https://commons.apache.org/proper/commons-dbcp/configuration.html>

[4] <https://commons.apache.org/proper/commons-dbcp/>

[5] <https://java-source.net/open-source/connection-pools/jakarta-dbcp>

[6] 数据库 MySQL连接 菜鸟教程: <https://www.runoob.com/java/java-mysql-connect.html>

[7] <https://stackoverflow.com/questions/36678555/building-a-dynamic-sql-query-based-on-user-input-in-java>

[8] <https://blog.jooq.org/tag/dynamic-sql/>

[9] <https://examples.javacodegeeks.com/core-java/sql/jdbc-query-builder-tutorial/>

因为图比较大, 不小心超过了页数😓, 辛苦了! 谢谢老师和助教哥的批改~