数据库原理 第五次实验报告

09019106 牟倪

目录

[一、应用程序概述 2](#_Toc90499460)

[1.1 应用场景 2](#_Toc90499461)

[1.2 主要需求 2](#_Toc90499462)

[二、数据库设计 3](#_Toc90499463)

[2.1 实体与关系设计 3](#_Toc90499464)

[2.2 数据库模式（database schema） 3](#_Toc90499465)

[三、功能设计 5](#_Toc90499466)

[四、软件接口实现 6](#_Toc90499467)

[4.1 SQL语句编写 6](#_Toc90499468)

[4.2 接口代码编写思路 7](#_Toc90499469)

[4.2.1 建立数据库连接 7](#_Toc90499470)

[4.2.2 与数据库交换信息 8](#_Toc90499471)

[4.3 实现模糊搜索 9](#_Toc90499472)

[五、软件接口测试 9](#_Toc90499473)

[六、实验体会 11](#_Toc90499474)

# 一、应用程序概述

## 1.1 应用场景

特展作为展品的全新组织形式，将具有某些相关性的展品（如同一年代、同一主题、同一形式）进行集中展览，一般由第三方主办，特定博物馆承办。一场精彩的特展，不仅聚集了丰富的展品，更聚集了大量领域爱好者甚至专家；人们不仅可以对展品大饱眼福，更可以了解领域的历史沿革，享受观点与思维的碰撞。

对特展从组织到展览的全过程进行考察，如下需求浮出水面：

* 在组织者的角度，举办一场特展需要确定展览地点和展品，需要统一的管理系统来记录特展的地点和展品信息。
* 在博物馆方的角度，如果没有统一的管理系统，很难考察文物在特定时间的所在地，进而很难确定文物在某段时间是否已经被特展借出、是否可以接受新的特展预定请求。
* 在游客的角度，游客可能希望参与特定主题的特展，希望参与位于特定城市/特定博物馆的特展，希望参与特定时间段的特展，希望去特展观看特定文物；建立统一的管理系统，就可以实现特展信息的多样化查询。同时，游客可能希望了解特展的口碑和人气，管理系统可以记录游客对特展的评价，并将评价进行展示以供其他游客参考。

因此，我们考虑建立统一的管理系统，对特展信息进行管理。

## 1.2 主要需求

* 特展组织者角度：
  + 高效管理（增删改查）特展信息（名称、地点、场所（博物馆）、起始时间、展品等）。
* 博物馆角度：
  + 高效管理馆藏文物信息，记录文物出展的时间表。
  + 对特展请求进行筛选，判断特展是否可以预定文物。
* 游客角度：
  + 对特展信息进行可视化展示。
  + 通过主题/城市/博物馆/日期/文物，查询特展信息。
  + 对特展进行评价。
  + 查看其他用户对特展的评价。

# 二、数据库设计

## 2.1 实体与关系设计

设计如下实体：

* 管理员：管理系统的管理员，具有id、密码属性。
* 用户：普通用户，具有id、账号、密码属性。
* 博物馆：具有博物馆id、名称、简介、图片、所在城市属性。
* 文物：具有文物id、名称、所属博物馆id属性。
* 特展：具有特展id、名称、开始日期、结束日期、简介、图片、所在博物馆id属性。

设计如下关系：

* <博物馆>拥有<文物>：1 – N。
* <博物馆>举办<特展>：M – N。
* <特展>展出<文物>：1 – N，在给定时刻，文物最多被一个特展展出。
* <用户>喜欢<特展>：M – N。
* <管理员>管理<特展>：M – N。

## 2.2 数据库模式（database schema）

database schema如下所示（标\*的属性为主键）：

admin (

\*adminid: auto-increment unsigned int,

password: non-**null** varchar(255)

)

exhibition (

\*eid: auto-increment unsigned int,

ename: non-**null** varchar(255),

start\_date: sql date,

end\_date: sql date,

eintro: non-**null** varchar(255),

epicture: non-**null** varchar(255),

mid: unsigned int

)

likelist (

\*uid: unsigned int,

\*eid: unsigned int

)

museum (

\*mid: auto-increment unsigned int,

mname: non-**null** varchar(255),

mintro: non-**null** varchar(255),

mpicture: non-**null** varchar(255),

city: non-**null** varchar(255)

)

showlist (

\*eid: unsigned int,

\*tid: unsigned int

)

treasure (

\*tid: auto-increment unsigned int,

tname: non-**null** varchar(255),

mid: unsigned int

)

user (

\*uid: auto-increment unsigned int,

uname: non-**null** varchar(255),

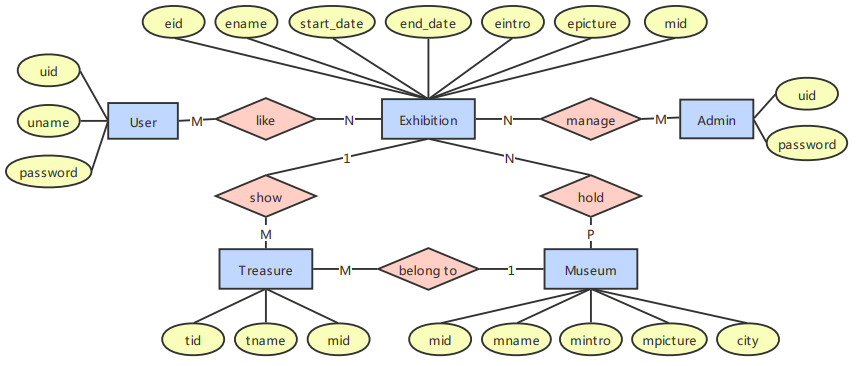
password: non-**null** varchar(255)

)

其中，

* showlist表：eid属性为关联到exhibition表的外键、tid属性为关联到treasure表的外键。
* likelist表：uid属性为关联到user表的外键、eid属性为关联到exhibition表的外键。

该data schema依据“一事一地”原则设计，不存在部分函数依赖，不存在属性对主键的传递依赖，属于关系数据库的第三范式。

ER图如下所示：

# 三、功能设计

应用程序与数据库交互的功能，大致分为以下3部分：

* 管理员 & 用户相关：
  + 通过管理员id，查询密码。
  + 通过用户id，查询密码。
  + 增删改exhibition。
  + 增删改showlist。
  + 增删likelist。
  + 通过用户id，查询用户所有喜欢特展的id。
* 特展相关：
  + 查询所有特展id+名称+开始时间+结束时间。
  + 通过特展名字，查询特展id+名称+开始时间+结束时间。
  + 通过日期，查询特展id+名称+开始时间+结束时间。
  + 通过城市，查询特展id+名称+开始时间+结束时间。
  + 通过文物名称，查询特展id+名称+开始时间+结束时间。
  + 通过博物馆名称，查询特展id+名称+开始时间+结束时间。
  + 通过特展id，查询博物馆id+名称。
  + 通过特展id，查询文物id+名称。
* 博物馆相关：
  + 查询所有博物馆id+名称。
  + 通过博物馆名字，查询博物馆id+名称。
  + 通过城市，查询博物馆id+名称。
  + 通过特展名称，查询博物馆id+名称。
  + 通过文物名称，查询博物馆id+名称。
  + 通过博物馆id，查询文物id+名称。
  + 通过博物馆id，查询特展id+名称+开始时间+结束时间。

由于该项目是我们小组选择的最终项目，因此小组成员对所有功能进行分工实现。队友叶若天负责的一部分，汤雯婧负责第三部分，我负责第二部分。

# 四、软件接口实现

## 4.1 SQL语句编写

|  |  |
| --- | --- |
| 查询所有  特展id+名称+开始时间+结束时间。 | SELECT E.eid, E.ename, E.start\_date, E.end\_date from exhibition E |
| 通过特展名字，  查询特展id+名称+开始时间+结束时间。 | SELECT E.eid, E.ename, E.start\_date, E.end\_date from exhibition E WHERE E.ename LIKE 'tempname ' |
| 通过日期，  查询特展id+名称+开始时间+结束时间。 | SELECT E.eid, E.ename, E.start\_date, E.end\_date from exhibition E WHERE 'date' BETWEEN E.start\_date AND E.end\_date |
| 通过城市，  查询特展id+名称+开始时间+结束时间。 | SELECT E.eid, E.ename, E.start\_date, E.end\_date from exhibition E WHERE E.mid IN (SELECT M.mid FROM museum M WHERE M.city LIKE 'tempname') |
| 通过文物名称，  查询特展id+名称+开始时间+结束时间。 | SELECT E.eid, E.ename, E.start\_date, E.end\_date FROM exhibition E  WHERE E.eid IN (SELECT S.eid FROM show S WHERE S.tid IN (SELECT T.tid FROM treasure T WHERE T.tname LIKE 'tempname')) |
| 通过博物馆名称，  查询特展id+名称+开始时间+结束时间。 | SELECT E.eid, E.ename, E.start\_date, E.end\_date FROM exhibition E WHERE E.mid IN (SELECT M.mid FROM museum M WHERE M.mname LIKE 'tempname') |
| 通过特展id，  查询博物馆id+名称。 | SELECT M.mid, M.mname FROM museum M WHERE M.mid IN (SELECT M.mid FROM exhibition E WHERE E.eid = eid) |
| 通过特展id，  查询文物id+名称。 | SELECT T.tid, T.tname, T.mid FROM treasure T WHERE T.tid IN (SELECT S.tid FROM showlist S WHERE S.eid = eid) |

## 4.2 接口代码编写思路

### 4.2.1 建立数据库连接

与数据库交互，首先需要建立数据库连接。关键代码如下：

1. static { *// 只做一次*
2. try {
3. Class.forName(driver);
4. con = DriverManager.getConnection(url, user, password);
5. if (!con.isClosed()) {
6. *//                System.out.println("数据库连接成功");*
7. }
8. } catch (ClassNotFoundException e) {
9. e.printStackTrace();
10. System.out.println("数据库驱动没有安装");
11. } catch (SQLException e) {
12. e.printStackTrace();
13. System.out.println("数据库连接或操作失败");
14. }
15. }

执行SQL查询并获得执行结果的关键代码如下：

1. public ResultSet search(String sql) { *// 数据库查询操作，sql为数据库操作指令*
2. ResultSet resultSet = null;
3. try {
4. resultSet = this.statement.executeQuery(sql);
5. } catch (SQLException e) {
6. e.printStackTrace();
7. }
8. return resultSet;
9. }

执行SQL增删改的关键代码如下：

1. public void adm(String sql) { *//adm为add delete modify 增删改的意思*
2. try {
3. statement.executeUpdate(sql);
4. } catch (SQLException e) {
5. e.printStackTrace();
6. }
7. }

这些代码被封装在DbOperation类中，DbOperation类用来与数据库建立连接、交换信息。

### 4.2.2 与数据库交换信息

利用DbOperation类，与数据库交换信息。基本流程如下：

* 声明新的数据库连接。
* 执行SQL语句，接收返回结果。
* 遍历并解析SQL语句的返回结果。

以查询所有特展的id+名称+开始日期+结束日期为例：

1. public static List<Exhi\_info3> getAllExhibition() throws SQLException {
2. DbOperation db = new DbOperation();
3. List<Exhi\_info3> lst = new ArrayList<>();
4. ResultSet res = db.search("SELECT E.eid, E.ename, E.start\_date, E.end\_date from exhibition E");
5. String eid = null, ename = null, start\_date = null, end\_date = null;
6. while (res.next()) {
7. eid = res.getString("eid");
8. ename = res.getString("ename");
9. start\_date = res.getString("start\_date");
10. end\_date = res.getString("end\_date");
11. Exhi\_info3 temp = new Exhi\_info3();
12. temp.setEid(eid);
13. temp.setEname(ename);
14. temp.setStart\_date(start\_date);
15. temp.setEnd\_date(end\_date);
16. lst.add(temp);
17. }
18. return lst;
19. }

## 4.3 实现模糊搜索

当通过特展名称/文物名称查询特展时，模糊搜索（而非精确查找）显然更符合实际应用场景。

可以利用SQL的LIKE运算符实现模糊搜索。得到查询关键词后，在字符串首尾、每个字与字之间都插入一个'%'（如 "123" → "%1%2%3" ），以实现正则表达式的匹配。

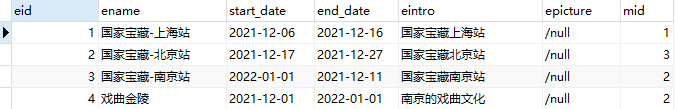
关键代码如下：

1. String tempname = "%";
2. for (int i = 0; i < ename.length(); ++i) {
3. tempname = tempname + origin.charAt(i) + "%";
4. }

# 五、软件接口测试

对特展相关的软件接口进行测试。数据库数据如下：

Exhibition：



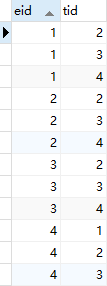
Museum：



Treasure：



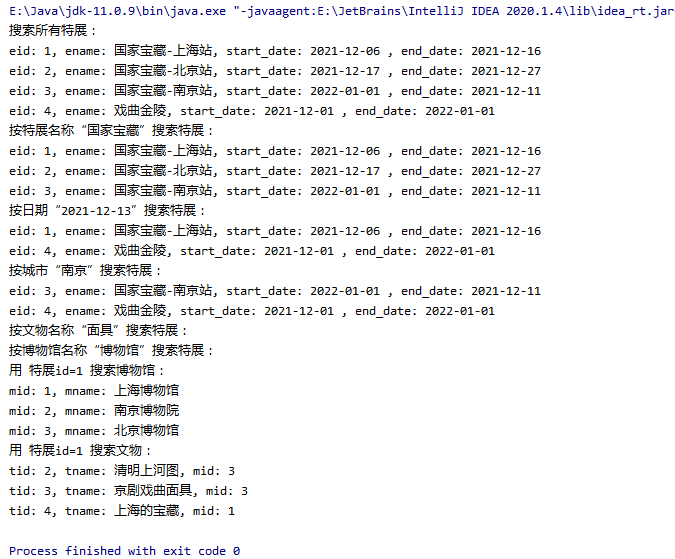
Showlist：



调用软件接口，分别进行以下操作

* 搜索所有特展；
* 按特展名称“国家宝藏”搜索特展；
* 按日期“2021-12-13”搜索特展；
* 按城市“南京”搜索特展；
* 按文物名称“面具”搜索特展；
* 按博物馆名称“博物馆”搜索特展；
* 用 特展id=1 搜索博物馆；
* 用 特展id=1 搜索文物。

实验结果如下所示：



运行结果正确，代码编写合理。

# 六、实验体会

在本次实验中，我们小组根据“一事一地”原则，仔细设计entity属性和关系，确保在没有数据冗余的情况下表格数目最少，data schema也达到了三范式。我们还初步接触了SQL与java的通信过程，非常感谢jdbc对java连接数据库的强大封装，使得我们通过几行代码即可实现两门语言间的交互。