**数据库系统实验报告**

# 公交安全管理系统

### 需求分析（系统数据和功能）

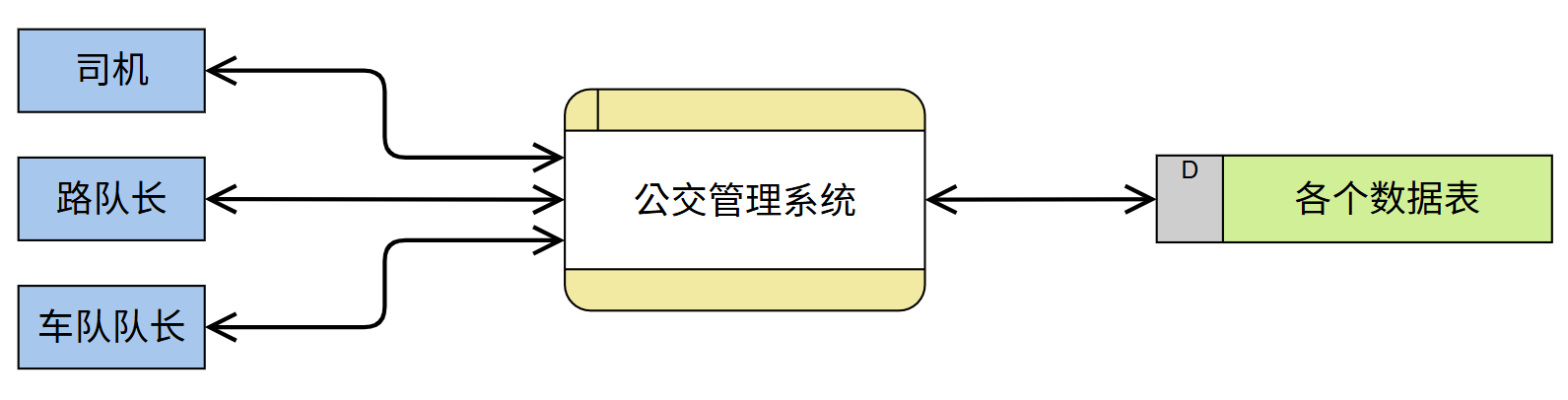
* 1. **数据库语义**

1. 公交公司有若干个车队，每个车队下有若干条线路；
2. 公交公司有若干辆汽车，每辆车属于一条线路；
3. 每个车队有一名队长，他只有管理工作，不开车；
4. 每条线路有若干名司机，其中有一名路队长，除开车外，还承 担管理工作；每名司机只在一条线路上开车；
5. 司机开车时会产生违章，包含：闯红灯、未礼让斑马线、压线、违章停车等；
6. 队长、路队长负责将司机的违章信息输入到系统，包含：司机、车辆、车队、线路、站点、时间、违章等。
   1. **数据库功能需求**
      1. **后台数据库**

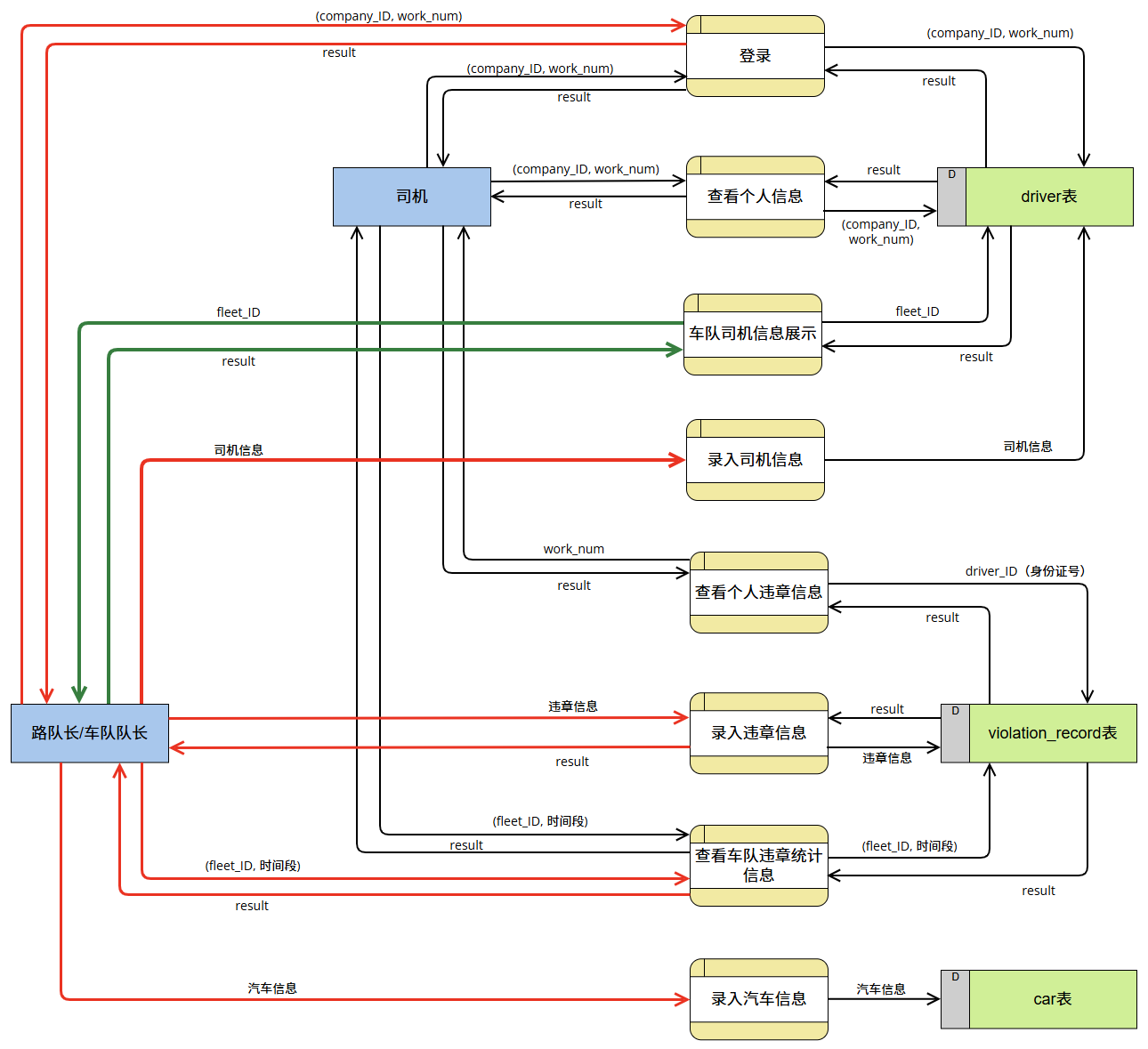
* 应用规范化理论进行数据库设计
* 设定关系完整性规则，包括实体完整性、参照完整性和用户自定义完整性。
* 使用索引优化查询速度。
* 利用视图、函数和存储过程简化系统设计。
* 可选用ERWin或PowerDesigner等工具辅助数据库设计。
  + 1. **前台程序**

前台程序应能实现以下功能：

* 录入司机基本信息（工号、姓名、性别等）。
* 录入汽车基本信息（车牌号、座数等）。
* 录入司机的违章信息。
* 查询某个车队下的所有司机基本信息。
* 查询某名司机在特定时间段内的违章详细信息。
* 查询某个车队在特定时间段内的违章统计信息。
  1. **数据流图**
     1. **顶层数据流图**



* + 1. **一级数据流图**



* 1. **数据字典**



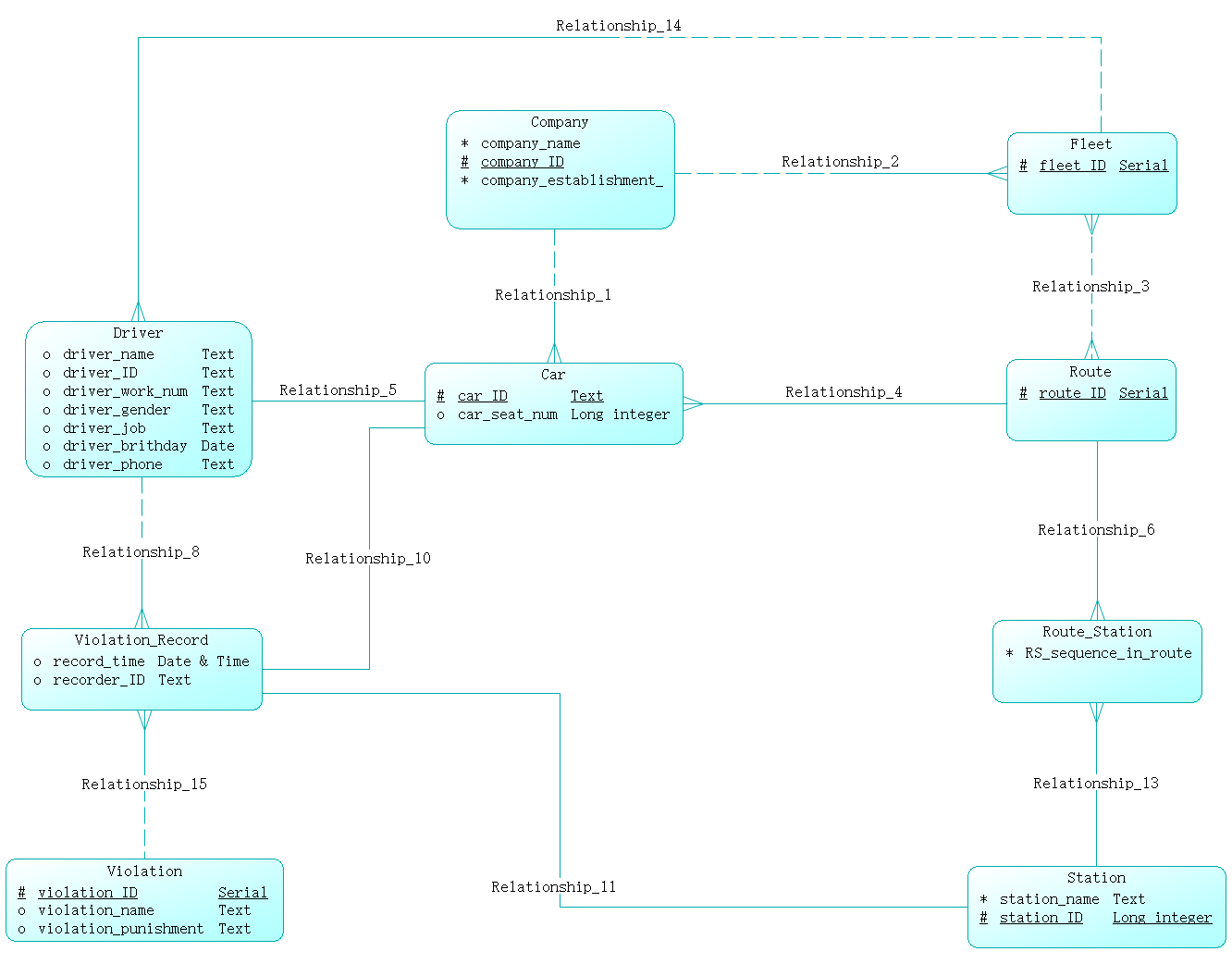




### 概念结构设计（E-R图设计）

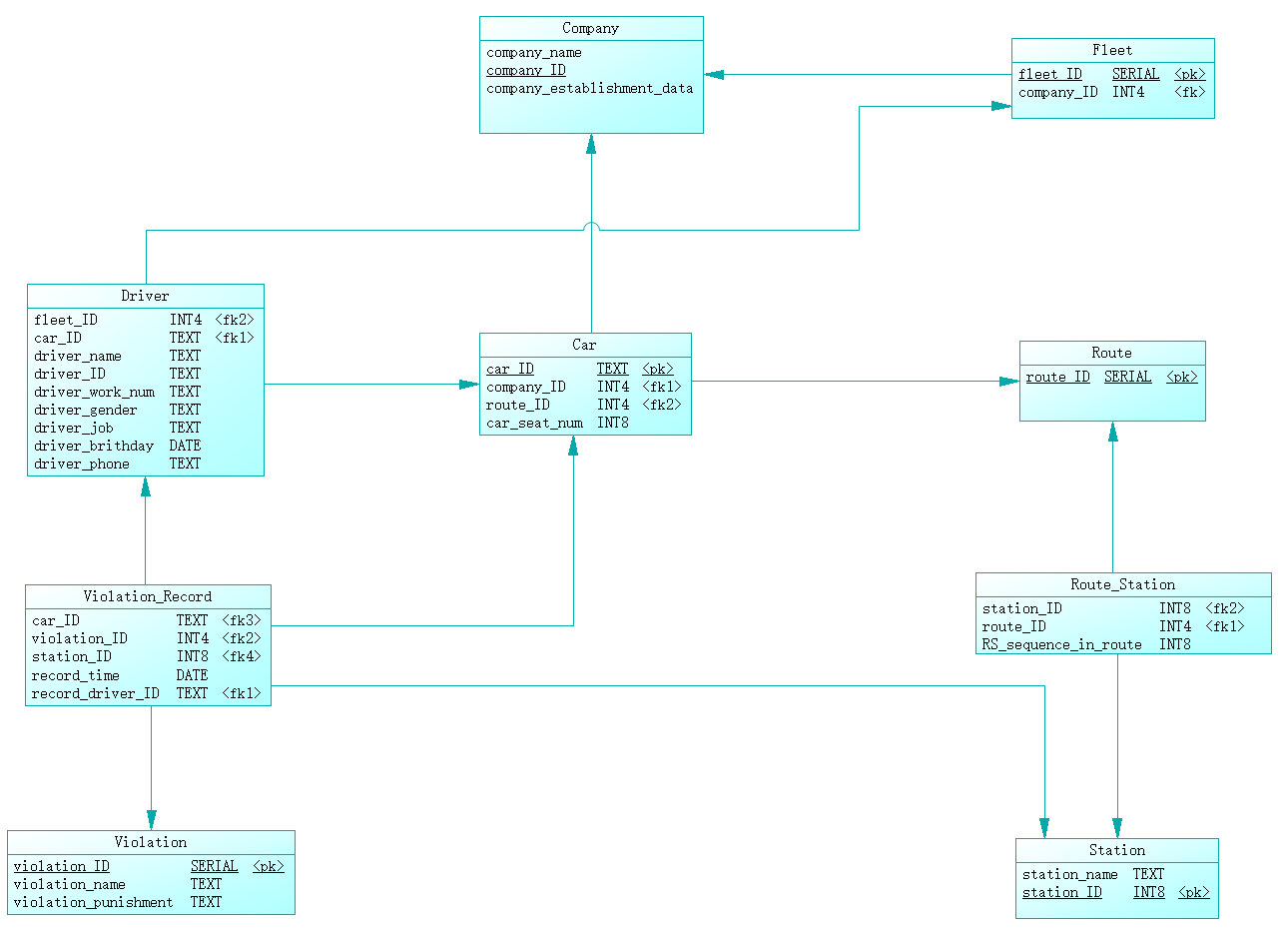
* 1. **E-R图设计**

使用Power Designer 设计E-R图如下：



* 1. **物理模型图**

通过Power Designer将设计的E-R图转换成如下物理模型图：



### 逻辑结构设计（E-R图转换为关系模型）

* 1. **关系模型**

关系模型给出表名，列名，列类型，主码，外码，其他约束，进行范式分析。



****





进行范式分析如下：

* 所有表都满足第一范式，因为每个字段都是原子值，且没有重复组。每个表中的每一列都只包含单一类型的值。
* 所有表也满足第二范式，因为它们已经消除了部分依赖。每个非主属性完全依赖于候选键，而不仅仅是其中的一部分。
* 大多数表都符合第三范式，即它们没有传递依赖。例如，在Driver表中，driver\_job、driver\_birthday等字段直接依赖于主键driver\_ID，而不是通过其他非主键字段间接依赖。
  1. **视图和函数**
     1. **视图**

1. **Driver\_View**

* **目的**：展示司机的基本信息及其关联的公司和车辆信息。
* **主要字段**：司机姓名、所属公司ID及名称、工作编号、职位、出生日期、电话、性别、车牌号。
* **连接表**：Driver、Company 和 Car。

1. **Driver\_Violation\_View**

* **目的**：展示司机的违章记录详情。
* **主要字段**：司机基本信息、违章记录ID、违章名称和惩罚、违章时间和地点等。
* **连接表**：Driver、Violation\_Record 和 Violation。
  + 1. **函数**

1. **get\_captain\_drivers**

* **输入**：公司ID (captain\_company\_id) 和路队长工号 (captain\_work\_num)。
* **输出**：该路队长管理下的所有普通司机的信息（姓名、工号、身份证号码、车牌号、联系方式）。
* **逻辑**：通过车队和司机表查找指定路队长下的普通司机。

1. **get\_fleet\_captain\_drivers**

* **输入**：公司ID (captain\_company\_id) 和车队队长工号 (captain\_work\_num)。
* **输出**：该车队队长管理下的所有司机（包括普通司机和路队长）的信息，并按线路ID排序。
* **逻辑**：通过车队和司机表查找指定车队队长下的所有司机，并排序。

### 程序开发环境及应用环境

* 1. **操作系统：Windows**
  2. **编程语言：SQL + Java**
  3. **开发工具：PostgreSQL + IDEA**

### 应用程序设计中遇到的问题及解决方法

* 1. **数据库设计中的问题。**

1. **循环引用导致的设计错误**

* **问题：**在数据库设计初期，我在Driver表中引入了fleet\_company\_ID字段以引用Fleet表中的company\_ID。然而，我又试图在Fleet表中添加一个名为fleet\_captain\_worknum的字段，直接引用Driver表中的driver\_work\_num字段，意图是指定每个车队的队长。这种设计最终导致了循环引用的问题。这种循环引用造成了以下问题：数据插入困难、更新约束冲突。
* **解决：**为了解决上述问题，我删除了Fleet表中的fleet\_captain\_worknum字段。

1. **冗余字段问题**

* **问题：**Driver表中存在driver\_fleet\_company\_ID和driver\_fleet\_ID两个字段同时存在，如果一个车队只能属于一个公司，那么driver\_fleet\_company\_ID是多余的，因为它可以通过driver\_fleet\_ID间接获得。
* **解决**：移除Driver表中的driver\_fleet\_company\_ID，以减少数据冗余并简化模式。

1. **索引优化不足**

* **问题：**设计中定义了许多外键关系，但并没有为这些外键创建索引。此外，对于频繁查询的字段（如driver\_name、car\_ID等），也缺乏索引优化措施。
* **解决：**添加必要索引，确保所有外键字段都有适当的索引，特别是那些用于频繁查询和连接操作的字段。
  1. **程序设计中的问题。**

1. **代码冗余**

* **问题：**在开发前端程序时，我为司机（Driver）、路队长（RouteCaptain）和车队队长（FleetCaptain）分别定义了独立的Java类。由于这三类角色在某些功能上具有相似性或完全相同的操作，例如获取个人信息、查看违章记录等，我在每个类中都实现了相同的函数。这种做法导致了大量的代码重复，增加了程序的维护成本和复杂度。
* **解决：**为了减少代码重复并提高程序的可维护性和扩展性，可以引入抽象基类或接口，将司机、路队长和车队队长共有的方法提取到基类中，并在具体角色类中继承该基类或实现接口。利用多态机制确保不同角色可以通过统一的方式调用相同的方法，具体实现由子类决定。

### 总结

通过《数据库系统》课程的学习和实验操作，我深入理解了数据库设计的基本原则，包括规范化理论、关系模型和完整性约束，并掌握了使用SQL语言进行数据库创建、表结构定义、索引建立、视图和函数开发等技能。在公交安全管理系统项目中，我经历了从需求分析到概念结构设计（E-R图）再到逻辑结构设计和应用程序开发的完整流程，积累了宝贵的实践经验。特别是在解决实际问题的过程中，如循环引用和代码冗余等问题，虽然尚未完全解决，但显著提升了我的问题分析和解决能力。

### 附录：建立数据库和应用程序的主要代码

* 1. **数据库设计代码**

1. /\* 创建公司表 \*/
2. **CREATE** **TABLE** Company (
3. company\_ID SERIAL, -- SERIAL自动递增整数类型
4. company\_name **VARCHAR**(255) NOT NULL,
5. company\_establishment\_date **DATE** NOT NULL, -- 公司成立日期
6. **PRIMARY** **KEY**(company\_ID)
7. );

10. /\* 创建车队表 \*/
11. **CREATE** **TABLE** Fleet (
12. fleet\_ID SERIAL, -- 车队ID
13. fleet\_company\_ID **INT** NOT NULL, -- 公司ID
14. **PRIMARY** **KEY** (fleet\_ID)
15. );

18. /\* 创建汽车表 \*/
19. **CREATE** **TABLE** Car (
20. car\_ID **VARCHAR**(20) NOT NULL, -- 车牌号
21. car\_company\_ID **INT**, -- 公司ID
22. car\_seat\_num **INT**, -- 座位数量
23. **PRIMARY** **KEY** (car\_ID) -- 主键
24. );
26. /\* 创建司机表 \*/
27. **CREATE** **TABLE** Driver (
28. driver\_name **VARCHAR**(255) NOT NULL, -- 司机姓名
29. driver\_fleet\_company\_ID **INT** NOT NULL, -- 司机所属车队所属公司ID
30. driver\_ID **CHAR**(18) NOT NULL **UNIQUE**, -- 身份证号码，作为唯一标识符
31. driver\_work\_num **VARCHAR**(20) NOT NULL, -- 工作编号常嘉乐龙益辰田涵越孙榕润江国栋
32. driver\_job **VARCHAR**(255) NOT NULL, -- 职位（字符串）
33. driver\_birthday **DATE**, -- 出生日期
34. driver\_phone **VARCHAR**(15), -- 电话号码
35. driver\_gender **CHAR**(1), -- 性别，例如'M'表示男性，'F'表示女性
36. driver\_fleet\_ID **INT** NOT NULL, -- 所属车队ID
37. driver\_car\_ID **VARCHAR**(20), -- 所属车辆ID
38. driver\_route\_ID **INT**,
39. **PRIMARY** **KEY** (driver\_fleet\_company\_ID, driver\_work\_num) -- 组合主键
40. );

43. /\* 创建线路表 \*/
44. **CREATE** **TABLE** Route (
45. route\_ID SERIAL, -- 线路ID
46. route\_beginning\_station\_ID **INT** NOT NULL, -- 起始站ID
47. route\_ending\_station\_ID **INT** NOT NULL, -- 终点站ID
48. **PRIMARY** **KEY** (route\_ID) -- 主键
49. );
51. /\* 创建站点表 \*/
52. **CREATE** **TABLE** Station (
53. station\_ID SERIAL, -- 站点ID
54. station\_name **VARCHAR**(255) NOT NULL, -- 站点名称
55. **PRIMARY** **KEY** (station\_ID) -- 主键
56. );

59. /\* 创建条规表 \*/
60. **CREATE** **TABLE** Violation (
61. violation\_ID SERIAL, -- 使用SERIAL以实现自动递增的违规ID
62. violation\_name **VARCHAR**(255) NOT NULL, -- 违规名称
63. violation\_punishment TEXT, -- 违规惩罚
64. **PRIMARY** **KEY** (violation\_ID) -- 主键
65. );

68. /\* 创建违规记录表 \*/
69. **CREATE** **TABLE** Violation\_Record (
70. record\_ID SERIAL, -- 违规记录ID，自动递增
71. record\_violation\_ID **INT** NOT NULL, -- 违规类型ID
72. record\_station\_ID **INT** NOT NULL, -- 发生违规的站点ID
73. record\_driver\_ID **CHAR**(18) NOT NULL, -- 司机身份证号码
74. record\_car\_ID **VARCHAR**(20) NOT NULL, -- 车牌
75. record\_time **TIMESTAMP** NOT NULL, -- 违规时间（年月日时分秒）
76. record\_recorder\_ID **CHAR**(18) NOT NULL, -- 记录人身份证
77. **PRIMARY** **KEY** (record\_ID) -- 显式定义主键
78. );
80. /\* 创建车站-线路关系表 \*/
81. **CREATE** **TABLE** Route\_Station (
82. RS\_station\_ID **INT** NOT NULL, -- 站点ID
83. RS\_route\_ID **INT** NOT NULL, -- 路线ID
84. RS\_sequence\_in\_route **INT** NOT NULL, -- 该站点在线路中的顺序
85. **PRIMARY** **KEY** (RS\_route\_ID, RS\_station\_ID) -- 组合主键
86. );


90. /\* 引入外码 \*/

93. -- 为Fleet表添加外键
94. **ALTER** **TABLE** Fleet
95. **ADD** **CONSTRAINT** fk\_fleet\_company\_ID
96. **FOREIGN** **KEY** (fleet\_company\_ID) **REFERENCES** Company(company\_ID);
98. -- 为Car表添加外键
99. **ALTER** **TABLE** Car
100. **ADD** **CONSTRAINT** fk\_car\_company
101. **FOREIGN** **KEY** (car\_company\_ID) **REFERENCES** Company(company\_ID);
103. -- 为Driver表添加外键
104. **ALTER** **TABLE** Driver
105. **ADD** **CONSTRAINT** fk\_driver\_fleet
106. **FOREIGN** **KEY** (driver\_fleet\_company\_ID, driver\_fleet\_ID) **REFERENCES** Fleet(fleet\_company\_ID, fleet\_ID),
107. **ADD** **CONSTRAINT** fk\_driver\_car
108. **FOREIGN** **KEY** (driver\_car\_ID) **REFERENCES** Car(car\_ID),
109. **ADD** **CONSTRAINT** fk\_driver\_route
110. **FOREIGN** **KEY** (driver\_route\_ID) **REFERENCES** Route(route\_ID);
112. -- 为Route表添加外键
113. **ALTER** **TABLE** Route
114. **ADD** **CONSTRAINT** fk\_route\_beginning\_station
115. **FOREIGN** **KEY** (route\_beginning\_station\_ID) **REFERENCES** Station(station\_ID),
116. **ADD** **CONSTRAINT** fk\_route\_ending\_station
117. **FOREIGN** **KEY** (route\_ending\_station\_ID) **REFERENCES** Station(station\_ID);
119. -- 为Violation\_Record表添加外键
120. **ALTER** **TABLE** Violation\_Record
121. **ADD** **CONSTRAINT** fk\_violation\_record\_violation
122. **FOREIGN** **KEY** (record\_violation\_ID) **REFERENCES** Violation(violation\_ID),
123. **ADD** **CONSTRAINT** fk\_violation\_record\_station
124. **FOREIGN** **KEY** (record\_station\_ID) **REFERENCES** Station(station\_ID),
125. **ADD** **CONSTRAINT** fk\_violation\_record\_driver
126. **FOREIGN** **KEY** (record\_driver\_ID) **REFERENCES** Driver(driver\_ID),
127. **ADD** **CONSTRAINT** fk\_violation\_record\_car
128. **FOREIGN** **KEY** (record\_car\_ID) **REFERENCES** Car(car\_ID),
129. **ADD** **CONSTRAINT** fk\_violation\_record\_recorder
130. **FOREIGN** **KEY** (record\_recorder\_ID) **REFERENCES** Driver(driver\_ID); -- 记录人也是司机
132. -- 为Route\_Station表添加外键
133. **ALTER** **TABLE** Route\_Station
134. **ADD** **CONSTRAINT** fk\_route\_station\_station
135. **FOREIGN** **KEY** (RS\_station\_ID) **REFERENCES** Station(station\_ID),
136. **ADD** **CONSTRAINT** fk\_route\_station\_route
137. **FOREIGN** **KEY** (RS\_route\_ID) **REFERENCES** Route(route\_ID);

140. /\* 创建约束条件 \*/
142. -- driver的性别字段、出生年龄、号码、身份证号、职位
143. **ALTER** **TABLE** Driver
144. **ADD** **CONSTRAINT** chk\_driver\_gender
145. **CHECK** (driver\_gender IN ('M', 'F')),
146. **ADD** **CONSTRAINT** chk\_driver\_birthday
147. **CHECK** (driver\_birthday < **CURRENT\_DATE** AND driver\_birthday > **CURRENT\_DATE** - INTERVAL '150 years'),
148. **ADD** **CONSTRAINT** chk\_driver\_phone
149. **CHECK** (driver\_phone ~ '^[1][0-9]{10}$'),
150. **ADD** **CONSTRAINT** chk\_driver\_id
151. **CHECK** (driver\_ID ~ '^[0-9]{17}[0-9Xx]$'),
152. **ADD** **CONSTRAINT** chk\_driver\_job
153. **CHECK** (driver\_job IN ('司机', '路队长', '车队队长')),
154. **ADD** **CONSTRAINT** chk\_driver\_job\_and\_associated\_fields
155. **CHECK** (
156. (driver\_job = '车队队长' AND driver\_car\_ID **IS** NULL AND driver\_route\_ID **IS** NULL)
157. OR
158. (driver\_job IN ('司机', '路队长') AND driver\_car\_ID **IS** NOT NULL AND driver\_route\_ID **IS** NOT NULL)
159. );

162. -- Car的座位数，车牌号
163. **ALTER** **TABLE** Car
164. **ADD** **CONSTRAINT** chk\_car\_seat\_num
165. **CHECK** (car\_seat\_num > 0),
166. **ADD** **CONSTRAINT** chk\_car\_id\_format
167. **CHECK** (car\_ID ~ '^[京津沪渝冀豫云辽黑湘皖鲁新苏浙赣鄂桂甘晋蒙陕吉闽贵青藏川宁琼港澳]([A-HJ-NP-Z]{1}[A-Z0-9]{5}|[A-HJ-NP-Z]{2}[A-Z0-9]{4})$');
169. -- 公司成立时间
170. **ALTER** **TABLE** Company
171. **ADD** **CONSTRAINT** chk\_company\_establishment\_date
172. **CHECK** (company\_establishment\_date < **CURRENT\_DATE** AND company\_establishment\_date > **CURRENT\_DATE** - INTERVAL '200 years');
174. -- 违规记录的时间
175. **ALTER** **TABLE** Violation\_Record
176. **ADD** **CONSTRAINT** chk\_record\_time
177. **CHECK** (record\_time <= CURRENT\_TIMESTAMP);


181. /\* 路队长 \*/
182. **CREATE** OR REPLACE **FUNCTION** get\_captain\_drivers(captain\_company\_id **INT**, captain\_work\_num **VARCHAR**)
183. **RETURNS** **TABLE** (
184. driver\_name **VARCHAR**(255),
185. driver\_work\_num **VARCHAR**(20),
186. driver\_ID **CHAR**(18),
187. car\_ID **VARCHAR**(20),
188. driver\_phone **VARCHAR**(15)
189. ) **AS** $$
190. **BEGIN**
191. **RETURN** QUERY
192. **WITH** CaptainInfo **AS** (
193. **SELECT**
194. f.fleet\_company\_ID,
195. f.fleet\_ID,
196. d.driver\_route\_ID
197. **FROM**
198. Fleet f
199. JOIN
200. Driver d **ON** (f.fleet\_company\_ID, f.fleet\_ID) = (d.driver\_fleet\_company\_ID, d.driver\_fleet\_ID)
201. **WHERE**
202. d.driver\_job = '路队长'
203. AND d.driver\_fleet\_company\_ID = captain\_company\_id
204. AND d.driver\_work\_num = captain\_work\_num
205. )
206. **SELECT**
207. d.driver\_name, -- 司机姓名
208. d.driver\_work\_num, -- 工号
209. d.driver\_ID,
210. d.driver\_car\_ID, -- 车牌号
211. d.driver\_phone -- 联系方式
212. **FROM**
213. Driver d
214. JOIN
215. CaptainInfo ci **ON** d.driver\_fleet\_company\_ID = ci.fleet\_company\_ID AND d.driver\_fleet\_ID = ci.fleet\_ID AND d.driver\_route\_ID = ci.driver\_route\_ID
216. **WHERE**
217. d.driver\_job = '司机'; -- 仅选择职位为“司机”的记录
218. **END**;
219. $$ LANGUAGE plpgsql;

222. /\* 车队队长 \*/
223. **CREATE** OR REPLACE **FUNCTION** get\_fleet\_captain\_drivers(captain\_company\_id **INT**, captain\_work\_num **VARCHAR**)
224. **RETURNS** **TABLE** (
225. driver\_route\_ID **INT**,
226. driver\_job **VARCHAR**(255),
227. driver\_name **VARCHAR**(255),
228. driver\_work\_num **VARCHAR**(20),
229. car\_ID **VARCHAR**(20),
230. driver\_phone **VARCHAR**(15)
231. ) **AS** $$
232. **BEGIN**
233. **RETURN** QUERY
234. **WITH** CaptainInfo **AS** (
235. **SELECT**
236. f.fleet\_company\_ID,
237. f.fleet\_ID
238. **FROM**
239. Fleet f
240. JOIN
241. Driver d **ON** (f.fleet\_company\_ID, f.fleet\_ID) = (d.driver\_fleet\_company\_ID, d.driver\_fleet\_ID)
242. **WHERE**
243. d.driver\_job = '车队队长'
244. AND d.driver\_fleet\_company\_ID = captain\_company\_id
245. AND d.driver\_work\_num = captain\_work\_num
246. )
247. **SELECT**
248. d.driver\_route\_ID,
249. d.driver\_job,
250. d.driver\_name, -- 司机姓名
251. d.driver\_work\_num, -- 工号
252. d.driver\_car\_ID, -- 车牌号
253. d.driver\_phone -- 联系方式
254. **FROM**
255. Driver d
256. JOIN
257. CaptainInfo ci **ON** d.driver\_fleet\_company\_ID = ci.fleet\_company\_ID AND d.driver\_fleet\_ID = ci.fleet\_ID
258. **where**
259. d.driver\_job = '司机' OR d.driver\_job = '路队长'
260. **ORDER** **BY**
261. d.driver\_route\_ID **ASC**; -- 按照线路ID递增排序
262. **END**;
263. $$ LANGUAGE plpgsql;

266. /\* 司机视图 \*/
267. **CREATE** OR REPLACE **VIEW** Driver\_View **AS**
268. **SELECT**
269. d.driver\_name,
270. d.driver\_fleet\_company\_ID **AS** company\_ID,
271. c.company\_name,
272. d.driver\_work\_num,
273. d.driver\_ID,
274. d.driver\_job,
275. d.driver\_birthday,
276. d.driver\_phone,
277. d.driver\_route\_ID,
278. d.driver\_fleet\_ID,
279. d.driver\_gender,
280. c2.car\_ID
281. **FROM**
282. Driver d
283. JOIN
284. Company c **ON** d.driver\_fleet\_company\_ID = c.company\_ID
285. LEFT JOIN
286. Car c2 **ON** d.driver\_car\_ID = c2.car\_ID;

289. /\* 违章记录视图 \*/
290. **CREATE** OR REPLACE **VIEW** Driver\_Violation\_View **AS**
291. **SELECT**
292. d.driver\_name,
293. d.driver\_fleet\_company\_ID **AS** company\_ID,
294. d.driver\_work\_num,
295. d.driver\_job,
296. d.driver\_phone,
297. d.driver\_gender,
298. d.driver\_fleet\_ID,
299. d.driver\_route\_ID,
300. v.record\_ID,
301. v2.violation\_name,
302. v2.violation\_punishment,
303. v.record\_station\_ID,
304. v.record\_car\_ID,
305. v.record\_time,
306. v.record\_recorder\_ID
307. **FROM**
308. Driver d
309. JOIN
310. Violation\_Record v **ON** d.driver\_ID = v.record\_driver\_ID
311. JOIN
312. Violation v2 **ON** v.record\_violation\_ID = v2.violation\_ID;
     1. **应用程序主要代码（**FleetCaptainView.java**）**
313. **import** java.sql.Connection;
314. **import** java.sql.PreparedStatement;
315. **import** java.sql.ResultSet;
316. **import** java.sql.SQLException;
317. **import** java.time.LocalDateTime;
318. **import** java.sql.Date;
319. **import** java.sql.Timestamp;
320. **import** java.util.ArrayList;
321. **import** java.util.List;
323. **public** **class** FleetCaptainView **implements** AutoCloseable{
324. **private** **final** Connection connection;
325. **private** **int** companyID;
326. **private** String workNum;
328. // 构造函数接收一个已有的数据库连接，并进行登录验证
329. **public** FleetCaptainView(**int** companyID, String workNum) **throws** SQLException, DriverView.LoginFailedException {
330. **this**.connection = DatabaseConnection.connect();
331. **this**.companyID = companyID;
332. **this**.workNum = workNum;
334. **if** (!login()) {
335. **throw** **new** DriverView.LoginFailedException("登录失败: 未找到该车队队长的信息");
336. }
337. }
339. // 执行登录验证
340. **private** **boolean** login() **throws** SQLException {
341. **final** String QUERY\_VERIFY\_DRIVER = "SELECT 1 FROM Driver\_View WHERE company\_ID = ? AND driver\_work\_num = ? AND driver\_job = '车队队长' FETCH FIRST 1 ROW ONLY";

344. **try** (PreparedStatement pstmt = connection.prepareStatement(QUERY\_VERIFY\_DRIVER)) {
345. pstmt.setInt(1, companyID);
346. pstmt.setString(2, workNum);
348. **try** (ResultSet rs = pstmt.executeQuery()) {
349. **return** rs.next(); // 如果查询结果存在，则表示登录成功
350. }
351. }
352. }
354. // 查询车队队长的基本信息
355. **public** **void** queryRouteCaptainInfo(**int** companyID, String workNum) **throws** SQLException {
356. **final** String QUERY\_DRIVER\_INFO =
357. "SELECT driver\_name, company\_name, driver\_work\_num, driver\_ID, driver\_job, driver\_birthday, driver\_phone, driver\_fleet\_ID, driver\_gender " +
358. "FROM Driver\_View WHERE company\_ID = ? AND driver\_work\_num = ?";
360. **try** (PreparedStatement pstmt = connection.prepareStatement(QUERY\_DRIVER\_INFO)) {
361. pstmt.setInt(1, companyID);
362. pstmt.setString(2, workNum);
364. **try** (ResultSet rs = pstmt.executeQuery()) {
365. **if** (rs.next()) {
366. System.out.println("司机基本信息:");
367. System.out.println("姓名: " + rs.getString("driver\_name"));
368. System.out.println("公司名称: " + rs.getString("company\_name"));
369. System.out.println("工作编号: " + rs.getString("driver\_work\_num"));
370. System.out.println("身份证号码: " + rs.getString("driver\_ID"));
371. System.out.println("职位: " + rs.getString("driver\_job"));
372. System.out.println("出生日期: " + rs.getDate("driver\_birthday"));
373. System.out.println("电话号码: " + rs.getString("driver\_phone"));
374. System.out.println("车队编号: " + rs.getInt("driver\_fleet\_ID"));
375. System.out.println("性别: " + rs.getString("driver\_gender"));
376. } **else** {
377. System.out.println("未找到该车队队长的信息");
378. }
379. }
380. }
381. }
383. // 展示由自己管理的司机的信息
384. **public** **void** displayManagedDriversInTable() {
385. String functionCall = "SELECT \* FROM get\_fleet\_captain\_drivers(?, ?)";
387. **try** (PreparedStatement stmt = connection.prepareStatement(functionCall)) {
388. stmt.setInt(1, **this**.companyID);
389. stmt.setString(2, **this**.workNum);
391. **try** (ResultSet rs = stmt.executeQuery()) {
392. printTableHeader(); // 打印表头
393. **boolean** hasResults = **false**;
394. **while** (rs.next()) {
395. hasResults = **true**;
396. printTableRow(
397. rs.getInt("driver\_route\_ID"),
398. rs.getString("driver\_job"),
399. rs.getString("driver\_name"),
400. rs.getString("driver\_work\_num"),
401. rs.getString("car\_ID"),
402. rs.getString("driver\_phone")
403. );
404. }
405. **if** (!hasResults) {
406. System.out.println("没有找到相关记录。");
407. }
408. }
409. } **catch** (SQLException e) {
410. e.printStackTrace();
411. }
412. }
414. **private** **void** printTableHeader() {
415. System.out.printf("%-15s | %-15s | %-20s | %-20s | %-20s | %-20s%n",
416. "线路ID", "职位", "司机姓名", "工号", "车牌号", "联系方式");
417. printSeparatorLine();
418. }
420. **private** **void** printSeparatorLine() {
421. System.out.println("--------------------------------------------------------------" +
422. "-------------------------------------------------------------");
423. }
425. **private** **void** printTableRow(**int** driverRouteID, String driverJob, String driverName, String driverWorkNum, String carID, String driverPhone) {
426. System.out.printf("%-15d | %-15s | %-20s | %-20s | %-20s | %-20s%n",
427. driverRouteID, driverJob, driverName, driverWorkNum, carID, driverPhone);
428. }
430. // 查询自己在某个时间段的违章详细信息
431. **public** List<DriverView.ViolationDetail> queryViolationDetails(**int** companyID, String workNum, LocalDateTime startTime, LocalDateTime endTime) **throws** SQLException {
432. **final** String QUERY\_VIOLATION\_DETAILS =
433. "SELECT dv.driver\_name, dv.violation\_name, dv.violation\_punishment, dv.record\_time, dv.record\_station\_ID, dv.record\_car\_ID " +
434. "FROM Driver\_Violation\_View dv " +
435. "WHERE dv.company\_ID = ? AND dv.driver\_work\_num = ? AND dv.record\_time BETWEEN ? AND ?";
437. List<DriverView.ViolationDetail> violations = **new** ArrayList<>();
439. **try** (PreparedStatement pstmt = connection.prepareStatement(QUERY\_VIOLATION\_DETAILS)) {
440. pstmt.setInt(1, companyID);
441. pstmt.setString(2, workNum);
442. pstmt.setTimestamp(3, java.sql.Timestamp.valueOf(startTime));
443. pstmt.setTimestamp(4, java.sql.Timestamp.valueOf(endTime));
445. **try** (ResultSet rs = pstmt.executeQuery()) {
446. **while** (rs.next()) {
447. violations.add(**new** DriverView.ViolationDetail(
448. rs.getString("driver\_name"),
449. rs.getString("violation\_name"),
450. rs.getString("violation\_punishment"),
451. rs.getTimestamp("record\_time").toLocalDateTime(),
452. rs.getInt("record\_station\_ID"),
453. rs.getString("record\_car\_ID")
454. ));
455. }
456. }
457. }
459. **if** (violations.isEmpty()) {
460. System.out.println("该司机在此时间段内无违章记录");
461. }
463. **return** violations;
464. }

467. // 判断给定的司机是否由当前车队队长管理
468. **public** **boolean** isDriverManagedByRouteCaptain(**int** companyID, String workNum) {
469. String sql = "SELECT driver\_fleet\_ID FROM Driver "
470. + "WHERE driver\_fleet\_company\_ID = ? AND driver\_work\_num = ? ";
472. **int** route\_captain\_fleet\_ID = 0;
474. **int** driver\_fleet\_ID = 0;
476. // SQL查询语句：检查路队长的车队ID、线路ID
477. **try** (PreparedStatement stmt = connection.prepareStatement(sql)) {
478. stmt.setInt(1, **this**.companyID);
479. stmt.setString(2, **this**.workNum);
481. **try** (ResultSet rs = stmt.executeQuery()) {
482. **if** (rs.next()) {
483. route\_captain\_fleet\_ID = rs.getInt("driver\_fleet\_ID");
484. }
485. }
486. } **catch** (SQLException e) {
487. System.err.println("查询失败: " + e.getMessage());
488. }
490. // SQL查询语句：检查司机的车队ID、线路ID
491. **try** (PreparedStatement stmt = connection.prepareStatement(sql)) {
492. stmt.setInt(1, companyID);
493. stmt.setString(2, workNum);
495. **try** (ResultSet rs = stmt.executeQuery()) {
496. **if** (rs.next()) {
497. driver\_fleet\_ID = rs.getInt("driver\_fleet\_ID");
498. }
499. }
500. } **catch** (SQLException e) {
501. System.err.println("查询失败: " + e.getMessage());
502. }
504. **if**(companyID == **this**.companyID && driver\_fleet\_ID == route\_captain\_fleet\_ID)
505. {
506. **return** **true**;
507. }
508. **else**{
509. **return** **false**;
510. }
511. }
513. // 判断给定的司机是否由当前车队队长管理
514. **public** **boolean** isDriverManagedByRouteCaptain(String driverID) {
515. String sql = "SELECT driver\_fleet\_ID FROM Driver "
516. + "WHERE driver\_fleet\_company\_ID = ? AND driver\_work\_num = ? ";
518. **int** route\_captain\_fleet\_ID = 0;
520. **int** driver\_company\_ID = 0;
521. **int** driver\_fleet\_ID = 0;
523. // SQL查询语句：检查路队长的车队ID、线路ID
524. **try** (PreparedStatement stmt = connection.prepareStatement(sql)) {
525. stmt.setInt(1, **this**.companyID);
526. stmt.setString(2, **this**.workNum);
528. **try** (ResultSet rs = stmt.executeQuery()) {
529. **if** (rs.next()) {
530. route\_captain\_fleet\_ID = rs.getInt("driver\_fleet\_ID");
531. }
532. }
533. } **catch** (SQLException e) {
534. System.err.println("查询失败: " + e.getMessage());
535. }
537. sql = "SELECT driver\_fleet\_company\_ID, driver\_fleet\_ID FROM Driver "
538. + "WHERE driver\_ID = ?";
540. // SQL查询语句：检查司机的车队ID、线路ID
541. **try** (PreparedStatement stmt = connection.prepareStatement(sql)) {
542. stmt.setString(1, driverID);
544. **try** (ResultSet rs = stmt.executeQuery()) {
545. **if** (rs.next()) {
546. driver\_company\_ID = rs.getInt("driver\_fleet\_company\_ID");
547. driver\_fleet\_ID = rs.getInt("driver\_fleet\_ID");
548. }
549. }
550. } **catch** (SQLException e) {
551. System.err.println("查询失败: " + e.getMessage());
552. }
554. **if**(driver\_company\_ID == **this**.companyID && driver\_fleet\_ID == route\_captain\_fleet\_ID)
555. {
556. **return** **true**;
557. }
558. **else**{
559. **return** **false**;
560. }
561. }
563. // 查询某司机所属车队在某个时间段的违章统计信息，并直接打印结果
564. **public** **void** queryFleetViolationStatsForRouteCaptain(LocalDateTime startTime, LocalDateTime endTime) {
565. // 第一步：根据(company\_ID, workNum)查询司机所属的fleet\_ID
566. **int** fleetID = -1;
567. **try** {
568. fleetID = getDriverFleetID();
570. **if** (fleetID == -1) {
571. System.out.println("未找到该司机的信息或司机不属于任何车队");
572. **return**;
573. }
575. // 第二步：查询该车队在指定时间段的违章统计信息并打印
576. printFleetViolationStats(fleetID, startTime, endTime);
577. } **catch** (SQLException e) {
578. System.err.println("查询过程中发生错误: " + e.getMessage());
579. }
580. }
582. // 获取司机所属的fleet\_ID
583. **private** **int** getDriverFleetID() **throws** SQLException {
584. **final** String QUERY\_DRIVER\_FLEET\_ID =
585. "SELECT driver\_fleet\_ID " +
586. "FROM Driver " +
587. "WHERE driver\_fleet\_company\_ID = ? AND driver\_work\_num = ? " +
588. "LIMIT 1";
590. **try** (PreparedStatement pstmt = connection.prepareStatement(QUERY\_DRIVER\_FLEET\_ID)) {
591. pstmt.setInt(1, **this**.companyID);
592. pstmt.setString(2, **this**.workNum);
594. **try** (ResultSet rs = pstmt.executeQuery()) {
595. **if** (rs.next()) {
596. **return** rs.getInt("driver\_fleet\_ID");
597. } **else** {
598. **return** -1; // 表示未找到对应的车队ID
599. }
600. }
601. }
602. }
604. // 打印某车队在某个时间段的违章统计信息
605. **private** **void** printFleetViolationStats(**int** fleetID, LocalDateTime startTime, LocalDateTime endTime) **throws** SQLException {
606. **final** String QUERY\_FLEET\_VIOLATION\_STATS =
607. "SELECT dv.violation\_name, COUNT(\*) AS violation\_count " +
608. "FROM Driver\_Violation\_View dv " +
609. "WHERE dv.driver\_fleet\_ID = ? AND dv.record\_time BETWEEN ? AND ? " +
610. "GROUP BY dv.violation\_name";
612. StringBuilder summary = **new** StringBuilder("车队(" + fleetID + ")在指定时间段内");
613. **boolean** hasViolations = **false**;
614. **int** totalViolations = 0;
616. **try** (PreparedStatement pstmt = connection.prepareStatement(QUERY\_FLEET\_VIOLATION\_STATS)) {
617. pstmt.setInt(1, fleetID);
618. pstmt.setTimestamp(2, java.sql.Timestamp.valueOf(startTime));
619. pstmt.setTimestamp(3, java.sql.Timestamp.valueOf(endTime));
621. **try** (ResultSet rs = pstmt.executeQuery()) {
622. **while** (rs.next()) {
623. hasViolations = **true**;
624. String violationName = rs.getString("violation\_name");
625. **int** count = rs.getInt("violation\_count");
626. totalViolations += count;
627. summary.append(", ").append(violationName).append(": ").append(count).append("次");
628. }
629. **if** (!hasViolations) {
630. summary.append("无违章记录");
631. } **else** {
632. summary.insert(summary.indexOf("内") + 2, "共有" + totalViolations + "次违章");
633. }
634. System.out.println(summary.toString().replaceFirst(", ", " "));
635. }
636. }
637. }
639. // 录入汽车信息
640. **public** **boolean** addCar(String carID, **int** seatNum) {
641. String sql = "INSERT INTO Car (car\_ID, car\_company\_ID, car\_seat\_num) VALUES (?, ?, ?)";
643. **try** (PreparedStatement stmt = connection.prepareStatement(sql)) {
644. stmt.setString(1, carID);
645. stmt.setInt(2, companyID); // 关联到当前路队长的公司
646. stmt.setInt(3, seatNum);
648. **return** stmt.executeUpdate() > 0;
649. } **catch** (SQLException e) {
650. System.err.println("录入汽车信息失败: " + e.getMessage());
651. }
652. **return** **false**;
653. }
655. // 录入司机信息
656. **public** **boolean** addDriver(String workNum, String name, **char** gender, String id, String phone, Date birthday, String carID, **int** route\_ID, String job) {
657. String sql1 = "SELECT driver\_fleet\_ID FROM Driver "
658. + "WHERE driver\_fleet\_company\_ID = ? AND driver\_work\_num = ? ";
660. **int** route\_captain\_fleet\_ID = 0;
662. // SQL查询语句：检查路队长的车队ID、线路ID
663. **try** (PreparedStatement stmt = connection.prepareStatement(sql1)) {
664. stmt.setInt(1, **this**.companyID);
665. stmt.setString(2, **this**.workNum);
667. **try** (ResultSet rs = stmt.executeQuery()) {
668. **if** (rs.next()) {
669. route\_captain\_fleet\_ID = rs.getInt("driver\_fleet\_ID");
670. }
671. }
672. } **catch** (SQLException e) {
673. System.err.println("查询失败: " + e.getMessage());
674. }
676. **if**(job.equals("路队长"))
677. {
678. String sql2 = "SELECT 1 FROM Driver "
679. + "WHERE driver\_fleet\_company\_ID = ? AND driver\_route\_ID = ? AND driver\_job = '路队长'";
681. **try** (PreparedStatement stmt = connection.prepareStatement(sql2)) {
682. stmt.setInt(1, **this**.companyID);
683. stmt.setInt(2, route\_ID);
685. **try** (ResultSet rs = stmt.executeQuery()) {
686. **if** (rs.next()) {
687. System.out.printf("该线路已经有路队长，不能重复设置!!!\n");
688. **return** **false**;
689. }
690. }
691. } **catch** (SQLException e) {
692. System.err.println("查询失败: " + e.getMessage());
693. }
694. }

697. String sql3 = "INSERT INTO Driver (driver\_name, driver\_fleet\_company\_ID, driver\_fleet\_ID, driver\_ID, driver\_work\_num, driver\_job, driver\_birthday, driver\_phone, driver\_gender, driver\_route\_ID, driver\_car\_ID) "
698. + "VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
700. **try** (PreparedStatement stmt = connection.prepareStatement(sql3)) {
701. stmt.setString(1, name);
702. stmt.setInt(2, companyID); // 关联到当前车队队长的公司
703. stmt.setInt(3, route\_captain\_fleet\_ID);
704. stmt.setString(4, id);
705. stmt.setString(5, workNum);
706. stmt.setString(6, job);
707. stmt.setDate(7, birthday);
708. stmt.setString(8, phone);
709. stmt.setString(9, Character.toString(gender));
710. stmt.setInt(10, route\_ID);
711. stmt.setString(11, carID);
713. **return** stmt.executeUpdate() > 0;
714. } **catch** (SQLException e) {
715. System.err.println("录入司机信息失败: " + e.getMessage());
716. }
717. **return** **false**;
718. }
720. // 录入违章信息
721. **public** **boolean** recordViolation(**int** violationID, **int** stationID, String driverID, String carID, LocalDateTime violationTime, String recorderID) {
722. String sql = "INSERT INTO Violation\_Record (record\_violation\_ID, record\_station\_ID, record\_driver\_ID, record\_car\_ID, record\_time, record\_recorder\_ID) "
723. + "VALUES (?, ?, ?, ?, ?, ?)";
725. **try** (PreparedStatement stmt = connection.prepareStatement(sql)) {
726. stmt.setInt(1, violationID);
727. stmt.setInt(2, stationID);
728. stmt.setString(3, driverID);
729. stmt.setString(4, carID);
730. stmt.setTimestamp(5, Timestamp.valueOf(violationTime));
731. stmt.setString(6, recorderID);
733. **return** stmt.executeUpdate() > 0;
734. } **catch** (SQLException e) {
735. System.err.println("记录违章信息失败: " + e.getMessage());
736. }
737. **return** **false**;
738. }
740. // 内部类：用于封装违章详细信息
741. **public** **static** **class** ViolationDetail {
742. **private** **final** String driverName;
743. **private** **final** String violationName;
744. **private** **final** String violationPunishment;
745. **private** **final** LocalDateTime recordTime;
746. **private** **final** **int** recordStationID;
747. **private** **final** String recordCarID;
749. **public** ViolationDetail(String driverName, String violationName, String violationPunishment,
750. LocalDateTime recordTime, **int** recordStationID, String recordCarID) {
751. **this**.driverName = driverName;
752. **this**.violationName = violationName;
753. **this**.violationPunishment = violationPunishment;
754. **this**.recordTime = recordTime;
755. **this**.recordStationID = recordStationID;
756. **this**.recordCarID = recordCarID;
757. }
759. @Override
760. **public** String toString() {
761. **return** "违章详细信息:\n" +
762. "司机姓名: " + driverName + "\n" +
763. "违规名称: " + violationName + "\n" +
764. "处罚措施: " + violationPunishment + "\n" +
765. "违规时间: " + recordTime + "\n" +
766. "违规地点（站点ID）: " + recordStationID + "\n" +
767. "违规车辆（车牌号）: " + recordCarID + "\n" +
768. "-----------------------------";
769. }
770. }
772. // 定义自定义异常类以处理登录失败的情况
773. **public** **static** **class** LoginFailedException **extends** Exception {
774. **public** LoginFailedException(String message) {
775. **super**(message);
776. }
777. }
779. @Override
780. **public** **void** close() **throws** Exception {
781. **if** (connection != **null** && !connection.isClosed()) {
782. connection.close();
783. }
784. }
785. }