

## 实验二 高级SQL语言的使用

2022秋



# 本学期实验总体安排

本学期实验课程共 16 个学时，5 个实验项目，总成绩为 30 分。

实验项目	实验一	实验二	实验三	实验四		实验五
学时	2	2	2	4	2	4
实验内容	MySQL及SQL语言的使用	高级SQL语言的使用	openGauss的AI特性实验	一个小型系统的设计与实现		查询处理算法的模拟实现
分数	4	4	4	10		8

# 目录

---

1

实验目的

2

实验内容

3

实验原理

4

实验步骤

5

作业提交

# 实验目的

---

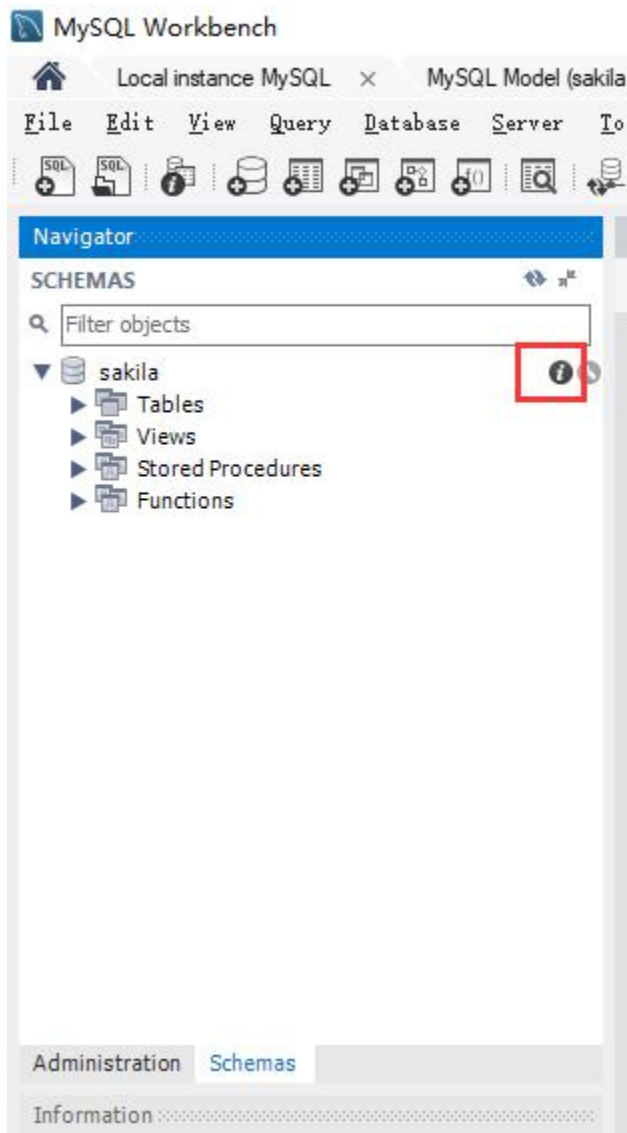
1. 理解视图、触发器、约束的基本概念，掌握它们的用法;
2. 能结合实例设计合适的视图、触发器和约束;
3. 结合实验加深对数据库完整性和安全性的理解。

# 实验内容

---

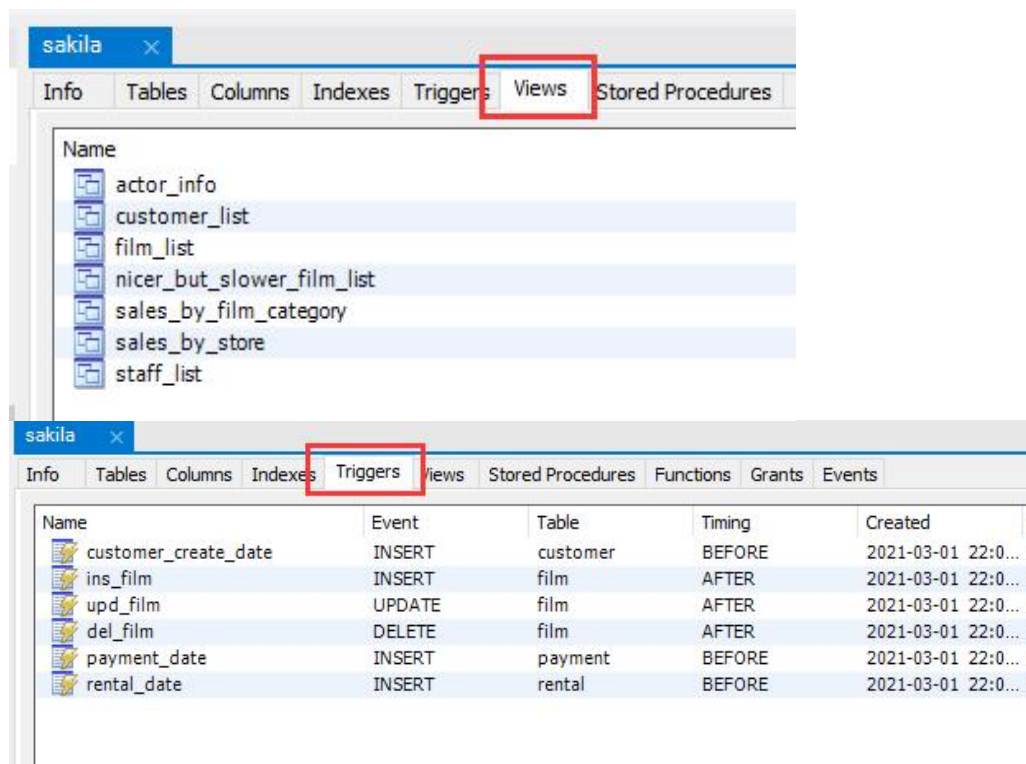
1. 理解Sakila数据库中的视图、触发器和约束;
2. 根据场景, 为Sakila数据库设计并实现合理的视图和触发器;
3. 创建新的数据库用户, 并为其分配权限。

# 实验原理



## Sakila样例数据库包括:

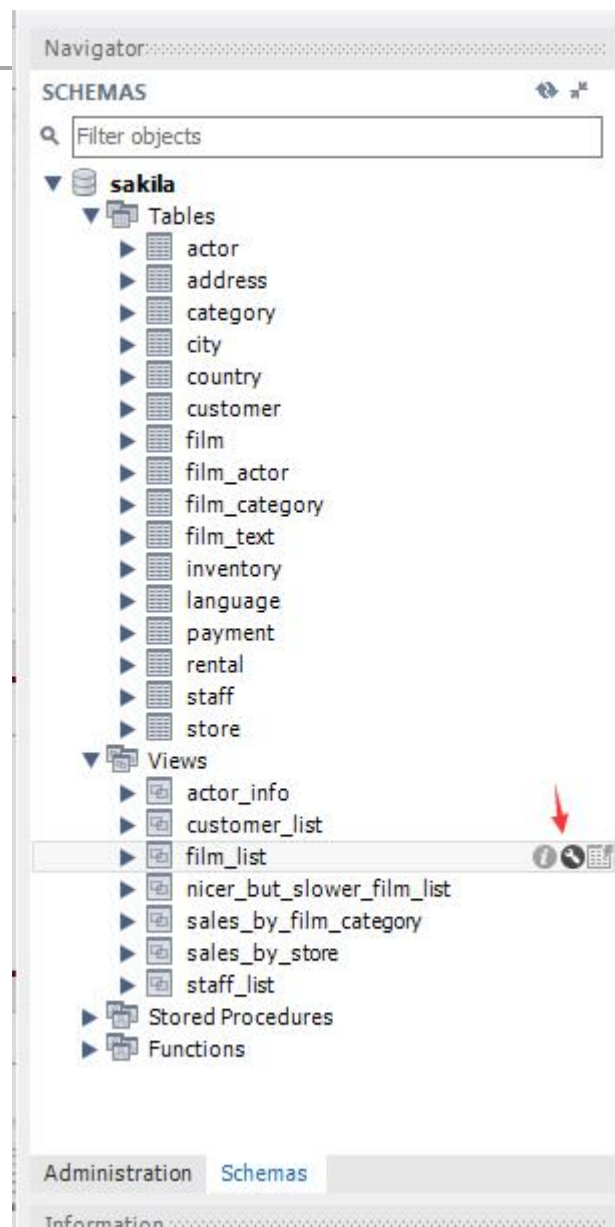
- 16 张表
- 7个视图
- 6个触发器



# 实验原理

## 视图 (Views)

- 简单性
- 安全性
- 逻辑数据独立性



# 实验原理

## 视图 (Views)

### the staff\_list view



Name: staff\_list

The name of the view is parsed automatically from the DDL statement. The DDL is parsed automatically while you type.

DDL:



```
1  CREATE
2  ALGORITHM = UNDEFINED
3  DEFINER = `root`@`localhost`
4  SQL SECURITY DEFINER
5  VIEW `sakila`.`staff_list` AS
6  SELECT
7      `s`.`staff_id` AS `ID`,
8      CONCAT(`s`.`first_name`,
9             utf8mb4 ' ',
10             `s`.`last_name`) AS `name`,
11      `a`.`address` AS `address`,
12      `a`.`postal_code` AS `zip code`,
13      `a`.`phone` AS `phone`,
14      `sakila`.`city`.`city` AS `city`,
15      `sakila`.`country`.`country` AS `country`,
16      `s`.`store_id` AS `SID`
17  FROM
18      (((`sakila`.`staff` `s`
19      JOIN `sakila`.`address` `a` ON ((`s`.`address_id` = `a`.`address_id`)))
20      JOIN `sakila`.`city` ON ((`a`.`city_id` = `sakila`.`city`.`city_id`)))
21      JOIN `sakila`.`country` ON ((`sakila`.`city`.`country_id` = `sakila`.`country`.`country_id`)))
```



# 实验原理

## 视图 (Views)

```
1 • SELECT
2     `s`.`staff_id` AS `ID`,
3     CONCAT(`s`.`first_name`,
4         _utf8mb4 ' ',
5         `s`.`last_name`) AS `name`,
6     `a`.`address` AS `address`,
7     `a`.`postal_code` AS `zip code`,
8     `a`.`phone` AS `phone`,
9     `city`.`city` AS `city`,
10    `country`.`country` AS `country`,
11    `s`.`store_id` AS `SID`
12 FROM
13    (((`staff` `s`
14 JOIN `address` `a` ON (`s`.`store_id` = `a`.`store_id` AND `s`.`staff_id` = `a`.`staff_id`)
15 JOIN `city` `c` ON (`a`.`city_id` = `c`.`city_id`)
16 JOIN `country` `co` ON (`a`.`country_id` = `co`.`country_id`))
```

Limit to 2000 rows

1 • select \* from staff\_list;

2

3

Result Grid

Filter Rows:

Export:

Wrap Cell Content: IA

	ID	name	address	zip code	phone	city	country	SID
▶	1	Mike Hillyer	23 Workhaven Lane		14033335568	Lethbridge	Canada	1
▶	2	Jon Stephens	1411 Lillydale Drive		6172235589	Woodridge	Australia	2

# 实验原理

## 触发器 (Triggers)

The screenshot displays the MySQL Workbench interface for the 'sakila' database, specifically the 'film' table. The interface is divided into several sections:

- Navigator:** Shows the database structure. The 'film' table is selected, and a red arrow labeled '1' points to its icon.
- Table Information:** Displays details for the 'film' table, including Schema (sakila), Charset/Collation (utf8mb4), and Engine (InnoDB).
- Triggers:** A list of triggers for the 'film' table is shown. The 'upd\_film' trigger is highlighted, and a red arrow labeled '3' points to it.
- SQL Editor:** Contains the SQL code for creating the 'upd\_film' trigger. A red arrow labeled '2' points to the 'Triggers' tab at the bottom of the editor.

The SQL code in the editor is as follows:

```
1 CREATE DEFINER='root'@'localhost' TRIGGER `upd_film` AFTER UPDATE ON `film` FOR EACH ROW BEGIN
2   IF (old.title != new.title) OR (old.description != new.description) OR (old.film_id != new.film_id)
3   THEN
4     UPDATE film_text
5       SET title=new.title,
6           description=new.description,
7           film_id=new.film_id
8     WHERE film_id=old.film_id;
9   END IF;
10 END
```

# 实验原理

## 触发器 (Triggers)

```
CREATE DEFINER='root'@'localhost' TRIGGER `upd_film` AFTER UPDATE ON `film` FOR EACH ROW BEGIN
  IF (old.title != new.title) OR (old.description != new.description) OR (old.film_id != new.film_id)
  THEN
    UPDATE film_text
      SET title=new.title,
          description=new.description,
          film_id=new.film_id
    WHERE film_id=old.film_id;
  END IF;
END
```

触发器创建语法四要素：

- ① 监视点(table)
- ② 监视事件(insert/update/delete)
- ③ 触发时间(after/before)
- ④ 触发事件(insert/update/delete)

# 实验原理

## 触发器 (Triggers)



The screenshot shows a MySQL IDE interface. The main editor displays SQL code for creating a trigger named 'upd\_film' in the 'sakila' database. The code is as follows:

```
1 use sakila;
2
3 CREATE DEFINER='root'@'localhost' TRIGGER `upd_film` AFTER UPDATE ON `film` FOR EACH ROW BEGIN
4     IF (old.title != new.title) OR (old.description != new.description) OR (old.film_id != new.film_id)
5     THEN
6         UPDATE film_text
7         SET title=new.title,
8             description=new.description,
9             film_id=new.film_id
10        WHERE film_id=old.film_id;
11    END IF;
12 END
```

Line 8 has a red squiggly line under 'description', indicating a syntax error. A yellow box with the text '新建trigger报错!' (Error creating trigger!) is overlaid on the right side of the code editor.

Below the code editor is the 'Output' panel, which shows the execution results:

#	Time	Action	Message
1	15:49:24	use sakila	0 row(s) affected
2	15:49:24	CREATE DEFINER='root'@'localhost' TRIGGER `upd_film` AFTER U...	Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near "" at line 8

# 实验原理

## 触发器 (Triggers)

```
209 --
210 -- Triggers for loading film_text from film
211 --
212
213 DELIMITER ;;
214 CREATE TRIGGER `ins_film` AFTER INSERT ON `film` FOR EACH ROW BEGIN
215     INSERT INTO film_text (film_id, title, description)
216         VALUES (new.film_id, new.title, new.description);
217 END;;
218
219
220 CREATE TRIGGER `upd_film` AFTER UPDATE ON `film` FOR EACH ROW BEGIN
221     IF (old.title != new.title) OR (old.description != new.description) OR (old.film_id != new.film_id)
222     THEN
223         UPDATE film_text
224             SET title=new.title,
225                 description=new.description,
226                 film_id=new.film_id
227         WHERE film_id=old.film_id;
228     END IF;
229 END;;
230
231
232 CREATE TRIGGER `del_film` AFTER DELETE ON `film` FOR EACH ROW BEGIN
233     DELETE FROM film_text WHERE film_id = old.film_id;
234 END;;
235
236 DELIMITER ;
237
```

Annotations in the image:

- A red box highlights `DELIMITER ;;` on line 213.
- A red box highlights `END;;` on line 229.
- A red box highlights `DELIMITER ;` on line 236.
- Red arrows point from the text "字段名" (Field Name) to `old.film_id` and `new.film_id` in the `WHERE` clause on line 227.
- Red arrows point from the text "值" (Value) to `old.film_id` and `new.film_id` in the `WHERE` clause on line 227.

# 实验原理

## 触发器 (Triggers)



Table Name: customer

Schema: sakila

Charset/Collation: utf8mb4

utf8mb4\_0900\_ai\_ci

Engine: InnoDB

Comments:

### ▼ BEFORE INSERT

customer\_create\_date

total\_name

AFTER INSERT

BEFORE UPDATE

AFTER UPDATE

BEFORE DELETE

```
1 CREATE DEFINER=`root`@`localhost` TRIGGER `customer_create_date` BEFORE INSERT ON `customer` FOR EACH ROW
2 SET NEW.create_date = NOW();
```

**注意：**new.字段的值可以在before类型的触发器中进行赋值和取值，在after类型的触发器中只能取值。

# 实验原理

---

## 约束 (Constraint)

**作用：**为了确保表中的数据**正确性、准确性、完整性**。

### 常用约束：

- 主键： primary key
- 非空约束： not null
- 唯一约束： unique
- 外键约束： foreign key
- .....



# 实验原理

## 约束 (Constraint)

```
CREATE TABLE rental (  
    rental_id INT NOT NULL AUTO INCREMENT,  
    rental_date DATETIME NOT NULL,  
    inventory_id MEDIUMINT UNSIGNED NOT NULL,  
    customer_id SMALLINT UNSIGNED NOT NULL,  
    return_date DATETIME DEFAULT NULL,  
    staff_id TINYINT UNSIGNED NOT NULL,  
    last_update TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,  
    PRIMARY KEY (rental_id),  
    UNIQUE KEY (rental_date,inventory_id,customer_id),  
    KEY idx_fk_inventory_id (inventory_id),  
    KEY idx_fk_customer_id (customer_id),  
    KEY idx_fk_staff_id (staff_id),  
    CONSTRAINT fk_rental_staff FOREIGN KEY (staff_id) REFERENCES staff (staff_id) ON DELETE RESTRICT ON UPDATE CASCADE,  
    CONSTRAINT fk_rental_inventory FOREIGN KEY (inventory_id) REFERENCES inventory (inventory_id) ON DELETE RESTRICT ON UPDATE CASCADE,  
    CONSTRAINT fk_rental_customer FOREIGN KEY (customer_id) REFERENCES customer (customer_id) ON DELETE RESTRICT ON UPDATE CASCADE  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```



# 实验原理

---

## 结构化查询语言(Structured Query Language, SQL)

- DDL - Data Definition Language, 数据定义语言
- DML - Data Manipulation Language, 数据处理语言
- **DCL - Data Control Language, 数据控制语言**  
控制数据的访问权限, 只有被授权的用户才能操作数据。

### 1. 创建用户

create user [用户名] identify by [登录密码];

### 2. 删除用户

drop user [用户名];

### 3. 用户授权

grant [权限1,权限2,...] on [数据库名].[表名] to [用户名];

### 4. 撤销授权

revoke [权限1,权限2,...] on [数据库名].[表名] from [用户名];

# 实验步骤

---

1. 跟随实验指导书的指引观察和分析Sakila数据库中的视图、触发器和约束，并回答实验指导书中的问题。

2. 根据应用场景，为Sakila数据库合理地设计并实现：

- 1个视图
- 1个触发器

**要求：**视图需关联至少2个表，触发器需验证是否生效。  
请在报告中提供创建语句和调用结果截图。

3. 根据实验指导书要求，创建新用户并为之分配权限。

4. 【附加题】思考问题：在阿里开发规范里有一条 “**【强制】不得使用外键与级联，一切外键概念必须在应用层解决。**” 请分析一下原因。外键是否没有存在的必要？

# 作业提交

---

- **课后提交**：提交实验报告至作业提交平台（截止日期参考平台发布）

作业平台入口：<http://grader.tery.top:8000/#/login>

用户名、密码默认是你的**学号**

- 推荐使用 **Chrome** 浏览器
- 注意提交 **pdf** 格式的报告

