



实验二 高级SQL语言的使用

2024秋



本学期实验总体安排

本学期**实验**课程共 **16** 个学时， **3**个实验项目， 总成绩为 **30** 分。

实验项目	MySQL及SQL的 使用		数据库设计	数据库系统功能 实现	
学时	2	2	4	4	4
分数	8		8	14	

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

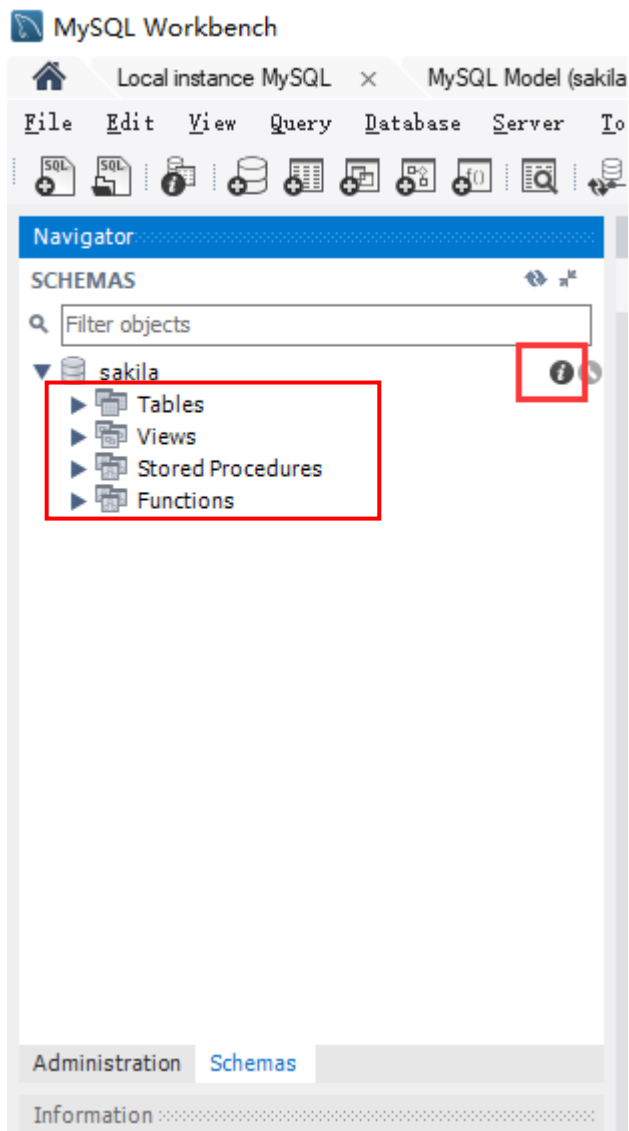
实验目的

1. 理解视图、触发器、约束、存储过程和函数的基本概念，掌握它们的用法；
2. 能结合实例设计合适的视图、触发器和存储过程；
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. 理解和分析Sakila数据库中的视图、触发器、约束、存储过程和函数；
2. 根据场景，为Sakila数据库设计并实现合理的视图、触发器和存储过程；
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:

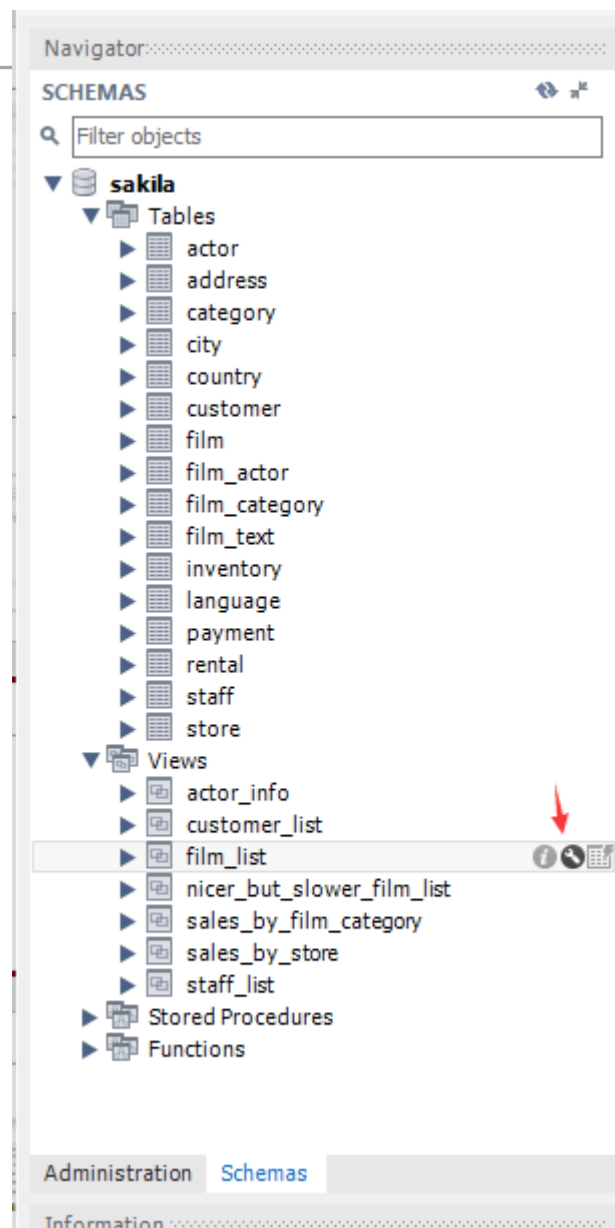
- 16 张表
- 7个视图
- 6个触发器
- 3个存储过程
- 3个函数

Info	Tables	Columns	Indexes	Triggers	Views	Stored Procedures	Functions	Grants	Events
Name	Event	Table	Timing	Created					
customer_create_date	INSERT	customer	BEFORE	2022-08-11					
ins_film	INSERT	film	AFTER	2022-08-11					
upd_film	UPDATE	film	AFTER	2022-08-11					
del_film	DELETE	film	AFTER	2022-08-11					
payment_date	INSERT	payment	BEFORE	2022-08-11					
rental_date	INSERT	rental	BEFORE	2022-08-11					

实验原理

视图 (Views)

- 简单
- 安全
- 数据独立



实验原理

视图 (Views)

staff_list



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)

The screenshot displays a SQL IDE interface. On the left, a complex SQL query is written across 16 lines. It uses subqueries to select staff information from tables like 'staff_list', 'address', 'city', and 'country'. The query includes aliases for 'ID', 'name', 'address', and 'SID'. On the right, a simplified version of the query is shown: `select * from staff_list;`. Below the queries, a 'Result Grid' is visible, showing two rows of data. The grid has columns for ID, name, address, zip code, phone, city, country, and SID. The data rows correspond to Mike Hillyer and Jon Stephens.

```
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`
15     JOIN `city` ON (`a`.`city_id` = `city`.`city_id`)
16     JOIN `country` ON (`a`.`country_id` = `country`.`country_id`))
```

select * from staff_list;

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)

Navigation: sakila | film - Table

Table Name: film | **Schema:** sakila

Charset/Collation: utf8mb4 | utf8mb4_0900_ai_ci | **Engine:** InnoDB

Comments:

Triggers:

- BEFORE INSERT
- ▼ AFTER INSERT
 - ins_film
- BEFORE UPDATE
- ▼ AFTER UPDATE
 - upd_film** (3)
- BEFORE DELETE
- ▼ AFTER DELETE
 - del_film

SQL Code:

```
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
```

Table: film

Columns:

- film_id: smallint(5) UN AI PK
- title: varchar(128)
- description: text
- release_year: year(4)
- language_id: tinyint(3) UN
- original_language_id: tinyint(3) UN
- rental_duration: decimal(4,2)
- rental_rate: decimal(5,2)
- length: smallint(5) UN
- replacement_cost: decimal(5,2)

Navigation: Columns | Indexes | Foreign Keys | **Triggers** (2) | Partitioning | Options

实验原理

触发器 (Triggers)

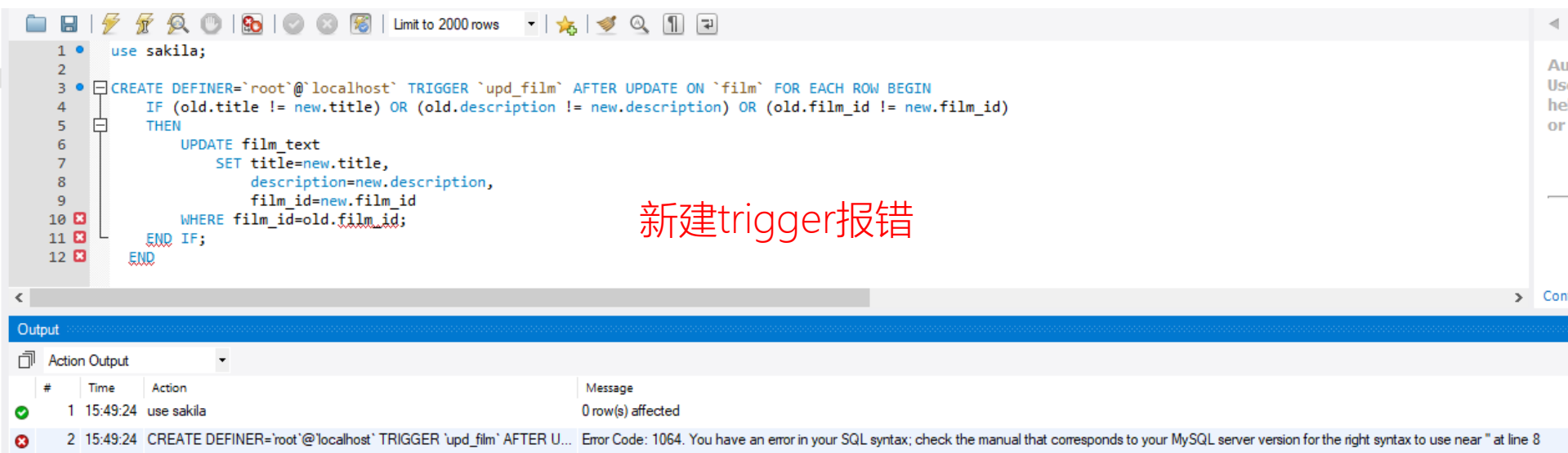
```
CREATE DEFINER='root'@'localhost' TRIGGER `upd_film` AFTER UPDATE ON `film` FOR EACH ROW BEGIN
  IF (old.title != new.title) OR (old.description != new.description) OR (old.film_id != new.film_id)
  THEN
    UPDATE film_text
      SET title=new.title,
          description=new.description,
          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 window with a SQL script editor and an output pane. The script defines a trigger named 'upd_film' that updates the 'film_text' table whenever the 'film' table is updated. The trigger logic 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
```

The output pane shows the results of the script execution:

#	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

The error message indicates a syntax error near the empty string at line 8 of the script. The red text "新建trigger报错" (Error creating trigger) is overlaid on the screenshot.

实验原理

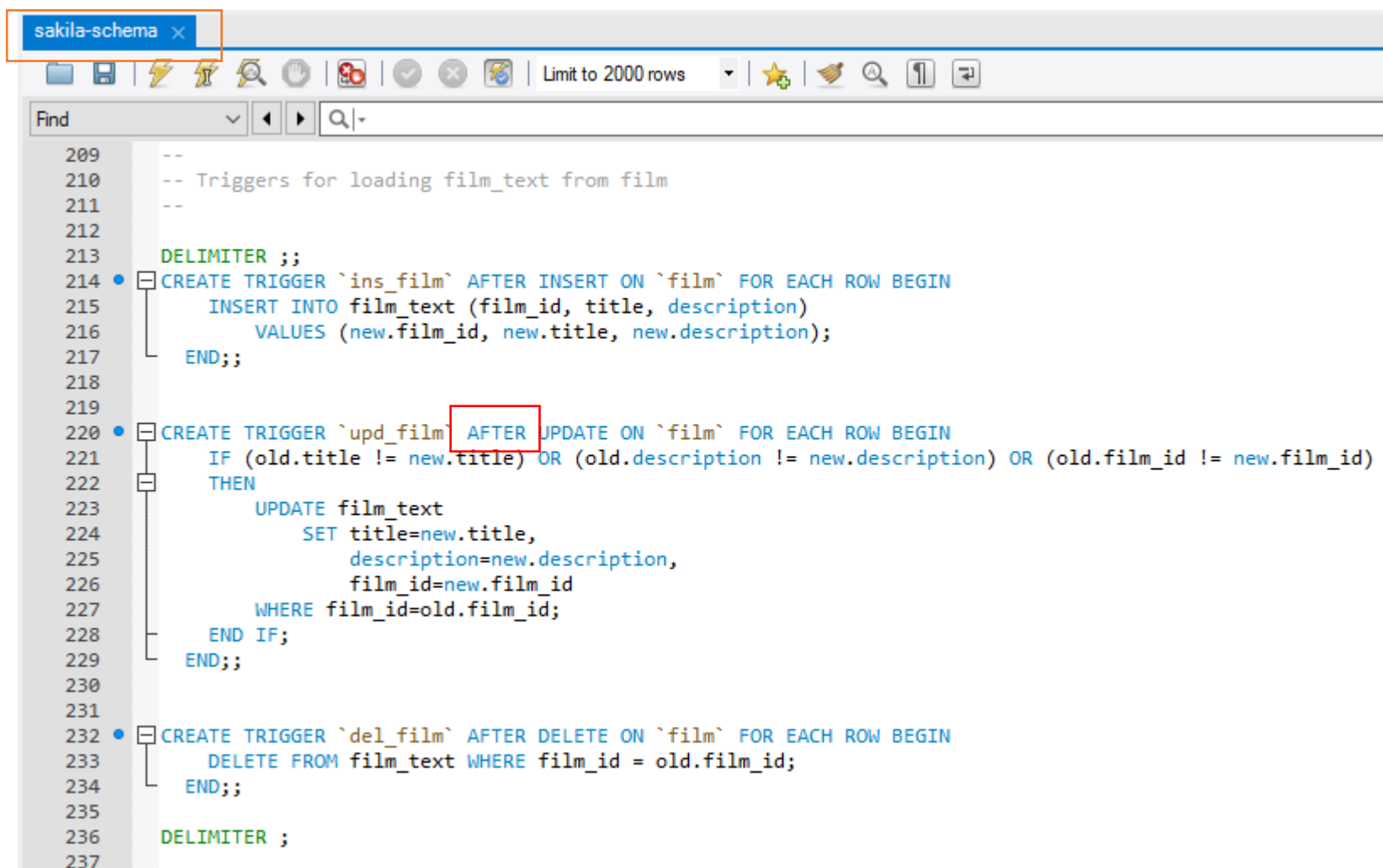
触发器 (Triggers)

```
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 the `DELIMITER ;;` statement on line 213.
- A red box highlights the `CREATE TRIGGER` statement on line 220.
- A red box highlights the `END;;` statement on line 229.
- Red arrows point to the `film_id` field names in the `WHERE` clause on line 227, with the label "字段名" (Field Name).
- Red arrows point to the `old.film_id` and `new.film_id` values in the `WHERE` clause on line 227, with the label "值" (Value).

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  CREATE TRIGGER `upd_film` AFTER UPDATE ON `film` FOR EACH ROW BEGIN
220      IF (old.title != new.title) OR (old.description != new.description) OR (old.film_id != new.film_id)
221      THEN
222          UPDATE film_text
223              SET title=new.title,
224                  description=new.description,
225                  film_id=new.film_id
226              WHERE film_id=old.film_id;
227      END IF;
228  END;;
229
230
231  CREATE TRIGGER `del_film` AFTER DELETE ON `film` FOR EACH ROW BEGIN
232      DELETE FROM film_text WHERE film_id = old.film_id;
233  END;;
234
235  DELIMITER ;
236
237
```

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()
```

NEW 关键字用于引用触发事件过程中对某表行（记录）的新版本；
OLD 关键字用来引用操作前的状态。

思考：为什么在 **AFTER** 触发器中只能对 **NEW** 取值，不能对 **NEW** 进行赋值？

约束 (Constraint)

作用：为了确保表中的数据**正确性、有效性、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

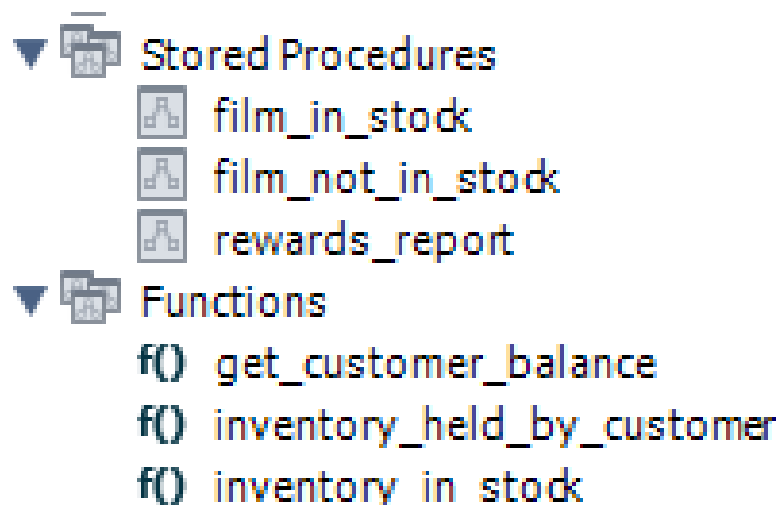
存储过程 (Procedure) & 函数 (Function)

- 可统称为存储程序
- 是一组为了完成特定功能的SQL语句集
- 优点:
 - ① 复用：存储的程序可重用
 - ② 安全：仅授予某些用户访问存储程序的权限，而不需要提供访问基础数据库表的任何权限

实验原理

存储过程 (Procedure) & 函数 (Function)

- sakila已有



实验原理

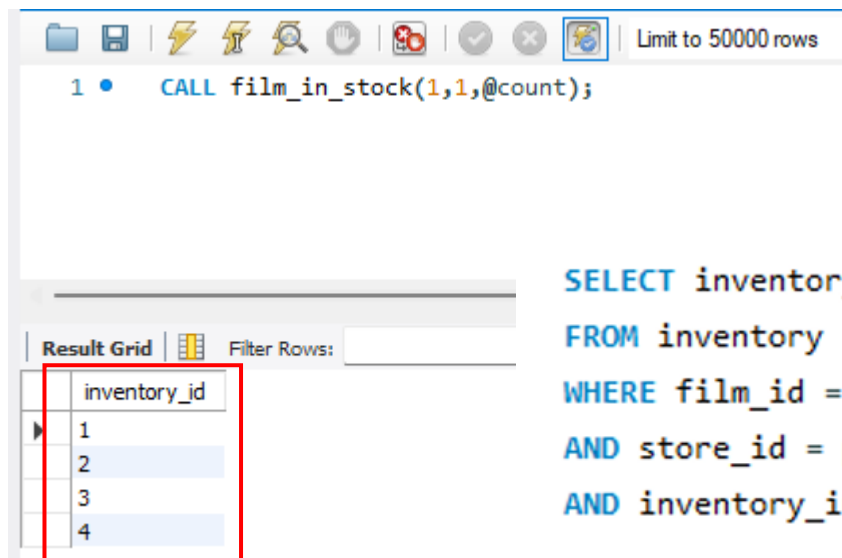
存储过程 (Procedure)

```
587 DELIMITER $$
588
589 • CREATE PROCEDURE film_in_stock(IN p_film_id INT, IN p_store_id INT, OUT p_film_count INT)
590 READS SQL DATA
591 BEGIN
592     SELECT inventory_id
593     FROM inventory
594     WHERE film_id = p_film_id
595     AND store_id = p_store_id
596     AND inventory_in_stock(inventory_id);
597
598     SELECT COUNT(*)
599     FROM inventory
600     WHERE film_id = p_film_id
601     AND store_id = p_store_id
602     AND inventory_in_stock(inventory_id)
603     INTO p_film_count;
604 END $$
605
606 DELIMITER ;
```

实验原理

存储过程 (Procedure)

- 调用



The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, execution, and a 'Limit to 50000 rows' option. The SQL editor contains the following code:

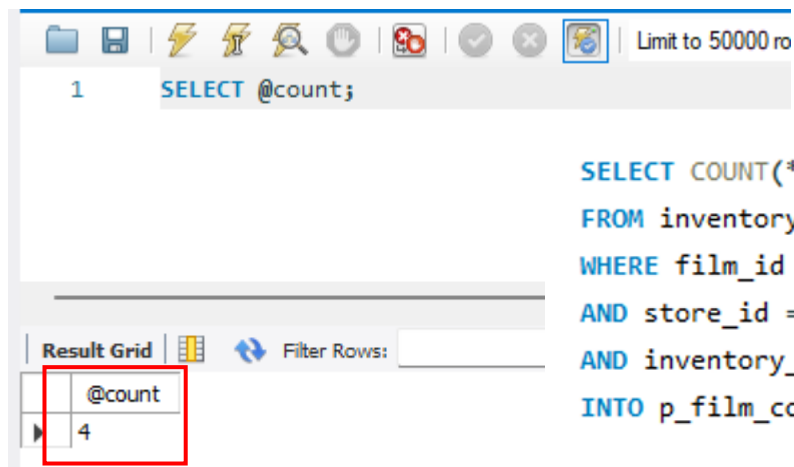
```
1 • CALL film_in_stock(1,1,@count);
```

Below the editor is the 'Result Grid' tab. It displays a table with one column, 'inventory_id', and four rows with values 1, 2, 3, and 4. The first row is highlighted with a red border.

inventory_id
1
2
3
4

```
SELECT inventory_id  
FROM inventory  
WHERE film_id = p_film_id  
AND store_id = p_store_id  
AND inventory_in_stock(inventory_id);
```

- 获得返回值



The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, execution, and a 'Limit to 50000 rows' option. The SQL editor contains the following code:

```
1 SELECT @count;
```

Below the editor is the 'Result Grid' tab. It displays a table with one column, '@count', and one row with the value 4. The first row is highlighted with a red border.

@count
4

```
SELECT COUNT(*)  
FROM inventory  
WHERE film_id = p_film_id  
AND store_id = p_store_id  
AND inventory_in_stock(inventory_id)  
INTO p_film_count;
```

函数 (Function)

```
647 DELIMITER $$
648
649 • CREATE FUNCTION inventory_in_stock(p_inventory_id INT) RETURNS BOOLEAN
650 READS SQL DATA
651 BEGIN
652     DECLARE v_rentals INT;
653     DECLARE v_out     INT;
654
655     #AN ITEM IS IN-STOCK IF THERE ARE EITHER NO ROWS IN THE rental TABLE
656     #FOR THE ITEM OR ALL ROWS HAVE return_date POPULATED
657
658     SELECT COUNT(*) INTO v_rentals
659     FROM rental
660     WHERE inventory_id = p_inventory_id;
661
662     IF v_rentals = 0 THEN
663         RETURN TRUE;
664     END IF;
665
666     SELECT COUNT(rental_id) INTO v_out
667     FROM inventory LEFT JOIN rental USING(inventory_id)
668     WHERE inventory.inventory_id = p_inventory_id
669     AND rental.return_date IS NULL;
670
671     IF v_out > 0 THEN
672         RETURN FALSE;
673     ELSE
674         RETURN TRUE;
675     END IF;
676 END $$
677 DELIMITER ;
```

实验原理

函数 (Function)

```
647 DELIMITER $$
648
649 • CREATE FUNCTION inventory_in_stock(p_inventory_id INT) RETURNS BOOLEAN
650 READS SQL DATA
651 BEGIN
    .....主要逻辑
676 END $$
677
678 DELIMITER ;
```

函数 (Function)

```
651 BEGIN
652     DECLARE v_rentals INT;
653     DECLARE v_out     INT;
654
655     #AN ITEM IS IN-STOCK IF THERE ARE EITHER NO ROWS IN THE rental TABLE
656     #FOR THE ITEM OR ALL ROWS HAVE return_date POPULATED
657
658     SELECT COUNT(*) INTO v_rentals
659     FROM rental
660     WHERE inventory_id = p_inventory_id;
661
662     IF v_rentals = 0 THEN
663         RETURN TRUE;
664     END IF;
665
666     SELECT COUNT(rental_id) INTO v_out
667     FROM inventory LEFT JOIN rental USING(inventory_id)
668     WHERE inventory.inventory_id = p_inventory_id
669     AND rental.return_date IS NULL;
670
671     IF v_out > 0 THEN
672         RETURN FALSE;
673     ELSE
674         RETURN TRUE;
675     END IF;
676 END $$
```

该inventory_id从来没有出租过，返回true

该inventory_id借出去没有归还，返回false
否则，返回true

实验原理

函数 (Function)

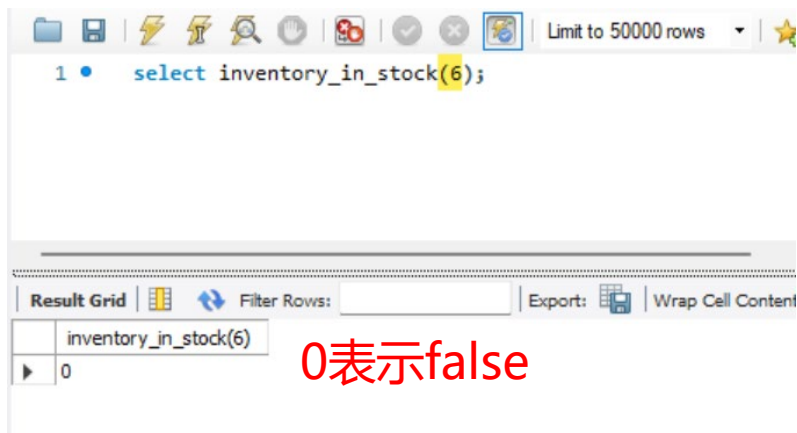
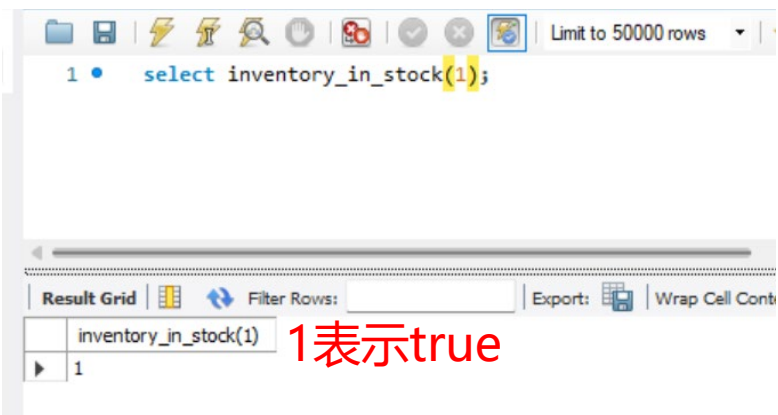
- 使用

```
587 DELIMITER $$
588
589 • CREATE PROCEDURE film_in_stock(IN p_film_id INT, IN p_store_id INT, OUT p_film_count INT)
590 READS SQL DATA
591 BEGIN
592     SELECT inventory_id
593     FROM inventory
594     WHERE film_id = p_film_id
595     AND store_id = p_store_id
596     AND inventory_in_stock(inventory_id);
597
598     SELECT COUNT(*)
599     FROM inventory
600     WHERE film_id = p_film_id
601     AND store_id = p_store_id
602     AND inventory_in_stock(inventory_id)
603     INTO p_film_count;
604 END $$
605
606 DELIMITER ;
```

实验原理

函数 (Function)

- 使用



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] identified 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、约束、存储过程和函数，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
- 1个存储过程

要求：视图需关联至少2个表，触发器需验证是否生效，存储过程需调用
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. 完成实验报告。

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



**同学们
请开始实验吧!**