实验二报告

一、 观察并回答问题

1. 关于视图

（1） sakila.mwb 模型图中共有几个 View？

7 个视图：actor_info、customer_list、film_list、nicer_but_slower_film_list、sales_by_film_category、sales_by_store、staff_list。

（2） 分析以下 3 个视图，回答以下问题：

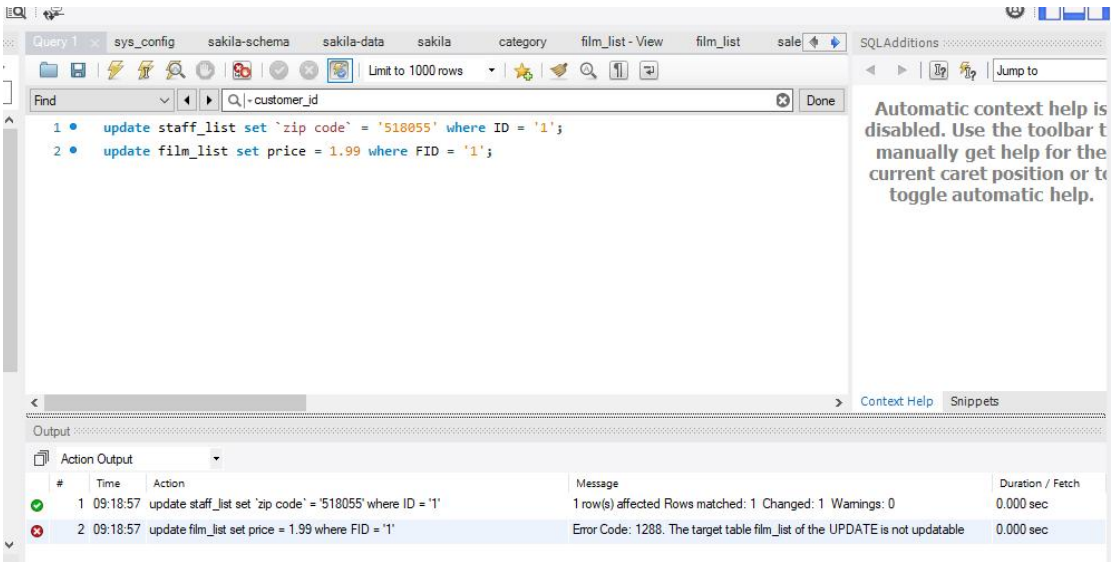
视图名	关联表	作用
actor_info	actor、film、film_actor、film_category、category	提供所有演员的列表，按照类别分类，列出他们演过的电影，
film_list	film、actor、film_category、category、film_actor	提供所有电影的视图，每个电影包含一个逗号分开的演员列表
sales_by_store	city、country、payment、rental、inventory、store、address、staff	提供 2 个商店的总销售额列表，包含商店位置、经理姓名和总销售额

（3） 分别执行以下 2 句 SQL 语句：

```
update staff_list set `zip code` = '518055' where ID = '1';
```

```
update film_list set price = 1.99 where FID = '1';
```

截图执行结果，并分析一下视图在什么情况下可以进行 update 操作，什么情况下不能？



如图，第一句执行成功，第二句执行报错。

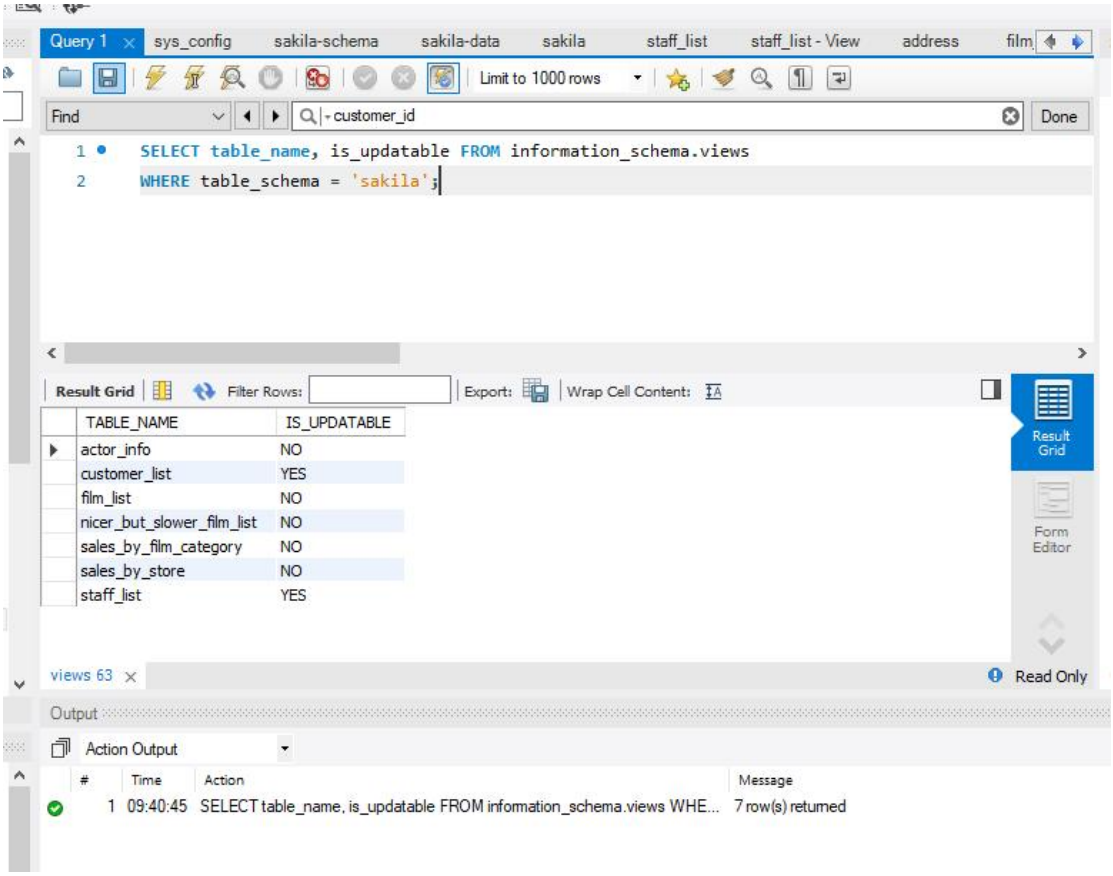
视图的更新要反映到基本表的更新上，且操作有较大约束。film_list 视图的构造中含有 group by 子句，在这种情况下不能更新；staff_list 是多表连接导出的，其 select 目标列不含有聚合函数，select 子句不含 unique 或 distinct，不含由算术表达式计算出来的列，也不含 group by 子句，可以更新。

一般来说，更新视图需要满足以下条件：select 目标列不含有聚合函数，select 子句不含 unique 或 distinct，不含由算术表达式计算出来的列，也不含 group by 子句。若视图是由单个表的列构成，则需要包括主键。

（4） 执行以下命令查询 sakila 数据库中的视图是否可更新，截图执行结果：

```
SELECT table_name, is_updatable FROM information_schema.views
WHERE table_schema = 'sakila';
```

查询结果如下：



2. 关于触发器

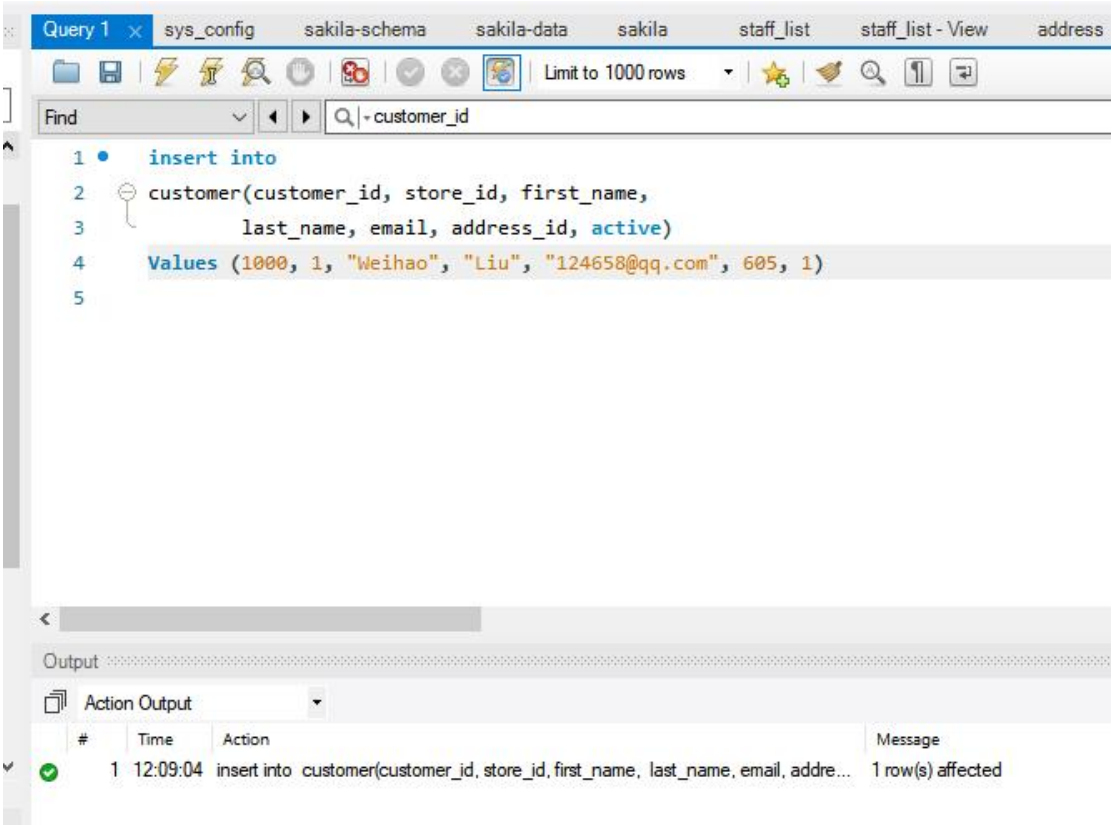
（1） 触发器 customer_create_date 建在哪个表上？这个触发器实现什么功能？

建立在 customer 表上。当 customer 表执行插入操作时，这个触发器将 create_date 列设置为当前的时间和日期。

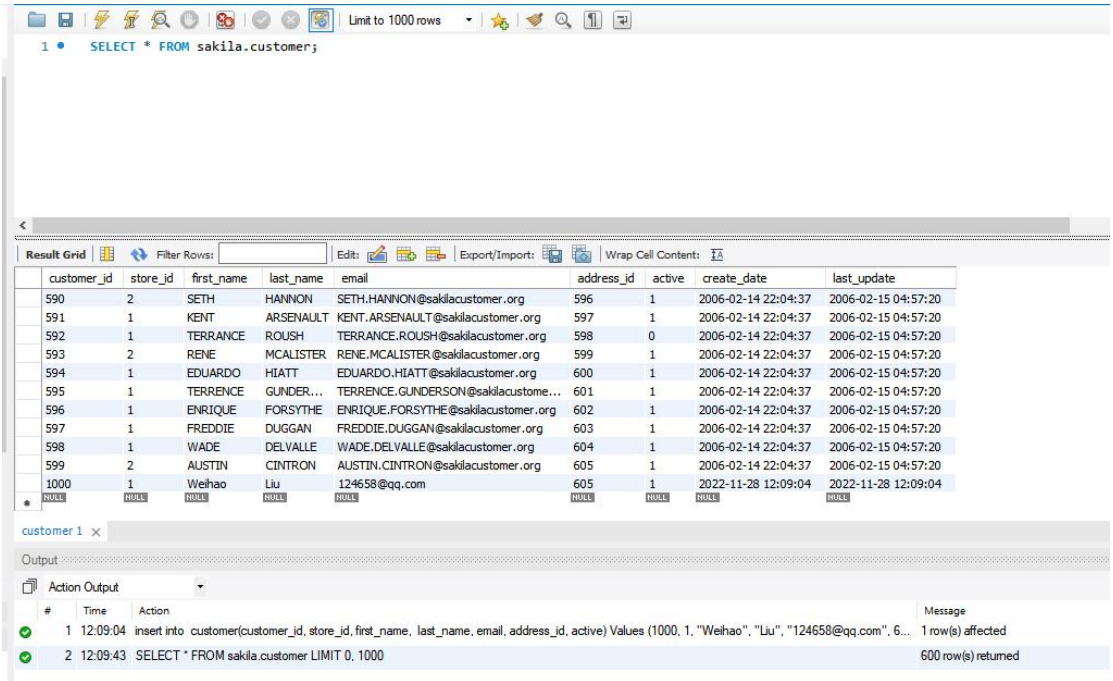
（2） 在这个表上新增一条数据，验证一下触发器是否生效。（截图语句和执行结果）

```
insert into
customer(customer_id, store_id, first_name,
        last_name, email, address_id, active)
Values (1000, 1, "Weihao", "Liu", "124658@qq.com", 605, 1)
```

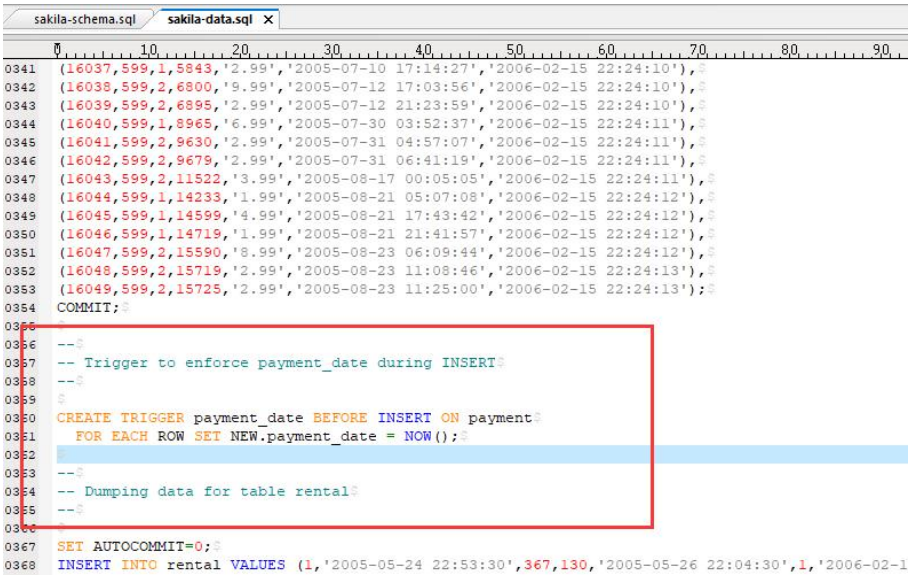
执行结果：



查看 customer 表，发现 create_date 列确实被设置为了当前时间：



（3） 我们可以看到 sakila-schema.sql 里的语句是用于创建数据库的结构，包括表、视图、触发器等，而 sakila-data.sql 主要是用于往表写入数据。但 sakila-data.sql 里有这样一个建立触发器 payment_date 的语句，这个触发器是否可以移到 sakila-schema.sql 里去执行？为什么？



不能。这里是在 payment 插入数据后，才声明了触发器。这样能使触发器只作用于后续插入的过程，而图中 30354 行之前的插入不受影响。

如果移到 sakila-schema.sql 里去执行，那 sakila-data.sql 中插入 payment 表的元组的 payment_date 字段会受到影响。

3. 关于约束

（1） store 表上建了哪几种约束？这些约束分别实现什么功能？（至少写 3 个）

约束类型	功能
主键约束	在 CREATE TABLE 语句中，通过 PRIMARY KEY 关键字来指定主键，作为行数据的唯一标识
非空约束	创建表时可以使用 NOT NULL 关键字设置非空约束，可限制相关列非空
唯一约束	在定义完列之后直接使用 UNIQUE 关键字指定唯一约束，限制不允许两个元组的该列相同
外键约束	在 CREATE TABLE 语句中，通过 FOREIGN KEY 关键字来指定外键，体现出两个表的关联

（2） 图中 sakila-schema.sql 第 343 行的 ON DELETE RESTRICT 和 ON UPDATE CASCADE 是什么意思？

ON DELETE RESTRICT 表示当在父表（即 staff 表）中删除对应记录时，首先在 store 中检查该记录是否有对应外键 manager_staff_id，如果有则不允许删除。

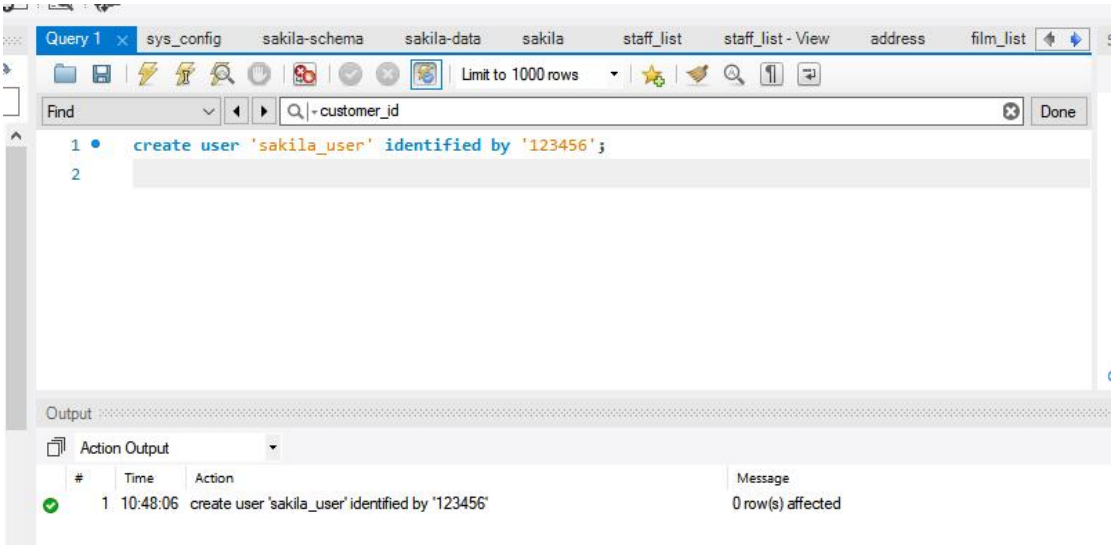
ON UPDATE CASCADE 表示当在父表（即 staff 表）中更新对应记录时，首先检查该记录是否有对应外键，如果有则也更新外键在子表（即 store 表）中的记录。

二、 创建新用户并分配权限

（截图语句和执行结果）

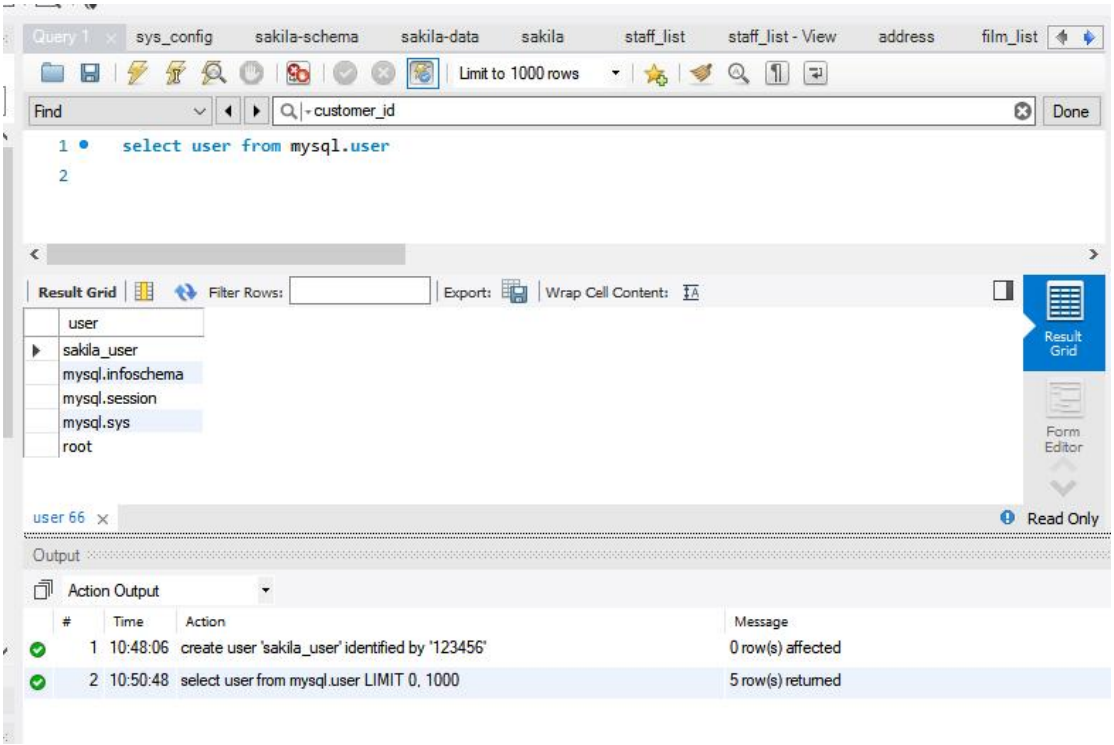
（1） 执行命令新建 sakila_user 用户（密码 123456）；

```
create user 'sakila_user' identified by '123456';
```



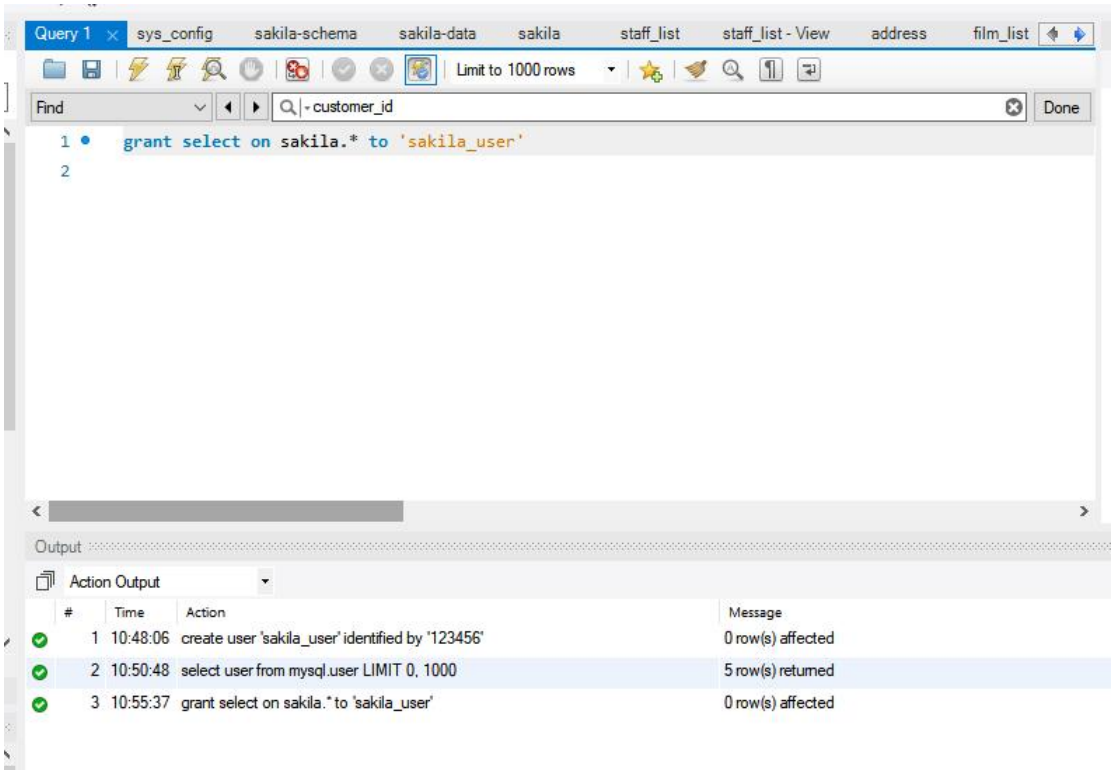
（2） 执行命令查看当前已有用户；

```
select user from mysql.user
```



（3） 执行命令把 sakila 数据库的访问权限赋予 sakila_user 用户；

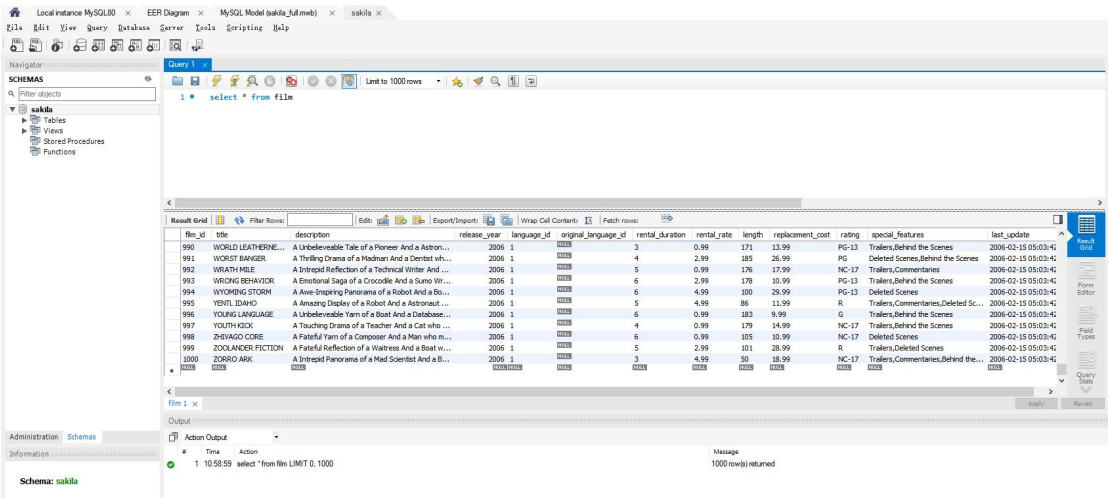
```
grant select on sakila.* to 'sakila_user'
```



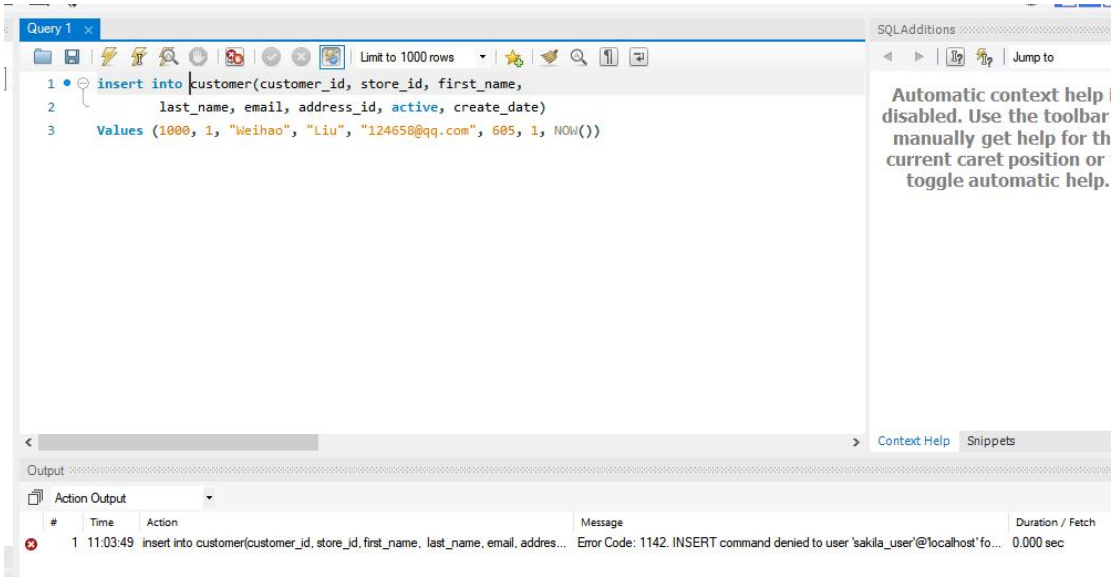
这里只分配了 select 权限，sakila_user 只能进行查看和查询操作。

（4） 切换到 sakila_user 用户，执行 select * from film 操作。

```
select * from film
```



【附】尝试发现，由于权限不足，sakila_user 的 insert 操作会失败



三、 设计并实现

根据应用场景，为 Sakila 数据库合理地设计并实现：

（截图语句和执行结果）

1. 设计 1 个视图，至少关联 2 个表；

（1） 执行新建视图的语句，并截图 SQL 和执行结果：

站在 manager 的角度，创建视图 sales_by_category_and_store，记录一系列销售额，按照商店和影片类别分类，每个商店按照销售额从高到低排序。

```
create view sales_by_category_and_store as(
    select s.store_id as store_id,
           c.name as category,
           sum(p.amount) as total_sales
    from payment p
        inner join rental r on p.rental_id = r.rental_id
        inner join inventory i on r.inventory_id = i.inventory_id
        inner join store s on s.store_id = i.store_id
        inner join film f on i.film_id = f.film_id
        inner join film_category fc on f.film_id = fc.film_id
        inner join category c on fc.category_id = c.category_id
    group by c.category_id, s.store_id
    order by s.store_id asc, total_sales desc
)
```

排序时，先按照商店 id 顺序排序，再按照销售额倒序排序。

```
create view sales_by_category_and_store as(
    select  s.store_id as store_id,
            c.name as category,
            sum(p.amount) as total_sales
    from payment p
        inner join rental r on p.rental_id = r.rental_id
        inner join inventory i on r.inventory_id = i.inventory_id
        inner join store s on s.store_id = i.store_id
        inner join film f on i.film_id = f.film_id
        inner join film_category fc on f.film_id = fc.film_id
        inner join category c on fc.category_id = c.category_id
    group by c.category_id, s.store_id
    order by s.store_id asc, total_sales desc
)
```

Output

#	Time	Action	Message
1	12:38:18	create view sales_by_category_and_store as(select s.store_id as store_id, c.name as category, sum(p.amount) as total_sales from payment p inner j...	0 row(s) affected

（2） 执行 select * from [视图名]，截图执行结果：

```
select * from sales_by_category_and_store
```

```
select * from sales_by_category_and_store
```

store_id	category	total_sales
1	Drama	2573.24
1	Sports	2488.46
1	New	2402.98
1	Comedy	2377.97
1	Action	2342.04
1	Animation	2297.29
1	Foreign	2289.77
1	Sci-Fi	2203.79
1	Family	2166.44
1	Games	2083.46
1	Documen...	1869.51
1	Music	1795.66
1	Children	1758.47
1	Classics	1717.51
1	Horror	1673.14
1	Travel	1640.06
2	Sports	2825.75
2	Sci-Fi	2553.19
2	Animation	2359.01
2	Documen...	2348.01
2	Games	2197.87
2	Family	2059.63
2	Horror	2049.40
2	Action	2033.81
2	Drama	2014.15
2	Comedy	2005.61
2	Foreign	1980.90
2	New	1948.64
2	Classics	1922.08
2	Travel	1909.58
2	Children	1897.08
2	Music	1622.06

sales_by_category_and_store 68

Output

#	Time	Action	Message
1	12:38:18	create view sales_by_category_and_store as(select s.store_id as store_id, c.name as category, sum(p.amount) as total_sales from payment p inne...	0 row(s) affected
2	12:40:10	select * from sales_by_category_and_store LIMIT 0, 1000	32 row(s) returned

2. 设计 1 个触发器，需要体现触发器生效。

（1） 执行新建触发器的语句，并截图 SQL 和执行结果：

首先新建表 customer_history，用于记录删除的客户信息，可以理解成一个简单的回收站。customer_id 为主键，delete_time 表示该客户被删除的时间，其他 first_name、last_name、email、create_date 等字段的含义同 customer 表。

每次删除客户时，触发器生效，将该客户信息插入 customer_history。需要注意的是，可能客户 id 在 customer_history 中已经存在，即存在“创建客户-删除客户-创建客户-删除客户”的情况。这里处理方式是：对于每个 customer_id，新的信息将直接覆盖掉旧的客户信息，采用 replace into 实现。

新建表、初始化触发器、验证生效的代码一并如下：

```
create table customer_history (
    customer_id SMALLINT UNSIGNED NOT NULL,
    first_name VARCHAR(45) NOT NULL,
    last_name VARCHAR(45) NOT NULL,
    email VARCHAR(50) DEFAULT NULL,
    create_date DATETIME NOT NULL,
    delete_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
    PRIMARY KEY  (customer_id)
)ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

DELIMITER ;;
create trigger customer_history_trigger before delete on customer for
each row begin
    replace into customer_history(customer_id, first_name, last_name,
email, create_date, delete_time) (
        select customer_id, first_name, last_name, email, create_date,
NOW() delete_time
        from customer
        where customer_id = old.customer_id
    );
end;;
DELIMITER ;

insert into customer(customer_id, store_id, first_name, last_name, email,
address_id, active)
        Values (1000, 1, "Weihao", "Liu", "124658@qq.com", 605, 1);
insert into customer(customer_id, store_id, first_name, last_name, email,
address_id, active)
        Values (1001, 1, "wh", "l", "124658@163.com", 604, 1);
insert into customer(customer_id, store_id, first_name, last_name, email,
address_id, active)
        Values (1002, 2, "fds", "hrd", "124658@gmail.com", 603, 1);
delete from customer where customer_id = 1000;
```

```
delete from customer where customer_id = 1001;
delete from customer where customer_id = 1002;
```

sys_config | sakila-data | sakila | staff_list | sales_by_store | sales_by_film_category - View | customer | sakila-schema | SQL File 8* | SQL File 9*

Limit to 1000 rows

```
create table customer_history (
    customer_id SMALLINT UNSIGNED NOT NULL,
    first_name VARCHAR(45) NOT NULL,
    last_name VARCHAR(45) NOT NULL,
    email VARCHAR(50) DEFAULT NULL,
    create_date DATETIME NOT NULL,
    delete_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    PRIMARY KEY  (customer_id)
)ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
DELIMITER ;;
create trigger customer_history_trigger before delete on customer for each row begin
    replace into customer_history(customer_id, first_name, last_name, email, create_date, delete_time) (
        select customer_id, first_name, last_name, email, create_date, NOW() delete_time
        from customer
        where customer_id = old.customer_id
    );
end;;
DELIMITER ;
insert into customer(customer_id, store_id, first_name, last_name, email, address_id, active)
        Values (1000, 1, "Weihao", "Liu", "124658@qq.com", 605, 1);
insert into customer(customer_id, store_id, first_name, last_name, email, address_id, active)
        Values (1001, 1, "wh", "l", "124658@163.com", 604, 1);
insert into customer(customer_id, store_id, first_name, last_name, email, address_id, active)
        Values (1002, 2, "fds", "hrd", "124658@gmail.com", 603, 1);
delete from customer where customer_id = 1000;
delete from customer where customer_id = 1001;
delete from customer where customer_id = 1002;
```

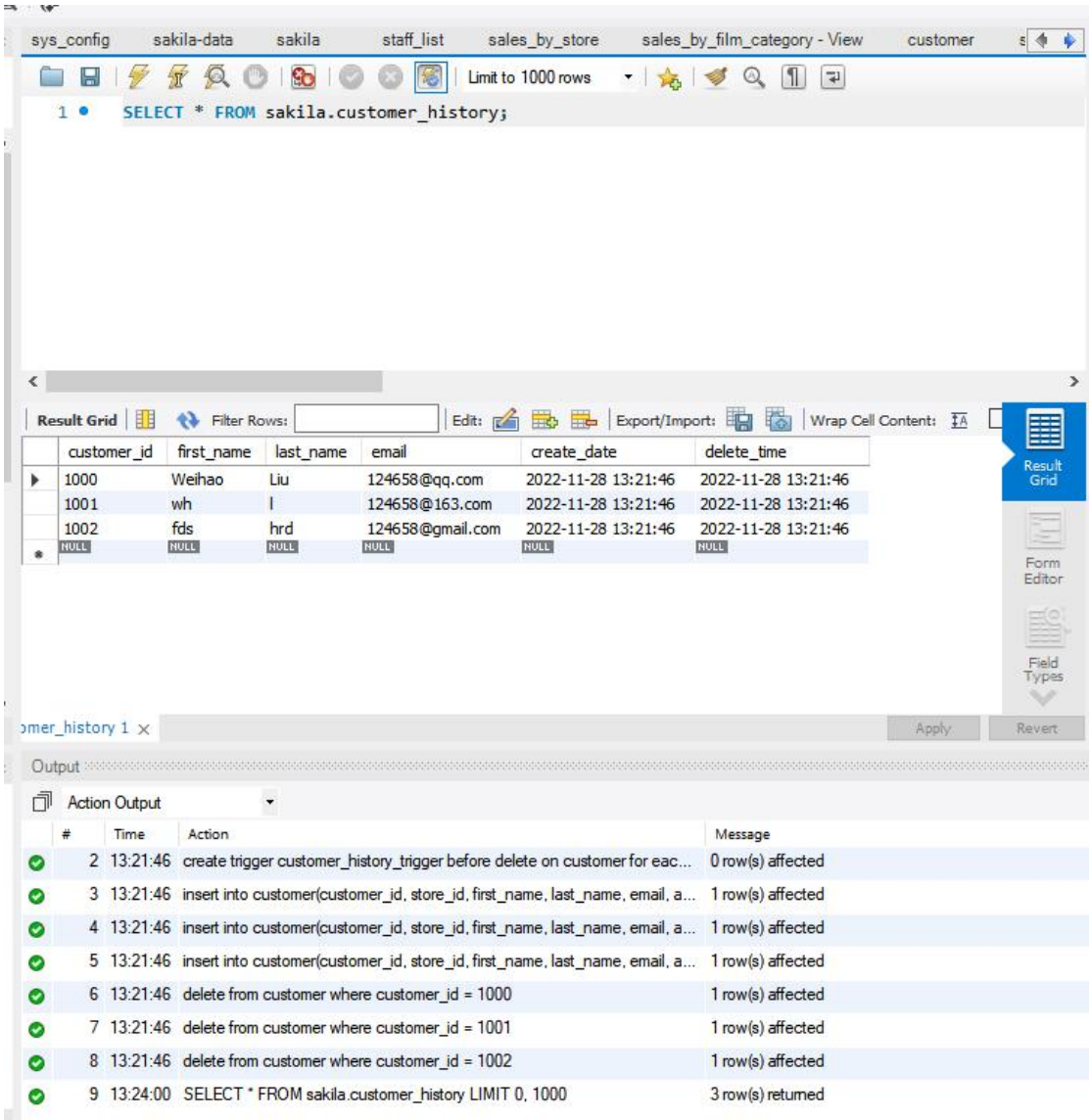
Output

Action Output

#	Time	Action	Message
1	13:21:46	create table customer_history (customer_id SMALLINT UNSIGNED NOT NULL, first_name VARCHAR(45) NOT NULL, last_name VARCHAR(45)...	0 row(s) affected
2	13:21:46	create trigger customer_history_trigger before delete on customer for each row begin replace into customer_history(customer_id, first_name, last_na...	0 row(s) affected
3	13:21:46	insert into customer(customer_id, store_id, first_name, last_name, email, address_id, active) Values (1000, 1, "Weihao", "Liu", "124658@qq.com", ...	1 row(s) affected
4	13:21:46	insert into customer(customer_id, store_id, first_name, last_name, email, address_id, active) Values (1001, 1, "wh", "l", "124658@163.com", 604, 1)	1 row(s) affected
5	13:21:46	insert into customer(customer_id, store_id, first_name, last_name, email, address_id, active) Values (1002, 2, "fds", "hrd", "124658@gmail.com", 60...	1 row(s) affected
6	13:21:46	delete from customer where customer_id = 1000	1 row(s) affected
7	13:21:46	delete from customer where customer_id = 1001	1 row(s) affected
8	13:21:46	delete from customer where customer_id = 1002	1 row(s) affected

（2） 验证触发器是否生效，截图验证过程：

查看表 customer_history，得：



可以发现，删除掉的数据都成功插入 customer_history 表中，表明触发器已生效。

四、 思考题

（这部分不是必做题，供有兴趣的同学思考）

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。你认为外键是否没有存在的必要？

我认为具体问题具体分析。阿里的数据库规模较大，如果使用外键和级联，一方面每次插入数据，就需要往外键对应的表查询，带来额外的时间开销，造成性能下降；另一方面查询需要获取锁，在高并发大流量的业务中会大大增加死锁风险，综合各种因素他们强制规定不得使用外键。

但我们做的工程和实验都是小规模的数据，暂时不用考虑死锁、性能、并发等问题，所以外键就随便用了。毕竟外键能将逻辑判断转移到数据库上，级联操作方便，又能减少程序代码量。