实验二报告

## 观察并回答问题

### 关于视图

1. sakila.mwb模型图中共有几个View？

sakila.mwb模型图中共有7个View分别为actor\_info、customer\_list、film\_list、nicer\_but\_slower\_film\_list、sales\_by\_film\_category、sales\_by\_store、staff\_list

1. 分析以下3个视图，回答以下问题：

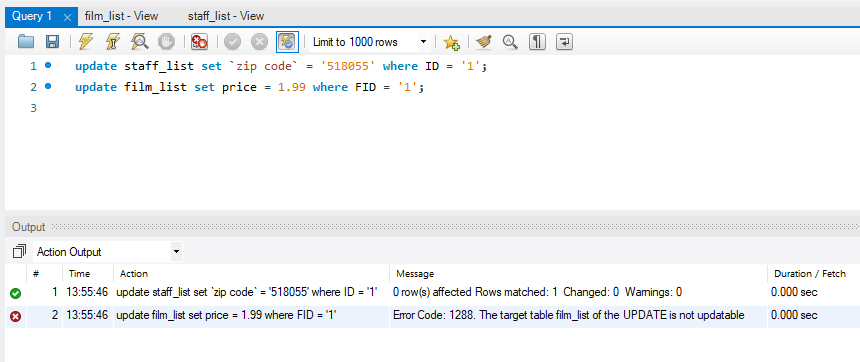
|  |  |  |
| --- | --- | --- |
| 视图名 | 关联表 | 作用 |
| actor\_info | film, actor, category, film\_actor, film\_category | 提供了所有演员的列表，包括了他们的id和姓名，以及出演的按类别划分的电影（以逗号分开）。 |
| film\_list | film, category, film\_category, actor, film\_actor | 提供了一个格式化的电影表视图，包括了它们的id、标题、描述、所属类别、价格、长度、等级，其中每个电影的演员列表以逗号分隔。 |
| sales\_by\_store | city, country, payment, rental, inventory, store, address, staff | 提供按商店细分的总销售额列表，包括商店位置、经理名称和总销售额。 |

1. 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？



如上图所示，第一条语句执行成功，但第二条报错，显示Error Code: 1288. The target table film\_list of the UPDATE is not updatable，表示film\_list这个视图不可被更新。

观察视图可知，staff\_list视图中，将staff、address、city、country表连接了起来，但update操作涉及的属性zip code并未被作为外键使用，仅在address表中出现，因此可以被update。而film\_list视图中，在获取整张视图时用到了group by语句，因此不能被update。

视图可以进行update的情况：

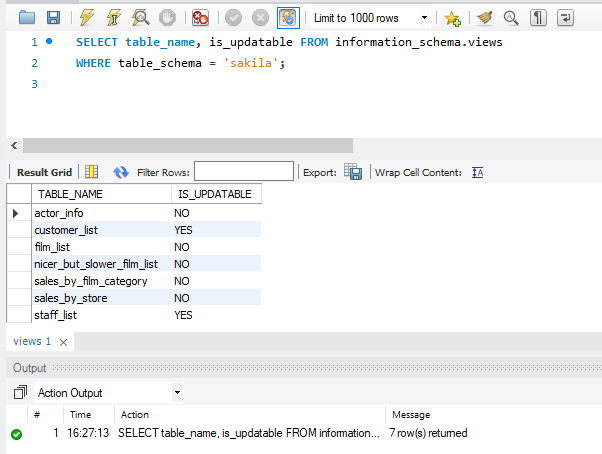
1. 视图的生成中不涉及分组和聚合函数。
2. 待update的结果列涉及的select语句中不包含TOP，GROUP BY，HAVING，UNION(除非视图是分区视图)或DISTINCT子句。
3. 待update的结果列涉及的select语句中没有任何使用非简单列表达式(使用函数、加法或减法运算符等)。
4. 若视图是基于多个表使用连接、并、交或差操作而导出的，那么对这个视图执行update操作时，每次只能影响其中的一个表。
5. 待update的结果列涉及的where子句不能包含任何引用了FROM子句的表的嵌套SELECT操作。
6. FROM子句至少引用一个表。select语句不能只包含非表格格式的表达式(即不是从表派生出的表达式)。
7. 待update的结果列不涉及子查询。
8. 没有违反基本表的约束。

只要上述条件中有一条不满足，则不可以进行update操作。

1. 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';



查询结果如上，其中actor\_info、film\_list、nicer\_but\_slower\_film\_list、sales\_by\_film\_category、sales\_by\_store视图不能被更新，而staff\_list、customer\_list视图可以被更新。

### 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？

触发器customer\_create\_date建在customer表上。该触发器实现的功能：在向custome表中插入数据之前，设置其属性create\_date为当前的日期和时间。

1. 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）

在表中新增数据，此处不用now()函数显示地设置create\_date：

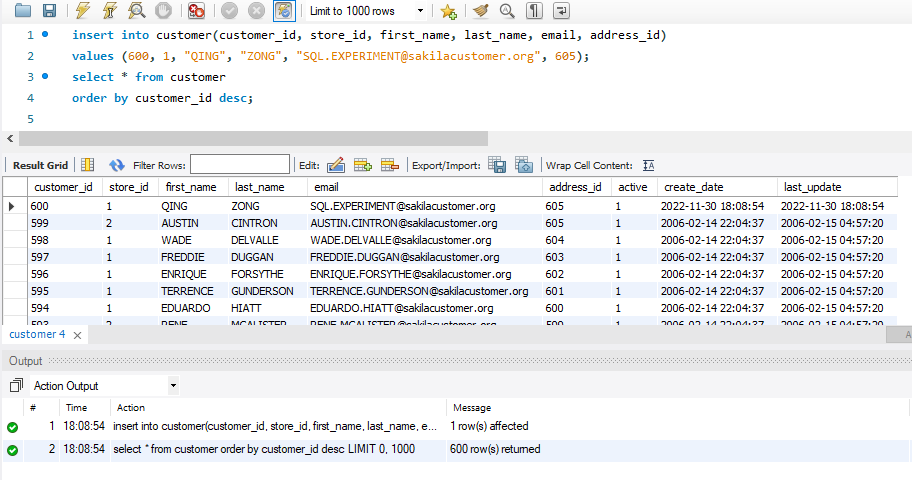
insert into customer(customer\_id, store\_id, first\_name, last\_name, email, address\_id)

values (600, 1, "QING", "ZONG", "SQL.EXPERIMENT@sakilacustomer.org", 605);

然后查看添加数据之后的表：

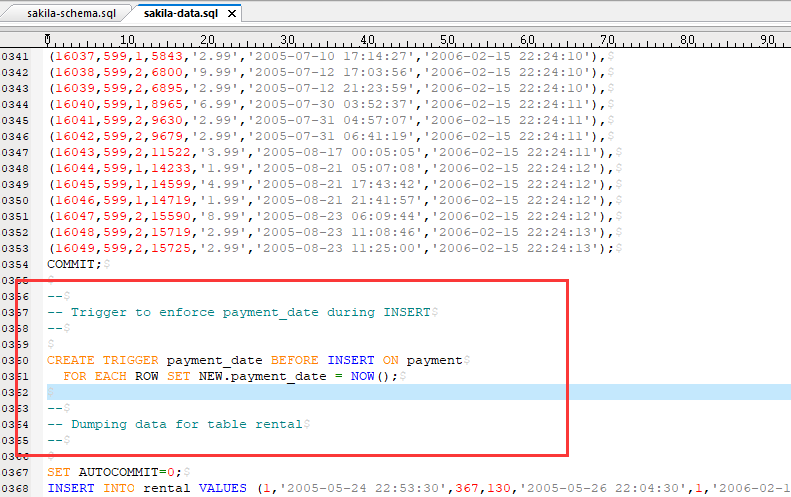
select \* from customer

order by customer\_id desc;



如上图所示，create\_time已经自动设置为了当前日期和时间，说明触发器已生效。

1. 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，这个触发器是用来将插入payment表的数据的payment\_date设置为当前时间。

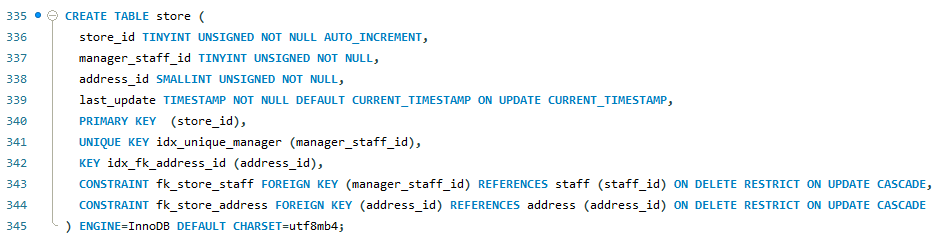
这段建立触发器的语句放在此处是为了先向payment表中插入原始数据，再创建触发器。这使得在sakila-data.sql中插入表中的数据，即0353行及其之前的原有数据不受影响，因为这些数据的payment\_date已知。

若移到sakila-schema.sql里去执行，那么在sakila-data.sql中向payment表中插入原有数据之前，这个触发器就已经被建立，那么插入的原始数据的payment\_date均会被设置为当前时间，而原有的payment\_date则会被覆盖，这是错误的。

所以，该触发器不可以移到sakila-schema.sql里去执行。

### 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）



store表的建立过程如上图所示。

|  |  |
| --- | --- |
| 约束类型 | 功能 |
| 主键（primary key） | 将该字段指定为主键，要求非空且唯一。如store表中的store\_id字段。 |
| 非空约束（not null） | 保证该字段不能为空。如store表中的manager\_staff\_id字段。 |
| 唯一约束（unique key） | 保证该字段若填写内容，那么内容不能重复。该字段可为空。如store表中的manager\_staff\_id字段。 |
| 外键约束（foreign key） | 将该字段指定为外键，添加或修改该字段时必须参照其对应的父表的相应字段。如store表中的manager\_staff\_id字段。 |

1. 图中第343行的ON DELETE RESTRICT和ON UPDATE CASCADE是什么意思？

第343行的ON DELETE RESTRICT表示当在父表（即外键的来源表，此处为staff）中删除对应记录时，首先检查该记录是否有对应外键（此处为manager\_staff\_id），如果有则不允许删除。

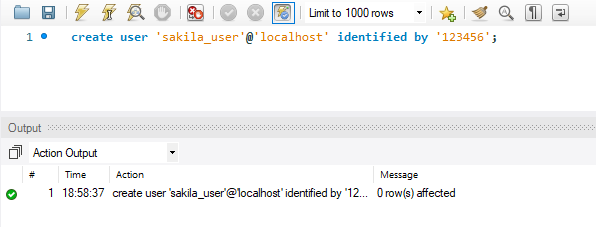
第343行的ON UPDATE CASCADE表示当在父表（即外键的来源表，此处为staff）中更新对应记录时，首先检查该记录是否有对应外键（此处为manager\_staff\_id），如果有则也要更新外键在子表（即包含外键的表，此处为store）中的记录。

## 创建新用户并分配权限

（截图语句和执行结果）

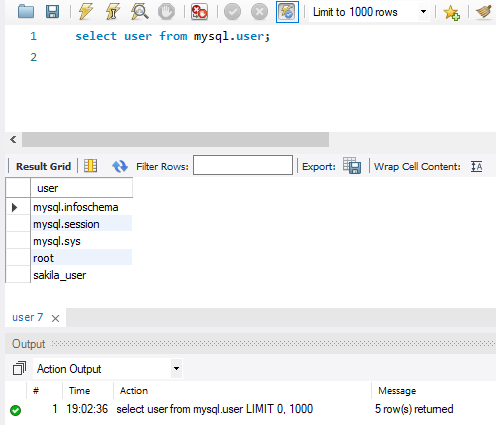
1. 执行命令新建sakila\_user用户（密码123456）；

create user 'sakila\_user'@'localhost' identified by '123456';



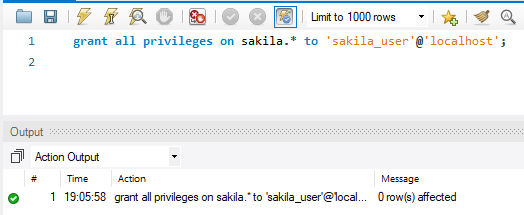
1. 执行命令查看当前已有用户；

select user from mysql.user;

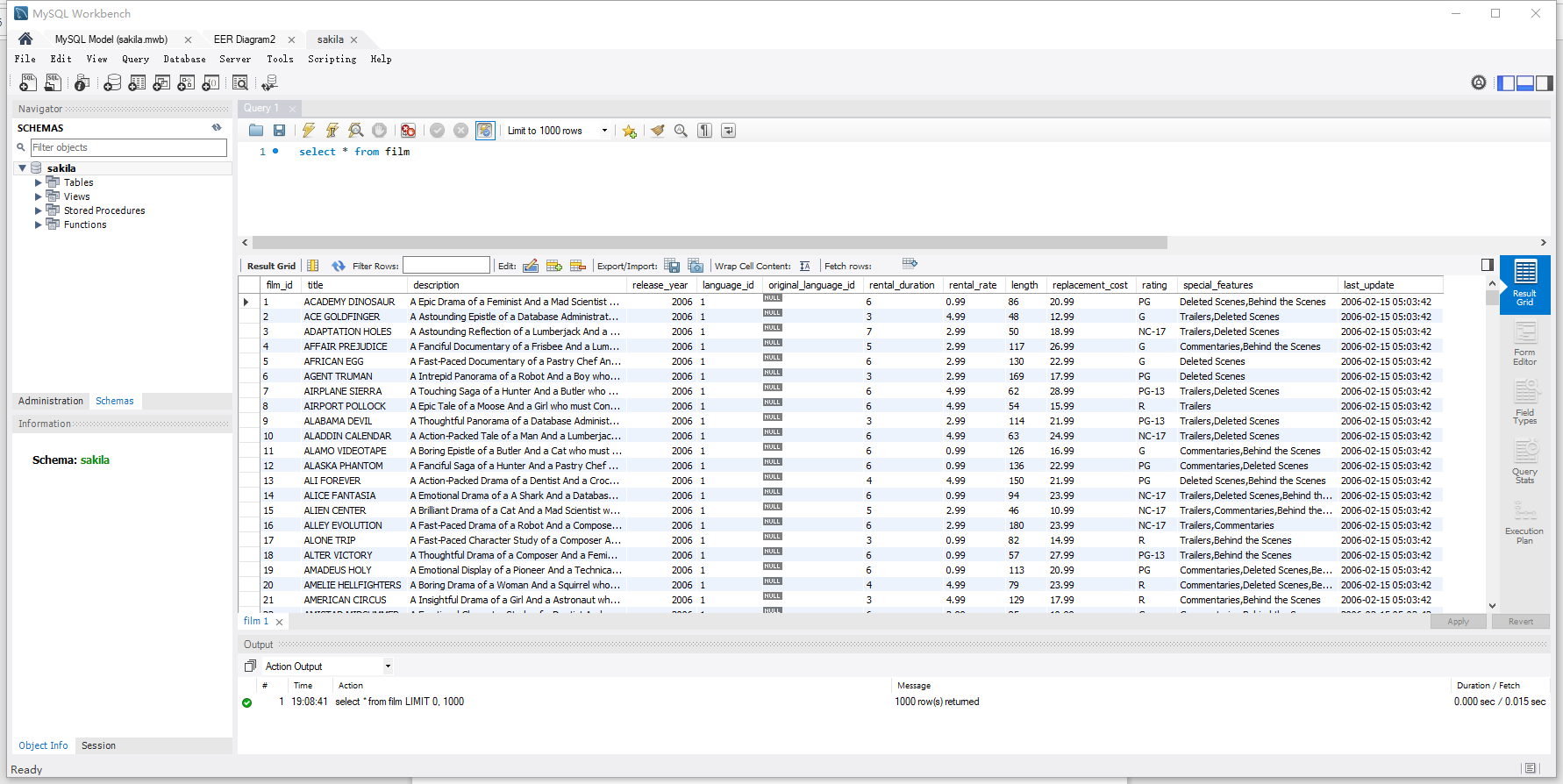


1. 执行命令把sakila数据库的访问权限赋予sakila\_user用户；

grant all privileges on sakila.\* to 'sakila\_user'@'localhost';



1. 切换到sakila\_user用户，执行select \* from film操作。



## 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；
2. 执行新建视图的语句，并截图SQL和执行结果：

设计一个顾客分布customer\_distribution的视图，统计各商店的顾客在每个国家的分布情况，按照store\_id升序后customer\_num降序排序。该视图可以体现每个商店的顾客主要来自的国家，从而帮助商店制定合适的发展战略。

create view customer\_distribution

as

select

store\_id,

c.country\_id,

c.country,

count(\*) as customer\_num

from customer

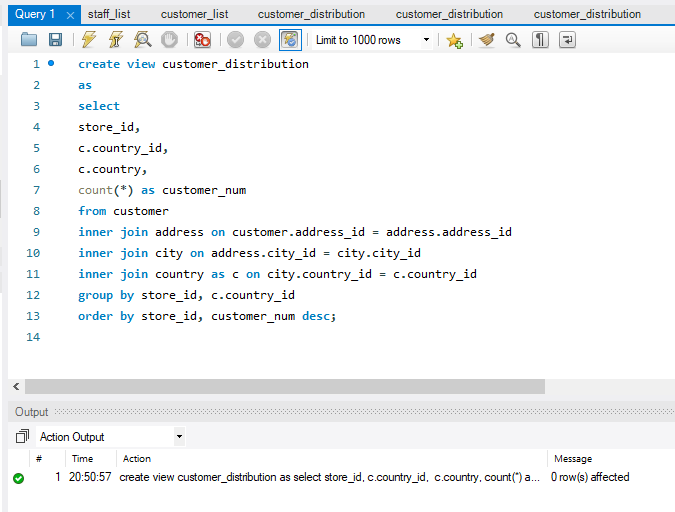
inner join address on customer.address\_id = address.address\_id

inner join city on address.city\_id = city.city\_id

inner join country as c on city.country\_id = c.country\_id

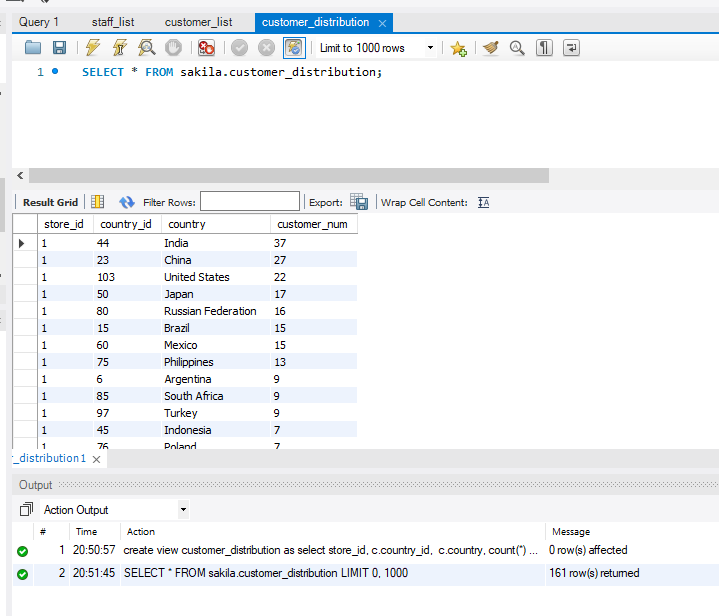
group by store\_id, c.country\_id

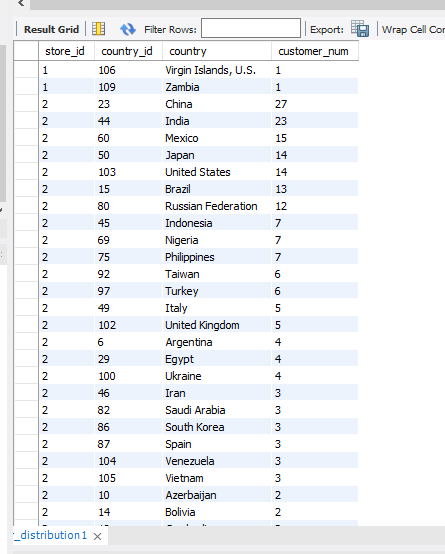
order by store\_id, customer\_num desc;



1. 执行select \* from [视图名]，截图执行结果：

SELECT \* FROM sakila.customer\_distribution;





1. 设计1个触发器，需要体现触发器生效。
2. 执行新建触发器的语句，并截图SQL和执行结果：

设计一个统计每个商店所拥有的顾客数量的触发器。

触发器add\_customer\_num：当向customer表中insert一条数据时，需要将对应store\_id的商店所拥有的顾客数量加一。

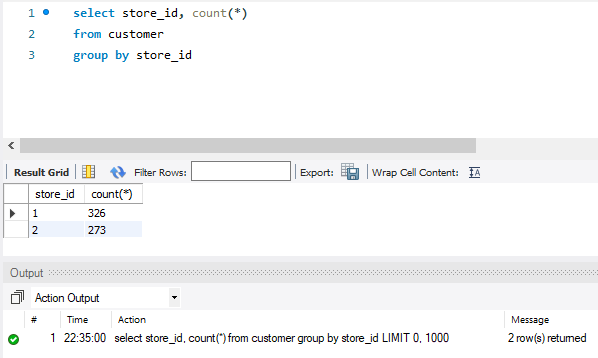
触发器minus\_customer\_num：当向customer表中delete一条数据时，需要将对应store\_id的商店所拥有的顾客数量减一。

首先，计算出当前每个商店所拥有的顾客数量：

select store\_id, count(\*)

from customer

group by store\_id



然后，创建一个customer\_num表用于存放每个商店所拥有的顾客数量信息，同时将当前的数量信息加入表中，然后定义触发器add\_customer\_num和触发器minus\_customer\_num实现上述功能：

CREATE TABLE customer\_num (

store\_id TINYINT UNSIGNED NOT NULL,

customer\_num SMALLINT UNSIGNED NOT NULL,

PRIMARY KEY (store\_id)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

INSERT INTO customer\_num VALUES (1, 326), (2, 273);

DELIMITER ;;

CREATE TRIGGER `add\_customer\_num` AFTER INSERT ON `customer` FOR EACH ROW BEGIN

update customer\_num

set customer\_num = customer\_num + 1

where store\_id = new.store\_id;

END;;

CREATE TRIGGER `minus\_customer\_num` AFTER DELETE ON `customer` FOR EACH ROW

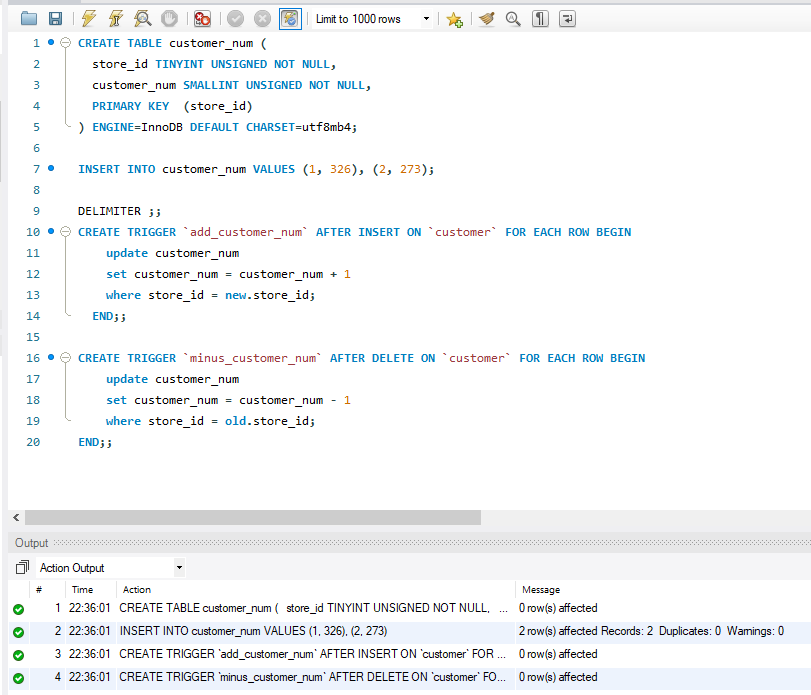
BEGIN

update customer\_num

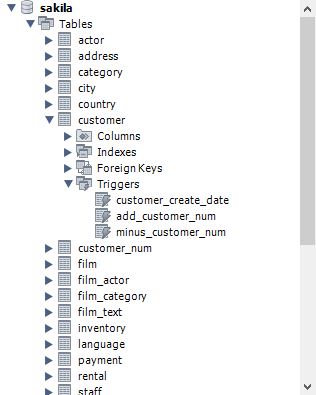
set customer\_num = customer\_num - 1

where store\_id = old.store\_id;

END;;



可看出，当前customer\_num表以及两个触发器均已创建：

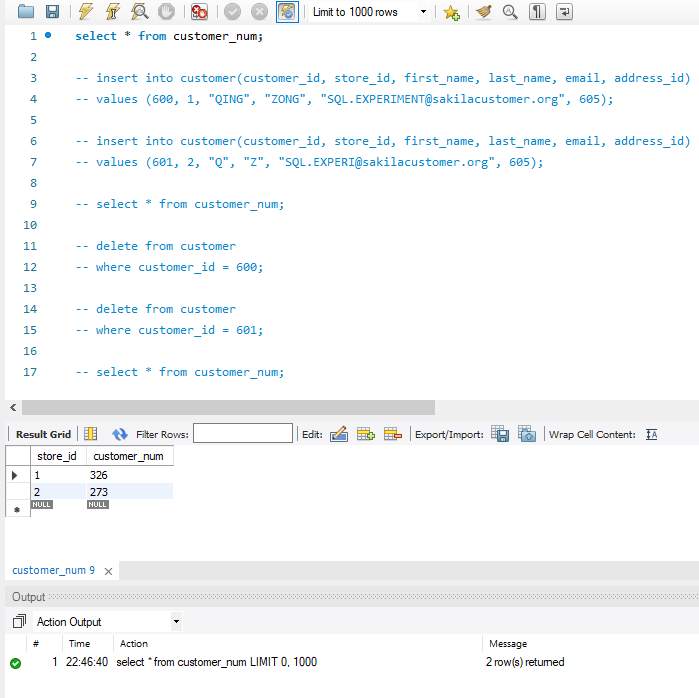


1. 验证触发器是否生效，截图验证过程：

首先，查看当前customer\_num表中的数据：

select \* from customer\_num;

如下图，符合预期：



然后向customer表中分别insert两条数据，随后查看当前的customer\_num表：

insert into customer(customer\_id, store\_id, first\_name, last\_name, email, address\_id)

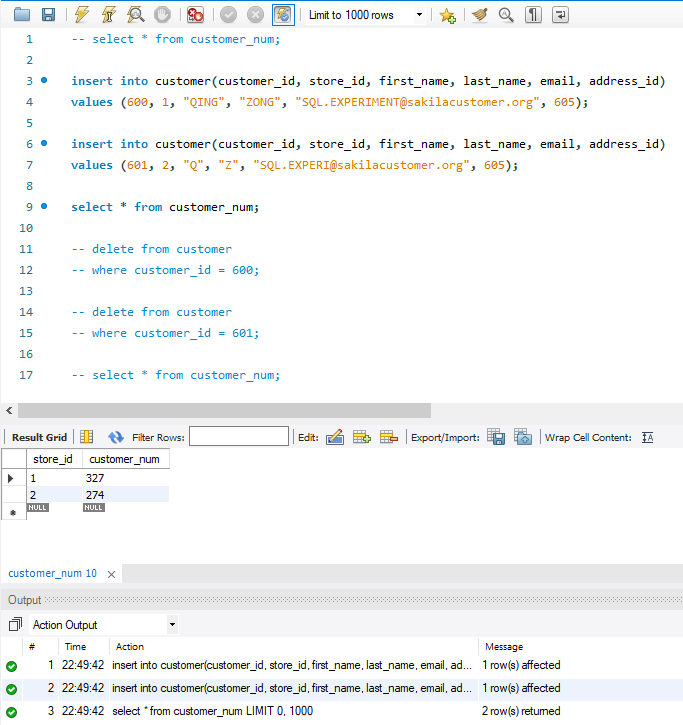
values (600, 1, "QING", "ZONG", "SQL.EXPERIMENT@sakilacustomer.org", 605);

insert into customer(customer\_id, store\_id, first\_name, last\_name, email, address\_id)

values (601, 2, "Q", "Z", "SQL.EXPERI@sakilacustomer.org", 605);

select \* from customer\_num;

如下图，两个商店所拥有的顾客数量各自加一，符合预期：



最后，将刚刚添加的两条数据删除，随后查看当前的customer\_num表：

delete from customer

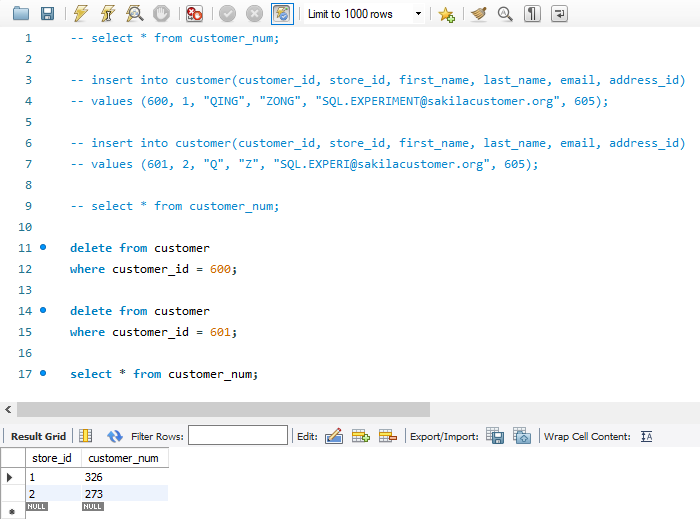
where customer\_id = 600;

delete from customer

where customer\_id = 601;

select \* from customer\_num;

如下图，两个商店所拥有的顾客数量均恢复为初始值，符合预期：



由此可验证触发器确实生效。

## 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？

这与阿里的实际业务相关，使用外键与级联需要对涉及到的表进行搜索，而阿里的数据库规模异常庞大，若对整张表进行搜索，那么将消耗大量的时间与资源，这会造成极大的浪费，也会使得阿里所提供的服务性能下降。并且，在查询表时需要上锁，而阿里的业务量很大，极易产生死锁。同时外键与级联会降低数据库的可扩展性，也会使得阿里的数据耦合性强，迁移维护较为困难，更可能导致重要数据被错误删除。

尽管如此，外键是有存在的必要的。当数据库规模较小，搜索整张表花费的时间和资源开销在可接受范围内时，使用外键与级联会大大简化对数据库的操作，减小代码量，更可以保证数据的完整性和一致性。