

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？

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

(2) 分析以下 3 个视图，回答以下问题：

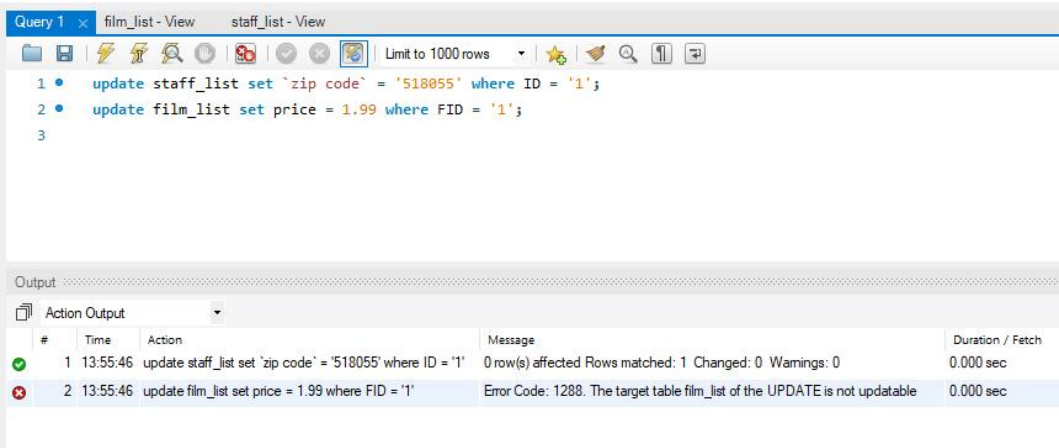
视图名	关联表	作用
actor_info	film, actor, category, film_actor, film_category	提供了所有演员的列表, 包括了他们的 id 和姓名, 以及出演的按类别划分的电影 (以逗号分开)。
film_list	film, category, film_category, actor, film_actor	提供了一个格式化的电影表视图, 包括了它们的 id、标题、描述、所属类别、价格、长度、等级, 其中每个电影的演员列表以逗号分隔。
sales_by_store	city, country, payment, rental, inventory, 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。

(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。

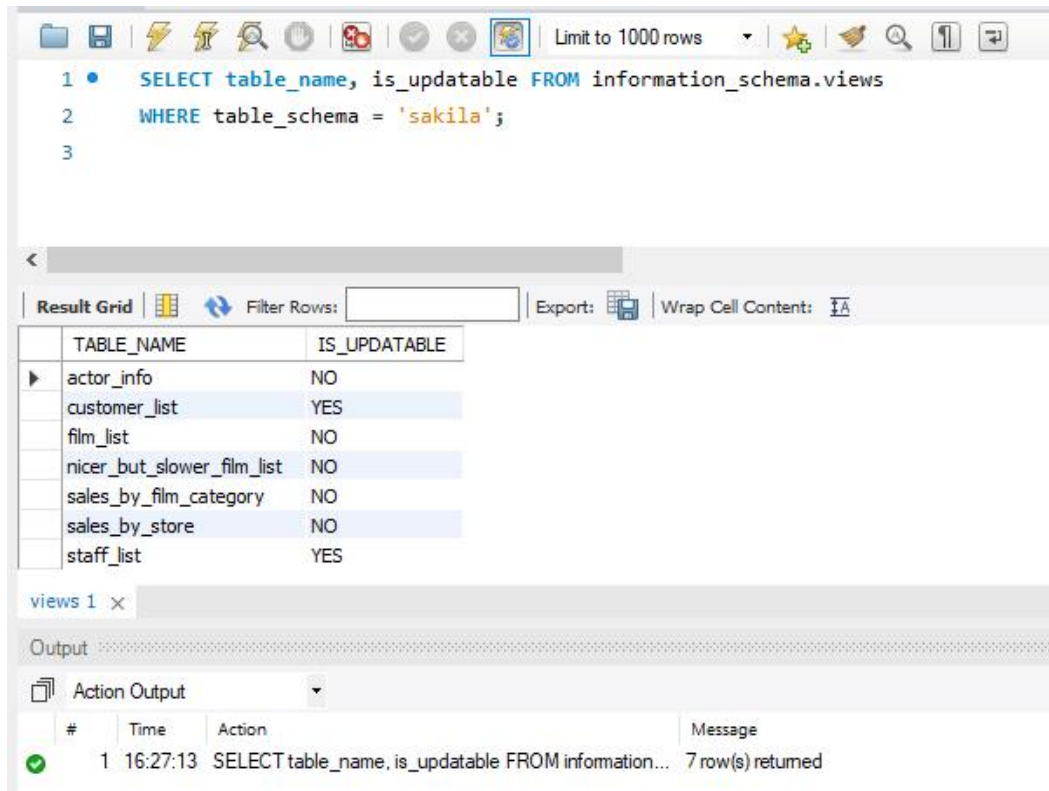
视图可以进行 update 的情况:

- 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。

(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。

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？

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

- (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）

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;
```

```

1 • insert into customer(customer_id, store_id, first_name, last_name, email, address_id)
2   values (600, 1, "QING", "ZONG", "SQL.EXPERIMENT@sakilacustomer.org", 605);
3 • select * from customer
4   order by customer_id desc;
5

```

customer_id	store_id	first_name	last_name	email	address_id	active	create_date	last_update
600	1	QING	ZONG	SQL.EXPERIMENT@sakilacustomer.org	605	1	2022-11-30 18:08:54	2022-11-30 18:08:54

```

# Time Action Message
1 18:08:54 insert into customer(customer_id, store_id, first_name, last_name, e... 1 row(s) affected
2 18:08:54 select * from customer order by customer_id desc LIMIT 0, 1000 600 row(s) returned

```

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。

- (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？

```

0341 (16037,599,1,5843,'2.99','2005-07-10 17:14:27','2006-02-15 22:24:10'),
0342 (16038,599,2,6800,'9.99','2005-07-12 17:03:56','2006-02-15 22:24:10'),
0343 (16039,599,2,6895,'2.99','2005-07-12 21:23:59','2006-02-15 22:24:10'),
0344 (16040,599,1,8965,'6.99','2005-07-30 03:52:37','2006-02-15 22:24:11'),
0345 (16041,599,2,9630,'2.99','2005-07-31 04:57:07','2006-02-15 22:24:11'),
0346 (16042,599,2,9679,'2.99','2005-07-31 06:41:19','2006-02-15 22:24:11'),
0347 (16043,599,2,11522,'3.99','2005-08-17 00:05:05','2006-02-15 22:24:11'),
0348 (16044,599,1,14233,'1.99','2005-08-21 05:07:08','2006-02-15 22:24:12'),
0349 (16045,599,1,14599,'4.99','2005-08-21 17:43:42','2006-02-15 22:24:12'),
0350 (16046,599,1,14719,'1.99','2005-08-21 21:41:57','2006-02-15 22:24:12'),
0351 (16047,599,2,15590,'8.99','2005-08-23 06:09:44','2006-02-15 22:24:12'),
0352 (16048,599,2,15719,'2.99','2005-08-23 11:08:46','2006-02-15 22:24:13'),
0353 (16049,599,2,15725,'2.99','2005-08-23 11:25:00','2006-02-15 22:24:13');
0354 COMMIT;
0355
0356 --
0357 -- Trigger to enforce payment_date during INSERT
0358 --
0359
0360 CREATE TRIGGER payment_date BEFORE INSERT ON payment
0361   FOR EACH ROW SET NEW.payment_date = NOW();
0362
0363 --
0364 -- Dumping data for table rental
0365 --
0366
0367 SET AUTOCOMMIT=0;
0368 INSERT INTO rental VALUES (1,'2005-05-24 22:53:30',367,130,'2005-05-26 22:04:30',1,'2006-02-1

```

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。

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）

```
335 CREATE TABLE store (  
336     store_id TINYINT UNSIGNED NOT NULL AUTO_INCREMENT,  
337     manager_staff_id TINYINT UNSIGNED NOT NULL,  
338     address_id SMALLINT UNSIGNED NOT NULL,  
339     last_update TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,  
340     PRIMARY KEY (store_id),  
341     UNIQUE KEY idx_unique_manager (manager_staff_id),  
342     KEY idx_fk_address_id (address_id),  
343     CONSTRAINT fk_store_staff FOREIGN KEY (manager_staff_id) REFERENCES staff (staff_id) ON DELETE RESTRICT ON UPDATE CASCADE,  
344     CONSTRAINT fk_store_address FOREIGN KEY (address_id) REFERENCES address (address_id) ON DELETE RESTRICT ON UPDATE CASCADE  
345 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

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。

(2) 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。

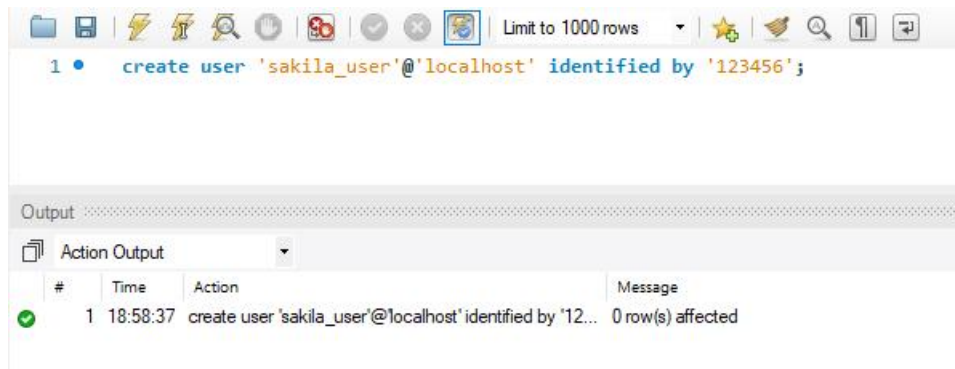
第 343 行的 ON UPDATE CASCADE 表示当在父表（即外键的来源表，此处为 staff）中更新对应记录时，首先检查该记录是否有对应外键（此处为 manager_staff_id），如果有则也要更新外键在子表（即包含外键的表，此处为 store）中的记录。

二、创建新用户并分配权限

（截图语句和执行结果）

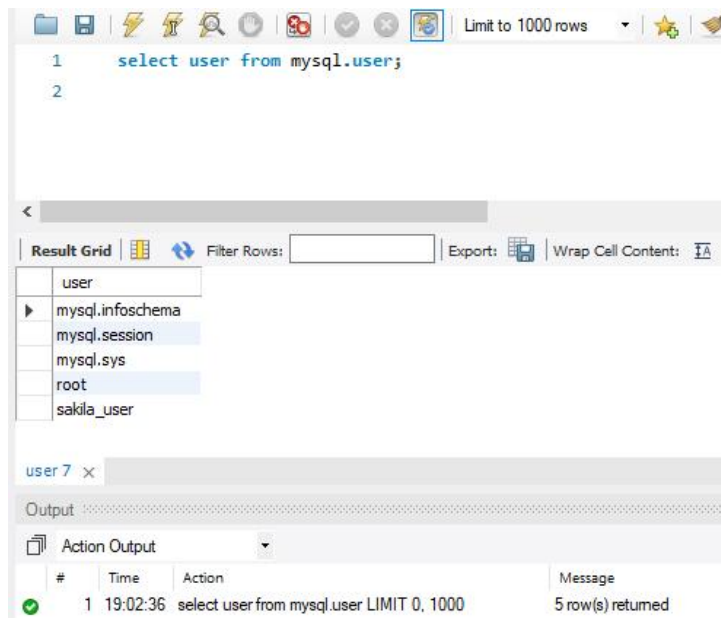
(1) 执行命令新建 sakila_user 用户（密码 123456）；

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



(2) 执行命令查看当前已有用户；

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

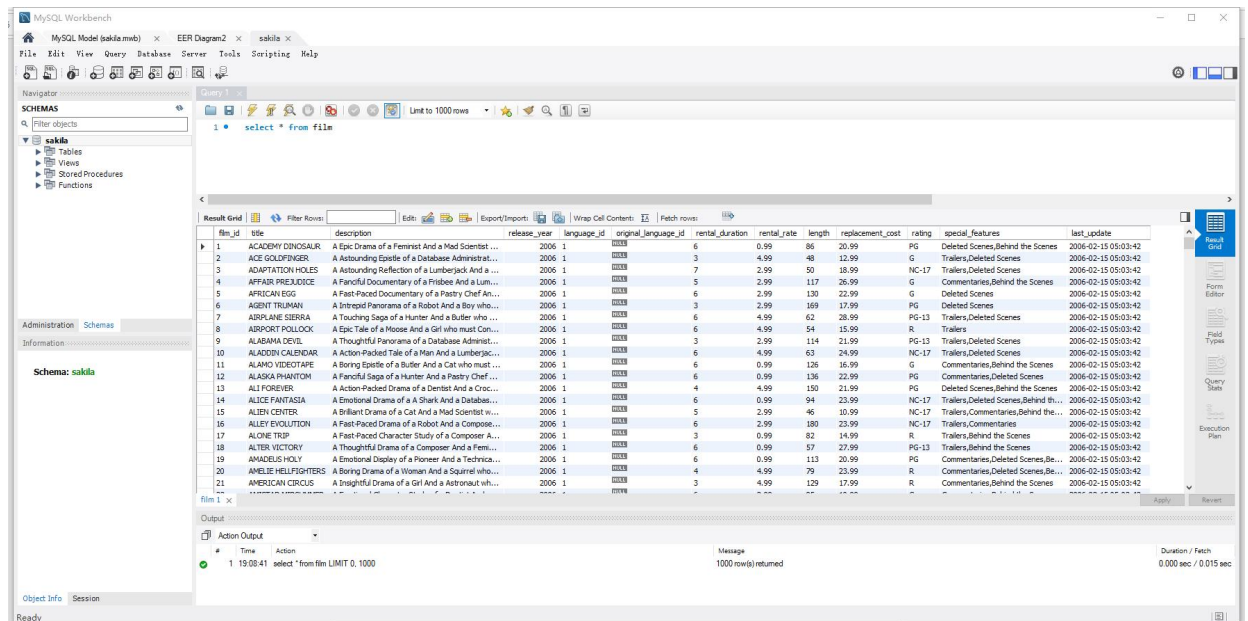


(3) 执行命令把 sakila 数据库的访问权限赋予 sakila_user 用户；

```
grant all privileges on sakila.* to 'sakila_user'@'localhost';
```



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



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：

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;
```

Query 1 x staff_list customer_list customer_distribution customer_distribution customer_distribution

Limit to 1000 rows

```
1 • create view customer_distribution
2   as
3   select
4   store_id,
5   c.country_id,
6   c.country,
7   count(*) as customer_num
8   from customer
9   inner join address on customer.address_id = address.address_id
10  inner join city on address.city_id = city.city_id
11  inner join country as c on city.country_id = c.country_id
12  group by store_id, c.country_id
13  order by store_id, customer_num desc;
14
```

Output

Action Output

#	Time	Action	Message
✓ 1	20:50:57	create view customer_distribution as select store_id, c.country_id, c.country, count(*) a...	0 row(s) affected

(2) 执行 select * from [视图名], 截图执行结果:

```
SELECT * FROM sakila.customer_distribution;
```


Query 1 staff_list customer_list **customer_distribution** x

Limit to 1000 rows

1 • `SELECT * FROM sakila.customer_distribution;`

Result Grid Filter Rows: Export: Wrap Cell Content: `⌂`

	store_id	country_id	country	customer_num
▶	1	44	India	37
	1	23	China	27
	1	103	United States	22
	1	50	Japan	17
	1	80	Russian Federation	16
	1	15	Brazil	15
	1	60	Mexico	15
	1	75	Philippines	13
	1	6	Argentina	9
	1	85	South Africa	9
	1	97	Turkey	9
	1	45	Indonesia	7
	1	76	Poland	7

Output

Action Output

#	Time	Action	Message
✓ 1	20:50:57	create view customer_distribution as select store_id, c.country_id, c.country, count(*) ...	0 row(s) affected
✓ 2	20:51:45	SELECT * FROM sakila.customer_distribution LIMIT 0, 1000	161 row(s) returned

Result Grid Filter Rows: Export: Wrap Cell Cor

	store_id	country_id	country	customer_num
	1	106	Virgin Islands, U.S.	1
	1	109	Zambia	1
	2	23	China	27
	2	44	India	23
	2	60	Mexico	15
	2	50	Japan	14
	2	103	United States	14
	2	15	Brazil	13
	2	80	Russian Federation	12
	2	45	Indonesia	7
	2	69	Nigeria	7
	2	75	Philippines	7
	2	92	Taiwan	6
	2	97	Turkey	6
	2	49	Italy	5
	2	102	United Kingdom	5
	2	6	Argentina	4
	2	29	Egypt	4
	2	100	Ukraine	4
	2	46	Iran	3
	2	82	Saudi Arabia	3
	2	86	South Korea	3
	2	87	Spain	3
	2	104	Venezuela	3
	2	105	Vietnam	3
	2	10	Azerbaijan	2
	2	14	Bolivia	2

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：

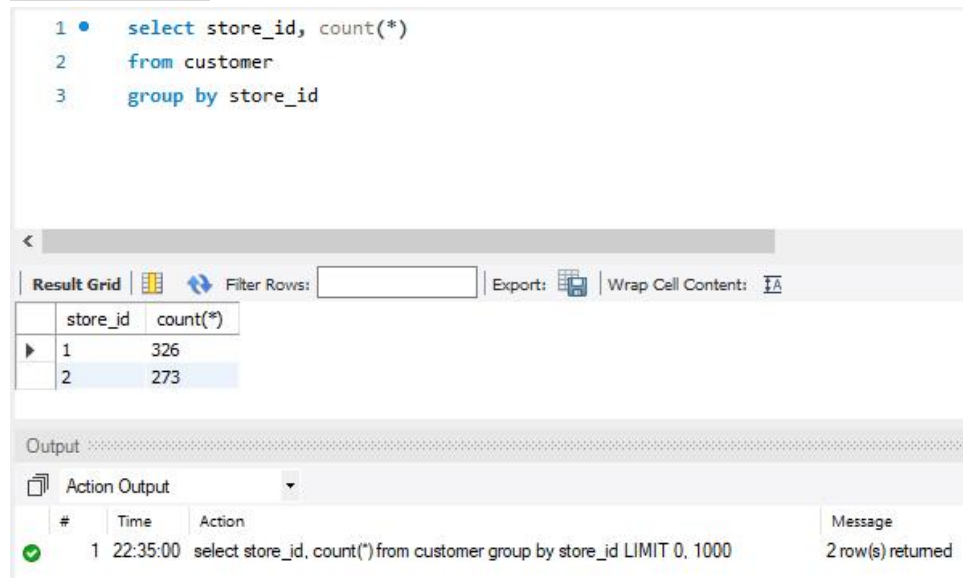
设计一个统计每个商店所拥有的顾客数量的触发器。

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 ;;
```

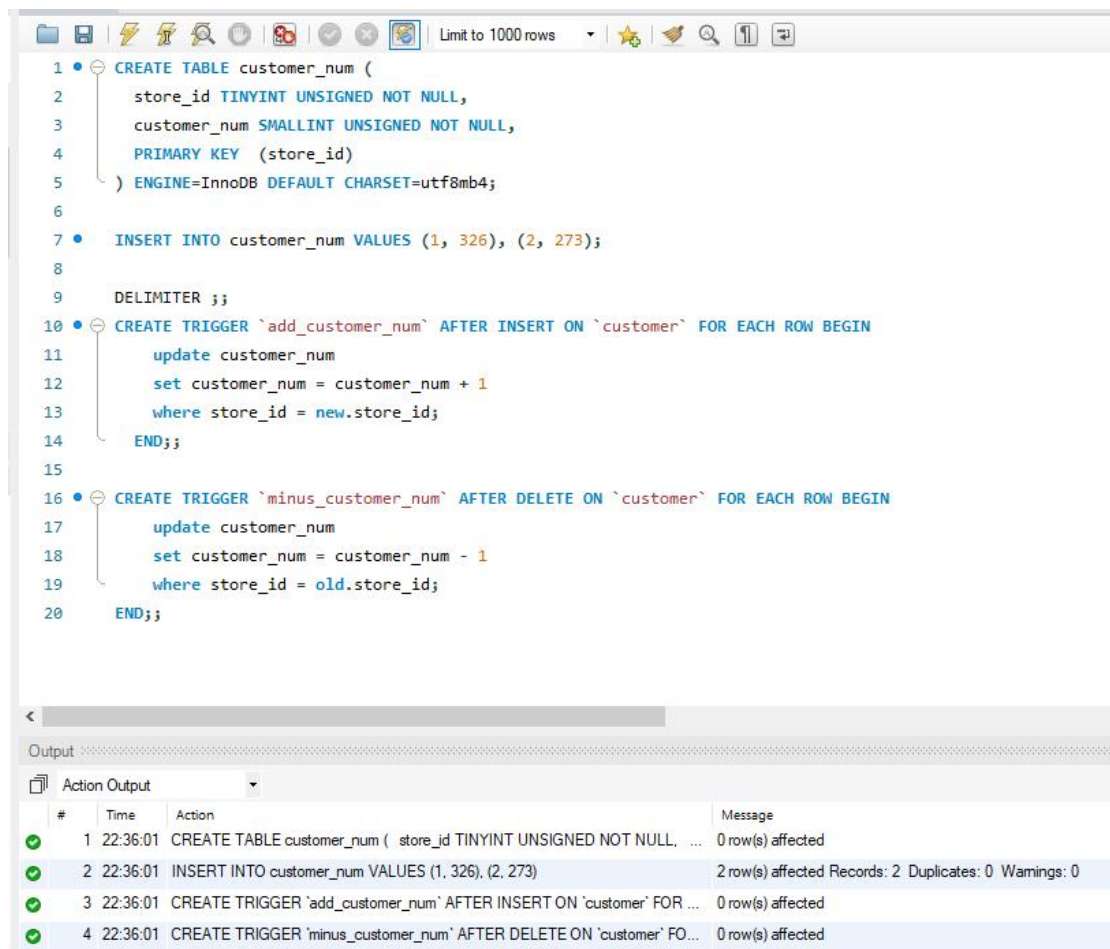
```
CREATE TRIGGER `add_customer_num` AFTER INSERT ON `customer` FOR EACH ROW BEGIN  
  update customer_num  
  set customer_num = customer_num + 1
```

```

        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;;

```



The screenshot shows a MySQL IDE window with a toolbar at the top. The main area contains SQL code for creating a table, inserting data, and creating two triggers. The bottom panel shows the 'Output' window with 'Action Output' selected, displaying a log of the executed statements and their results.

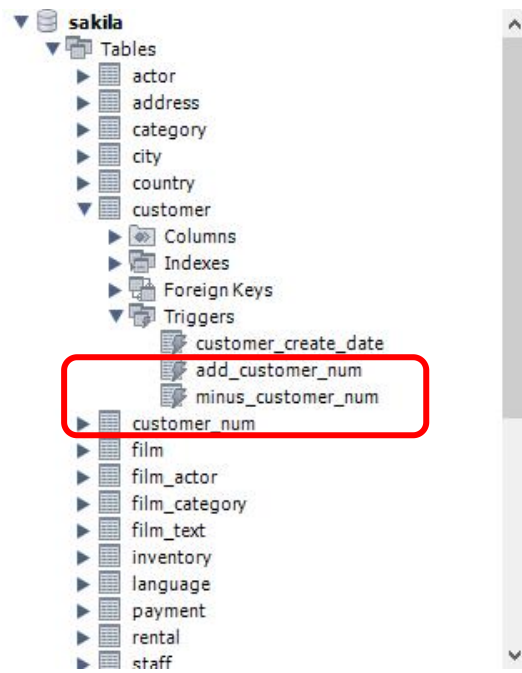
```

1 • CREATE TABLE customer_num (
2     store_id TINYINT UNSIGNED NOT NULL,
3     customer_num SMALLINT UNSIGNED NOT NULL,
4     PRIMARY KEY (store_id)
5 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
6
7 • INSERT INTO customer_num VALUES (1, 326), (2, 273);
8
9 DELIMITER ;;
10 • CREATE TRIGGER `add_customer_num` AFTER INSERT ON `customer` FOR EACH ROW BEGIN
11     update customer_num
12     set customer_num = customer_num + 1
13     where store_id = new.store_id;
14 END;;
15
16 • CREATE TRIGGER `minus_customer_num` AFTER DELETE ON `customer` FOR EACH ROW BEGIN
17     update customer_num
18     set customer_num = customer_num - 1
19     where store_id = old.store_id;
20 END;;

```

#	Time	Action	Message
1	22:36:01	CREATE TABLE customer_num (store_id TINYINT UNSIGNED NOT NULL, ...	0 row(s) affected
2	22:36:01	INSERT INTO customer_num VALUES (1, 326), (2, 273)	2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0
3	22:36:01	CREATE TRIGGER `add_customer_num` AFTER INSERT ON `customer` FOR ...	0 row(s) affected
4	22:36:01	CREATE TRIGGER `minus_customer_num` AFTER DELETE ON `customer` FO...	0 row(s) affected

可看出，当前 `customer_num` 表以及两个触发器均已创建：



(2) 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：

The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, execution, and a 'Limit to 1000 rows' dropdown. The script editor contains the following SQL code:

```
1 select * from customer_num;
2
3 -- insert into customer(customer_id, store_id, first_name, last_name, email, address_id)
4 -- values (600, 1, "QING", "ZONG", "SQL.EXPERIMENT@sakilacustomer.org", 605);
5
6 -- insert into customer(customer_id, store_id, first_name, last_name, email, address_id)
7 -- values (601, 2, "Q", "Z", "SQL.EXPERI@sakilacustomer.org", 605);
8
9 -- select * from customer_num;
10
11 -- delete from customer
12 -- where customer_id = 600;
13
14 -- delete from customer
15 -- where customer_id = 601;
16
17 -- select * from customer_num;
```

Below the script editor is the 'Result Grid' section, which displays a table with two columns: 'store_id' and 'customer_num'. The table contains two rows of data:

store_id	customer_num
1	326
2	273

Below the result grid is the 'Output' section, which shows the 'Action Output' log. The log contains one entry:

#	Time	Action	Message
1	22:46:40	select * from customer_num LIMIT 0, 1000	2 row(s) returned

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：

```
1  -- select * from customer_num;
2
3  • insert into customer(customer_id, store_id, first_name, last_name, email, address_id)
4  values (600, 1, "QING", "ZONG", "SQL.EXPERIMENT@sakilacustomer.org", 605);
5
6  • insert into customer(customer_id, store_id, first_name, last_name, email, address_id)
7  values (601, 2, "Q", "Z", "SQL.EXPERI@sakilacustomer.org", 605);
8
9  • select * from customer_num;
10
11 -- delete from customer
12 -- where customer_id = 600;
13
14 -- delete from customer
15 -- where customer_id = 601;
16
17 -- select * from customer_num;
```

Result Grid

	store_id	customer_num
▶	1	327
	2	274
✱	NULL	NULL

customer_num 10 x

Output

Action Output

#	Time	Action	Message
✓ 1	22:49:42	insert into customer(customer_id, store_id, first_name, last_name, email, ad...	1 row(s) affected
✓ 2	22:49:42	insert into customer(customer_id, store_id, first_name, last_name, email, ad...	1 row(s) affected
✓ 3	22:49:42	select * from customer_num LIMIT 0, 1000	2 row(s) returned

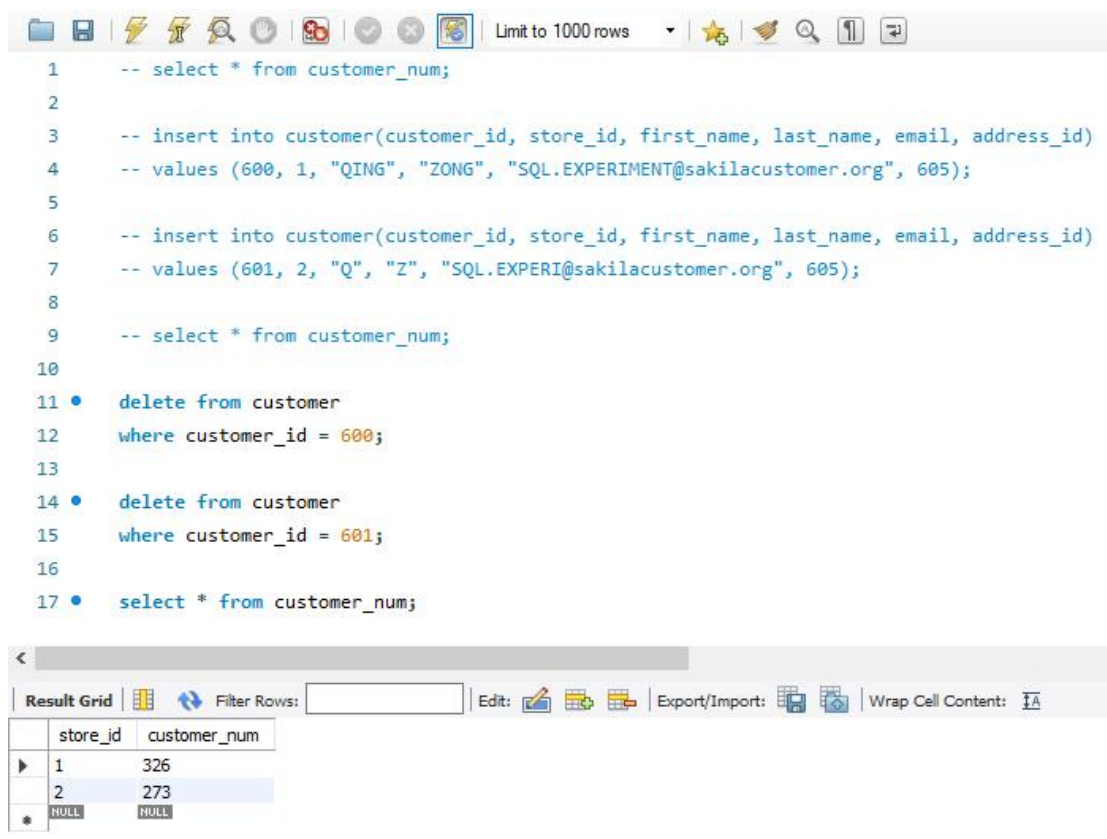
最后，将刚刚添加的两条数据删除，随后查看当前的 customer_num 表：

```
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：



The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, execution, and a 'Limit to 1000 rows' dropdown. The script area contains the following SQL code:

```
1  -- select * from customer_num;
2
3  -- insert into customer(customer_id, store_id, first_name, last_name, email, address_id)
4  -- values (600, 1, "QING", "ZONG", "SQL.EXPERIMENT@sakilacustomer.org", 605);
5
6  -- insert into customer(customer_id, store_id, first_name, last_name, email, address_id)
7  -- values (601, 2, "Q", "Z", "SQL.EXPERI@sakilacustomer.org", 605);
8
9  -- select * from customer_num;
10
11 • delete from customer
12   where customer_id = 600;
13
14 • delete from customer
15   where customer_id = 601;
16
17 • select * from customer_num;
```

Below the script, the 'Result Grid' shows the output of the final query. It has columns 'store_id' and 'customer_num'.

store_id	customer_num
1	326
2	273
NULL	NULL

由此可验证触发器确实生效。

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？

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。