

深 圳 大 学

实 验 报 告

课程名称： 数据库系统

实验序号： 实验 3

实验名称： DMBS应用

学 号： 2022155028

姓 名： 黄亮铭

实验完成日期： 2024 年 11 月 06 日

观察并回答问题

关于视图-回答问题:

(1) 分析以下 3 个视图，回答以下问题:

```
SELECT DBMS_METADATA.GET_DDL('VIEW', 'ACTOR_INFO', 'SAKILA') DDL FROM DUAL;  
SELECT DBMS_METADATA.GET_DDL('VIEW', 'FILM_LIST', 'SAKILA') DDL FROM DUAL;  
SELECT DBMS_METADATA.GET_DDL('VIEW',  
'SALES_BY_FILM_CATEGORY', 'SAKILA') DDL FROM DUAL;
```

视图名	关联表	作用
actor_info	ACTOR	快速访问演员信息
film_list	FILM	方便展示电影信息
sales_by_film_category	FILM_CATEGORY	便于分析电影票房

(2) 创建视图分别执行以下 3 句 SQL 语句:

创建视图:

```
CREATE OR REPLACE VIEW sakila.customer_simple_view AS  
SELECT customer_id,  
first_name,  
last_name,  
email,  
address_id,  
active,  
create_date,  
last_update  
FROM customer;
```

执行 SQL 语句:

```
(1)update sakila.customer_simple_view set email =  
'newemail@example.com' where customer_id = 1;  
(2)update sakila.staff_list set "zip code" = '518055' where  
ID = '1';  
(3)update sakila.film_list set price = 1.99 where FID = '1';  
6
```

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？MYSQL 和崖山数据库有什么区别

创建视图的命令执行结果如下图所示。



图 1：创建视图

三条 SQL 语句执行结果如图 2 所示。由执行结果可以得出，只有第一条 SQL 语句执行成功，第二条和第三条 SQL 语句执行失败，发生了不同的错误。

第二条 SQL 语句错误的原因是尝试更新了非键保持的表；第三条 SQL 语句错误的原因是视图上的数据操作不合法。这两种错误的可能原因为：1) 视图包含了聚合函数、group by 子句、distinct 关键字和子查询等；2) 视图没有直接映射到一个具体的表，并且更新操作影响了多个表。

通过上述分析，我认为视图在满足下列条件的前提下，可以进行 update 操作。如果不满足下列条件，则不能对视图进行 update 操作。需要满足的条件：1) 创建的视图不包含聚合函数、group by 子句、distinct 关键字和子查询等；2) 视图是基于多个表生成时，一次 update 操作只能影响一个表；3) 创建视图时，select 语句选择的列没有派生列。

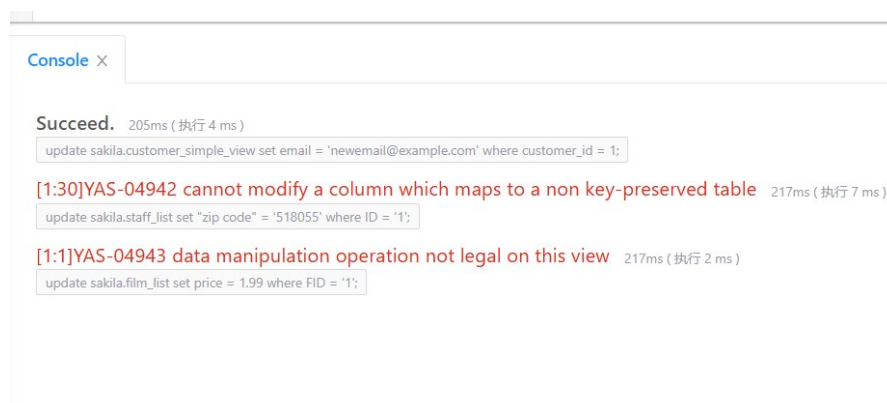


图 2：SQL 执行结果

关于触发器-回答问题:

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

触发器 `rental_date` 建立在表 `RENTAL` 上。这个触发器实现了如下功能: 当有新记录插入表中时, 触发器会自动将新记录的 `RENTAL_DATE` 字段设置为当前系统的日期和时间。

执行下表格中的 SQL 语句。

```
INSERT INTO
  SAKILA.RENTAL (
    RENTAL_ID,
    RENTAL_DATE,
    INVENTORY_ID,
    CUSTOMER_ID,
    RETURN_DATE,
    STAFF_ID,
    LAST_UPDATE
  )
VALUES
  (
    432802,
    '2024-10-20 00:02:21',
    1824,
    399,
    NULL,
    1,
    '2024-10-21 21:30:53'
  );
```

执行结果如下图所示。

```
1 INSERT INTO
2 SAKILA.RENTAL (
3 RENTAL_ID,
4 RENTAL_DATE,
5 INVENTORY_ID,
6 CUSTOMER_ID,
7 RETURN_DATE,
8 STAFF_ID,
9 LAST_UPDATE
10 )
11 VALUES
12 (
13 432802,
14 '2024-10-20 00:02:21',
15 1824,
16 399,
17 NULL,
18 1,
19 '2024-10-21 21:30:53'
20 );
```

Console X

Succeed. 233ms (执行 3 ms)

INSERT INTO SAKILA.RENTAL (RENTAL_ID, RENTAL_DATE, INVENTORY_ID, CUSTOMER_ID, RETURN...

图 3: SQL 执行结果

然后，我们通过 RENTAL_ID 字段查找上面插入的数据，如果触发器生效，则 RENTAL_DATE 字段会被修改为 '2024-10-30 HH:MM:SS'。如果触发器不生效，则 RENTAL_DATE 字段不会被修改。

执行如下表的查询语句。

```
SELECT * FROM SAKILA.RENTAL WHERE RENTAL_ID = 432802;
```

执行结果如下图所示。发现 RENTAL_DATE 字段被修改，说明触发器生效了。

RENTAL_ID	RENTAL_DATE	INVENTORY_ID	CUSTOMER_ID	RETURN_DATE	STAFF_ID	LAST_UPDATE
432802	2024-10-30 12:26:40	1824	399		1	2024-10-21 21:30:53

图 4: 查询结果

- (2) 触发器 del_film 建在哪个表上？这个触发器实现什么功能？在这个表上删除一条记录，验证一下触发器是否生效。（截图语句和执行结果）

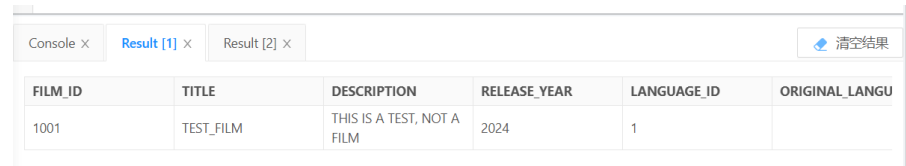
触发器 del_film 建立在表 FILM 上。这个触发器实现如下功能：当表 FILM 删除一条记录时，表 FILE_TEXT 会同步删除 FILE_TEXT.FILE_ID = OLD.FILE_ID 的记录。

首先执行如下插入语句和查询语句。

```
INSERT INTO SAKILA.FILM
(FILM_ID, TITLE, DESCRIPTION, RELEASE_YEAR, LANGUAGE_ID,
ORIGINAL_LANGUAGE_ID, RENTAL_DURATION, RENTAL_RATE, LENGTH,
REPLACEMENT_COST, RATING, SPECIAL_FEATURES, LAST_UPDATE)
VALUES
(1001, 'TEST_FILM', 'THIS IS A TEST, NOT A FILM', 2024, 1, NULL, 1,
9.99, 120, 29.99, NULL, NULL, SYSDATE);

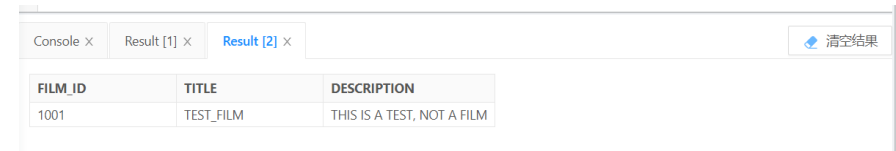
SELECT * FROM SAKILA.FILM WHERE FILM_ID = 1001;
SELECT * FROM SAKILA.FILM_TEXT WHERE FILM_ID = 1001;
```

查询结果如下图所示，说明数据被成功插入，FILM 和 FILM_TEXT 中均存在 FILM_ID=1001 的数据记录，其中 FILM_TEXT 中的记录为触发器自动插入。



FILM_ID	TITLE	DESCRIPTION	RELEASE_YEAR	LANGUAGE_ID	ORIGINAL_LANGU
1001	TEST_FILM	THIS IS A TEST, NOT A FILM	2024	1	

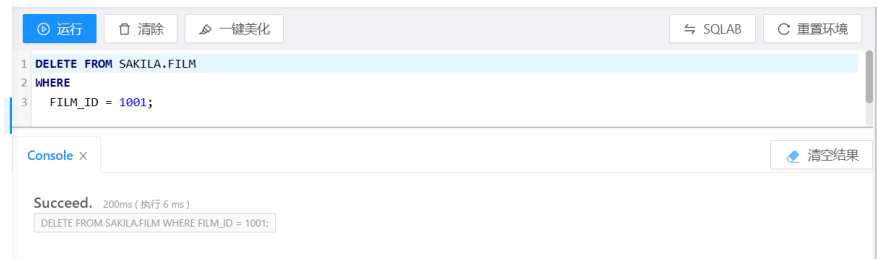
图 5a: FILM 表查询结果



FILM_ID	TITLE	DESCRIPTION
1001	TEST_FILM	THIS IS A TEST, NOT A FILM

图 5b: FILM_TEXT 表查询结果

执行删除 FILM_ID 为 1001 的数据记录的 SQL 语句，执行结果如下图所示，说明 FILM 中 FILM_ID=1001 的数据记录被成功地删除。



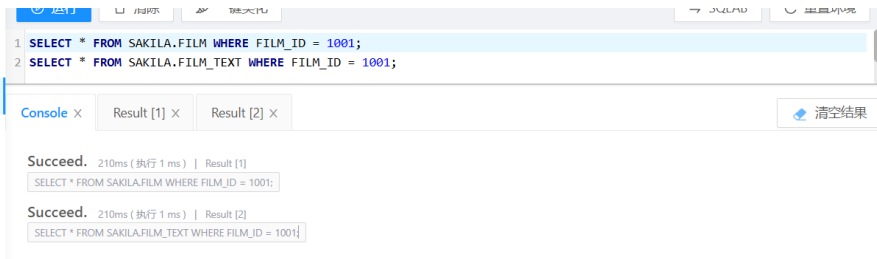
```
1 DELETE FROM SAKILA.FILM
2 WHERE
3 FILM_ID = 1001;
```

Succeed. 200ms (执行 6 ms)

DELETE FROM SAKILA.FILM WHERE FILM_ID = 1001;

图 6: 执行结果

执行如下的查询语句，执行语句成功，但是查询结果为空。说明相关记录已经被删除，进而证明触发器生效。



```
1 SELECT * FROM SAKILA.FILM WHERE FILM_ID = 1001;
2 SELECT * FROM SAKILA.FILM_TEXT WHERE FILM_ID = 1001;
```

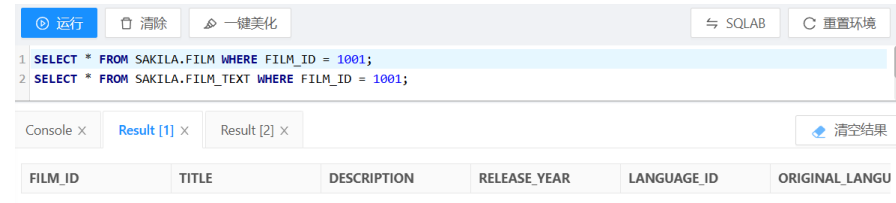
Succeed. 210ms (执行 1 ms) | Result [1]

SELECT * FROM SAKILA.FILM WHERE FILM_ID = 1001;

Succeed. 210ms (执行 1 ms) | Result [2]

SELECT * FROM SAKILA.FILM_TEXT WHERE FILM_ID = 1001;

图 7a: 执行 SQL 语句结果



FILM_ID	TITLE	DESCRIPTION	RELEASE_YEAR	LANGUAGE_ID	ORIGINAL_LANGU
---------	-------	-------------	--------------	-------------	----------------

图 7b: FILM 表查询结果



FILM_ID	TITLE	DESCRIPTION
---------	-------	-------------

图 7c: FILM_TEXT 表查询结果

- (3) 现在有这样一个建立触发器的语句，如下图所示。此时需要迁移 payment 表历史记录，请同学们思考，这个触发器是否会生效？如果生效的话，有什么办法可以避免迁移记录的 payment_date 被修改？

```
CREATE OR REPLACE TRIGGER payment_date
BEFORE INSERT ON payment
FOR EACH ROW
BEGIN
    :NEW.payment_date := SYSTIMESTAMP;
END;
```

分析：

触发器 PAYMENT_DATE 设置为在 PAYMENT 表上执行 INSERT 操作之前触发，并且会将新插入行的 payment_date 字段设置为系统当前时间戳。

在迁移 payment 表历史记录时会涉及到 INSERT 操作，因此，触发器会生效。触发器会自动地将新插入的数据记录的 payment_date 字段设置为系统当前时间戳。

想要迁移记录的 payment_date 不被修改，我们可以 1) 在迁移记录开始之前禁用触发器，在迁移记录结束之后重新开启触发器；2) 使用数据库自带的导入导出工具。

关于约束-回答问题

现有 rental 表的建表语句如下：

```
CREATE TABLE SAKILA.RENTAL
(
    RENTAL_ID BIGINT DEFAULT SAKILA.SEQ_RENTAL_RENTAL_ID.NEXTVAL NOT NULL ,
    RENTAL_DATE TIMESTAMP(6) NOT NULL ,
    INVENTORY_ID INTEGER NOT NULL ,
    CUSTOMER_ID INTEGER NOT NULL ,
    RETURN_DATE TIMESTAMP(6),
    STAFF_ID SMALLINT NOT NULL ,
    LAST_UPDATE TIMESTAMP(6) DEFAULT CURRENT_TIMESTAMP NOT NULL ,
    CONSTRAINT FK_RENTAL_CUSTOMER FOREIGN KEY (CUSTOMER_ID) REFERENCES SAKILA.CUSTOMER (CUSTOMER_ID) ,
    CONSTRAINT FK_RENTAL_INVENTORY FOREIGN KEY (INVENTORY_ID) REFERENCES SAKILA.INVENTORY (INVENTORY_ID) ,
    CONSTRAINT FK_RENTAL_STAFF FOREIGN KEY (STAFF_ID) REFERENCES SAKILA.STAFF (STAFF_ID) ,
    CONSTRAINT IDX_RENTAL_RENTAL_DATE_INVENTORY_ID_CUSTOMER_ID UNIQUE (RENTAL_DATE, INVENTORY_ID, CUSTOMER_ID)
    PRIMARY KEY (RENTAL_ID)
);
```

- (1) rental 表上建了哪几种约束？这些约束分别实现什么功能？

约束类型	功能
NOT NULL，非空约束	字段不能为空，确保字段总是有效
FOREIGN KEY，外键约束	确保引用的字段的值出现在另一个表中，维护参照完整性
UNIQUE，唯一约束	确保某一列或某一组合的值是唯一的
PRIMARY KEY，	确保某一列或某一组合的值是唯一的，每一行记录都可以被其唯一标识

对于 rental 表来说，1) NOT NULL 确保 RENTAL_ID、RENTAL_DATE、INVENTORY_ID、CUSTOMER_ID、RETURN_DATE 和 STAFF_ID 字段不为空；2) FOREIGN KEY 确保 rental 表中的 CUSTOMER_ID 列的值必须在 SAKILA.CUSTOMER 表的 CUSTOMER_ID 列中存在，确保 rental 表中的 INVENTORY_ID 列的值必须在 SAKILA.INVENTORY 表的 INVENTORY_ID 列中存在，确保 rental 表中的 STAFF_ID 列的值必须在 SAKILA.STAFF 表的 STAFF_ID 列中存在；3) UNIQUE 确保 RENTAL_DATE、INVENTORY_ID 和 CUSTOMER_ID 这三个列的组合值在 rental 表上是唯一的；4) PRIMARY KEY 确保 RENTAL_ID 列的值是唯一的，并且每一行记录都可以被唯一标识。

- (2) 图中的 ON DELETE RESTRICT 和 ON UPDATE CASCADE 是什么意思？

ON DELETE RESTRICT 和 ON UPDATE CASCADE 是外键约束的行为选项，用于定义当被引用的数据（即外键指向的主键）被删除或更新时，外键如何处理。

ON DELETE RESTRICT：当尝试删除主键表中的记录时，如果存在外键表中的数据引用了这个主键，那么删除操作会被限制，即不允许删除。这确保了数据的参照完整性，防止出现不合法的引用记录。

ON UPDATE CASCADE：当主键表中的记录被更新时，如果存在外键表中的数据引用了这个主键，那么外键表中相应的记录也会自动更新，以保持数据的一致性。

关于存储过程-回答问题

- (1) 这个存储过程实现了什么功能？输出参数 count_rewardees 是什么？

从 payment 表中查询上个月内购买金额超过 min_dollar_amount_purchased 且购买次数超过 min_monthly_purchases 的客户，将他们放到一个集合里，然后输出每个客户的详细信息。

输出参数 count_rewardees 是满足上述要求的客户的数量。

- (2) last_month_start := ADD_MONTHS (TRUNC (SYSDATE, 'MM'), -1); 具体是怎么获取到上个月第一天的日期的？请展开说明。

首先使用 SYSDATE 函数获取当前系统日期和时间，然后使用 TRUNC (SYSDATE, 'MM') 将当前系统日期和时间截断到月份的第一天，即去掉日和时间部分，只保留年和月，最后使用 ADD_MONTHS (TRUNC (SYSDATE, 'MM'), -1) 函数将当前系统日期的月份减 1，即获取上个月的第一天。

关于函数-回答问题：

观察 SAKILA 用户里面的 get_customer_balance 函数。

- (1) 这个函数实现了什么功能？返回值是什么？

这个函数的功能是计算给定客户 ID 和特定生效日期下的当前余额，影响因素有租赁费用、逾期费用、超期租赁费用和已支付的款项。

返回值是一个 NUMBER 类型的数值，代表计算出的当前余额。

- (2) 这个函数体中用到了 3 个函数，是哪几个函数？这 3 个函数的作用分别是？

函数	作用
NVL (a, b)	NVL 函数用于替换 NULL 值为指定的值。如果 a 为 NULL，则 NVL 函数将其替换为 b
SUM(a)	计算 a 列的总和
TRUNC (a)	截断日期，去掉日期中的时间部分，只保留到日

创建新用户并授予权限

创建新用户

- (1) 查看当前已有用户：select username from dba_users;

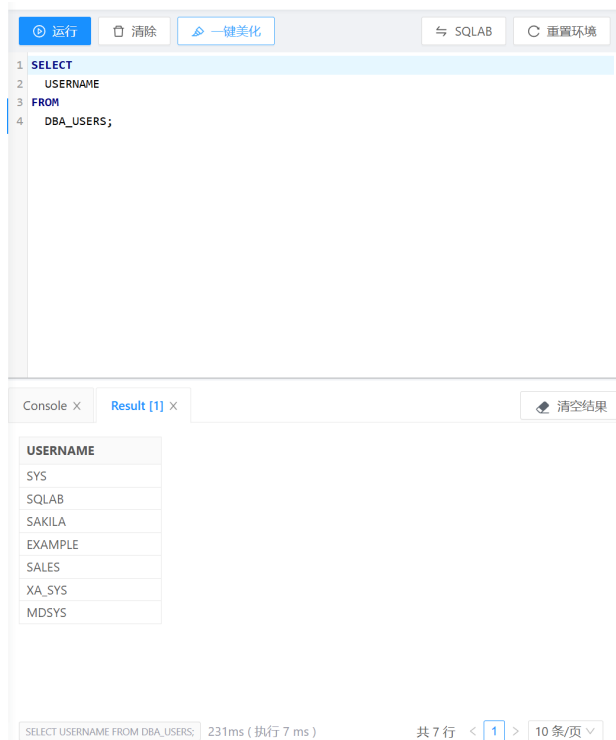


图 8：查看用户

- (2) 新建 SAKILA_TESE 用户（密码 123456）：create user SAKILA_TEST identified by 123456; 执行结果如下图所示，说明新建用户成功。

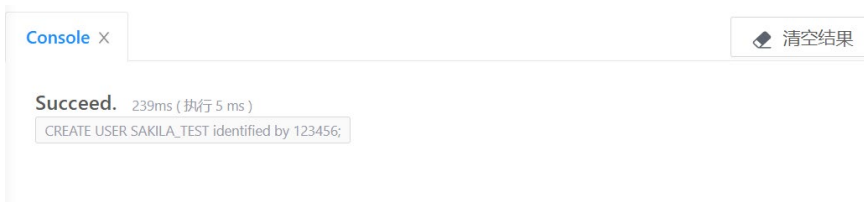


图 9：语句执行结果

- (3) 再次查看已有用户：select username from dba_users; 新用户已经被添加到列表中。



图 10：新建用户结果

为新用户 SAKILA_TEST 赋予访问 sakila 的权限

- (1) 查看新用户当前的权限。

```
select * from dba_sys_privs where grantee='SAKILA_TEST';
select * from dba_tab_privs where grantee='SAKILA_TEST';
select * from dba_role_privs where grantee='SAKILA_TEST';
```

执行结果如下图所示，发现新用户没有任何权限。

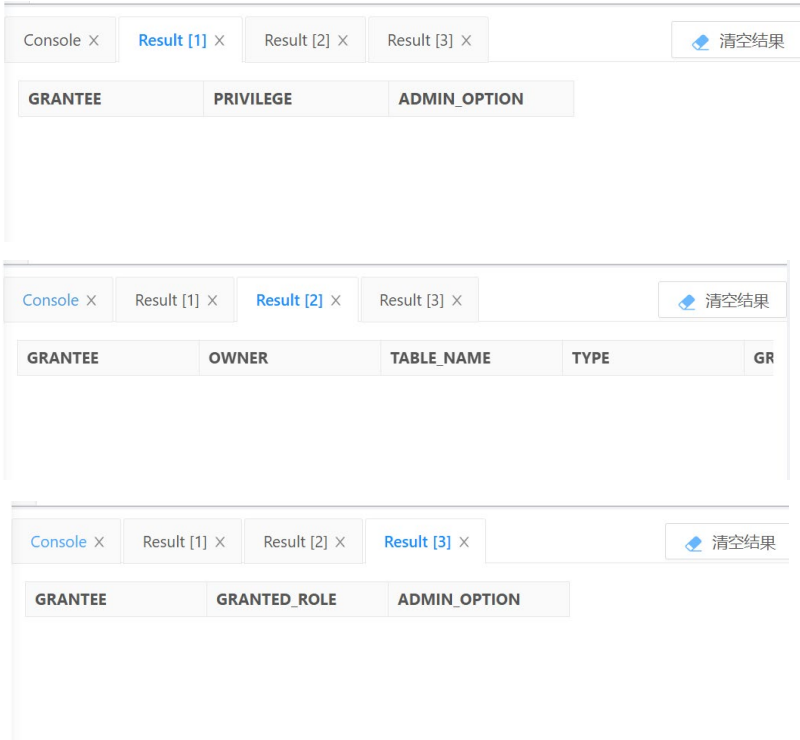


图 11：权限查询结果

- (2) 赋予新用户 connect 角色（CONNECT 角色具有 CREATE SESSION 权限，通过赋予 CONNECT 角色使用户能够登录会话。）：grant connect to SAKILA_TEST;赋予新用户 resource 角色（RESOURCE 角色具有 CREATE TABLE、CREATE SEQUENCE、CREATE PROCEDURE、CREATE TRIGGER、CREATE TYPE 的权限。）：grant resource to SAKILA_TEST;



图 12: 赋予权限的语句执行结果

- (3) 使用新用户连接数据库。

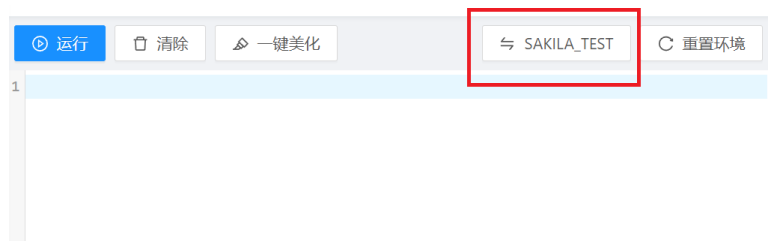


图 13: 新用户登录

- (4) 把原来的 SAKILA 用户中所有的表的访问权限赋予 SAKILA_TEST 用户: 执行语句 `SELECT 'GRANT SELECT ON SAKILA.' || TABLE_NAME || ' TO SAKILA_TEST;' FROM ALL_TABLES WHERE OWNER = 'SAKILA'`;然后将得到的赋予权限的语句输入到命令行。执行结果如下图所示, 说明赋予 SAKILA_TEST 用户权限成功。

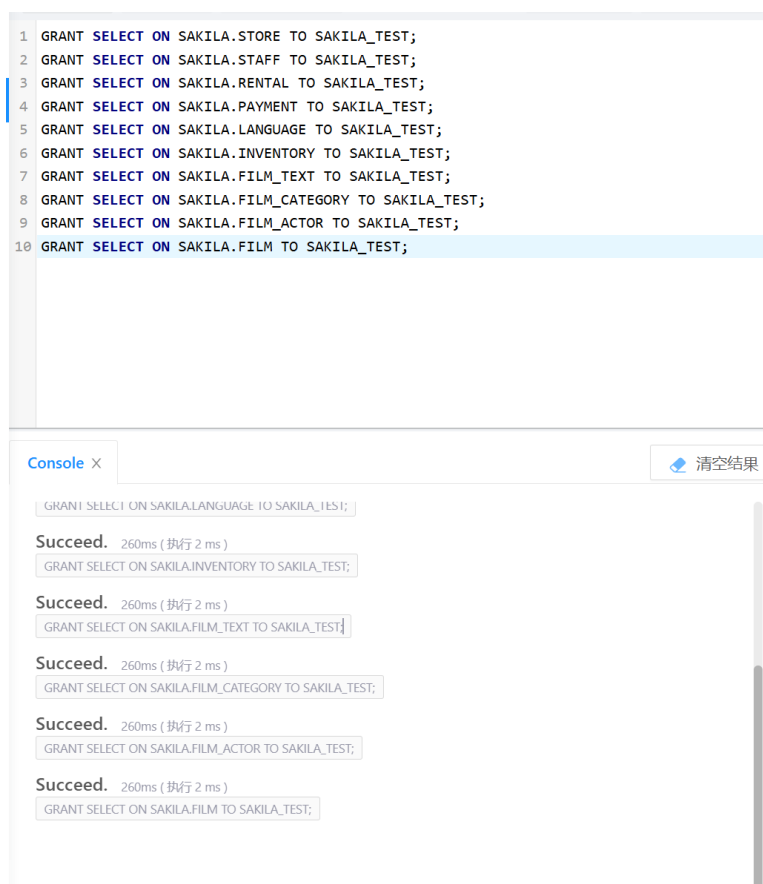


图 14: 赋予权限结果

- (5) 切换到 sakila_test 用户, 这时可以看到 sakila 用户的所有表, 可以进行查询操作: `SELECT * FROM ALL_TABLES;`

OWNER	TABLE_NAME	TABLE_TYPE	TABLESPACE_NAME	ST.
SAKILA	STORE	HEAP	USERS	VA
SAKILA	STAFF	HEAP	USERS	VA
SAKILA	RENTAL	HEAP	USERS	VA
SAKILA	PAYMENT	HEAP	USERS	VA
SAKILA	LANGUAGE	HEAP	USERS	VA
SAKILA	INVENTORY	HEAP	USERS	VA
SAKILA	FILM_TEXT	HEAP	USERS	VA
SAKILA	FILM_CATEGORY	HEAP	USERS	VA
SAKILA	FILM_ACTOR	HEAP	USERS	VA
SAKILA	FILM	HEAP	USERS	VA

图 15：查询结果

设计并实现

根据应用场景，为 Sakila 数据库合理地设计并实现：

1. 设计 1 个视图，通过关联 customer 和 address 2 个表，可以查看客户及其地址信息的列表，方便用户查看地址信息；

设计思路：查看 customer 表，一共有 8 个字段。其中，customer_id, first_name, last_name, email 和 address_id 是需要通过视图展示出来的客户信息。同时，我们还需要根据 address_id 找到对应的 address 表中的信息。查看 address 表，其中需要我们关注的字段为 ADDRESS, ADDRESS2, DISTRICT, CITY_ID, POSTAL_CODE 和 PHONE。此时，我们还可以根据 CITY_ID 去 city 表中找到对应的城市名，但是题目要求我们只需要关联 customer 和 address2 个表。

综合上述分析，我们只需要创建视图显示上面提到的字段即可。创建视图语句如图 16 所示。

```
CREATE VIEW sakila.customer_address_view AS
SELECT  c.customer_id,  c.first_name,  c.last_name,  c.email,  c.address_id,
a.address, a.address2, a.district, a.city_id, a.postal_code, a.phone
FROM sakila.customer AS c
JOIN sakila.address AS a ON c.address_id = a.address_id;
```

图 16：创建视图

创建视图语句运行结果如下图所示。终端提示 Succeed 说明创建视图语句成功运行。

```
1 CREATE VIEW sakila.customer_address_view AS
2 SELECT
3   c.customer_id,
4   c.first_name,
5   c.last_name,
6   c.email,
7   c.address_id,
8   a.address,
9   a.address2,
10  a.district,
11  a.city_id,
12  a.postal_code,
13  a.phone
14 FROM
15  sakila.customer AS c
16  JOIN sakila.address AS a ON c.address_id = a.address_id;
```

Console X

Succeed. 202ms (执行 5 ms)

CREATE VIEW sakila.customer_address_view AS SELECT c.customer_id, c.first_name, c.last_name, c.e...

图 17：创建语句运行结果

接下来我们通过视图查阅客户及其地址信息，查看上述语句创建的视图是否能正确显示客户及其地址信息。

```
1 SELECT
2 *
3 FROM
4 sakila.customer_address_view;
```

Console X Result [1] X 清空结果

CUSTOMER_ID	FIRST_NAME	LAST_NAME	EMAIL	ADDRESS_ID	A
1	MARY	SMITH	MARY.SMITH@sakilacustomer.org	5	1!
2	PATRICIA	JOHNSON	PATRICIA.JOHNSON@sakilacustomer.org	6	1
3	LINDA	WILLIAMS	LINDA.WILLIAMS@sakilacustomer.org	7	6!
4	BARBARA	JONES	BARBARA.JONES@sakilacustomer.org	8	1!
5	ELIZABETH	BROWN	ELIZABETH.BROWN@sakilacustomer.org	9	5!
6	JENNIFER	DAVIS	JENNIFER.DAVIS@sakilacustomer.org	10	1!
7	MARIA	MILLER	MARIA.MILLER@sakilacustomer.org	11	9!
8	SUSAN	WILSON	SUSAN.WILSON@sakilacustomer.org	12	4!
9	MARGARET	MOORE	MARGARET.MOORE@sakilacustomer.org	13	6!
10	DOROTHY	TAYLOR	DOROTHY.TAYLOR@sakilacustomer.org	14	1!

图 18：显示客户及其地址信息

由图 18 可以看出，视图可以正确显示客户及其地址信息。

- 设计 1 个触发器，融合 ins_film、upd_film 和 del_film 的功能，需要在报告里体现触发器生效；

我们首先将三个触发器实现的功能写到一个新的触发器中，然后修改触发状态，根据不同的触发状态进入不同的选择分支中。通过上述步骤，实现一个触发器融合三个触发器的功能。

```
CREATE OR REPLACE TRIGGER sakila.film_sync_trigger
BEFORE INSERT OR UPDATE OR DELETE ON sakila.film
FOR EACH ROW
BEGIN
    IF INSERTING THEN
        INSERT INTO sakila.film_text (film_id, title, description)
        VALUES (:NEW.film_id, :NEW.title, :NEW.description);
```

```

END IF;
IF UPDATING THEN
    IF :OLD.title != :NEW.title OR :OLD.description != :NEW.description
OR :OLD.film_id != :NEW.film_id THEN
        UPDATE sakila.film_text
        SET title = :NEW.title,
            description = :NEW.description,
            film_id = :NEW.film_id
        WHERE film_id = :OLD.film_id;
    END IF;
END IF;
IF DELETING THEN
    DELETE FROM sakila.film_text
    WHERE film_id = :OLD.film_id;
END IF;
END;

```

图 19：创建触发器

上述创建触发器的语句运行结果如下图。由下图终端显示的 Succeed 可知，触发器创建成功。

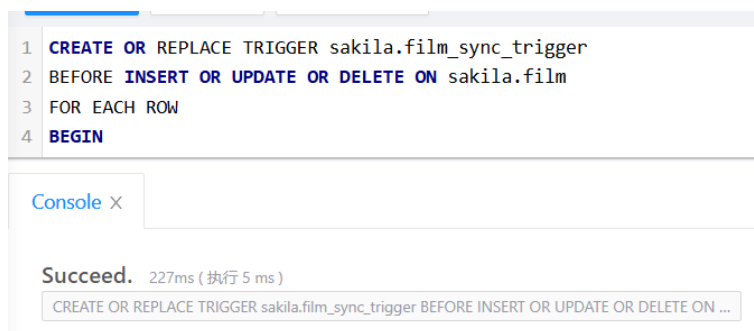


图 20：创建结果

接下来我们需要知道触发器是否生效。查阅崖山数据库的手册得知，我们需要输入命令：*select * from DBA_TRIGGERS where owner = 'SAKILA' and table_name = 'FILM' and trigger_name = 'FILM_SYNC_TRIGGER'*；。然后找到STATUS字段，如果字段的值为'ENABLED'，则说明触发器已经生效，否则没有生效。

TRIGGERING NAMES	WHEN CLAUSE	STATUS	DESCRIPTION	ACTION TYPE	TRIGGER_BODY
ENDING NEW AS OLD AS OLD		ENABLED	film_sync_trigger BEFORE INSERT OR UPDATE OR DELETE ON sakila.film FOR EACH ROW	PL/SQL	BEGIN IF INSERTING THEN INSERT INTO sakila.film_text (film_id, title, description) VALUES (:NEW.film_id, :NEW.title, :NEW.description); END IF; IF UPDATING THEN IF :OLD.title != :NEW.title OR :OLD.description != :NEW.description OR :OLD.film_id != :NEW.film_id THEN UPDATE sakila.film_text SET title = :NEW.title,

图 21：查看触发器是否生效

由上图得知，触发器已经生效，其`STATUS`字段的值已经是'`ENABLED`'。

- 设计 1 个存储过程，用于获取特定客户在指定时间段内的租赁历史记录。它将返回客户的租赁信息，包括租赁日期、归还日期、租赁的电影标题以及租赁费用。须在报告里调用，并展示结果。

设计的存储过程如下所示。1) 参数为：客户的 id、查询的起始时间和查询的结束时间；2) 定义相关变量，用于临时存储一条租赁历史记录；3) 定义一个游标用于遍历所有符合记录的租赁历史；4) 使用游标和循环结构遍历所有符合条件的记录并展示。

```
DELIMITER $$

CREATE PROCEDURE get_customer_rental_history(
    IN p_customer_id INT,
    IN p_start_date TIMESTAMP,
    IN p_end_date TIMESTAMP
)
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE v_rental_date TIMESTAMP;
    DECLARE v_return_date TIMESTAMP;
    DECLARE v_film_title VARCHAR(128);
    DECLARE v_rental_fee DECIMAL(4, 2);

    DECLARE rental_cursor CURSOR FOR
        SELECT r.rental_date, r.return_date, f.title, f.rental_rate
        FROM rental AS r
        JOIN inventory AS i ON r.inventory_id = i.inventory_id
        JOIN film AS f ON i.film_id = f.film_id
        WHERE r.customer_id = p_customer_id
        AND r.rental_date BETWEEN p_start_date AND p_end_date;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    OPEN rental_cursor;

    read_loop: LOOP
        FETCH rental_cursor INTO v_rental_date, v_return_date, v_film_title,
v_rental_fee;
        IF done THEN
            LEAVE read_loop;
        END IF;

        SELECT v_rental_date AS 'Rental Date',
            v_return_date AS 'Return Date',
            v_film_title AS 'Film Title',
            v_rental_fee AS 'Rental Fee';
    END LOOP;
END
```

```

END LOOP;

CLOSE rental_cursor;
END $$

DELIMITER ;

```

执行结果如下图所示，红框中的 OK 说明创建存储过程成功。

```

mysql> DELIMITER ;
mysql> DELIMITER $$
mysql>
mysql> CREATE PROCEDURE get_customer_rental_history(
-> IN p_customer_id INT,
-> IN p_start_date TIMESTAMP,
-> IN p_end_date TIMESTAMP
-> )
-> BEGIN
-> DECLARE done INT DEFAULT FALSE;
-> DECLARE v_rental_date TIMESTAMP;
-> DECLARE v_return_date TIMESTAMP;
-> DECLARE v_film_title VARCHAR(128);
-> DECLARE v_rental_fee DECIMAL(4, 2);
->
-> DECLARE rental_cursor CURSOR FOR
-> SELECT r.rental_date, r.return_date, f.title, f.rental_rate
-> FROM rental AS r
-> JOIN inventory AS i ON r.inventory_id = i.inventory_id
-> JOIN film AS f ON i.film_id = f.film_id
-> WHERE r.customer_id = p_customer_id
-> AND r.rental_date BETWEEN p_start_date AND p_end_date;
->
-> DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
->
-> OPEN rental_cursor;
->
-> read_loop: LOOP
-> FETCH rental_cursor INTO v_rental_date, v_return_date, v_film_title, v_rental_fee;
-> IF done THEN
-> LEAVE read_loop;
-> END IF;
->
-> SELECT v_rental_date AS 'Rental Date',
-> v_return_date AS 'Return Date',
-> v_film_title AS 'Film Title',
-> v_rental_fee AS 'Rental Fee';
->
-> END LOOP;
->
-> CLOSE rental_cursor;
-> END $$
Query OK, 0 rows affected (0.00 sec)
mysql>
mysql> DELIMITER ;

```

图 22：创建存储过程

以客户 1 为例，首先查看客户 1 的所有租赁记录（按照租赁日期的升序排序）：
select rental_date, return_date, customer_id from rental where customer_id = 1 order by rental_date;

我们的目标是调用存储过程，查找客户 1 在 2005 年 6 月的租赁记录。因此，下图只展示属于我们的目标日期的部分。可以看到，符合要求的一共右 7 条记录。

```
mysql> select rental_date, return_date, customer_id from rental
```

rental_date	return_date	customer_id
2005-05-25 11:30:37	2005-06-03 12:00:37	1
2005-05-28 10:35:23	2005-06-03 06:32:23	1
2005-06-15 00:54:12	2005-06-23 02:42:12	1
2005-06-15 18:02:53	2005-06-19 15:54:53	1
2005-06-15 21:08:46	2005-06-25 02:26:46	1
2005-06-16 15:18:57	2005-06-17 21:05:57	1
2005-06-18 08:41:48	2005-06-22 03:36:48	1
2005-06-18 13:33:59	2005-06-19 17:40:59	1
2005-06-21 06:24:45	2005-06-28 03:28:45	1
2005-07-08 03:17:05	2005-07-14 01:19:05	1
2005-07-08 07:33:56	2005-07-12 13:25:56	1
2005-07-09 13:24:07	2005-07-14 14:01:07	1

图 23：客户 1 的租赁记录（部分）

调用上述存储过程：*CALL get_customer_rental_history(1, '2005-06-01', '2005-06-30');*。结果如下图所示。一共有 7 条记录，并且记录的租赁日期和归还日期与上相符，说明我设计的存储过程的正确性。


```
mysql> CALL get_customer_rental_history(1, '2005-06-01', '2005-06-30');
```

Rental Date	Return Date	Film Title	Rental Fee
2005-06-15 00:54:12	2005-06-23 02:42:12	MUSKETEERS WAIT	4.99

1 row in set (0.00 sec)

Rental Date	Return Date	Film Title	Rental Fee
2005-06-15 18:02:53	2005-06-19 15:54:53	DETECTIVE VISION	0.99

1 row in set (0.01 sec)

Rental Date	Return Date	Film Title	Rental Fee
2005-06-15 21:08:46	2005-06-25 02:26:46	FERRIS MOTHER	2.99

1 row in set (0.02 sec)

Rental Date	Return Date	Film Title	Rental Fee
2005-06-16 15:18:57	2005-06-17 21:05:57	CLOSER BANG	4.99

1 row in set (0.03 sec)

Rental Date	Return Date	Film Title	Rental Fee
2005-06-18 08:41:48	2005-06-22 03:36:48	ATTACKS HATE	4.99

1 row in set (0.05 sec)

Rental Date	Return Date	Film Title	Rental Fee
2005-06-18 13:33:59	2005-06-19 17:40:59	SAVANNAH TOWN	0.99

1 row in set (0.08 sec)

Rental Date	Return Date	Film Title	Rental Fee
2005-06-21 06:24:45	2005-06-28 03:28:45	YOUTH KICK	0.99

1 row in set (0.10 sec)

Query OK, 0 rows affected (0.13 sec)

图 24：调用存储过程查询结果

思考题

在 MySQL 开发规范中，通常要求为每张表创建一个主键，但是在 YashanDB 中，表可以没有主键。请分析一下原因。主键是否没有存在的必要？

答：主键可以不存在的原因有：1）数据记录的字段包含足够的信息来唯一标识记录或者数据记录的字段可以组合成候选码；2）可能对于一些数据记录来说，并不需要保证唯一性；3）为了避免索引的开销。

但是，主键仍然有存在的必要。原因如下：1）使用主键可以保证数据的唯一性，没有重复的记录；2）在关系数据库中，其他表可以通过外键与主键相关联，从而维护数据的参照完整性；3）主键通常会自动创建索引，可以提高查询效率；4）主键是实体完整性的一部分，有助于维护数据的准确性和一致性。