پایگاه داده

تمرین سری سوم

پارسا ملکیان ۴۰۲۱۷۱۰۷۵

سوالات تئوری

سوال۱

الف) ایجاد VIEW با نام کارمند، نام دپارتمان و حقوق

```sql
CREATE VIEW EmployeeDepartmentInfo AS
SELECT e.name AS employee_name, d.name AS department_name, e.salary
FROM Employees e
JOIN Departments d ON e.department_id = d.department_id;
```

سناریوهای کاربردی این VIEW:

- ساده‌سازی کوئری‌های پیچیده و حذف نیاز به نوشتن JOIN مکرر
- کنترل دسترسی و امنیت: دادن دسترسی به کاربران بدون نمایش کل جداول
- گزارش‌گیری: استفاده آسان برای گزارش‌های مربوط به کارمندان و دپارتمان‌ها
- انتزاع داده: پنهان کردن پیچیدگی ساختار دیتابیس از کاربران نهایی
- ثبات: اطمینان از اینکه همه کاربران به ساختار یکسانی از داده دسترسی دارند

ب) تأثیر حذف دپارتمان بر VIEW

اثر حذف یک دپارتمان بر VIEW به محدودیت‌های کلید خارجی بستگی دارد:

۱. **بدون محدودیت کلید خارجی:**
   - کارمندان با department_id حذف شده در VIEW باقی می‌مانند، اما department_nameآنها NULL خواهد بود.

۲. **با محدودیت کلید خارجی و CASCADE DELETE :**
   - با حذف دپارتمان، تمام کارمندان آن نیز حذف می‌شوند و دیگر در VIEW ظاهر نمی‌شوند.

۳. **با محدودیت کلید خارجی و SET NULL :**
   - با حذف دپارتمان، مقدار department_id کارمندان آن به NULL تبدیل می‌شود و در VIEW ، department_nameآنها NULL خواهد بود.

۴. **با محدودیت کلید خارجی و RESTRICT یا NO ACTION :**
   - از حذف دپارتمان جلوگیری می‌شود و VIEW بدون تغییر باقی می‌ماند.

ج) ایجاد VIEW برای کارمندان با حقوق بیش از ۵۰ میلیون

```sql
CREATE VIEW HighPaidEmployees AS
SELECT e.name AS employee_name, d.name AS department_name, e.salary
FROM Employees e
JOIN Departments d ON e.department_id = d.department_id
WHERE e.salary > 50000000;
```

د) استفاده از WITH CHECK OPTION

```sql
CREATE VIEW HighPaidEmployees AS
SELECT e.name AS employee_name, d.name AS department_name, e.salary
FROM Employees e
JOIN Departments d ON e.department_id = d.department_id
WHERE e.salary > 50000000
WITH CHECK OPTION;
```

تأثیرWITH CHECK OPTION :

- تضمین می‌کند که هر عملیات INSERT یا UPDATE از طریقVIEW ، شرط WHERE را رعایت کند.
- اگر تلاش کنید از طریق این VIEW کارمندی با حقوق کمتر از ۵۰ میلیون اضافه کنید یا حقوق کارمندی را به کمتر از ۵۰ میلیون تغییر دهید، عملیات رد می‌شود.

آیا استفاده از آن الزامی است؟

استفاده از WITH CHECK OPTION الزامی نیست، اما برای حفظ یکپارچگی داده‌ها مفید است.

بدون این گزینه، می‌توان از طریقVIEW ، رکوردهایی را اضافه یا به‌روزرسانی کرد که بلافاصله از VIEWناپدید می‌شوند چون شرط WHERE را برآورده نمی‌کنند.

سوال ۲

الف) قیود یکپارچگی که ممکن است نقض شوند

۱ .قید "تکمیل prerequisite پیش از ثبت‌نام":

نقض قید :ثبت‌نام دانشجو در درسی که پیش‌نیاز آن را نگذرانده است

- **پیامدهای عملی :**
  - دانشجویان بدون دانش پایه لازم وارد درس می‌شوند
  - احتمال افت تحصیلی و شکست در درس افزایش می‌یابد
  - برنامه درسی توالی منطقی خود را از دست می‌دهد
- **پیامدهای آموزشی :**
  - استاد باید زمان بیشتری صرف تدریس مفاهیم پایه کند
  - کاهش سطح کلی کلاس و کندی پیشرفت درس
  - عدم درک کامل مفاهیم پیشرفته توسط دانشجویان

۲ .قید "رعایت محدودیت max_seats برای هر:"Course

نقض قید :ثبت‌نام بیش از ظرفیت مجاز در یک درس

- **پیامدهای عملی :**
  - کمبود فضای فیزیکی در کلاس
  - کمبود منابع آموزشی و امکانات لازم
  - مشکلات تدارکات و برنامه‌ریزی
- **پیامدهای آموزشی :**
  - کاهش کیفیت تدریس و توجه استاد به تک‌تک دانشجویان
  - کاهش فرصت مشارکت دانشجویان در کلاس
  - فشار بر زیرساخت‌های آموزشی و افت کیفیت یادگیری

ب) نوشتن ASSERTION در SQL

قید "رعایت محدودیت max_seats":

```sql
CREATE ASSERTION check_max_seats AS
CHECK (NOT EXISTS (
    SELECT c.course_id, COUNT(e.student_id) AS enrolled_count, c.max_seats
    FROM Course c
    JOIN Enrollment e ON c.course_id = e.course_id
    GROUP BY c.course_id, c.max_seats
    HAVING COUNT(e.student_id) > c.max_seats
));
```

اگر این ASSERTION در حین اجرای یک تراکنش نقض شود:

- تراکنش با خطا مواجه می‌شود
- تمام تغییرات انجام شده در تراکنش با ROLLBACK به حالت قبل برمی‌گردد
- عملیات COMMIT انجام نمی‌شود
- به کاربر پیام خطا نمایش داده می‌شود
- هیچ داده‌ای که باعث نقض یکپارچگی شود در پایگاه داده ذخیره نمی‌شود

ج) پیاده‌سازی با TRIGGER در PostgreSQL

```sql
CREATE OR REPLACE FUNCTION check_max_seats_func()
RETURNS TRIGGER AS $$
DECLARE
    current_enrollment INTEGER;
    max_seats_limit INTEGER;
BEGIN
    SELECT COUNT(*) INTO current_enrollment
    FROM Enrollment
    WHERE course_id = NEW.course_id;

    SELECT max_seats INTO max_seats_limit
    FROM Course
    WHERE course_id = NEW.course_id;

    IF (current_enrollment + 1) > max_seats_limit THEN
        RAISE EXCEPTION 'The capacity of the course with code % is full. (Maximum capacity: %)',
                        NEW.course_id, max_seats_limit;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER check_max_seats_trigger
BEFORE INSERT ON Enrollment
FOR EACH ROW
EXECUTE FUNCTION check_max_seats_func();
```

این TRIGGER قبل از هر عملیات INSERT در جدول Enrollment اجرا می‌شود و بررسی می‌کند که آیا با افزودن این ثبت‌نام جدید، تعداد دانشجویان از ظرفیت مجاز درس بیشتر می‌شود یا خیر. اگر تعداد از حد مجاز بیشتر باشد، با پیام خطای مناسب عملیات را متوقف می‌کند.

سوال ۳

الف) طراحی view برای order_summary

```sql
CREATE VIEW order_summary AS
SELECT o.order_id, o.customer_id, o.order_date,
       SUM(oi.quantity * oi.price) AS total_amount
FROM orders o
JOIN order_items oi ON o.order_id = oi.order_id
GROUP BY o.order_id, o.customer_id, o.order_date;
```

ب) نوشتن trigger از نوع INSTEAD OF روی view

```sql
CREATE OR REPLACE FUNCTION update_order_summary()
RETURNS TRIGGER AS $$
DECLARE
    current_total NUMERIC;
    difference NUMERIC;
    target_product_id INT;
    target_price NUMERIC;
    target_quantity NUMERIC;
    proportional_reduction NUMERIC;
    max_item_id RECORD;
BEGIN
    SELECT SUM(quantity * price) INTO current_total
    FROM order_items
    WHERE order_id = NEW.order_id;

    difference := NEW.total_amount - current_total;

    IF difference > 0 THEN
        SELECT product_id, unit_price INTO target_product_id, target_price
        FROM products
        ORDER BY product_id
        LIMIT 1;

        target_quantity := difference / target_price;

        INSERT INTO order_items (order_id, product_id, quantity, price)
        VALUES (NEW.order_id, target_product_id, target_quantity, target_price);

    ELSIF difference < 0 THEN
        SELECT * INTO max_item_id
        FROM order_items
        WHERE order_id = NEW.order_id
        ORDER BY quantity * price DESC
        LIMIT 1;

        proportional_reduction := LEAST(ABS(difference) / (max_item_id.quantity * max_item_id.price), 1);

        target_quantity := max_item_id.quantity * (1 - proportional_reduction);

        UPDATE order_items
        SET quantity = target_quantity
        WHERE order_id = NEW.order_id AND product_id = max_item_id.product_id;
```

```
        DELETE FROM order_items
        WHERE order_id = NEW.order_id AND quantity <= 0;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER instead_of_update_order_summary
INSTEAD OF UPDATE ON order_summary
FOR EACH ROW
EXECUTE FUNCTION update_order_summary();
```

ج) چالش‌های طراحی هنگام بروزرسانی ستون‌های محاسبه‌شده

۱. همگام‌سازی داده‌ها

چالش: تضمین اینکه تغییرات در چندین جدول به‌درستی در ستون‌های محاسبه‌شده منعکس شوند.

مثلا اگر رکوردی از order_items حذف شود، total_amount در order_summary باید بلافاصله به‌روز شود.

راهکار: استفاده از trigger های مناسب روی همه جداول مرتبط (AFTER INSERT/UPDATE/DELETE).

۲. مسائل عملکردی (Performance)

چالش: محاسبه مجدد ستون‌ها با هر تغییر می‌تواند عملکرد سیستم را کاهش دهد.

راهکار: استفاده از محاسبات افزایشی (incremental) به جای محاسبه کامل، یا استفاده از materialized viewها همراه با به‌روزرسانی دوره‌ای.

۳. پیچیدگی منطق تجاری

چالش: پیاده‌سازی منطق پیچیده کسب‌وکار در trigger ها دشوار است.

مثلا در مثال ما، تصمیم‌گیری برای انتخاب محصول بونوس یا کاهش کدام آیتم نیاز به منطق پیچیده‌ای دارد.

راهکار: انتقال منطق پیچیده به توابع جداگانه و قابل آزمون.

۴. مسائل همزمانی (Concurrency)

چالش: وقتی چندین کاربر همزمان داده‌ها را تغییر می‌دهند، شرایط مسابقه (race conditions) ممکن است رخ دهد.

مثلا دو کاربر همزمان total_amount را تغییر دهند و سبب ناهماهنگی شوند.

راهکار:استفاده از سطوح مناسب isolation در تراکنش‌ها و قفل‌گذاری.(locking)

۵ .مدیریت خطا

چالش :مدیریت مناسب خطاها در زنجیره‌ای از عملیات وابسته.

راهکار:پیاده‌سازی مکانیسم‌های مناسب برای ROLLBACK و گزارش خطا، استفاده از بلوک‌های EXCEPTION.

۶ .تست‌پذیری

چالش :تست کردن trigger ها و اعتبارسنجی رفتار سیستم در سناریوهای مختلف.

راهکار:ایجاد تست‌های خودکار و سناریوهای جامع برای بررسی رفتار سیستم.

۷ .نگهداری و توسعه‌پذیری

چالش :با پیچیده‌تر شدن سیستم، نگهداری و توسعه آن دشوارتر می‌شود.

راهکار:مستندسازی دقیق، استفاده از الگوهای طراحی استاندارد، و تقسیم منطق به بخش‌های کوچکتر و قابل مدیریت.

سوال٤.

الف) اضافه کردن فیلم جدید

```
INSERT INTO MovieProd (title, year, pName)
SELECT 'New DB', 1404, pName
FROM Producer
WHERE pID = 123;
```

این دستور یک رکورد جدید به ویو ذخیره‌شده اضافه می‌کند که اطلاعات فیلم جدید را با نام تولیدکننده مربوطه ترکیب می‌کند.

ب) حذف یک فیلم

```
DELETE FROM MovieProd
WHERE title = 'Old DB' AND year = 1403;
```

با حذف فیلم از جدول اصلی، باید رکورد مربوط به آن در ویو نیز حذف شود.

ج) اضافه کردن تولیدکننده جدید

```
INSERT INTO MovieProd (title, year, pName)
SELECT title, year, 'Rox'
FROM Movie
WHERE producerID = 456;
```

اگر فیلم‌هایی با producerID=456 وجود داشته باشند، باید اطلاعات آنها همراه با نام تولیدکننده جدید به ویو اضافه شوند.

د) حذف یک تولیدکننده

```
DELETE FROM MovieProd
WHERE pName = 'Bob';
```

با حذف یک تولیدکننده، تمام فیلم‌های مرتبط با آن تولیدکننده باید از ویو حذف شوند.

ه) به‌روزرسانی ارزش خالص تولیدکننده

```
-- هیچ اقدامی نیاز نیست
```

از آنجا که فیلد netWorth در ویو استفاده نشده است، تغییر آن تأثیری بر ویو ندارد و نیازی به به‌روزرسانی نیست.