

1.

(الف)

atomicity : مطمئن میشود که تمام عملیات ها داخل یک واحد عملیاتی به صورت atomic و یکپارچه انجام میگردد و عملیات موفقیت آمیز است. در غیر این صورت transaction متوقف شده و خطا نشان داده میشود و transaction به حالت اولیه بازمیگردد.

Isolation : به واسطه این خاصیت transaction های مختلف مستقل از هم عمل کرده و عملکرد یک transaction از دیگری جداست.

Consistency: در هنگام و پس از اجرای موفقیت آمیز یک transaction یکپارچگی دیتابیس حفظ میشود.

Durability: باعث میشود که نتیجه یک transaction موفقیت آمیز در هنگامی که دچار یک system failure میشود در سیستم باقی بماند.

(ب)

زیرا در این حالت این transaction یک bottleneck برای سیستم محسوب میشود و میتواند باعث کاهش performance شود در حالیکه به علت زیاد بودن تعداد درخواستها در یک سیستم سرعت اجرای transaction ها از اهمیت بالایی برخوردار است پس اجرای همزمان آنها باعث افزایش کارایی سیستم میشود.

(ج)

یک serial schedule در واقع ترتیب انجام زمانبندی transactionهاست و در آن عملیات های داخل یک transaction همگی با یکدیگر یک گروه شده و به ترتیب اجرا میشوند و یک transaction پس از transaction دیگر اجرا میشود. در اینجا ترتیب انجام transactionها از اهمیت بالایی برخوردار است زیرا در صورت ایجاد تغییر ممکن است به نتیجه دیگری برسیم. در یک serializable schedule یک serial schedule مانند s وجود دارد این دو ترتیب به یک نتیجه می انجامند.

(د)

بله زیرا در یک serial چه T1 اول اجرا شود چه T2 باعث میشود که متغیر A یا B مقدارش یک شود و در transaction بعدی شرط اجرا نشود و مقدار یکی از دو متغیر صفر باقی میماند.

2.

(الف) در simple view جدول جدیدی نداریم و با هر فراخوانی از view دستور نظیر ان view اجرا میشود اما در یک materialized عملاً یک جدول جدید ساخته شده و درخواستهای ما از ان جدول فراخوانی میشود.

(ب) هنگامی که دستور ما relation های زیادی دارد اما سریع اپدیت نشود و یا هنگامی که کویری به تعداد زیاد استفاده میشود بهتر است از materialized view استفاده شود. زیرا برخلاف حالت simple در حالتی که کویری زیادی داشته باشیم و جوین های زیادی به کار رفته باشد یک بار دستورات اجرا شده و نتایج در یک جدول ذخیره میشود و دفعات بعد هم از همان جدول ۹۱ استفاده میشود.

(ج) بله

(i) اگر شامل جوین بین چندین تبیل باشد هنگام insert کردن در view تنها میتوانید در یک جدول insert کنید و نمیتوان سطری از view را حذف کرد.

(ii) اطلاعات view هایی که از union یا مواردی مانند distinct یا group by استفاده میکنند را نمیتوان دستکاری کرد.

(iii) ستونهایی را که شامل aggregate ها هستند را نمیتوان ویرایش کرد.

(د)

اگر از materialized view استفاده شود به دلیل اینکه به روزرسانی جدول به صورت دوره ای است اگر جدول هایی که از آنها در view استفاده شده جدول هایی باشد که اطلاعات آنها سریع به روزرسانی میشود ممکن است اطلاعات با اطلاعات دیتابیس سینک نباشد.

در simple view عملاً چون با هر بار اجرای اطلاعات به روزرسانی میشود و کویری ها هم با هر بار اجرا دوباره اجرا میشوند از نظر بهینه بودن و بار پردازشی برای کویری هایی که زیاد استفاده میشوند بهینه نیست.

(3)

الف) spها در داخل select قابل اجرا نیستند اما functionها هستند.

ب) spها میتوانند هر نوع خروجی داشته باشند یا هیچ خروجی نداشته باشند ولی functionها حتماً باید خروجی داشته باشند.

ج) داخل sp میتوان از function استفاده کرد ولی عکس این حالت ممکن نیست.

د) تعداد ورودی مجاز sp بیشتر از function است.

-4-

(الف)

ب) با توجه به شناور بودن قیمت بلیت و ظرفیت در سیستم و تغییر قیمت در طول زمان، بستگی به خریدی که مسافر انجام میدهد ممکن است قیمت های متفاوتی برای مسافر اعمال شود. یکی از مزیت هایی که ذخیره سازی به این شکل دارد track کردن قیمت پرداختی مسافر بسته به زمانی که بلیت را خرید کرده است می باشد

ز) در یک trigger عملیاتی که قرار است انجام شود به صورت خودکار انجام میشود. در واقع با فعال شدن trigger عملیات انجام شده و کنترل بیشتری بر روی این عملیات داریم ولی در حالتی که با استفاده از transactionها انجام میدهم خود فردی که عملیات را انجام میدهد باید نظارت بر روی عملیات مورد نظر داشته باشد و در واقع عملیات trigger توسط DBMS مدیریت می شود ولی در این حالت توسعه دهنده عملیات را مدیریت میکند. مشکلی که پیش می آید هنگامی است که عمل کرد یک trigger شاید موجب

فعال شدن یک trigger دیگر شود که در این حالت برای پیاده سازی با استفاده از transaction دچار پیچیدگی خواهیم شد

5)

```
create recursive view fact(column1, (AS column2)
values(1, 1)
union
SELECT column1+1, column2 * (column1+1) from fact where column2 > 34 );
select column1 from fact ;
```

از int بیشتر شده و دچار overflow میشود.

6-روش اول ایجاد جداول history و انتقال حداقل داده مورد نیاز به آن‌هاست، اشکال این روش این است که در جداول history دیگر نمی‌توان از foreign key های جدول اصلی استفاده کرد ولی می‌توان روی این جداول query های لازم را اجرا کرد. روش دیگر پارتیشن کردن داده‌های جدول با توجه به زمان آن‌هاست که می‌تواند سرعت load را بالا ببرد ولی مشکلات دیگری می‌تواند ایجاد کند.

-8

(الف)

1	Seq Scan on payment (cost=0.00..290.45 rows=7304 width=26) (actual time=16.661..25.425 rows=7304 loops=1)
2	Filter: (staff_id = 2)
3	Rows Removed by Filter: 7292
4	Planning time: 26.810 ms
5	Execution time: 25.678 ms

مقدار 1 loops بیانگر این است که index scan دقیقاً یک بار انجام شده است و نهایتاً 7304 سطر پیدا شده و باید برگردانده شود. مقدار روبروی Filter بیانگر همان فیلترهایی است که ما در where بر روی سطرهای که select کرده ایم اعمال کرده ایم و همچنین بیان شده است که 7292 سطر بر اثر این فیلتر از خروجی نهایی حذف شده است. همچنین زمانی که صرف شده است تا planner بهترین راه برای اجرای بهینه ی این دستور را پیدا کند مقدار 52ms است planner time و زمانی که صرف شده تا راه حلی که planner پیدا کرده است، اجرا شود مقدار 20ms است

(ب)

با with

1	Nested Loop (cost=16.88..393.20 rows=134 width=220) (actual time=0.192..5.348 rows=189 loops=1)
2	Join Filter: (rt.staff_id = st.staff_id)
3	Rows Removed by Join Filter: 95
4	CTE italian_customers
5	-> Nested Loop (cost=4.87..16.71 rows=5 width=35) (actual time=0.059..0.118 rows=7 loops=1)
6	-> Nested Loop (cost=4.60..14.38 rows=6 width=22) (actual time=0.049..0.085 rows=7 loops=1)
7	-> Nested Loop (cost=4.32..12.09 rows=6 width=22) (actual time=0.039..0.053 rows=7 loops=1)
8	-> Seq Scan on country co (cost=0.00..2.36 rows=1 width=13) (actual time=0.014..0.022 rows=1 loops=1)
9	Filter: ((country)::text = 'Italy'::text)
10	Rows Removed by Filter: 108
11	-> Bitmap Heap Scan on city ci (cost=4.32..9.66 rows=6 width=15) (actual time=0.020..0.025 rows=7 loops=1)
12	Recheck Cond: (country_id = co.country_id)
13	Heap Blocks: exact=3
14	-> Bitmap Index Scan on idx_fk_country_id (cost=0.00..4.32 rows=6 width=0) (actual time=0.014..0.014 rows=7 loops=1)
15	Index Cond: (country_id = co.country_id)
16	-> Index Scan using idx_fk_city_id on address (cost=0.28..0.37 rows=1 width=6) (actual time=0.004..0.004 rows=1 loops=7)
17	Index Cond: (city_id = ci.city_id)
18	-> Index Scan using idx_fk_address_id on customer cu (cost=0.28..0.38 rows=1 width=19) (actual time=0.004..0.004 rows=1 loops=7)
19	Index Cond: (address_id = address.address_id)
20	-> Hash Join (cost=0.16..372.11 rows=134 width=218) (actual time=0.182..5.204 rows=189 loops=1)
21	Hash Cond: (rt.customer_id = ic.cid)
22	-> Seq Scan on rental rt (cost=0.00..310.44 rows=16044 width=4) (actual time=0.008..2.242 rows=16044 loops=1)
23	-> Hash (cost=0.10..0.10 rows=5 width=220) (actual time=0.134..0.134 rows=7 loops=1)
24	Buckets: 1024 Batches: 1 Memory Usage: 9kB
25	-> CTE Scan on italian_customers ic (cost=0.00..0.10 rows=5 width=220) (actual time=0.062..0.127 rows=7 loops=1)
26	-> Materialize (cost=0.00..1.03 rows=2 width=4) (actual time=0.000..0.000 rows=2 loops=189)
27	-> Seq Scan on staff st (cost=0.00..1.02 rows=2 width=4) (actual time=0.003..0.003 rows=2 loops=1)
28	Planning time: 1.013 ms
29	Execution time: 5.444 ms

بدون with:

```
23 rows
QUERY PLAN
1  Nested Loop  (cost=3.21..143.05 rows=1 width=17) (actual time=0.148..0.149 rows=0 loops=1)
2    -> Nested Loop  (cost=2.94..142.69 rows=1 width=19) (actual time=0.148..0.148 rows=0 loops=1)
3        Join Filter: (r.staff_id = staff.staff_id)
4        -> Nested Loop  (cost=2.94..141.65 rows=1 width=17) (actual time=0.148..0.148 rows=0 loops=1)
5            -> Nested Loop  (cost=2.65..19.06 rows=1 width=19) (actual time=0.148..0.148 rows=0 loops=1)
6                -> Hash Join  (cost=2.38..18.68 rows=1 width=6) (actual time=0.147..0.148 rows=0 loops=1)
7                    Hash Cond: (a.last_update = c3.last_update)
8                    -> Seq Scan on address a  (cost=0.00..14.03 rows=603 width=14) (actual time=0.008..0.054 rows=603 loops=1)
9                    -> Hash  (cost=2.36..2.36 rows=1 width=8) (actual time=0.016..0.016 rows=1 loops=1)
10                        Buckets: 1024  Batches: 1  Memory Usage: 9kB
11                        -> Seq Scan on country c3  (cost=0.00..2.36 rows=1 width=8) (actual time=0.009..0.013 rows=1 loops=1)
12                            Filter: ((country)::text = 'Italy'::text)
13                            Rows Removed by Filter: 108
14                -> Index Scan using idx_fk_address_id on customer  (cost=0.28..0.38 rows=1 width=19) (never executed)
15                    Index Cond: (address_id = a.address_id)
16            -> Index Scan using idx_unq_rental_rental_date_inventory_id_customer_id on rental r  (cost=0.29..122.31 rows=27 width=4) (never executed)
17                Index Cond: (customer_id = customer.customer_id)
18            -> Seq Scan on staff  (cost=0.00..1.02 rows=2 width=4) (never executed)
19        -> Index Only Scan using city_pkey on city c2  (cost=0.28..0.35 rows=1 width=4) (never executed)
20            Index Cond: (city_id = a.city_id)
21            Heap Fetches: 0
22 Planning time: 0.816 ms
23 Execution time: 0.220 ms
```

در حالت دوم ( ) nested به دلیل تو در تو بودن ها و nested loop بیشتر، زمان Planning time و execution time هر دو بیشتر از حالتی است که با استفاده از with نوشته شده است. نتیجه ای که میتوان گرفت این است که در استفاده از with به دلیل اینکه تعداد جوین های نهایی ممکن است کمتر شود و index گذاری های مناسب تری ممکن است اتخاذ شود زمان اجرا میتواند کمتر باشد

ج)

عمل کرد hash به منظور تبدیل یک رشته ی بزرگ به یک رشته ی کوچکتر توسط dbms استفاده میشود. پس از این کار میتوان موارد hash شده را در قالب جدول هایی و مجموعه هایی راحت تر دسته بندی و مرتب کرد و بنابراین میتوان آن ها را index گذاری کرد و نهایتا برای retrieve کردن داده ها با استفاده از مقادیر index ها سریع تر داده ها را پیدا و استفاده کرد.

همچنین همانطور که از Query Plan ها پیداست عملیات HASH قبل از عملیات index ها و قبل از عملیات های نظیر جست و جو و... استفاده شده است