ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ІВАНА ФРАНКА
Факультет прикладної математики та інформатики

Бази даних та інформаційні системи

ЛАБОРАТОРНА РОБОТА №8

Віртуальні таблиці SQL

(представлення VIEW)

Виконав:
Студент Процьків Назарій
Група ПМі-21
Оцінка - _____
Перевірила:
доц. Малець Р.Б.

2023

Тема: Віртуальні таблиці SQL.
Мета роботи: Ознайомлення з поняттям Віртуальні таблиці SQL, їх створенням та застосуванням.

Завдання (Варіант 17):

Розробити базу даних для сайту соціальної мережі. Соціальна мережа підтримує реєстрацію користувачів зі збереженням усіх їхніх деталей (ім'я, прізвище, дата народження, місце проживання, телефони, сайти/e-mail/skype іт.д.), місця і періоди перебування, місця і періоди навчання, роботи, служби, приєднані файли (зображення, фільми, аудіо), які можна пов’язувати із місцями з деталей. Крім того, кожен користувач має можливість розміщувати свої повідомлення на власній сторінці, отримувати на повідомлення «лайки» та коментарі, а також додавати інших користувачів в друзі, або в «чорний список». Додатково користувач повинен мати змогу шукати нових друзів за довільними критеріями.

Хід роботи

1. Опрацював теоретичний матеріал.
2. Відповідно до свого завдання написав 3 представлення SQL. Виконання і результати проілюстрував скрінами, а також поясненнями їх роботи.

Представлення №1 (змінюване):

Query Query History

```
1  create or replace view updatable_view as
2  select * from page
```

Data Output Messages Notifications

```
CREATE VIEW
```

Вище бачимо приклад змінюваного представлення. За замовчуванням усі прості представлення є змінюваними, проте є певні умови, при відсутності однієї з яких вони втрачають цю можливість.

Умови, які повинні виконуватись, щоб представлення було змінюваним:

1. Представлення може мати лиш одне entry в FROM, яким може бути table, або інше змінюване представлення.
2. Тіло представлення не має містити WITH, DISTINCT, GROUP BY, HAVING, LIMIT, OFFSET.
3. Тіло представлення не має містити операції множин, тобто UNION, INTERSECT, EXCEPT.
4. SELECT список не має містити ніяких функцій агрегації і т.д.

Як бачимо, усі ці умови виконані при створенні вказаного вище представлення, тому воно є змінюваним. Коли ми працюємо зі змінюваним представленням, значення в ньому оновлюються при кожній спробі взяти з нього певні дані, або ж додати дані в таблицю. Тобто це представлення постійно оновлюється, на відміну від матеріалізованого представлення, яке я буде пізніше.

Подивімося, як зміниться представлення коли спробуємо додати в нього дані.
Створене змінюване представлення:

Query Query History

```
1  select * from updatable_view
```

Data Output Messages Notifications

	pageid bigint	message_id text
1	2	#23
2	3	#33
3	4	#43
4	5	#53
5	6	#63

Спроба додати кортеж за допомогою insert:

Query Query History

```
1  insert into updatable_view values (7, '#73')
```

Data Output Messages Notifications

```
INSERT 0 1

Query returned successfully in 37 msec.
```

Перевірка чи кортеж доданий:

Query Query History

```
1  select * from updatable_view
```

Data Output Messages Notifications

	pageid bigint	message_id text
1	2	#23
2	3	#33
3	4	#43
4	5	#53
5	6	#63
6	7	#73

Кортеж додано успішно, бо видно, що новий кортеж (7, “#73”) додався в таблицю.

Тепер подивімося в таблицю page з якої будувалося це представлення:

```
Query  Query History
1  select * from page
```

Data Output | Messages | Notifications

	pageid bigint	message_id [PK] text
1	2	#23
2	3	#33
3	4	#43
4	5	#53
5	6	#63
6	7	#73

Як видно зі скріна, вона теж змінилась і має в собі тепер новий кортеж даних, який був доданий у представлення.

Віртуальна таблиця №2 (Незмінювана):
Створив незмінювану таблицю

```
Query  Query History
1  create or replace view non_updatable_view as
2  (
3    SELECT users.id, users.firstname, users.lastname,
4      COUNT(DISTINCT serviceplaces.serviceid) +
5      COUNT(DISTINCT beingplaces.beingid) +
6      COUNT(DISTINCT studyplaces.studyid) +
7      COUNT(DISTINCT workplaces.workid) as count_of_transfers
8    FROM users
9    JOIN serviceplaces ON users."servicePlace" = serviceplaces.serviceid AND serviceplaces.enddate >= CURRENT_DATE - INTERVAL '6 months'
10   JOIN beingplaces ON users."beingPlace" = beingplaces.beingid AND beingplaces.enddate >= CURRENT_DATE - INTERVAL '6 months'
11   JOIN studyplaces ON users."studyPlace" = studyplaces.studyid AND studyplaces.enddate >= CURRENT_DATE - INTERVAL '6 months'
12   JOIN workplaces ON users."workPlace" = workplaces.workid AND workplaces.enddate >= CURRENT_DATE - INTERVAL '6 months'
13   GROUP BY users.id, users.firstname, users.lastname
14 )
```

Data Output | Messages | Notifications

```
CREATE VIEW

Query returned successfully in 58 msec.
```

У запиті міститься JOIN, тому таблиця не мала б бути змінюваною, бо вона порушує одне з правил написаних вище, але потрібно це перевірити.

Створене незмінюване представлення

```
Query  Query History
1  select * from non_updatable_view
```

Data Output | Messages | Notifications

	id bigint	firstname text	lastname text	count_of_transfers bigint
1	1	PABLO	Protskiv	4
2	2	Oleksandr	Zhenchenko	4
3	3	Olena	Hatala	4
4	4	Sofiia	Hoshko	4
5	5	Yarema	Tymchyshyn	4

Спроба додати кортеж за допомогою insert:

Query Query History

```
1  insert into non_updatable_view values (6, 'Nazarii', 'Protskiv', 5)
```

Data Output Messages Notifications

```
ERROR: ПОМИЛКА:  вставити дані в подання "non_updatable_view" не можна
DETAIL:  Подання які містять GROUP BY не оновлюються автоматично.
HINT:  Щоб подання допускало додавання даних, встановіть тригер INSTEAD OF INSERT або безумовне правило ON INSERT DO INSTEAD.


SQL state: 55000
```

На скріні видно, що додати кортеж у це представлення неможливо.

Віртуальна таблиця №3 (Матеріалізована):

Query Query History

```
1  create materialized view materialized_view as
2  (
3    select * from commentary
4  )
```

Data Output Messages Notifications

```
SELECT 9

Query returned successfully in 108 msec.
```

Вище скрін прикладу матеріалізованого представлення. Суть цих представлень в тому, що вони не оновлюються автоматично і на відміну від звичайних представлень, займають певну пам’ять. Зазвичай їх використовують у випадках, коли дані з таблиці потрібно отримувати часто, але при тому ці вони не часто оновлюються. Дані такого типу представлень можуть бути оновлені вручну.

Query Query History

```
1  select * from materialized_view
```

Data Output Messages Notifications

	commentary_id text	userid bigint	message_id text	commentary_text text
1	&21	2	#11	Wow! Cool! 21
2	&22	2	#11	Wow! Cool! 22
3	&23	2	#11	Wow! Cool! 23
4	&41	4	#31	Wow! Cool! 41
5	&42	4	#31	Wow! Cool! 42
6	&43	4	#51	Wow! Cool! 43
7	&51	5	#11	Wow! Cool! 51
8	&52	5	#11	Wow! Cool! 52
9	&53	5	#11	Wow! Cool! 53

На цьому скріні показано усі дані, які є у новоствореному представленні. Тепер спробуємо додати щось у нашу таблицю commentary, з якої це було створене дане представлення, після чого спробуємо знову отримати усі дані з даного представлення.

Додаємо кортеж у commentary за допомогою insert:

Query | Query History

```
1  insert into commentary
2  values ('&24', 2, '#12', 'Wow! I like your post!')
```

Data Output | Messages | Notifications

```
INSERT 0 1

Query returned successfully in 64 msec.
```

Матеріалізоване представлення після операції INSERT.

Query | Query History

```
1  select * from materialized_view
```

Data Output | Messages | Notifications

	commentary_id text	userid bigint	message_id text	commentary_text text
1	&21	2	#11	Wow! Cool! 21
2	&22	2	#11	Wow! Cool! 22
3	&23	2	#11	Wow! Cool! 23
4	&41	4	#31	Wow! Cool! 41
5	&42	4	#31	Wow! Cool! 42
6	&43	4	#51	Wow! Cool! 43
7	&51	5	#11	Wow! Cool! 51
8	&52	5	#11	Wow! Cool! 52
9	&53	5	#11	Wow! Cool! 53

Як можна побачити зі скріншоту, дійсно нічого не змінилося в цьому матеріалізованому представленні. Для того щоб зміни все-таки внеслися в нього потрібно використати REFRESH або REFRESH … CONCURRENTLY.

Представлення після refresh-у:

Query | Query History

```
1  refresh materialized view materialized_view;
2  select * from materialized_view;
```

Data Output | Messages | Notifications

	commentary_id text	userid bigint	message_id text	commentary_text text
1	&21	2	#11	Wow! Cool! 21
2	&22	2	#11	Wow! Cool! 22
3	&23	2	#11	Wow! Cool! 23
4	&41	4	#31	Wow! Cool! 41
5	&42	4	#31	Wow! Cool! 42
6	&43	4	#51	Wow! Cool! 43
7	&51	5	#11	Wow! Cool! 51
8	&52	5	#11	Wow! Cool! 52
9	&53	5	#11	Wow! Cool! 53
10	&24	2	#12	Wow! I like your post!

Видно, що воно оновилось і зміни тепер внеслися в нього.

Висновок: ознайомився з поняттям віртуальні таблиці SQL, їх створенням та застосуванням. Написав власні 3 віртуальні таблиці.