Министерство образования и науки РФ

Санкт-Петербургский Политехнический университет Петра Великого

Институт компьютерных наук и кибербезопасности

Высшая школа программной инженерии

ОТЧЕТ

по лабораторной работе вариант №1

Контроль целостности данных

по дисциплине «Системы управления базы данных»

|  |  |
| --- | --- |
| Выполнил  студент гр. 5130904/10104  Преподаватель | Баженов Кирилл Максимович.  Прокофьев О. В. |

Санкт-Петербург

2024

**Практическое задание №1.4 Контроль целостности данных**

**Постановка задачи:**

Практическое задание посвящено контролю целостности данных, который

производится с помощью механизма транзакций и триггеров. Транзакции позволяют

рассматривать группу операций как единое целое, либо отрабатывают все операции,

либо ни одной. Это позволяет избегать несогласованности данных. Триггеры

позволяют проверять целостность данных в момент выполнения транзакций,

поддерживать целостность, внося изменения, и откатывать транзакции, приводящие к

потере целостности.

Необходимо подготовить SQL-скрипты для проверки наличия аномалий (потерянных

изменений, грязных чтений, неповторяющихся чтений, фантомов) при параллельном

исполнении транзакций на различных уровнях изолированности SQL/92 (READ

UNCOMMITTED, READ COMMITTED, REPEATABLE READ, SERIALIZABLE).

Подготовленные скрипты должны работать с таблицей, содержащей 3-4 атрибута и до

30 строк. Для проверки наличия аномалий потребуются два параллельных сеанса,

операторы в которых выполняются пошагово:

* Установить в обоих сеансах уровень изоляции READ COMMITED/READ

UNCOMMITTED (для postgresql это синонимы). Выполнить сценарии проверки

наличия аномалий потерянных изменений и грязных чтений.

* Установить в обоих сеансах уровень изоляции READ COMMITTED. Выполнить

сценарии проверки наличия аномалий грязных чтений и неповторяющихся чтений.

* Установить в обоих сеансах уровень изоляции REPEATABLE READ. Выполнить

сценарии проверки наличия аномалий неповторяющихся чтений и фантомов,

сериализации.

* Установить в обоих сеансах уровень изоляции SERIALIZABLE. Выполнить

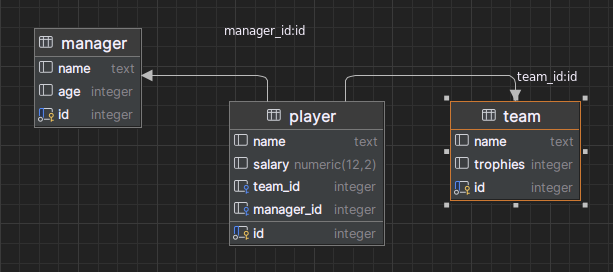
сценарий проверки фиксации аномалии сериализации.

Необходимо составить скрипт для создания 1-2х триггеров, а также подготовить

несколько запросов для проверки и демонстрации его полезных свойств:

* Изменение данных для сохранения целостности.
* Проверка транзакций и их откат в случае нарушения целостности.

**Архитектура базы данных**



**Транзакции**

*READ COMMITED/READ UNCOMITTED*

**Грязное чтение**

|  |  |
| --- | --- |
| Transaction A | Transaction B |
| begin transaction isolation level read committed; | begin transaction isolation level read committed; |
|  | select salary from player where id = 1; |
| update player set salary = 300000 where id = 1; |  |
|  | select salary from player where id = 1; |

В PostgreSQL не реализован уровень ReadUncomitted, поэтому грязного чтения нет.

**Неповторяющееся чтение**

|  |  |
| --- | --- |
| Transaction A | Transaction B |
| begin transaction isolation level read committed; | begin transaction isolation level read committed; |
|  | select salary from player where id = 1; |
| update player set salary = 500000 where id = 1; |  |
| commit; |  |
|  | select salary from player where id = 1; |

**Потерянное обновление**

|  |  |
| --- | --- |
| Transaction A | Transaction B |
| begin transaction isolation level read committed; | begin transaction isolation level read committed; |
| select salary from player where id = 1; |  |
| update player set salary = salary + 50000 where id = 1; |  |
|  |  |
|  | select salary from player where id = 1; |
|  | update player set salary = salary - 100000 where id = 1;  ждет пока транзакция A, не зафиксирует свои изменения |

Lost Update не поддерживается ни на одном уровне транзакций в PostgreSql.

*REPEATABLE READ*

**Неповторяющееся чтение**

|  |  |
| --- | --- |
| Transaction A | Transaction B |
| begin transaction isolation level repeatable read; | begin transaction isolation level repeatable read; |
|  | elect salary from player where id = 1; |
| update player set salary = salary + 50000 where id = 1; |  |
| commit; |  |
|  | select salary from player where id = 1; |

**Фантомное чтение**

|  |  |
| --- | --- |
| Transaction A | Transaction B |
| begin transaction isolation level repeatable read; | begin transaction isolation level repeatable read; |
|  | select count(\*) from player;  #count=20 |
| insert into player(name, salary, team\_id, manager\_id)  values ('Timofey Mozgov', 230000.00, 2, 2); |  |
| commit; |  |
|  | select count(\*) from player;  #count=20 |

**Аномалия сериализации**

|  |  |
| --- | --- |
| Transaction A | Transaction B |
| begin transaction isolation level repeatable read; | begin transaction isolation level repeatable read; |
| select \* from player where team\_id = 2; | select \* from player where team\_id = 1; |
| insert into player (name, salary, team\_id) VALUES ('Yura Babin', 1200.00, 1); | insert into player (name, salary, team\_id) VALUES ('Vladimir Babin', 1200.00, 2); |
| commit; | commit; |

*SERIALIZATION*

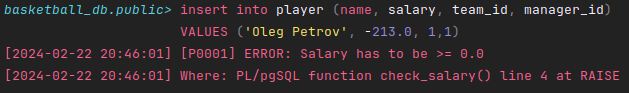
***Аномалия сериализации***

|  |  |
| --- | --- |
| Transaction A | Transaction B |
| begin transaction isolation level serializable; | begin transaction isolation level serializable; |
| select \* from player where team\_id = 2; | select \* from player where team\_id = 1; |
| insert into player (name, salary, team\_id) VALUES ('Yura Babin', 1200.00, 1); | insert into player (name, salary, team\_id) VALUES ('Vladimir Babin', 1200.00, 2); |
|  | commit; |
| Commit;  [40001] ERROR: could not serialize access due to read/write dependencies among transactions |  |

**Триггер**

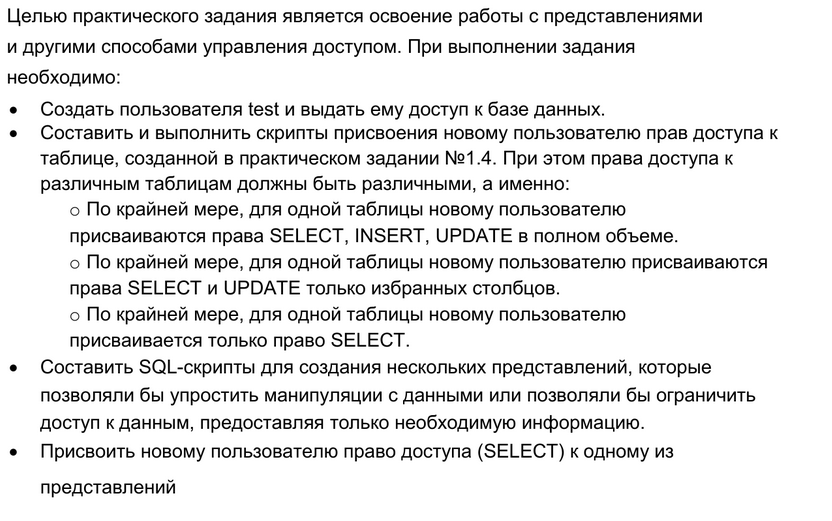
Данный триггер не повзоляет внести зарплату меньше 0.

create or replace function *check\_salary*() returns trigger language plpgsqlas$$begin if new.salary < 0.0 then raise exception 'Salary has to be >= 0.0'; end if; return old;end$$;create trigger salary\_trigger before insert on player for each rowexecute procedure *check\_salary*();



**Практическое задание №2.1 Управление доступом.**

**Постановка задачи:**

Создание пользователя происходит с помощью команды createuser:  
*createuser -h localhost -p 5432 -U stewie -P test*

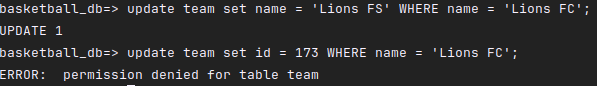
Теперь от имени пользователя Postgres назначаем права доступа в соответствии с заданием:

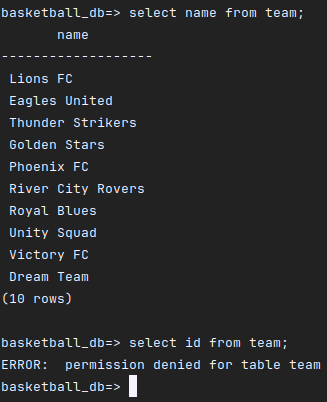
*GRANT SELECT, UPDATE, INSERT ON player to test;*

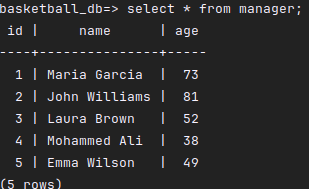
*GRANT SELECT(name), UPDATE(name) ON team to test;*

*GRANT SELECT ON manager to test;*

Проверим работу прав доступа:

****

****

****

Скрипт создания представлений:

*create view player\_view as*

*select name, salary*

*from player;*

*create view team\_view as*

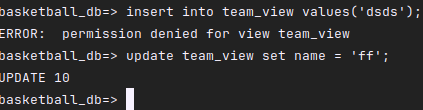
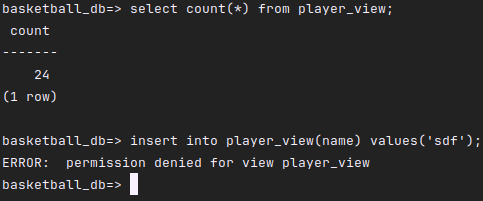
*select name*

*from team;*

Передача прав доступа для представлений:

*GRANT SELECT on player\_view to test;*

*GRANT UPDATE(name) on team\_view to test;*



Вызов команды \dp

