Федеральное государственное автономное образовательное учреждение высшего образования национальный исследовательский университет ИТМО

Факультет программной инженерии и компьютерной техники

Дисциплина «базы данных»

Отчёт

По лабораторной работе №5

Исполнитель:
Назирджонов Некруз
группа: P33211

Санкт-Петербург
2023

Триггер для проверки даты начала экскурсии:

```
CREATE OR REPLACE FUNCTION check_excursion_start_date()
   RETURNS TRIGGER AS $$
BEGIN
   IF NEW.start_datetime < NOW() THEN
       RAISE EXCEPTION 'Дата начала экскурсии не может быть в
прошлом';
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER check_excursion_start_date_trigger
   BEFORE INSERT ON Excursion
   FOR EACH ROW
EXECUTE FUNCTION check_excursion_start_date();
```

добавлял триггер, который будет проверять, чтобы стоимость билета не превышала 1000:

```
-- Триггер для проверки стоимости билета
CREATE OR REPLACE FUNCTION check_max_ticket_cost()
   RETURNS TRIGGER AS $$
BEGIN

   IF NEW.cost > 1000.00 THEN
       RAISE EXCEPTION 'Стоимость билета не может превышать
1000';
   END IF;

   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER check_max_ticket_cost_trigger
   BEFORE INSERT ON Ticket
   FOR EACH ROW
EXECUTE FUNCTION check_max_ticket_cost();
```

Функция для получения информации о туре по его идентификатору:

```
CREATE OR REPLACE FUNCTION get_tour_info(
    IN p_tour_id INT
)
    RETURNS TABLE (
                        tour_name VARCHAR(100),
                        tour_description TEXT,
                        route_name VARCHAR(100),
                        excursion_name VARCHAR(100),
                        booking_status VARCHAR(20)
                    )
AS $$
BEGIN
    RETURN QUERY
        SELECT
            t.name AS tour_name,
            t.description AS tour_description,
            tr.name AS route_name,
            e.name AS excursion_name,
            b.status AS booking_status
        FROM
            Tour t
```

```
                JOIN TimeRoute tr ON t.time_route_id =
tr.route_id

                JOIN Excursion e ON t.excursion_id =
e.excursion_id

                JOIN Booking b ON t.booking_id = b.booking_id

        WHERE

                t.tour_id = p_tour_id;

END;

$$ LANGUAGE plpgsql;
```

триггер для проверки максимального времени маршрута:

```
CREATE OR REPLACE FUNCTION check_max_route_duration()

    RETURNS TRIGGER AS $$

BEGIN

    IF NEW.duration > 10 THEN

        RAISE EXCEPTION 'Длительность временного маршрута не
может превышать 10 дней';

    END IF;

    RETURN NEW;

END;

$$ LANGUAGE plpgsql;



CREATE TRIGGER check_max_route_duration_trigger

    BEFORE INSERT ON TimeRoute
```

```
    FOR EACH ROW

EXECUTE FUNCTION check_max_route_duration();
```

Триггер для проверки статуса бронирования:

Проверьте, что статус бронирования (`status`) находится в списке допустимых значений.

```
CREATE OR REPLACE FUNCTION check_booking_status()
    RETURNS TRIGGER AS $$
BEGIN
    IF NEW.status NOT IN ('Забронировано', 'Подтверждено',
'Отменено') THEN
        RAISE EXCEPTION 'Недопустимый статус бронирования';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER check_booking_status_trigger
    BEFORE INSERT ON Booking
    FOR EACH ROW
EXECUTE FUNCTION check_booking_status();
```

Триггер для проверки корректности стоимости отеля:

Проверяет, что стоимость за ночь в отеле не отрицатеCREATE OR REPLACE FUNCTION check_valid_hotel_cost()

```
    RETURNS TRIGGER AS $$
BEGIN
    IF NEW.cost_per_night < 0 THEN
        RAISE EXCEPTION 'Стоимость отеля не может быть
отрицательной';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER check_valid_hotel_cost_trigger
    BEFORE INSERT ON Hotel
    FOR EACH ROW
EXECUTE FUNCTION check_valid_hotel_cost();
```

Триггер для проверки уникальности названия маршрута:

Гарантирует, что в таблице `TimeRoute` не может быть двух маршрутов с одинаковым названием.

```
CREATE OR REPLACE FUNCTION check_unique_route_name()
    RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (SELECT 1 FROM TimeRoute WHERE name = NEW.name) THEN
        RAISE EXCEPTION 'Маршрут с таким названием уже существует';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER check_unique_route_name_trigger
    BEFORE INSERT ON TimeRoute
    FOR EACH ROW
EXECUTE FUNCTION check_unique_route_name();
```

Функция, которая найдет максимальную продолжительность тура в таблице Tour. Предположим, что продолжительность тура хранится в столбце Duration

```
CREATE OR REPLACE FUNCTION find_max_tour_duration()

    RETURNS INT AS $$

DECLARE

    max_duration INT;

BEGIN

    SELECT MAX(duration) INTO max_duration

    FROM TimeRoute;
```

```
    RETURN max_duration;

END;

$$ LANGUAGE plpgsql;
```

процедуру, которая увеличит стоимость всех билетов в туре на определенное значение.

Создадал триггер, который будет отслеживать изменения в столбце "cost_per_night" таблицы "Hotel" и записывать эти изменения в отдельную таблицу "PriceChangeHistory". Эта таблица содержит историю изменений стоимости номеров в отелях.

```
CREATE TABLE IF NOT EXISTS PriceChangeHistory (

                                                    change_id
SERIAL PRIMARY KEY,

                                                    hotel_id INT,

                                                    old_cost
DECIMAL(10, 2),

                                                    new_cost
DECIMAL(10, 2),

change_datetime TIMESTAMP

);




CREATE OR REPLACE FUNCTION log_price_change()

    RETURNS TRIGGER AS $$

BEGIN

    IF NEW.cost_per_night <> OLD.cost_per_night THEN
```

```
        INSERT INTO PriceChangeHistory (hotel_id, old_cost,
new_cost, change_datetime)

        VALUES (NEW.hotel_id, OLD.cost_per_night,
NEW.cost_per_night, CURRENT_TIMESTAMP);

    END IF;

    RETURN NEW;

END;

$$ LANGUAGE plpgsql;



CREATE TRIGGER hotel_price_change_trigger

    AFTER UPDATE OF cost_per_night ON Hotel

    FOR EACH ROW

EXECUTE FUNCTION log_price_change();
```

```
CREATE OR REPLACE PROCEDURE increase_ticket_prices(tour_id
INT, increase_amount DECIMAL(10, 2))

AS $$

BEGIN

    UPDATE Ticket

    SET cost = cost + increase_amount

    WHERE tour_id = increase_ticket_prices.tour_id;

END;

$$ LANGUAGE plpgsql;
```

Процедура, которая повышает цену всех типов билетов на определённую сумму

```
CREATE OR REPLACE PROCEDURE
increase_all_ticket_prices(increase_amount DECIMAL(10, 2))

AS $$

BEGIN

    UPDATE Ticket

    SET cost = cost + increase_amount;

END;

$$ LANGUAGE plpgsql;
```

Выявление используемых индексов:

Определите индексы, которые часто используются:

Этот запрос выводит десять индексов с наибольшим количеством сканирований.

```
SELECT indexrelname, idx_scan, idx_tup_read

FROM pg_stat_all_indexes

WHERE schemaname = 'public'

ORDER BY idx_scan DESC

LIMIT 10;
```

```
-- Наиболее используемые индексы
SELECT indexrelname, idx_scan, idx_tup_read
FROM pg_stat_all_indexes
WHERE schemaname = 'public'
ORDER BY idx_scan DESC
LIMIT 5;

-- Наиболее используемые столбцы
```

```sql
SELECT column_name, count(*) AS column_usage_count
FROM information_schema.columns
WHERE table_schema = 'public'
GROUP BY column_name
ORDER BY column_usage_count DESC
LIMIT 5;
```

Сценарий: Поиск Туров по Экскурсии и Маршруту:

Индекс на столбцах `excursion_id` и `time_route_id` в таблице `Tour`:

```sql
CREATE INDEX idx_tour_excursion_time_route ON
Tour(excursion_id, time_route_id);
```

Сценарий: Поиск Бронирований по Статусу и Времени:

- Индекс на столбцах `status`, `start_datetime`, и `end_datetime` в таблице `Booking`:

```sql
CREATE INDEX idx_booking_status_datetime ON Booking(status,
start_datetime, end_datetime);
```

ценарий: Поиск Гидов по Имени и Фач]милии:

- Индекс на столбцах `first_name` и `last_name` в таблице `Guide`:

```sql
CREATE INDEX idx_guide_name ON Guide(first_name, last_name);
```

Сценарий: Поиск Билетов по Типу и Стоимости:

- Индекс на столбцах `ticket_type` и `cost` в таблице `Ticket`:

```sql
CREATE INDEX idx_ticket_type_cost ON Ticket(ticket_type,
cost);
```