**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc125995756)

[Теоретическая часть 5](#_Toc125995757)

[Практическая часть 10](#_Toc125995758)

[1. Учет действий над таблицами 11](#_Toc125995759)

[2. Запрос на вывод данных на момент времени 13](#_Toc125995760)

[3. Восстановление данных на момент времени 15](#_Toc125995761)

[4. Создание процедуры, которая, принимая на вход название таблицы, создает для неё систему логирования 17](#_Toc125995762)

[Дополнительное задание 3. Реализация партицирования таблицы с логами по времени 18](#_Toc125995763)

[Дополнительное задание 5. Вывод статистики по тому, как часто меняются те или иные колонки 21](#_Toc125995764)

[ЗАКЛЮЧЕНИЕ 24](#_Toc125995765)

# Введение

Предметной областью данной работы является банк по работе с физическими лицами.

Современный этап развития российской экономики характеризуется постоянным ростом спроса на банковские услуги со стороны физических лиц. В условиях высокой конкуренции банк заинтересован в сохранности данных, а также возможности предоставить клиентам оперативную помощь в решении их вопросов, касающихся банковских счетов. Возможны ситуации, когда сотрудники банка по невнимательности или намеренно компрометируют данные, потому что имеют доступ к счетам клиентов. Поэтому важно отслеживать изменения данных для обеспечения контроля и возможности восстановления базы данных на определенный момент времени.

*Приведем описание кейса:*

Клиент преклонного возраста потерял банковскую карту и написал заявление об утере в банк. При этом у него не было доступа к мобильному приложению банка, и, чтобы узнать текущий баланс карты, ему нужно было прийти к сотрудникам банка для установления номера банковского счета и изготовления новой карты. Некоторые сотрудники, узнав, что можно воспользоваться плохой памятью клиента, стали переводить деньги с утерянной карты клиента себе на счет и сообщать измененный текущий баланс карты клиенту. Так как баланс карты был известен только сотруднику банка и нигде не сохранялся, установить такое мошенничество было тяжело. Однако, когда управляющие банка начали получать жалобы, было принято решение сохранять информацию о текущем балансе клиента после каждой транзакции. Таким образом, можно было посмотреть все изменения на карте клиента до его прихода в банк, чтобы узнать, какая сумма там хранилась на момент утери. Также было принято решение сохранять информацию о новой карте, внесенной в базу данных, и старой карте, которую нужно было удалить из базы данных, чтобы контролировать изменения текущего баланса при этих действиях.

Цель работы: реализовать подсистему контроля и восстановления состояния базы данных на заданный момент времени в PostgreSQL.

Для выполнения поставленной цели необходимо выполнить ряд задач:

1. Исследовать возможные способы реализации подсистемы контроля и восстановления базы данных.
2. Написать скрипт, позволяющий производить учет действий над таблицами.
3. Создать запрос на вывод данных на момент времени.
4. Реализовать восстановление данных на момент времени.
5. Создать процедуру, которая, принимая на вход название таблицы, создает для неё систему логирования.
6. Реализовать партицирование таблицы с логами по времени и расписать, почему это может быть полезно.
7. Вывести статистику по тому, как часто меняются те или иные колонки.

# Теоретическая часть

Безопасность базы данных является одной из важных проблем администрирования. Существует несколько основных подходов к созданию подсистемы контроля и восстановления базы данных PostgreSQL.

1. Утилита pg\_dump - извлекает из PostgreSQL базу данных в скрипт-файл или в архив.

Идея, стоящая за этим методом, заключается в генерации текстового файла с командами SQL, которые при выполнении на сервере пересоздадут базу данных в том же самом состоянии, в котором она была на момент выгрузки.

Преимуществом данного способа является то, что вывод pg\_dump обычно можно загрузить в более новые версии PostgreSQL, в то время как резервная копия на уровне файловой системы и непрерывное архивирование зависят от версии сервера. Метод с применением pg\_dump будет работать при переносе базы данных на другую машинную архитектуру, например, при переносе с 32-битной на 64-битную версию сервера.

Дампы, создаваемые pg\_dump, являются внутренне согласованными, то есть, дамп представляет собой снимок базы данных на момент начала запуска pg\_dump. pg\_dump не блокирует другие операции с базой данных во время своей работы, исключением являются некоторые формы команды ALTER TABLE.

Текстовые файлы, созданные pg\_dump, предназначаются для последующего чтения программой psql. Дампы, выгруженные не в текстовом формате, восстанавливаются утилитой pg\_restore.

Программа pg\_dump выгружает только одну базу данных в один момент времени и не включает в дамп информацию о ролях и табличных пространствах (так как это информация уровня кластера, а не самой базы данных). Для удобства создания дампа всего содержимого кластера баз данных существует утилита pg\_dumpall, которая делает резервную копию всех баз данных кластера, а также сохраняет данные уровня кластера, такие как роли и определения табличных пространств.

1. Резервное копирование файловой системы.

Альтернативной стратегией является непосредственное копирование файлов, в которых PostgreSQL хранит содержимое базы данных. Однако, существуют два ограничения, которые делают этот метод непрактичным или как минимум менее предпочтительным по сравнению с pg\_dump:

* Необходимо остановить сервер баз данных. Также сервер нужно будет остановить и перед восстановлением данных.
* Нельзя копировать отдельные таблицы или базы данных в соответствующих файлах или каталогах. Копирование на уровне файловой системы будет работать, только если выполняется полное копирование и восстановление всего кластера баз данных.

Размер копии на уровне файлов обычно больше, чем дампа SQL. Однако копирование на уровне файлов может выполняться быстрее.

1. Непрерывное архивирование и восстановление на момент времени (Point-in-Time Recovery, PITR)

Всё время в процессе работы PostgreSQL ведёт журнал предзаписи (WAL). Все транзакции, такие как INSERT, UPDATE, DELETE, записываются в WAL до того, как клиент получает результат транзакции. Прежде всего, журнал существует для безопасного восстановления после краха сервера: если происходит крах, целостность СУБД может быть восстановлена в результате «воспроизведения» записей, зафиксированных после последней контрольной точки.

Программы pg\_dump и pg\_dumpall не создают копии на уровне файловой системы и не могут применяться как часть решения по непрерывной архивации. Создаваемые ими копии являются логическими и не содержат информации, необходимой для воспроизведения WAL.

Как и обычное резервное копирование файловой системы, этот метод позволяет восстанавливать только весь кластер баз данных целиком, но не его части. Кроме того, для архивов требуется большое хранилище: базовая резервная копия может быть объёмной, а нагруженные системы будут генерировать многие мегабайты трафика WAL, который необходимо архивировать. Тем не менее, этот метод предпочитается во многих ситуациях, где необходима высокая надёжность.

1. Логирование с помощью триггерной процедуры.

Методы на основе триггеров предполагают создание триггеров аудита в базе данных для регистрации всех событий, связанных с методами INSERT, UPDATE и DELETE. Когда функция срабатывает как триггер, в блоке верхнего уровня автоматически создаются несколько специальных переменных с определенными данными, которые можно использовать для последующего контроля и восстановления базы данных. Еще одним плюсом является то, что хранение сведений о событиях в базе данных позволяет при их анализе использовать возможности языка SQL – фильтрацию, поиск, и сортировку.

Итак, исследовав различные способы контроля и восстановления базы данных на определенный момент времени, можем сделать вывод, что в рамках данной курсовой работы целесообразно использовать метод триггерной процедуры, так как с помощью него удобнее всего восстанавливать данные о банковской карте на определенный момент времени и контролировать все изменения текущего баланса. Остальные методы необходимы, чтобы полностью восстанавливать копию таблицы базы данных или всей базы данных, что используется при полной утери информации и для чего потребуются большие объемы свободной памяти, ведь текущий баланс банковской карты клиента может меняться в любой момент времени и по многу раз, создавать при каждом случае копию всей базы данных не рационально.

Рассмотрим логирование изменений с помощью триггерной процедуры более подробно.

Логирование позволяет ответить на вопросы, что происходило, когда и при каких обстоятельствах. Без логов сложно понять, из-за чего появляется ошибка, если она возникает периодически и только при определенных условиях. При этом важно, чтобы пользователи, для которых ведутся логи, не имели к ним доступа. Ведение журнала логов является распространенным требованием во многих корпоративных базах данных с целью регистрации всех изменений.

Контроль изменений с помощью логов позволяет решить следующие задачи:

* Узнать кто, когда и откуда производил изменения данных, какие изменения;
* Настроить систему уведомлений в случае подозрительных действий в базе данных. Например, если в субботу ночью появится учетная запись с правами суперпользователя, система сразу зарегистрирует это событие и пришлет уведомление. Инженеры уточнят легитимность новой записи, чтобы предотвратить попытку несанкционированного доступа;
* Восстановить некоторые данные в соответствии с таблицей логов на определенный момент времени.

Журнал логов может помочь достичь нескольких дополнительных целей, таких как индивидуальная подотчетность, обнаружение вторжений, анализ проблем, сбор статистики. Примеры задач, решаемых с помощью логирования:

* Выявление причин возникновения технических проблем, сбоев. К примеру, можно восстановить последовательность событий при инциденте, сделать выводы и выработать решения, которые предотвратят появление таких инцидентов в будущем;
* Сбор статистики по нагрузке базы данных;
* Установление индивидуальной ответственности: пользователи базы данных будут нести личную ответственность за свои действия, которые отслеживаются журналом аудита, потому что в нем фиксируется, кто внес изменения.

Логирование - хороший способ эффективно контролировать состояние базы данных. Вместе с сервисами мониторинга логирование существенно экономит время при расследовании тех или иных инцидентов.

Таким образом, основная идея реализации логирования заключается в том, чтобы создать уникальную таблицу истории всех изменений. Для восстановления данных из логов можно использовать средства языка запросов SQL и процедурное расширение языка SQL - PL/pgSQL. В практической части работы будет реализован один из способов восстановления.

# Практическая часть

Основной сущностью, изменения в которой нужно отслеживать, является «Банковская карта». Она содержит данные о карте, привязанной к некоторому счету: номер карты, имя и фамилию клиента, которые могут измениться, текущий баланс, зависящий от проведенных транзакций, и некоторую другую информацию. В случае утери карты или по истечении ее срока действия изготавливают новую карту и соответственно вносят новую информацию в базу данных. При этом необходимо иметь доступ к предыдущим данным, чтобы обеспечить качественное предоставление услуг и в случае ошибки суметь вернуть исходные данные клиента, к примеру, баланс карты до ее утери.

ER-диаграмма:

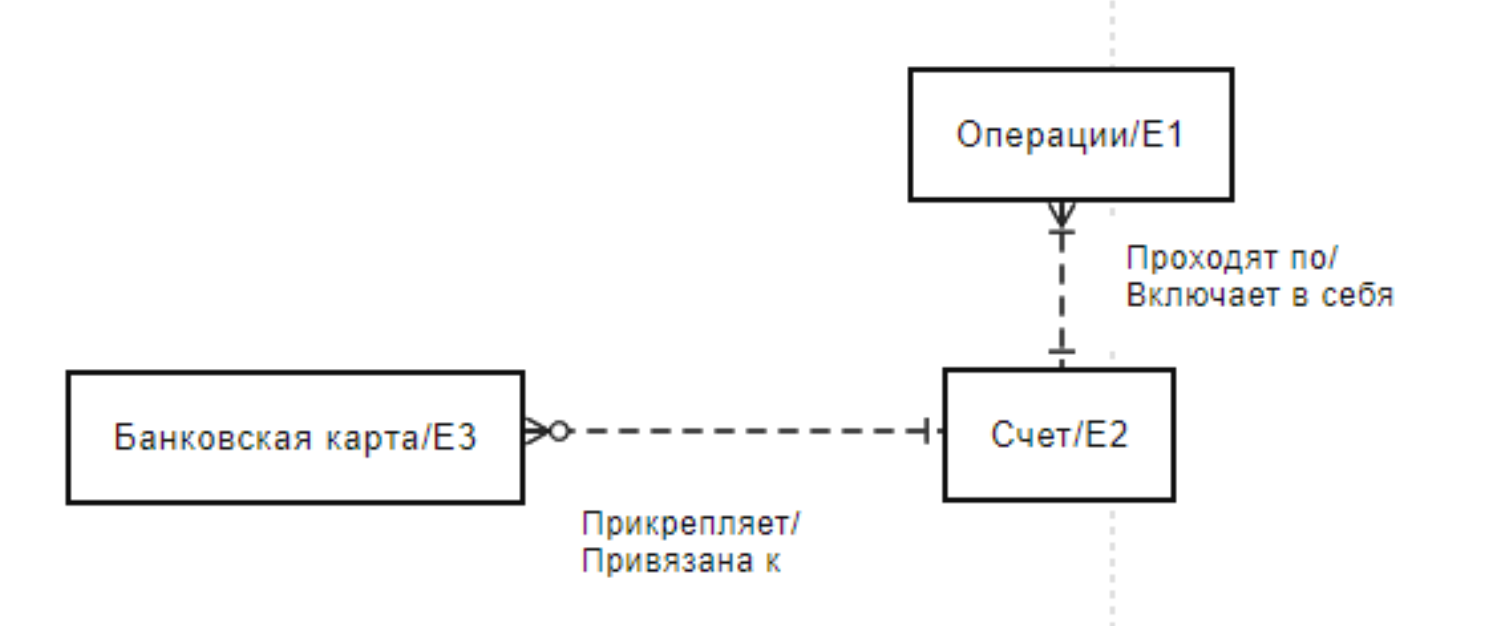


Рисунок . ER-диаграмма

FA-диаграмма имеет следующий вид:

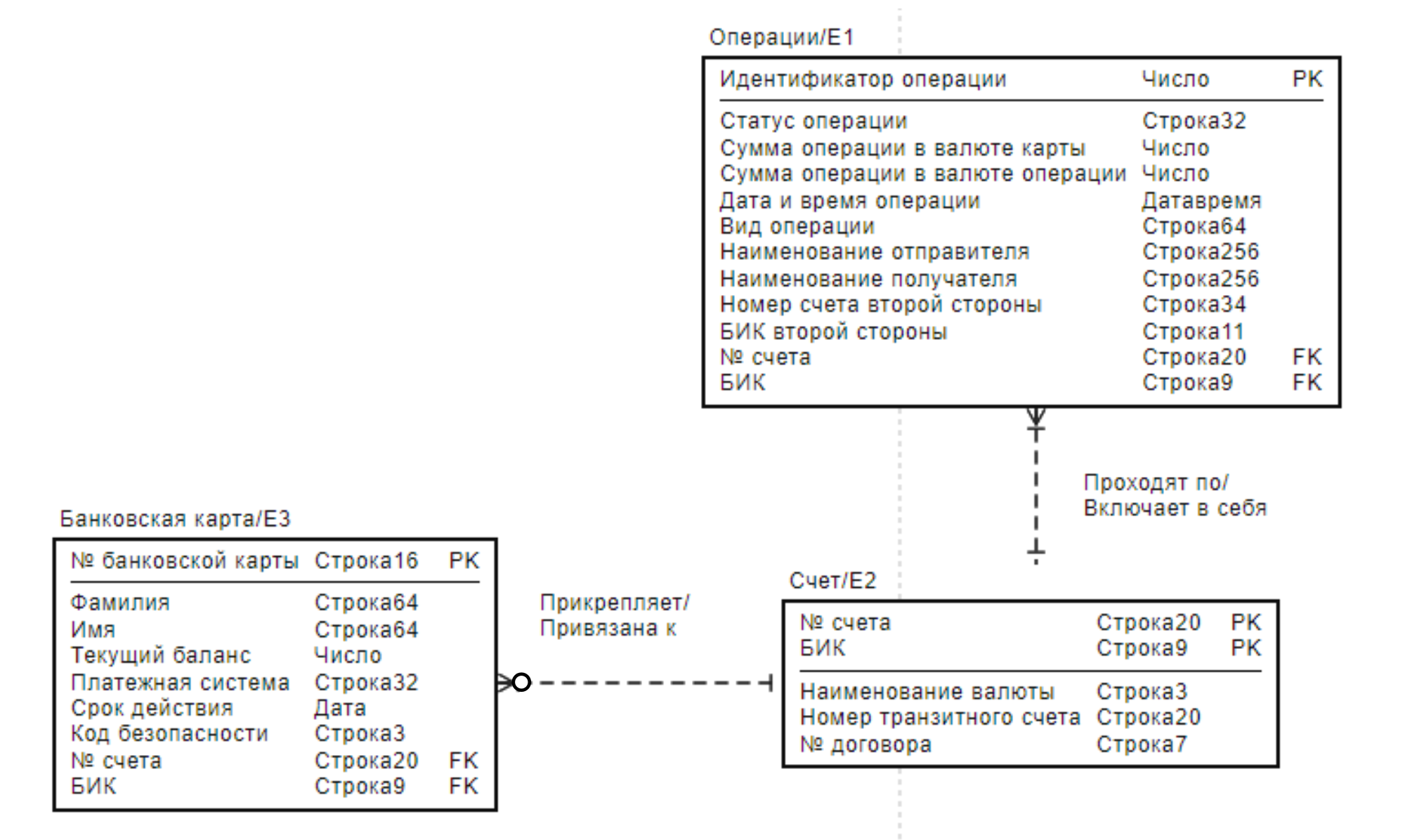


Рисунок . FA-диаграмма

Подготовительным этапом является создание и заполнение базы данных банка по работе с физическими лицами, для которой будем делать подсистему восстановления и контроля. Код создания приведен в приложении А.

## Учет действий над таблицами

Для того, чтобы логировать все изменения, происходящие с сущностью «Банковская карта», создадим таблицу card\_audit в схеме borisova, в которую будем вносить изменения. Таблица будет состоять из следующих столбцов:

* Столбцы, продублированные из таблицы «card», которые будут содержать соответственно для каждой колонки сохраняемое значение;
* Вид оператора DML: “U” - update, “D” - delete, “I” - insert;
* Имя пользователя, который внес изменения;
* Дата и время изменения;
* Названия колонок таблицы, в которых произошли изменения для оператора UPDATE, для операторов INSERT и DELETE – значение “All”, то есть изменены все колонки.

Далее создадим триггерную функцию card\_audit\_information в схеме borisova. Объявим переменную update\_columns в декларативной части, в которую будем вносить названия колонок, в которых произошли изменения.

В основной части функции будем проверять условие, является ли переменная TG\_OP, содержащая информацию о текущей операции, вызвавшей триггерную функцию, одной из: UPDATE, INSERT, DELETE. Если вызван оператор DELETE – будем сохранять значения каждого столбца таблицы до удаления, если INSERT – сохранять вставленные новые значения, если UPDATE – сохранять старые значения до изменения, так как новые будут храниться в таблице card. Также в случае с оператором UPDATE будем проверять, какие именно столбцы были изменены и сохранять их названия в переменную update\_columns через пробел.

Для того, чтобы функция card\_audit\_information вызывалась каждый раз, когда будут вноситься изменения в таблицу – создадим триггер для каждой строки, срабатывающий после INSERT, UPDATE или DELETE в таблице borisova.card.

Итак, приведем код, описанный выше:

-- Создание таблицы для хранения логов

CREATE TABLE borisova.card\_audit

AS

SELECT \*

FROM borisova.card

WHERE false;

alter table borisova.card\_audit

add column operation char(1) NOT NULL,

add column user\_name varchar(32) NOT NULL,

add column change\_time timestamp NOT NULL,

add column name\_col\_changed varchar(128);

select \* from borisova.card\_audit;

-- Создание триггерной функции

create or replace function borisova.card\_audit\_information()

returns trigger as

$card\_audit$

declare

update\_columns text;

begin

if (TG\_OP = 'DELETE') THEN

INSERT INTO borisova.card\_audit VALUES(OLD.\*, 'D', current\_user, now(), 'All');

elsif (TG\_OP = 'UPDATE') THEN

update\_columns:='';

if old.card\_number<>new.card\_number then

update\_columns:='card\_number';

end if;

if old.card\_surname<>new.card\_surname then

update\_columns:=update\_columns||' card\_surname';

end if;

if old.card\_name<>new.card\_name then

update\_columns:=update\_columns||' card\_name';

end if;

if old.card\_balance<>new.card\_balance then

update\_columns:=update\_columns||' card\_balance';

end if;

if old.card\_pay\_system<>new.card\_pay\_system then

update\_columns:=update\_columns||' card\_pay\_system';

end if;

if old.card\_validity<>new.card\_validity then

update\_columns:=update\_columns||' card\_validity';

end if;

if old.card\_code<>new.card\_code then

update\_columns:=update\_columns||' card\_code';

end if;

if old.acc\_number<>new.acc\_number then

update\_columns:=update\_columns||' acc\_number';

end if;

if old.acc\_bic<>new.acc\_bic then

update\_columns:=update\_columns||' acc\_bic';

end if;

insert into borisova.card\_audit VALUES(OLD.\*, 'U', current\_user, now(), update\_columns);

elsif (TG\_OP = 'INSERT') THEN

insert into borisova.card\_audit VALUES (NEW.\*, 'I', current\_user, now(), 'All');

end if;

return null;

end;

$card\_audit$

language plpgsql;

-- Создание триггера

create trigger card\_audit\_trigger

after insert or update or delete on borisova.card

for each row

execute procedure borisova.card\_audit\_information();

## Запрос на вывод данных на момент времени

Запрос будет состоять из нескольких запросов, объединенных между собой с помощью union:

* Выберем те строки из таблицы borisova.card, которые не были изменены. Для этого с помощью left join объединим таблицу card (table 1 на рисунке 2) и таблицу логов borisova.card\_audit (table 2 на рисунке 2) по значению card\_number.

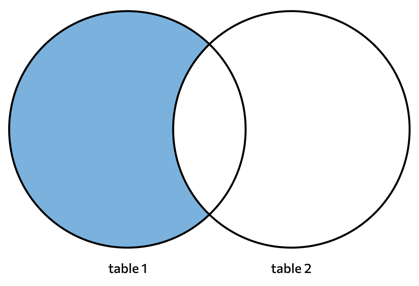


Рисунок 3. Left join c Null

* Выберем удаленные строки таблицы card из таблицы логов. Если какая-либо строка была удалена до момента восстановления – ее не должно быть в запросе на вывод, однако все строки, которые были удалены после назначенного времени восстановления, должны быть возвращены, поэтому их мы выбираем.
* Выберем строки из таблицы логов, которые были вставлены в таблицу card до момента восстановления, так как все строки, которые были вставлены после момента восстановления, не должны там быть. Также сделаем проверку на то, что вставленную строку не удалили до момента восстановления, потому что в таком случае ее не должно быть в запросе на вывод.
* Добавим строки, которые были изменены оператором UPDATE. Так как в логах мы сохраняем старые значения, новые значения находятся либо в последующих строках таблицы логов, либо в текущей таблице card. К примеру, если UPDATE какой-либо строки был в 19:00, а вывод данных назначен на 20:00, то нужно посмотреть был ли еще один UPDATE после 20:00, так как туда отправятся старые значения, установленные в 19:00. Либо, если UPDATE после 20:00 не было, то нужные нам значения хранятся в текущей таблице card.

Для реализации вывода строк из таблицы логов мы выбираем с помощью distinct единственное самое раннее по времени значение номера карты у строк с оператором UPDATE, произошедшего после выбранного времени вывода. Для реализации вывода строк из текущей таблицы card, делаем следующую проверку: если изменение произошло до времени вывода, и эту строку (с данным номером карты) не меняли после времени вывода, то добавляем ее из текущей таблицы card.

Приведем код ниже:

-- Строки таблицы card, которые не были изменены

select borisova.card.card\_number, borisova.card.card\_surname, borisova.card.card\_name, borisova.card.card\_balance, borisova.card.card\_pay\_system, borisova.card.card\_validity, borisova.card.card\_code, borisova.card.acc\_number, borisova.card.acc\_bic

from borisova.card

left join borisova.card\_audit on borisova.card.card\_number=borisova.card\_audit.card\_number

where borisova.card\_audit.card\_number is null

union

--Восстановление удаленных строк

select card\_number, card\_surname, card\_name, card\_balance, card\_pay\_system, card\_validity, card\_code, acc\_number, acc\_bic

from borisova.card\_audit

where operation='D' and change\_time>'2022-12-10 19:26:00'

union

--Восстановление вставленных строк

select a.card\_number, a.card\_surname, a.card\_name, a.card\_balance, a.card\_pay\_system, a.card\_validity, a.card\_code, a.acc\_number, a.acc\_bic

from borisova.card\_audit a

where a.operation='I' and a.change\_time<'2022-12-10 19:26:00' and a.card\_number not in

(

select r.card\_number

from borisova.card\_audit r

where r.operation='D' and r.change\_time<'2022-12-10 19:26:00' and a.change\_time<=r.change\_time

)

union

--Восстановление старых значений до update

(select distinct on(card\_number) card\_number, card\_surname, card\_name, card\_balance, card\_pay\_system, card\_validity, card\_code, acc\_number, acc\_bic

from borisova.card\_audit

where operation='U' and change\_time>'2022-12-10 19:26:00'

order by card\_number, change\_time asc)

union

select card\_number, card\_surname, card\_name, card\_balance, card\_pay\_system, card\_validity, card\_code, acc\_number, acc\_bic

from borisova.card c

where c.card\_number in

(select card\_number

from borisova.card\_audit a

where a.operation='U' and a.change\_time<'2022-12-10 19:26:00' and a.card\_number not in

(select card\_number

from borisova.card\_audit a

where a.operation='U' and a.change\_time>'2022-12-10 19:26:00'

)

)

order by card\_number;

## Восстановление данных на момент времени

Для восстановления данных на момент времени таблицу, созданную путем объединения запросов в предыдущем пункте, сохраним как card\_copy в схеме borisova. Далее удалим все значения из текущей таблицы card, а после вставим в нее все значения из таблицы card\_copy. Удалим копию таблицы.

Приведем код ниже:

--Сохраним таблицу, полученную для вывода данных на момент времени

create table borisova.card\_copy as

(

select borisova.card.card\_number, borisova.card.card\_surname, borisova.card.card\_name, borisova.card.card\_balance, borisova.card.card\_pay\_system, borisova.card.card\_validity, borisova.card.card\_code, borisova.card.acc\_number, borisova.card.acc\_bic

from borisova.card

left join borisova.card\_audit on borisova.card.card\_number=borisova.card\_audit.card\_number

where borisova.card\_audit.card\_number is null

union

select card\_number, card\_surname, card\_name, card\_balance, card\_pay\_system, card\_validity, card\_code, acc\_number, acc\_bic

from borisova.card\_audit

where operation='D' and change\_time>'2022-12-10 19:26:00'

union

select a.card\_number, a.card\_surname, a.card\_name, a.card\_balance, a.card\_pay\_system, a.card\_validity, a.card\_code, a.acc\_number, a.acc\_bic

from borisova.card\_audit a

where a.operation='I' and a.change\_time<'2022-12-10 19:26:00' and a.card\_number not in

(

select r.card\_number

from borisova.card\_audit r

where r.operation='D' and r.change\_time<'2022-12-10 19:26:00' and a.change\_time<=r.change\_time

)

union

(select distinct on(card\_number) card\_number, card\_surname, card\_name, card\_balance, card\_pay\_system, card\_validity, card\_code, acc\_number, acc\_bic

from borisova.card\_audit

where operation='U' and change\_time>'2022-12-10 19:26:00'

order by card\_number, change\_time asc)

union

select card\_number, card\_surname, card\_name, card\_balance, card\_pay\_system, card\_validity, card\_code, acc\_number, acc\_bic

from borisova.card c

where c.card\_number in

(select card\_number

from borisova.card\_audit a

where a.operation='U' and a.change\_time<'2022-12-10 19:26:00' and a.card\_number not in

(select card\_number

from borisova.card\_audit a

where a.operation='U' and a.change\_time>'2022-12-10 19:26:00'

)

)

order by card\_number

);

--Удалим все значения из текущей таблицы card

delete from borisova.card;

--Вставим значения из card\_copy в card

insert INTO borisova.card select \* FROM borisova.card\_copy;

--Удалим копию таблицы card

drop table borisova.card\_copy;

--Проверяем изменения

select \* from borisova.card;

## Создание процедуры, которая, принимая на вход название таблицы, создает для неё систему логирования

Создадим функцию sys\_log, которая на вход будет принимать название таблицы, сохранять его в переменную tab\_name и создавать для этой таблицы систему логирования.

Для этого будем использовать код, который создали в первом задании и выполнять его с помощью команды Execute, которая позволит динамически изменять команды и выполнять их внутри функции. Весь код оставим без изменений, но в качестве названия таблицы будем указывать переменную tab\_name.

Приведем код ниже:

--Создание функции для системы логирования

CREATE OR REPLACE FUNCTION borisova.sys\_log(tab\_name text)

returns text

AS $$

BEGIN

execute

--Таблица для хранения логов

'

CREATE TABLE if not exists '||tab\_name||'\_sys\_log

AS

SELECT \*

FROM '||tab\_name||'

WHERE false;

alter table '||tab\_name||'\_sys\_log

add column operation char(1) NOT NULL,

add column user\_name varchar(32) NOT NULL,

add column change\_time timestamp NOT NULL;

--Триггерная функция для логирования изменений

create or replace function '||tab\_name||'\_sys\_log\_func()

returns trigger as

$card\_audit$

begin

if (TG\_OP = ''DELETE'') THEN

INSERT INTO '||tab\_name||'\_sys\_log VALUES(OLD.\*, ''D'', current\_user, now());

elsif (TG\_OP = ''UPDATE'') THEN

insert into '||tab\_name||'\_sys\_log VALUES(OLD.\*, ''U'', current\_user, now());

elsif (TG\_OP = ''INSERT'') THEN

insert into '||tab\_name||'\_sys\_log VALUES (NEW.\*, ''I'', current\_user, now());

end if;

return null;

end;

$card\_audit$

language plpgsql;

--Триггер, срабатывающий при изменении таблицы

create trigger sys\_log\_trigger

after insert or update or delete on '||tab\_name||'

for each row

execute procedure '||tab\_name||'\_sys\_log\_func()

';

return tab\_name;

END; $$

LANGUAGE 'plpgsql';

-- Вызов функции для создания системы логирования

select \* from borisova.sys\_log('borisova.card');

## Дополнительное задание 3. Реализация партицирования таблицы с логами по времени

Партицированием данных называется разбиение одной большой логической таблицы на несколько меньших физических партиций. Так как партицирование можно реализовать только при создании таблицы, создадим новую таблицу логирования card\_audit\_partitioning.

По аналогии с первым заданием создадим триггерную функцию и триггер, которые будут логировать изменения таблицы card. Далее создадим таблицы-партиции, которые ссылаясь на основную таблицу, будут содержать логи за один день.

Приведем код ниже:

--Создание таблицы для хранения логов с партицированием по времени изменения

CREATE TABLE borisova.card\_audit\_partitioning

(

card\_number char(16),

card\_surname varchar(64),

card\_name varchar(64),

card\_balance real,

card\_pay\_system varchar(32),

card\_validity date,

card\_code char(3),

acc\_number char(20),

acc\_bic char(9),

operation char(1),

user\_name varchar(32),

change\_time timestamp,

name\_col\_changed varchar(128)

)

PARTITION BY RANGE (change\_time);

--Создание триггерной функции, записывающей изменения в таблицу логирования

create or replace function borisova.card\_audit\_part()

returns trigger as

$card\_audit\_part$

declare

update\_columns text;

begin

if (TG\_OP = 'DELETE') THEN

INSERT INTO borisova.card\_audit\_partitioning VALUES(OLD.\*, 'D', current\_user, now(), 'All');

elsif (TG\_OP = 'UPDATE') THEN

update\_columns:='';

if old.card\_number<>new.card\_number then

update\_columns:='card\_number';

end if;

if old.card\_surname<>new.card\_surname then

update\_columns:=update\_columns||' card\_surname';

end if;

if old.card\_name<>new.card\_name then

update\_columns:=update\_columns||' card\_name';

end if;

if old.card\_balance<>new.card\_balance then

update\_columns:=update\_columns||' card\_balance';

end if;

if old.card\_pay\_system<>new.card\_pay\_system then

update\_columns:=update\_columns||' card\_pay\_system';

end if;

if old.card\_validity<>new.card\_validity then

update\_columns:=update\_columns||' card\_validity';

end if;

if old.card\_code<>new.card\_code then

update\_columns:=update\_columns||' card\_code';

end if;

if old.acc\_number<>new.acc\_number then

update\_columns:=update\_columns||' acc\_number';

end if;

if old.acc\_bic<>new.acc\_bic then

update\_columns:=update\_columns||' acc\_bic';

end if;

insert into borisova.card\_audit\_partitioning VALUES(OLD.\*, 'U', current\_user, now(), update\_columns);

elsif (TG\_OP = 'INSERT') THEN

insert into borisova.card\_audit\_partitioning VALUES (NEW.\*, 'I', current\_user, now(), 'All');

end if;

return null;

end;

$card\_audit\_part$

language plpgsql;

--Создание триггера, срабатывающего при изменении таблицы

create trigger card\_audit\_partitioning\_tr

after insert or update or delete on borisova.card

for each row

execute procedure borisova.card\_audit\_part();

--Просмотр талицы логов

select \* from borisova.card\_audit\_partitioning;

--Создание партиций по дням

CREATE TABLE borisova.card\_audit\_partitioning\_2022\_12\_18 PARTITION OF borisova.card\_audit\_partitioning FOR VALUES FROM ('2022-12-18') TO ('2022-12-19');

CREATE TABLE borisova.card\_audit\_partitioning\_2022\_12\_19 PARTITION OF borisova.card\_audit\_partitioning FOR VALUES FROM ('2022-12-19') TO ('2022-12-20');

CREATE TABLE borisova.card\_audit\_partitioning\_2022\_12\_20 PARTITION OF borisova.card\_audit\_partitioning FOR VALUES FROM ('2022-12-20') TO ('2022-12-21');

Партицирование может принести следующую пользу:

* В определённых ситуациях оно кардинально увеличивает быстродействие, особенно когда большой процент часто запрашиваемых строк таблицы относится к одной или лишь нескольким партициям.
* Когда в выборке или изменении данных задействована большая часть одной партиции, производительность может возрасти, если будет выполняться последовательное сканирование этой партиции, а не поиск по индексу, сопровождаемый произвольным чтением данных, разбросанных по всей таблице.
* Массовую загрузку и удаление данных можно осуществлять, добавляя и удаляя партиции, если такой вариант использования был предусмотрен при проектировании партиций. Удаление отдельной партиции командой DROP TABLE и действие ALTER TABLE DETACH PARTITION выполняются гораздо быстрее, чем аналогичная массовая операция.
* Редко используемые данные можно перенести на более дешёвые и медленные носители.

Всё это обычно полезно только для очень больших таблиц. Какие именно таблицы логов выиграют от партицирования, зависит от конкретной базы данных.

## Дополнительное задание 5. Вывод статистики по тому, как часто меняются те или иные колонки

Создадим таблицу borisova.card\_statistics, которая будет считать количество вхождений подстроки с названием измененного столбца в строку name\_col\_changed таблицы borisova.card\_audit, в которую записывались названия колонок, измененных оператором UPDATE. Добавим также скопированную колонку name\_col\_changed таблицы borisova.card\_audit.

Теперь изменим значения количества вхождений в созданной таблице. С помощью оператора UPDATE вставим в соответствующие столбцы количество вхождений подстроки в строку. К примеру, если были изменены две колонки одновременно, то есть в таблице borisova.card\_audit в колонке name\_col\_changed в этой строке запись «card\_surname card\_balance», то напротив этой ячейки в соответствующих колонках таблицы borisova.card\_statistics будут обновлены значения - увеличены на единицу (рисунок 3).

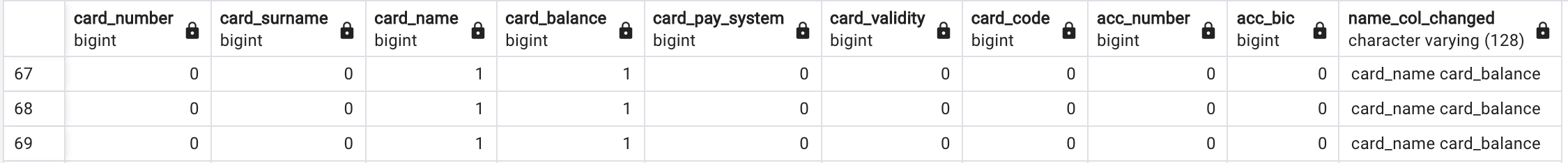


Рисунок 4. Пример определения измененных столбцов.

Осуществляется проверка с помощью функций Length и Replace. Replace заменяет вхождения названия столбца «card\_surname» на несуществующий в строке знак «~», то есть строка «card\_surname card\_balance» заменяется на «~ card\_balance», высчитывается длина с помощью Length. Далее Replace заменяет вхождения названия столбца «card\_surname» на пустой знак, также высчитывается длина строки. И разница между ними является количеством вхождений подстроки «card\_surname» в строку «card\_surname card\_balance».

Далее, чтобы получить количество изменений какого-нибудь столбца, мы делаем Select с агрегирующей функцией суммы и группируем по дню, извлеченному из даты изменения.

Приведем код ниже:

--Создание таблицы для подсчета количества измененй

create table borisova.card\_statistics

(

card\_number bigint,

card\_surname bigint,

card\_name bigint,

card\_balance bigint,

card\_pay\_system bigint,

card\_validity bigint,

card\_code bigint,

acc\_number bigint,

acc\_bic bigint,

name\_col\_changed varchar(128),

change\_time timestamp

);

--Вставка в таблицу значений из таблицы логирования

insert into borisova.card\_statistics (name\_col\_changed, change\_time) (select name\_col\_changed, change\_time from borisova.card\_audit);

--Просмотр таблицы

select \* from borisova.card\_statistics;

--Обновление ячеек таблицы для подсчета количества вхождений подстроки в строку

update borisova.card\_statistics

set card\_number = LENGTH(REPLACE(name\_col\_changed,'card\_number','~'))-LENGTH(REPLACE(name\_col\_changed, 'card\_number',''));

update borisova.card\_statistics

set card\_surname = LENGTH(REPLACE(name\_col\_changed,'card\_surname','~'))-LENGTH(REPLACE(name\_col\_changed, 'card\_surname',''));

update borisova.card\_statistics

set card\_name = LENGTH(REPLACE(name\_col\_changed,'card\_name','~'))-LENGTH(REPLACE(name\_col\_changed, 'card\_name',''));

update borisova.card\_statistics

set card\_balance = LENGTH(REPLACE(name\_col\_changed,'card\_balance','~'))-LENGTH(REPLACE(name\_col\_changed, 'card\_balance',''));

update borisova.card\_statistics

set card\_pay\_system = LENGTH(REPLACE(name\_col\_changed,'card\_pay\_system','~'))-LENGTH(REPLACE(name\_col\_changed, 'card\_pay\_system',''));

update borisova.card\_statistics

set card\_validity = LENGTH(REPLACE(name\_col\_changed,'card\_validity','~'))-LENGTH(REPLACE(name\_col\_changed, 'card\_validity',''));

update borisova.card\_statistics

set card\_code = LENGTH(REPLACE(name\_col\_changed,'card\_code','~'))-LENGTH(REPLACE(name\_col\_changed, 'card\_code',''));

update borisova.card\_statistics

set acc\_number = LENGTH(REPLACE(name\_col\_changed,'acc\_number','~'))-LENGTH(REPLACE(name\_col\_changed, 'acc\_number',''));

update borisova.card\_statistics

set acc\_bic = LENGTH(REPLACE(name\_col\_changed,'acc\_bic','~'))-LENGTH(REPLACE(name\_col\_changed, 'acc\_bic',''));

--Просмотр таблицы

select \* from borisova.card\_statistics;

--Суммирование количества вхождений подстроки в строку по столбцам и группировка строк по дням

select date\_trunc('day', change\_time) as data, sum(card\_number) as card\_number, sum(card\_surname) as card\_surname, sum(card\_name) as card\_name, sum(card\_balance) as card\_balance,

sum(card\_pay\_system) as card\_pay\_system, sum(card\_validity) as card\_validity, sum(card\_code) as card\_code, sum(acc\_number) as acc\_number, sum(acc\_bic) as acc\_bic

from borisova.card\_statistics

group by date\_trunc('day', change\_time);

# ЗАКЛЮЧЕНИЕ

Результатом работы является созданная подсистема контроля и восстановления состояния базы данных на определенный момент времени. С помощью данной системы удалось контролировать изменения в сущности «Банковская карта»: отслеживать текущий баланс карты, имя пользователя, который вносит изменения, а также внесенные в базу данных или удаленные карты.

В ходе работы был реализован запрос на вывод данных на момент времени, который позволит оператору в банке узнавать, какая сумма хранилась на карте, чтобы предоставить качественную помощь клиенту в управлении его финансами или выявить мошеннические операции. Был создан скрипт для восстановления данных на момент времени, который позволяет в случае сбоя или умышленного искажения данных вернуть прежнюю информацию. Для удобства последующей работы была написана функция, принимающая на вход название таблицы и создающая для нее систему логирования. Поэтому если понадобится создать систему контроля и восстановления для других таблиц, можно будет воспользоваться данной функцией.

Для реализации задач использовался язык запросов SQL и процедурное расширение языка - PL/pgSQL. Были изучены и применены триггерные функции, триггеры, составные запросы, объединенные оператором Union, агрегирующие функции и некоторые другие возможности языка SQL.

Логирование может быть полезно в любой базе данных, так как существует множество факторов, препятствующих ее функционированию: невнимательность оператора базы данных, превышение полномочий, технические сбои, несанкционированный доступ. Логирование позволяет не только выявить причины данных событий, но и установить точную историю, проанализировать ее и исправить ошибки, чтобы в будущем предотвратить похожие ситуации. В случае утери данных таблица логов позволит их восстановить.

Однако есть некоторые недостатки логирования, которые следует учесть при создании подсистемы контроля и восстановления базы данных на момент времени:

* Логирование не позволяет сохранить структуру базы данных, поэтому при утере всей базы данных, включая схемы, можно будет восстановить лишь наполнение таблиц значениями. Доработать механизм можно дополнительно реализовав систему контроля и восстановления всей базы данных способом, описанным в теоретической части работы.
* При изменении структуры основной таблицы, к примеру, при добавлении столбца, придется вручную изменять таблицу логов. В будущем следует добавить динамическое изменение таблицы логов в соответствии с изменением основной таблицы.
* Также недостатком является то, что база данных логов будет сильно расти, так как сохраняется вся информация, которая вносится в таблицу, причем большое количество раз. Следует разработать способ, который позволит сократить требуемую память для хранения логов.