|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ **Информатика и системы управления**

КАФЕДРА **Компьютерные системы и сети (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01**

**Отчет**

**по лабораторной работе №** **4**

**Название:** Функции языка PL/pgSQL. Индексы

**Дисциплина:** Базы данных

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ6-32Б |  |  | Н. И. Белоусов |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | Д. Н. Миков |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2022

**Задание:**

Для выполнения могут быть использованы механизмы секционирования, наследования и индексов. Для выполнения задания необходим достаточно большой объем данных, чтобы оптимизация была целесообразной порядка 1 млн. строк в каждой таблице.

Необходимо подготовить два запроса:

• Запрос к одной таблице, содержащий фильтрацию по нескольким

полям.

• Запрос к нескольким связанным таблицам, содержащий

фильтрацию по нескольким полям.

Для каждого из этих запросов необходимо провести следующие шаги:

• Получить план выполнения запроса без использования индексов.

• Получить статистику (IO и Time) выполнения запроса без

использования индексов.

• Создать нужные индексы, позволяющие ускорить запрос.

• Получить план выполнения запроса с использованием индексов и

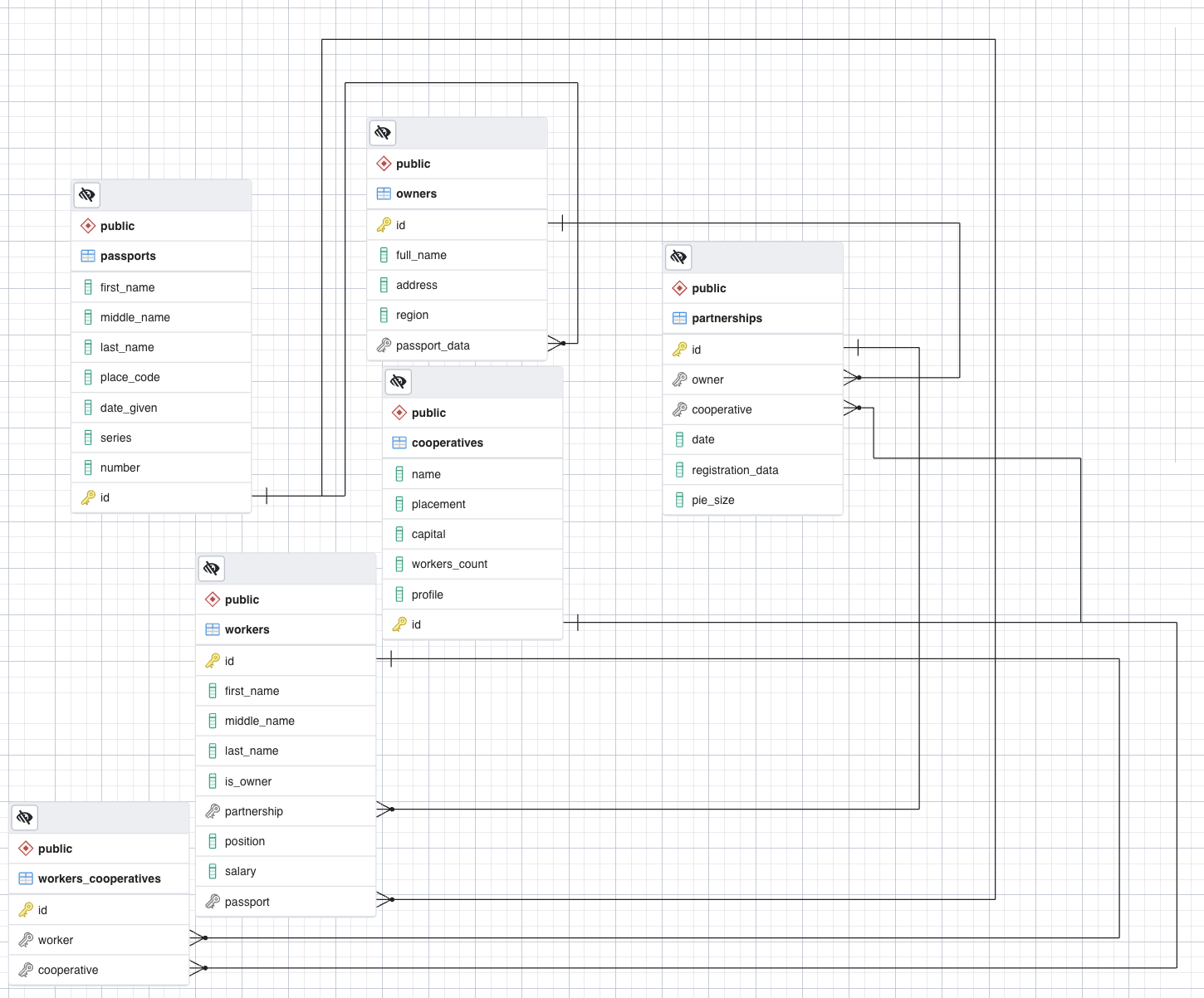
сравнить с первоначальным

• Получить статистику выполнения запроса с использованием

индексов и сравнить с первоначальной

• Оценить эффективность выполнения оптимизированного запроса

Схема для базы данных:



**Часть 1. Функции**

**Функция 1. Валидация паспортов.**

Создадим функцию, проверяющую соответствие имен рабочих и их паспортных данных

Код функции

*-- function: public:worker\_to\_owner()*

*-- validating every worker entity on passport correspondence*

*-- drop function public.check\_passports() if exists;*

*create* *or* replace *function* public.check\_passports()

*returns* *table* (id *bigint*, first\_name *varchar*, middle\_name *varchar*, last\_name *varchar*, is\_owner *boolean*, partnership *bigint*, "position" *varchar*, salary *varchar*, passport *bigint*)

*language* 'plpgsql'

*as* $body$

*declare*

current\_worker\_id *integer*;

workers\_count *integer*;

passport\_record record;

worker\_record record;

*begin*

*select* count(*\**) *into* workers\_count *from* public.workers;

*create* temp *table* *if* *not* *exists* *target* *as* *select* *\** *from* public.workers;

for current\_worker\_id *in* (*select* w.id *from* public.workers w)

*loop*

raise notice 'checking worker: %', (*select* w.last\_name *from* workers w *where* w.id *=* current\_worker\_id);

*select* *\** *from* workers w *where* w.id *=* current\_worker\_id *into* worker\_record;

*select* *\** *from* passports p *where* p.id *=* worker\_record.passport *into* passport\_record;

*if*(passport\_record.first\_name *!=* worker\_record.first\_name *or* passport\_record.middle\_name *!=* worker\_record.middle\_name *or* passport\_record.last\_name *!=* worker\_record.last\_name)

*then*

raise notice 'passport validation failed for %', (*select* w.last\_name *from* workers w *where* w.id *=* current\_worker\_id);

*insert* *into* *target* *select* *\** *from* workers w *where* w.id *=* current\_worker\_id;

*end* *if*;

*end* *loop*;

*return* query (*select* *\** *from* *target*);

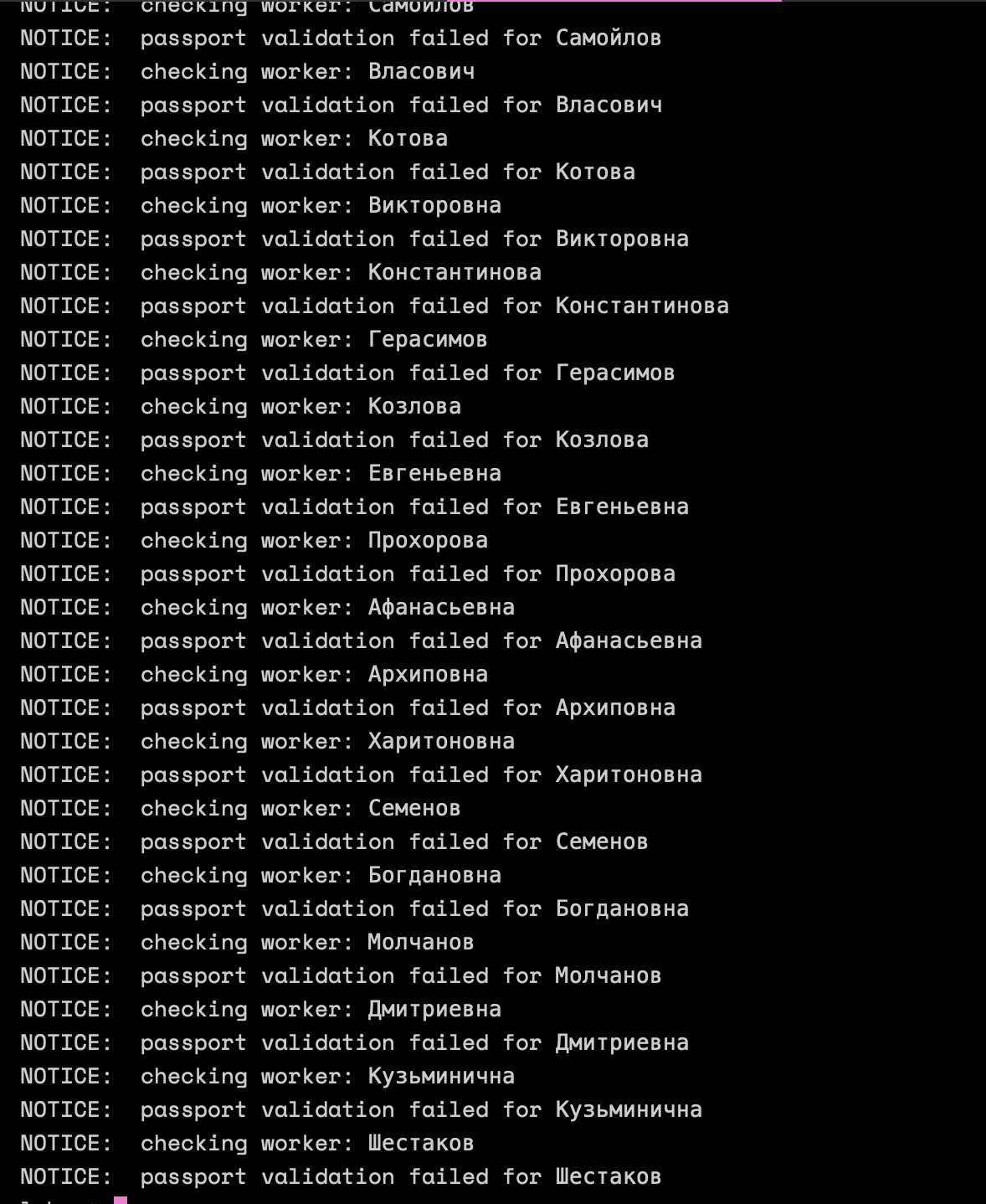
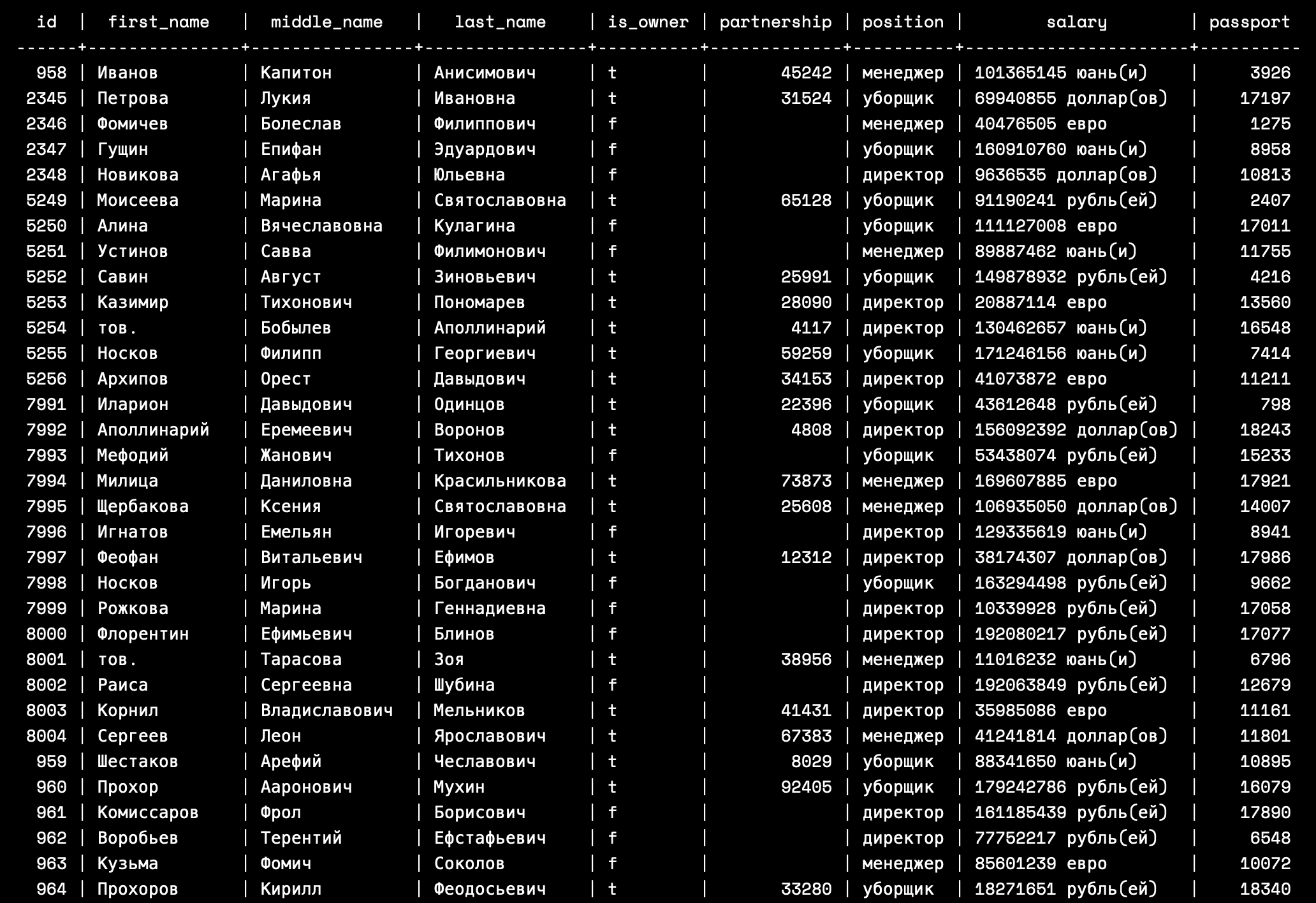
*end*;

$body$;

*-- alter function public.check\_passports() owner to postgres;*

Данная функция не только выведет таблицу всех недобросовестных рабочих, но также будет отдельно выбрасывать сообщения с их именами.

Результат работы:



Поскольку данные были сгенерированы с помощью скрипта, приведенного в лр2, таких очень много.

**Функция 2. Получение суммарного капитала.**

Реализуем функцию, подсчитывающую суммарный капитал со всех предприятий для данного владельца с заданным id.

Код функции:

*-- FUNCTION: public.all\_cooperatives\_capitals(bigint)*

*-- recieve all capitals from all cooperatives of owner*

*-- DROP FUNCTION IF EXISTS public.all\_cooperatives\_capitals(bigint);*

*CREATE* *OR* REPLACE *FUNCTION* public.all\_cooperatives\_capitals(

owner\_id *bigint*)

*RETURNS* *varchar*

*LANGUAGE* 'plpgsql'

COST 100

VOLATILE PARALLEL *UNSAFE*

*AS* $BODY$

*declare*

cooperative\_id *integer*;

capital *numeric*;

*target* *bigint*;

*begin*

*target* :*=* 0;

for cooperative\_id *in* (*select* p.cooperative *from* public.partnerships p *where* p.owner *=* owner\_id)

*loop*

raise notice 'cooperative %', cooperative\_id;

*SELECT* NULLIF(regexp\_replace( c.capital , '\D','','g'), '')::*numeric* *into* capital *from* cooperatives c *where* c.id *=* cooperative\_id;

*target* :*=* *target* *+* capital;

*end* *loop*;

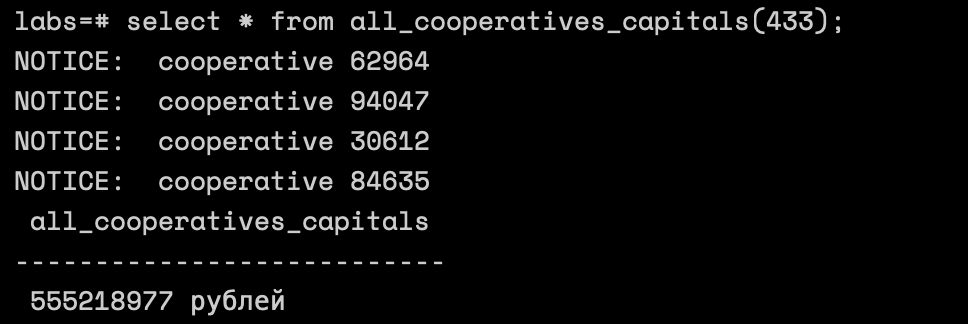
*return* *target*::*varchar* *||* ' рублей';

*end*;

$BODY$;

*ALTER* *FUNCTION* public.all\_cooperatives\_capitals(*bigint*)

*OWNER* *TO* postgres;

Результатом работы данной функции для владельца с идентификатором 433 будет:

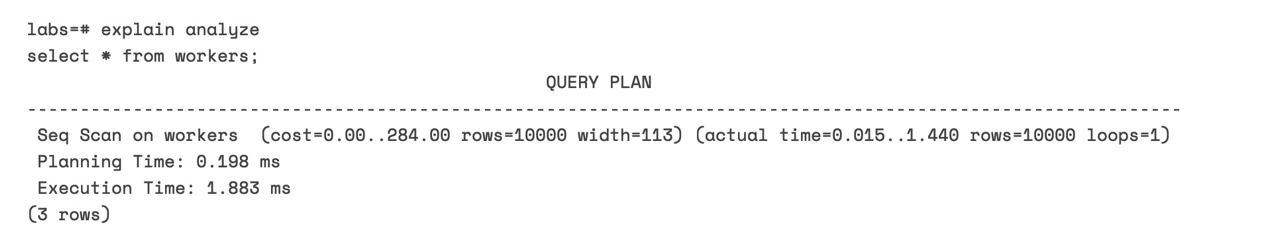
**Часть 2. Индексы**

**Запрос 1.**

Найти всех рабочих.

**Без индексов:**

Время выполнения: 1.906мс

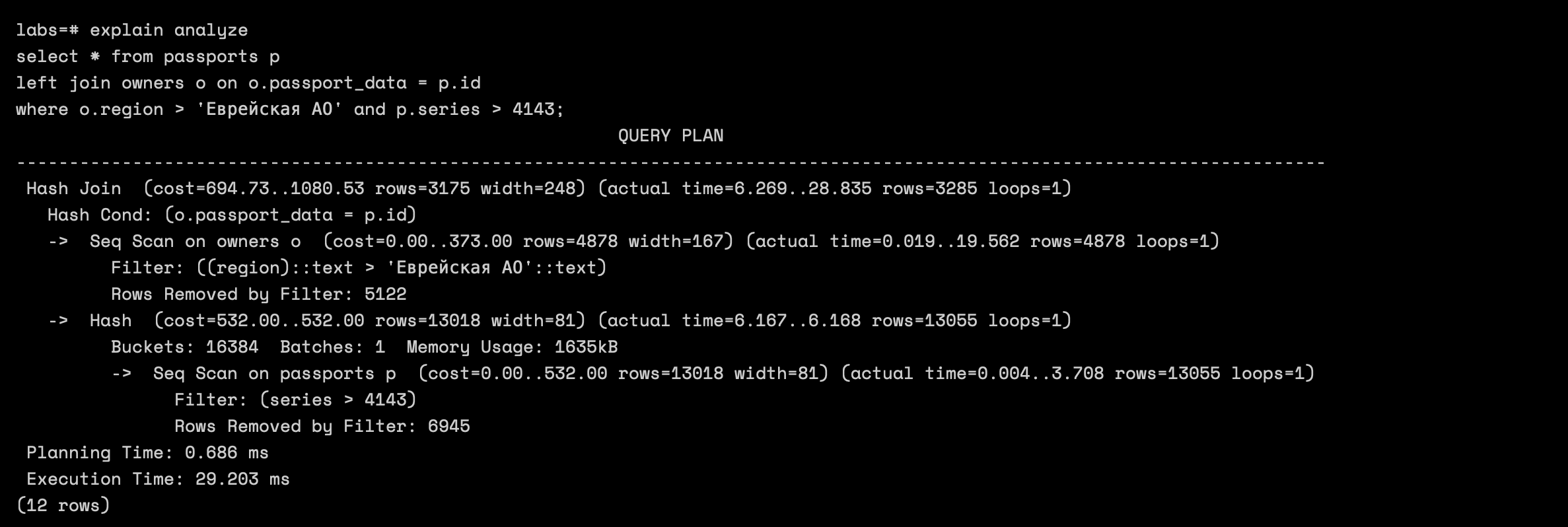
Создадим индекс для имени кооператива и посмотрим, насколько сильно изменилось время выполнения запроса.   
**С индексом**

Получаем время выполнения 1.883мс. Это на 1% быстрее, чем без индекса. Учитывая размер базы данных, при увеличении ее объема скорость выполнения также будет увеличена.

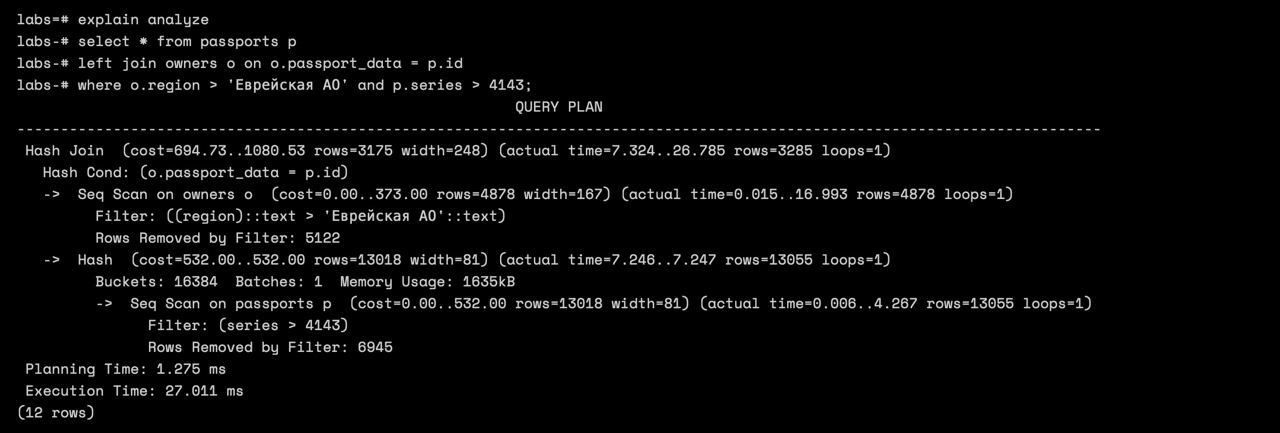
**Запрос 2.**

Найдем всех

**Без индексов**

****

Получаем время в 29.203мс.

**Добавим индекс на серию паспорта и регион проживания владельца.**

Как мы видим, время выполнения запроса уменьшилось на 2 миллисекунды.

**Контрольные вопросы:**

• В чём отличие первичного ключа и уникального индекса?

Когда для таблицы определяется ограничение уникальности или первичный ключ, PostgreSQL автоматически создаёт уникальный индекс по всем столбцам, составляющим это ограничение или первичный ключ (индекс может быть составным). Такой индекс и является механизмом, который обеспечивает выполнение ограничения.

• В каких случаях имеет смысл создавать индексы? Какие колонки следует включать в индекс и почему?

Индекс необходимо создавать, когда по какому-то полю или группе полей часто осуществляется поиск. Также, когда необходимо осуществлять полнотекстовый поиск по полю, с помощью индекса его можно подготовить к этому.

• Какие существуют способы внутренней организации индексов?

B-tree (по-умолчанию) – с помощью B-дерева GiST – generalized search tree – сбалансированное дерево, которое позволяет хранить данные, которые нельзя сравнить SP-GiST – в отличие от GiST, допускает несбалансированность GIN – Generalized Inverted Index – работает с составными типами данных.

• Рассказать о проблеме фрагментации индексов. Как бороться с фрагментацией?

В индексах сбалансированного дерева (rowstore) фрагментацией называют такое состояние, когда для некоторых страниц индекса логический порядок, основанный на значении ключа, не совпадает с физическим порядком страниц индексов. Это в итоге приводит к тому, что данные в индексе будут разбросаны по БД, а это в свою очередь снижает скорость выполнения запроса. Чтобы снизить фрагментацию индекса можно реорганизовать или перестроить индекс.

• Имеет ли значение порядок указания колонок при создании индекса?

Индекс можно использовать для точного поиска или сканирования диапазона. Точный поиск — это когда заданы значения для всех столбцов в индексе и запрос попадает точно в интересующую строку. Для поиска порядок столбцов не имеет значения. Сканирование диапазона выполняется, когда указаны только некоторые столбцы, и в этом случае порядок становится важным.

• В чём разница между Index Scan и Index Seek?

Index seek означает просмотр индекса по B-Дереву, Index Scan - обычная операция просмотра всех записей таблицы, аналогичная всем известной Table Scan.

• В чём разница между секционированием и наследованием?

С декларативным секционированием поддерживается только разбиение по спискам, по диапазонам и по хэшу, тогда как с наследованием таблиц данные можно разделять по любому критерию, выбранному пользователем.

• Зачем нужен ANALYZE?

ANALYZE собирает статистическую информацию о содержимом таблиц в базе данных и сохраняет результаты в системном каталоге pg\_statistic. Впоследствии планировщик запросов будет использовать эту статистику для выбора наиболее эффективных планов выполнения запросов.

• Могут ли индексы ухудшать производительность?

Индексы ухудшают производительность системы во время изменений записи. В любое время при выполнении запроса на изменение данных в таблице индекс должен также изменяться. Примером является последний скриншот в лабораторной работе.

• На что влияет порядок сортировки (ASC\DESC) при создании индекса?

Порядок сортировки индекса имеет значение только для многоколоночного индекса. Определение правильного порядка сортировки потенциально может устранить необходимость в шаге сортировки в плане запроса при определении порядка в инструкции select. При использовании сортировки в индексе результат запроса будет так же сразу отсортирован.

**Вывод:** в ходе выполнения лабораторной работы научился анализировать время выполнения различных запросов, а также работать с индексами.