Министерство образования и науки РФ

Федеральное государственное автономное

образовательное учреждение высшего образования

«Санкт-Петербургский национальный исследовательский университет

ИТМО»

**факультет программной инженерии и компьютерной техники**

**ЛАБОРАТОРНАЯ РАБОТА № 4**

по дисциплине

‘Базы данных’

Вариант №313121

*Выполнил:*

Студент группы P3131

Дворкин Борис Александрович

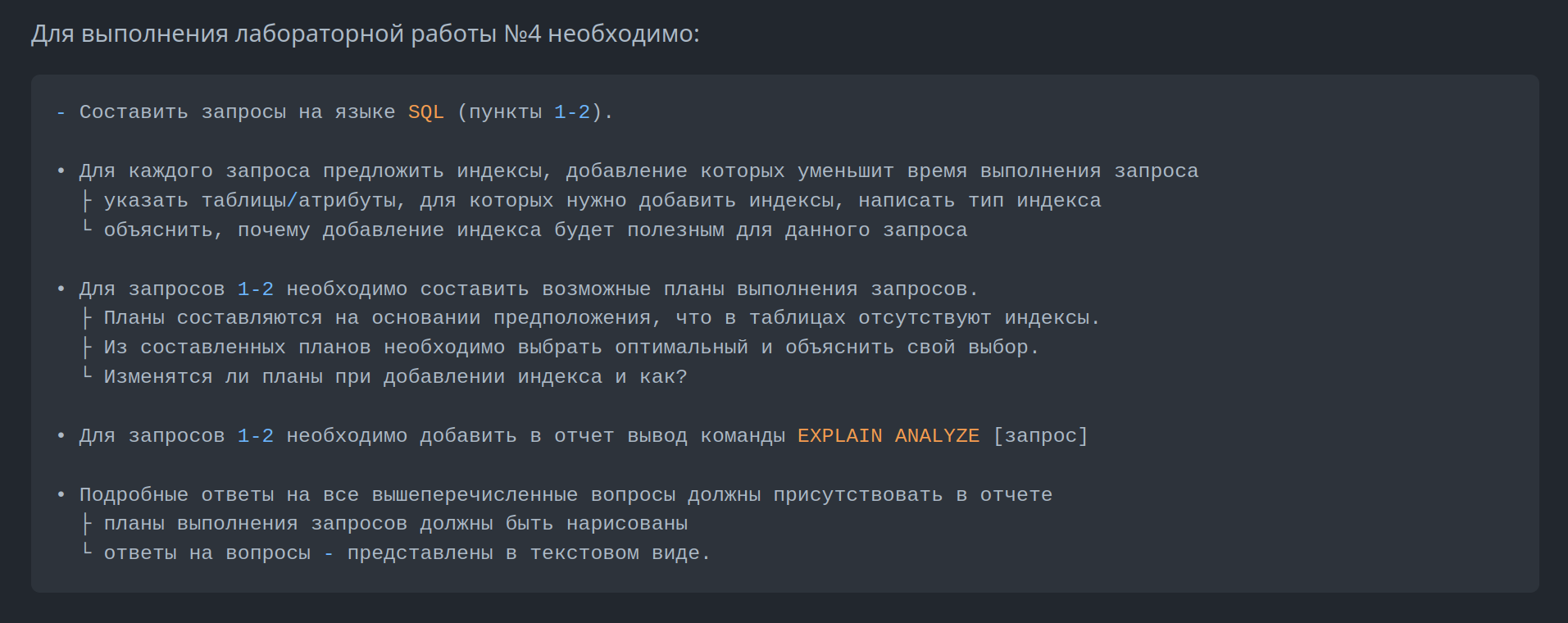
*Преподаватель:*

Наумова Надежда Александровна



Санкт-Петербург, 2023

1. **Текст задания**



1. **Реализация запросов на SQL**

|  |
| --- |
| -- 1.  -- Сделать запрос для получения атрибутов из указанных таблиц, применив фильтры по —-- указанным условиям:  -- Таблицы: Н\_ТИПЫ\_ВЕДОМОСТЕЙ, Н\_ВЕДОМОСТИ.  -- Вывести атрибуты: Н\_ТИПЫ\_ВЕДОМОСТЕЙ.НАИМЕНОВАНИЕ, Н\_ВЕДОМОСТИ.ИД.  -- Фильтры (AND):  -- a) Н\_ТИПЫ\_ВЕДОМОСТЕЙ.НАИМЕНОВАНИЕ > Ведомость.  -- b) Н\_ВЕДОМОСТИ.ДАТА < 1998-01-05. <- год 2003, там больше народу  -- Вид соединения: INNER JOIN.  SELECT  Н\_ТИПЫ\_ВЕДОМОСТЕЙ.НАИМЕНОВАНИЕ,  Н\_ВЕДОМОСТИ.ИД  FROM  Н\_ТИПЫ\_ВЕДОМОСТЕЙ  INNER JOIN  Н\_ВЕДОМОСТИ ON Н\_ТИПЫ\_ВЕДОМОСТЕЙ.ИД = Н\_ВЕДОМОСТИ.ТВ\_ИД  WHERE  Н\_ТИПЫ\_ВЕДОМОСТЕЙ.НАИМЕНОВАНИЕ > 'Ведомость' AND  Н\_ВЕДОМОСТИ.ДАТА < '2003-01-05';  ---------------------------------------------------------  -- 2.  -- Сделать запрос для получения атрибутов из указанных таблиц, применив фильтры по -  -- указанным условиям:  -- Таблицы: Н\_ЛЮДИ, Н\_ВЕДОМОСТИ, Н\_СЕССИЯ.  -- Вывести атрибуты: Н\_ЛЮДИ.ОТЧЕСТВО, Н\_ВЕДОМОСТИ.ЧЛВК\_ИД, Н\_СЕССИЯ.ИД.  -- Фильтры (AND):  -- a) Н\_ЛЮДИ.ИМЯ = Роман.  -- b) Н\_ВЕДОМОСТИ.ДАТА = 1998-01-05. <- аналогично год 2003, там больше народу  -- c) Н\_СЕССИЯ.ЧЛВК\_ИД = 106059. <- убрать, иначе ничего не выведется  -- Вид соединения: INNER JOIN.  SELECT Н\_ЛЮДИ.ОТЧЕСТВО, Н\_ВЕДОМОСТИ.ЧЛВК\_ИД, Н\_СЕССИЯ.ИД  FROM Н\_ЛЮДИ  INNER JOIN Н\_ВЕДОМОСТИ ON Н\_ЛЮДИ.ИД = Н\_ВЕДОМОСТИ.ЧЛВК\_ИД  INNER JOIN Н\_СЕССИЯ ON Н\_ВЕДОМОСТИ.СЭС\_ИД = Н\_СЕССИЯ.СЭС\_ИД  WHERE Н\_ЛЮДИ.ИМЯ = 'Роман' AND Н\_ВЕДОМОСТИ.ДАТА = '2003-01-05' AND Н\_СЕССИЯ.ЧЛВК\_ИД = 106059; |

1. **Уменьшение выполнения времени 1 запроса**

|  |
| --- |
| Индексы, добавление которых уменьшит время выполнения запроса:  a) На таблице Н\_ТИПЫ\_ВЕДОМОСТЕЙ:   * Индекс на атрибуте НАИМЕНОВАНИЕ (B-tree) Это ускорит фильтрацию строк с условием "Н\_ТИПЫ\_ВЕДОМОСТЕЙ.НАИМЕНОВАНИЕ > 'Ведомость'".   b) На таблице Н\_ВЕДОМОСТИ:   * Индекс на атрибуте ДАТА (B-tree) Это ускорит фильтрацию строк с условием "Н\_ВЕДОМОСТИ.ДАТА < '2003-01-05'". * Индекс на атрибуте ТВ\_ИД (B-tree) Это ускорит соединение таблиц по атрибуту ТВ\_ИД.   Добавление индексов на указанные столбцы позволит ускорить поиск нужных записей по фильтрам в запросе.  Возможные планы выполнения запросов без индексов:  План 1:   * Полный скан таблицы Н\_ТИПЫ\_ВЕДОМОСТЕЙ. * Полный скан таблицы Н\_ВЕДОМОСТИ с применением фильтра по условию "Н\_ВЕДОМОСТИ.ДАТА < '2003-01-05'". * Соединение таблиц с использованием Nested Loops Join по атрибуту ТВ\_ИД. * Фильтрация результата соединения по условию "Н\_ТИПЫ\_ВЕДОМОСТЕЙ.НАИМЕНОВАНИЕ > 'Ведомость'".   План 2:   * Полный скан таблицы Н\_ТИПЫ\_ВЕДОМОСТЕЙ с применением фильтра по условию "Н\_ТИПЫ\_ВЕДОМОСТЕЙ.НАИМЕНОВАНИЕ > 'Ведомость'". * Полный скан таблицы Н\_ВЕДОМОСТИ с применением фильтра по условию "Н\_ВЕДОМОСТИ.ДАТА < '2003-01-05'". * Соединение таблиц с использованием Nested Loops Join по атрибуту ТВ\_ИД.   Оптимальный план:   * План 2, потому что фильтрация данных выполняется до соединения таблиц, что уменьшает количество строк для обработки.   При добавлении индексов планы выполнения запросов изменятся:   * Вместо полного скана таблиц будет использоваться индексный скан. * Nested Loops Join станет быстрее благодаря индексу на атрибуте ТВ\_ИД. |

1. **План выполнения 1 запроса**

|  |
| --- |
| 1st QUERY PLAN  ------------------------------------------------------------------------------------------------------------------  Hash Join (cost=727.38..5993.93 rows=21249 width=422) (actual time=8.755..24.468 rows=1633 loops=1)  Hash Cond: ("Н\_ВЕДОМОСТИ"."ТВ\_ИД" = "Н\_ТИПЫ\_ВЕДОМОСТЕЙ"."ИД")  -> Bitmap Heap Scan on "Н\_ВЕДОМОСТИ" (cost=726.33..5589.15 rows=63746 width=8) (actual time=2.614..15.400 rows=64243 loops=1)  Recheck Cond: ("ДАТА" < '2003-01-05 00:00:00'::timestamp without time zone)  Heap Blocks: exact=1476  -> Bitmap Index Scan on "ВЕД\_ДАТА\_I" (cost=0.00..710.39 rows=63746 width=0) (actual time=2.388..2.389 rows=64243 loops=1)  Index Cond: ("ДАТА" < '2003-01-05 00:00:00'::timestamp without time zone)  -> Hash (cost=1.04..1.04 rows=1 width=422) (actual time=0.062..0.063 rows=2 loops=1)  Buckets: 1024 Batches: 1 Memory Usage: 9kB  -> Seq Scan on "Н\_ТИПЫ\_ВЕДОМОСТЕЙ" (cost=0.00..1.04 rows=1 width=422) (actual time=0.047..0.050 rows=2 loops=1)  Filter: (("НАИМЕНОВАНИЕ")::text > 'Ведомость'::text)  Rows Removed by Filter: 1  Planning Time: 1.332 ms  Execution Time: 24.661 ms |

1. **Уменьшение выполнения времени 2 запроса**

|  |
| --- |
| Индексы, добавление которых уменьшит время выполнения запроса:  a) На таблице Н\_ЛЮДИ:   * Индекс на атрибуте ИМЯ (B-tree). Это ускорит фильтрацию строк с условием "Н\_ЛЮДИ.ИМЯ = 'Роман'".   b) На таблице Н\_ВЕДОМОСТИ:   * Индекс на атрибуте ДАТА (B-tree). Это ускорит фильтрацию строк с условием "Н\_ВЕДОМОСТИ.ДАТА = '2003-01-05'".   c) На таблице Н\_СЕССИЯ:   * Индекс на атрибуте ЧЛВК\_ИД (B-tree). Это ускорит фильтрацию строк с условием "Н\_СЕССИЯ.ЧЛВК\_ИД = 106059".   Добавление индексов на указанные столбцы позволит ускорить поиск нужных записей по фильтрам в запросе.  Возможные планы выполнения запросов без индексов:  План 1:   * Полный скан таблицы Н\_ЛЮДИ с применением фильтра по условию "Н\_ЛЮДИ.ИМЯ = 'Роман'". * Полный скан таблицы Н\_ВЕДОМОСТИ с применением фильтра по условию "Н\_ВЕДОМОСТИ.ДАТА = '2003-01-05'". * Соединение таблиц Н\_ЛЮДИ и Н\_ВЕДОМОСТИ с использованием Nested Loops Join по атрибуту ЧЛВК\_ИД. * Полный скан таблицы Н\_СЕССИЯ с применением фильтра по условию "Н\_СЕССИЯ.ЧЛВК\_ИД = 106059". * Соединение результатов предыдущего этапа с таблицей Н\_СЕССИЯ с использованием Nested Loops Join по атрибуту СЭС\_ИД.   План 2:   * Полный скан таблицы Н\_ЛЮДИ с применением фильтра по условию "Н\_ЛЮДИ.ИМЯ = 'Роман'". * Полный скан таблицы Н\_ВЕДОМОСТИ с применением фильтра по условию "Н\_ВЕДОМОСТИ.ДАТА = '2003-01-05'". * Соединение таблиц Н\_ЛЮДИ и Н\_ВЕДОМОСТИ с использованием Hash Join по атрибуту ЧЛВК\_ИД. * Полный скан таблицы Н\_СЕССИЯ с применением фильтра по условию "Н\_СЕССИЯ.ЧЛВК\_ИД = 106059". * Соединение результатов предыдущего этапа с таблицей Н\_СЕССИЯ с использованием Hash Join по атрибуту СЭС\_ИД.   Оптимальный план:   * План 1, так как Nested Loops Join предпочтительнее Hash Join в случаях, когда обрабатываемые наборы данных небольшие, и заранее известно, что результаты фильтрации будут содержать малое количество строк..   При добавлении индексов планы выполнения запросов изменятся:   * Вместо полного скана таблиц будет использоваться индексный скан. * Nested Loops Join станет быстрее благодаря индексам на атрибутах ЧЛВК\_ИД и СЭС\_ИД. |

1. **План выполнения 2 запроса**

|  |
| --- |
| 2nd QUERY PLAN  ------------------------------------------------------------------------------------------------------------------  Nested Loop (cost=14.02..1736.27 rows=7 width=28) (actual time=0.863..2.760 rows=8 loops=1)  -> Nested Loop (cost=13.74..1727.12 rows=11 width=28) (actual time=0.085..2.724 rows=5 loops=1)  -> Seq Scan on "Н\_ЛЮДИ" (cost=0.00..163.97 rows=88 width=24) (actual time=0.021..0.878 rows=88 loops=1)  Filter: (("ИМЯ")::text = 'Роман'::text)  Rows Removed by Filter: 5030  -> Bitmap Heap Scan on "Н\_ВЕДОМОСТИ" (cost=13.74..17.75 rows=1 width=8) (actual time=0.019..0.019 rows=0 loops=88)  Recheck Cond: (("ЧЛВК\_ИД" = "Н\_ЛЮДИ"."ИД") AND ("ДАТА" = '2003-01-05 00:00:00'::timestamp without time zone))  Heap Blocks: exact=5  -> BitmapAnd (cost=13.74..13.74 rows=1 width=0) (actual time=0.018..0.018 rows=0 loops=88)  -> Bitmap Index Scan on "ВЕД\_ЧЛВК\_FK\_IFK" (cost=0.00..4.08 rows=68 width=0) (actual time=0.006..0.006 rows=37 loops=88)  Index Cond: ("ЧЛВК\_ИД" = "Н\_ЛЮДИ"."ИД")  -> Bitmap Index Scan on "ВЕД\_ДАТА\_I" (cost=0.00..9.24 rows=660 width=0) (actual time=0.016..0.016 rows=635 loops=57)  Index Cond: ("ДАТА" = '2003-01-05 00:00:00'::timestamp without time zone)  -> Index Scan using "СЕС\_СЭС\_FK" on "Н\_СЕССИЯ" (cost=0.28..0.81 rows=2 width=8) (actual time=0.005..0.006 rows=2 loops=5)  Index Cond: ("СЭС\_ИД" = "Н\_ВЕДОМОСТИ"."СЭС\_ИД")  Planning Time: 1.082 ms  Execution Time: 2.808 ms |

1. **Вывод**

В ходе выполнения лабораторной работы я освоил работу с реляционной алгеброй и научился строить планы выполнения запросов, а также их диаграммы. Я изучил различные виды индексов и узнал, как использовать их для оптимизации скорости выполнения запросов. Теперь я могу применять полученные знания для эффективной работы с базами данных и повышения производительности SQL-запросов.

Доп.задание:

|  |
| --- |
| -- Доп: написать триггер, который будет выводить информацию об объекте,  -- который был добавлен/удалён в табличку ship\_type.  --    -- Удаляем существующие триггеры, если они существуют  DROP TRIGGER IF EXISTS ship\_type\_after\_insert ON ship\_type;  DROP TRIGGER IF EXISTS ship\_type\_before\_delete ON ship\_type;  DROP TRIGGER IF EXISTS ship\_type\_delete\_cascade\_trigger ON ship\_type;    -- Создаем функцию, которая вызывается при вставке записи в таблицу ship\_type  CREATE OR REPLACE FUNCTION ship\_type\_insert\_trigger() RETURNS TRIGGER AS $$  BEGIN  RAISE NOTICE 'INSERT: Ship Type with ID %, Type %, Capacity %, Max Speed %, and Range % has been added.', NEW.id, NEW.ship\_type, NEW.ship\_capacity, NEW.max\_speed, NEW.range;  RETURN NEW;  END;  $$ LANGUAGE plpgsql;    -- Создаем функцию, которая вызывается при удалении записи из таблицы ship\_type  CREATE OR REPLACE FUNCTION ship\_type\_delete\_trigger() RETURNS TRIGGER AS $$  BEGIN  RAISE NOTICE 'DELETE: Ship Type with ID %, Type %, Capacity %, Max Speed %, and Range % has been deleted.', OLD.id, OLD.ship\_type, OLD.ship\_capacity, OLD.max\_speed, OLD.range;  RETURN OLD;  END;  $$ LANGUAGE plpgsql;    -- Создаем триггер, который вызывает функцию ship\_type\_insert\_trigger при вставке записи в таблицу ship\_type  CREATE TRIGGER ship\_type\_after\_insert  AFTER INSERT ON ship\_type  FOR EACH ROW  EXECUTE FUNCTION ship\_type\_insert\_trigger();    -- Создаем триггер, который вызывает функцию ship\_type\_delete\_trigger при удалении записи из таблицы ship\_type  CREATE TRIGGER ship\_type\_before\_delete  BEFORE DELETE ON ship\_type  FOR EACH ROW  EXECUTE FUNCTION ship\_type\_delete\_trigger();    -- Создаем функцию, которая вызывается перед удалением записи из таблицы ship\_type  CREATE OR REPLACE FUNCTION ship\_type\_delete\_cascade() RETURNS TRIGGER AS $$  BEGIN  DELETE FROM ship\_troubles WHERE ship\_id IN (SELECT id FROM ship WHERE ship\_type = OLD.id);  DELETE FROM ship WHERE ship\_type = OLD.id;  RETURN OLD;  END;  $$ LANGUAGE plpgsql;    -- Создаем триггер, который вызывает функцию ship\_type\_delete\_cascade при удалении записи из таблицы ship\_type  CREATE TRIGGER ship\_type\_delete\_cascade\_trigger  BEFORE DELETE ON ship\_type  FOR EACH ROW  EXECUTE FUNCTION ship\_type\_delete\_cascade();    ---------------------------------------------------------------------------------------------    -- Давайте покажем функциональность написанных триггеров и процедурных функций    INSERT INTO ship\_type (ship\_type, ship\_capacity, max\_speed, range)  VALUES ('Cargo', 100, 400, 3000),  ('Passenger', 200, 600, 5000);    DELETE FROM ship\_type WHERE id = 1; |