Университет ИТМО

Факультет программной инженерии и компьютерной техники

Курсовая работа (этап 2, 3, 4)

по предмету «Информационные системы и базы данных»

Веб-приложение для повторения флеш-карт

Выполнил:

Студент группы P33301

Хакимов Никита Минерахимович

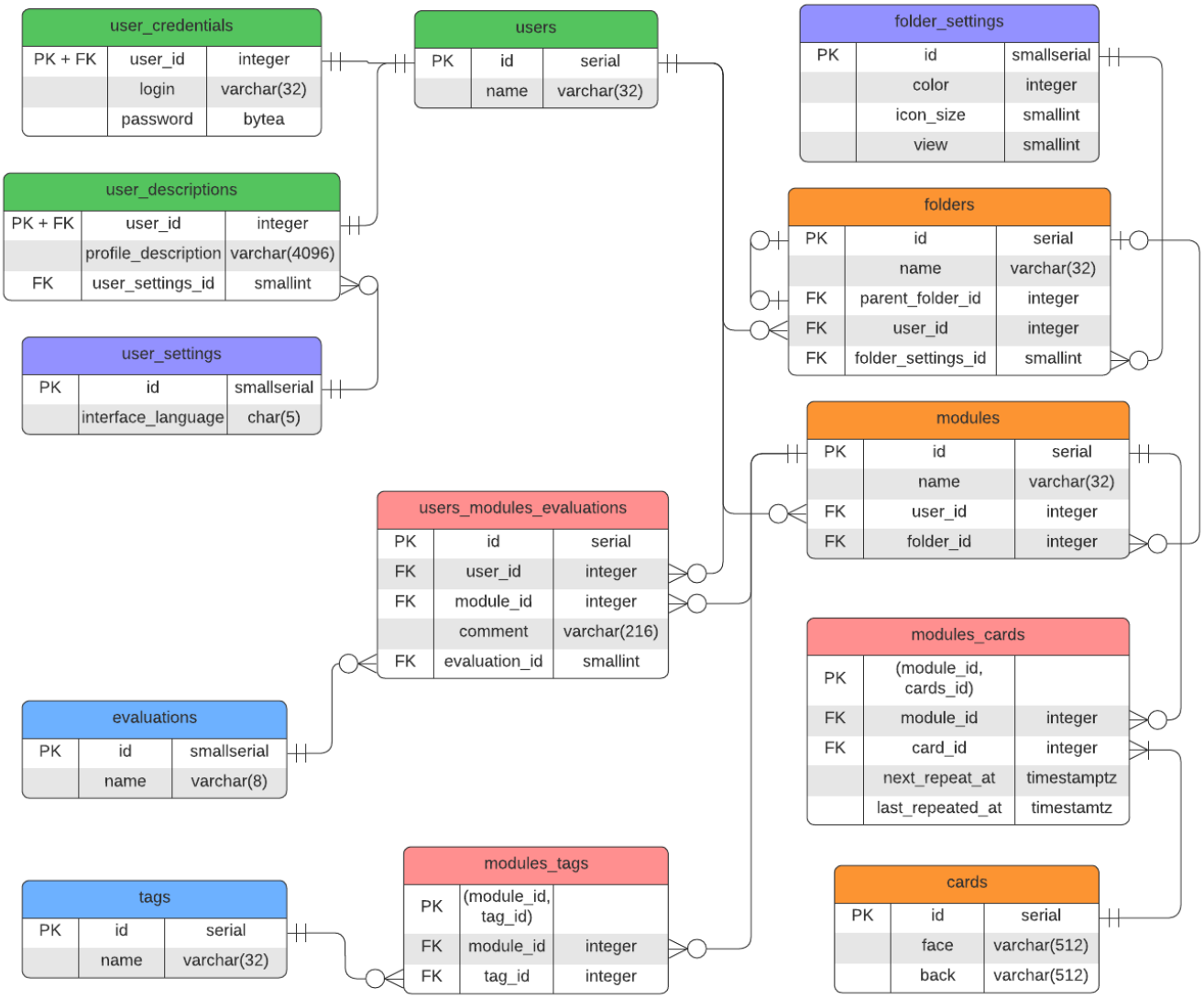
Преподаватели:

Харитонова Анастасия Евгеньевна

Санкт-Петербург

2022

1. Даталогическая модель



2. Реализация таблиц в СУБД PostgreSQL

1. Таблица user_settings

```
CREATE TABLE user_settings (
   id smallserial primary key,
   interface_language char(5)
);
```

Таблица необходима для хранения настроек пользователей. Сейчас храниться только выбранный язык, в дальнейшем настройки возможно расширять. Так как настройки у пользователей будут достаточно однотипными, организована связь 1 ко многим.

id использует тип smallserial для экономии памяти в таблицах с FK на user_settings, этого значения вполне хватит так, как вариация настроек ограничена

Вид данных хранящихся в таблице:

Id	interface_language
1	EN-en
2	RU-ru

2. Таблица users

```
CREATE TABLE users (
   id serial primary key,
   name varchar(32) NOT NULL
);
```

Таблица хранит имя пользователя и его id. Почти все таблицы будут ссылаться на id таблицы users.

3. Таблица user_descriptions

```
CREATE TABLE user_descriptions (
   profile_description varchar(4096),
   user_id integer PRIMARY KEY REFERENCES users ON DELETE CASCADE,
   user_settings_id smallint NOT NULL DEFAULT 1 REFERENCES user_settings
);
```

Характеристическая таблица для таблицы users. Была отделена для логического разделения описания пользователя, в будущем возможно расширение. Хранит указатель на настройки пользователя.

4. Таблица user_credentials

```
CREATE TABLE user_credentials (
   login varchar(32) UNIQUE NOT NULL,
   password bytea NOT NULL,
   user_id integer PRIMARY KEY UNIQUE REFERENCES users ON DELETE CASCADE
);
```

Также характеристическая таблица для таблицы users. Хранит его конфиденциальные данные для аутентификации.

5. Таблица folder_settings

```
CREATE TABLE folder_settings (
   id smallserial primary key,
   color integer NOT NULL,
   icon_size smallint NOT NULL,
   view smallint NOT NULL
);
```

Таблица предназначена для хранения настроек папок пользователя (визуальный цвет, размер иконки, вид: список, плитка и тд)

Логика выбора типа данных для id (smallserial) аналогичен с user_settings

Вид данных хранящихся в таблице:

Id	color	icon_size	view
1	2463422	16	1
2	15382134	10	1

6. Таблица folders

```
CREATE TABLE folders (
   id serial primary key,
   name varchar(32) NOT NULL,
   parent_folder_id integer REFERENCES folders ON DELETE CASCADE DEFAULT NULL,
   user_id integer NOT NULL REFERENCES users ON DELETE CASCADE,
   folder_settings_id smallint NOT NULL DEFAULT 1 REFERENCES folder_settings
);
```

Таблица предназначена для хранения папок пользователей. Папки могут быть вложенными для этого есть поле parent_folder_id, если оно равно NULL папка находиться в корневой папке. Папки не могут ссылаться циклично, это ограничение достигается настройкой триггера on_folder_insert_or_update (см. далее).

7. Таблица modules

```
CREATE TABLE modules (
   id serial primary key,
   name varchar(32) NOT NULL,
   user_id integer REFERENCES users ON DELETE CASCADE,
   folder_id integer REFERENCES folders ON DELETE CASCADE
);
```

Таблица предназначена для хранения модулей пользователя. В модулях хранятся карточки для повторения. Поле folder_id отвечает за вложенность в соответствующую папку, если оно равно NULL модуль находится в корневой папке.

8. Таблица cards

```
CREATE TABLE cards (
   id serial primary key,
   face varchar(512),
   back varchar(512)
);
```

В таблице cards хранятся непосредственно сами карточки для повторения.
Карты могут быть разделяемыми, то есть при копировании карты или модуля, копия карты не создается до тех пор, пока она не будет изменена, при этом карта должна использовать в более чем одном модуле. Логика реализуется на уровне приложения.
По факту карты не принадлежат пользователям, а их связь и время повторение достигается с помощью следующей таблицы modules_cards.
При удалении модуля карта не будет удалена, если используется в других модулях. Достигается настройкой триггера on_module_delete.

9. Таблица modules_cards

```
CREATE TABLE modules_cards (
   module_id integer REFERENCES modules ON DELETE CASCADE,
   card_id integer REFERENCES cards ON DELETE CASCADE,
   next_repeat_at timestamptz NOT NULL DEFAULT now(),
   last_repeated_at timestamptz NOT NULL DEFAULT now(),
   PRIMARY KEY (card_id, module_id)
);
```

Таблица хранит в себе соответствие карт, их принадлежность к модулю, время следующего повторения, время последнего повторения. Так как в таблице modules, есть поле, указывающее на владельца модуля (user_id), то для каждого пользователя время повторения определяется однозначно.

10. Таблица tags

```
CREATE TABLE tags (
    id serial PRIMARY KEY,
    name varchar(32) NOT NULL UNIQUE
);
```

Хранит в себе список тегов, под которыми публикуются модули.
Модуль считается опубликованным, если он связан с тегом, представленным в данной таблице.

11. Таблица modules_tags

```
CREATE TABLE modules_tags (
    module_id integer REFERENCES modules,
    tag_id integer REFERENCES tags,
    PRIMARY KEY (module_id, tag_id)
);
```

Таблица предназначена для раскрытия связи many-to-many между таблицами modules_tags.

12. Таблица evaluations

```
CREATE TABLE evaluations (
    id smallserial PRIMARY KEY,
    name varchar(8) NOT NULL UNIQUE
);
```

Таблица хранит в себе все возможные вариации оценки пользователем.
Логика выбора типа данных для id (smallserial) аналогична с user_settings.

Вид данных хранящихся в таблице:

Id	name
1	Liked
2	Disliked
3	Empty

Оценка Empty необходима в случае, когда пользователь оставляет только комментарий к модулю без оценки.

13. Таблица users_modules_evaluations

```
CREATE TABLE users_modules_evaluations (
    id serial PRIMARY KEY,
    user_id integer REFERENCES users ON DELETE SET NULL,
    module_id integer REFERENCES modules ON DELETE CASCADE,
    comment varchar(216),
    evaluation_id smallint REFERENCES evaluations
);
```

Предназначена для хранения оценок и комментариев от пользователей к публичным модулям.
Для возможности зануления user_id и сохранения оценки от удаленного пользователя в качестве PK выбрано поле id (serial), а не комбинация (user_id, module_id), как в похожих таблицах.

3. Реализация функций и триггеров в СУБД PostgreSQL

1. Функция ready_cards

```
CREATE OR REPLACE FUNCTION ready_cards(_module_id integer)
RETURNS TABLE(id integer, face varchar, back varchar)
LANGUAGE SQL AS $$
    SELECT cards.id, face, back FROM cards
    JOIN modules_cards mc ON cards.id = mc.card_id
    WHERE mc.next_repeat_at < now() AND mc.module_id = _module_id
    ORDER BY mc.last_repeated_at DESC;
$$;
```

Предназначена для получения карт готовых для повторения отсортированных в порядке «сначала новые».
Ныне устаревшая так, как логичнее производить данную операцию на стороне клиента.

2. Функция create_card

```
CREATE OR REPLACE FUNCTION create_card(_user_id integer, _module_id integer,
      face varchar, back varchar)
RETURNS integer
LANGUAGE PLPGSQL AS $$
   DECLARE
      _card_id integer;
   BEGIN
      INSERT INTO cards(face, back) VALUES (face, back) RETURNING id INTO _card_id;
        INSERT INTO modules_cards(card_id, module_id, next_repeat_at,
last_repeated_at)
            VALUES (_card_id, _module_id, now(), now());
      RETURN _card_id;
   END;
$$;
```

Предназначена для создания новой карты.
Реализует логику вставки в несколько таблиц и возвращает id созданной карты.

3. Функция create_user

```
CREATE OR REPLACE FUNCTION create_user(_login varchar, _password varchar)
RETURNS integer
LANGUAGE PLPGSQL AS $$
    DECLARE
      _user_id integer;
    BEGIN
        INSERT INTO users(name) VALUES (_login) RETURNING id INTO _user_id;
        INSERT INTO user_descriptions(user_id) VALUES (_user_id);
        INSERT INTO user_credentials(login, password, user_id) VALUES (_login,
_password, _user_id);
        RETURN _user_id;
    END;
$$;
```

Предназначена для создания нового пользователя. Реализует вставку в несколько таблиц и возвращает id созданного пользователя.

4. Триггер on_module_delete

```
CREATE OR REPLACE FUNCTION count_owners(_card_id integer)
RETURNS integer
LANGUAGE SQL AS $$
        SELECT COUNT(DISTINCT module_id) FROM modules_cards WHERE card_id =
_card_id;
$$;
```

Функция count_owners предназначена для подсчета количества модулей, что владеют картой с предоставленным id.

```
CREATE OR REPLACE PROCEDURE delete_unknown_owner_cards(_module_id integer)
LANGUAGE SQL AS $$
    DELETE FROM cards
    WHERE id IN (
        SELECT cards.id FROM cards
        JOIN modules_cards mc on cards.id = mc.card_id
        WHERE module_id = _module_id
        )
    AND count_owners(id) = 0;
$$;
```

Процедура *delete_unknown_owner_cards* удаляет карту, если она не принадлежит ни одному модулю.

```
CREATE OR REPLACE FUNCTION delete_unlinked_cards()
RETURNS trigger
LANGUAGE PLPGSQL AS $$
    BEGIN
        CALL delete_unknown_owner_cards(OLD.id);
        RETURN NEW;
    END;
$$;
```

Функция, вызываемая триггером on_module_delete

```
CREATE OR REPLACE TRIGGER on_module_delete
    AFTER DELETE ON modules
    FOR EACH ROW
    EXECUTE FUNCTION delete_unlinked_cards();
```

Триггер вызываемый на каждую строчку, удаляемую в таблице modules.

5. Триггер on_folder_insert_or_update

```
CREATE OR REPLACE FUNCTION check_cycle_parents()
RETURNS trigger
LANGUAGE PLPGSQL AS $$
    DECLARE
        folder_id integer;
    BEGIN
        IF NEW.id = NEW.parent_folder_id THEN
            RAISE EXCEPTION 'the folder(id = %) can''t be a parent(id = %) of
itself', NEW.id, NEW.parent_folder_id;
        END IF;
        FOR folder_id IN
            SELECT id FROM folders
            WHERE parent_folder_id = OLD.id
        LOOP
            UPDATE folders SET parent_folder_id = OLD.parent_folder_id WHERE id =
folder_id;
        END LOOP;
        RETURN NEW;
    END;
$$;
```

Функция, вызываемая триггером on_folder_insert_or_update. Вызывает ошибку при попытке назначить текущую папку как родителя самому себе, а также меняет дерево вложенности папок при образовании циклических связей.

```
CREATE OR REPLACE TRIGGER on_folder_insert_or_update
    BEFORE INSERT OR UPDATE OF parent_folder_id ON folders
    FOR EACH ROW
    WHEN (pg_trigger_depth() = 0)
    EXECUTE FUNCTION check_cycle_parents();
```

Триггер, вызываемый на вставку новой папки или изменения поля parent_folder_id уже существующей.
Также стоит отметить строку **WHEN** (*pg_trigger_depth*() = 0), которая предназначена для избегания рекурсии триггера.

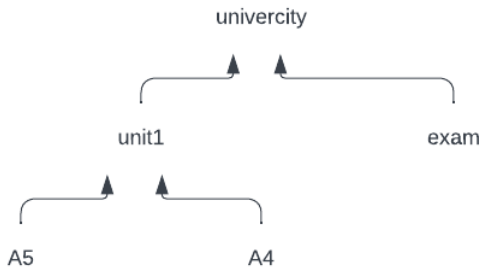
Пример работы:

```
SELECT * FROM folders WHERE user_id = 1;
```

Все папки, что принадлежат пользователю с id = 1:

id	name	parent_folder_id	user_id	fodler_settings_id
1	university	<NULL>	1	1
2	unit1	1	1	1
3	exam	1	1	1
4	A5	2	1	1
5	A4	2	1	1

В виде дерева:



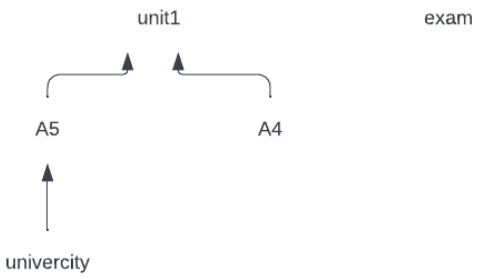
Допустим пользователем драг-энд-дропом перетащил папку univecity в A5:

```
UPDATE folders SET parent_folder_id = 5 WHERE id = 1;
```

Снова взглянем на структуру папок:

```
SELECT * FROM folders WHERE user_id = 1;
```

id	name	parent_folder_id	user_id	fodler_settings_id
1	university	5	1	1
2	unit1	<NULL>	1	1
3	exam	<NULL>	1	1
4	A5	2	1	1
5	A4	2	1	1



Теперь папки unit1 и exam находятся в корневой папке и не ссылаются на university.

Полный скрипт по созданию таблиц, триггеров и функций можно найти по ссылке:
https://github.com/NikosEvenTrue/DBMS_course/blob/dev/postgresql/create.sql

Заполнение маленьким набором тестовых данных реализуется в файле:
https://github.com/NikosEvenTrue/DBMS_course/blob/dev/postgresql/fill_test.sql

PowerShell скрипт для создания и заполнения бд:
https://github.com/NikosEvenTrue/DBMS_course/blob/dev/postgresql/reset.ps1

PowerShell скрипт с реквизитами PostgreSql ifmo:
https://github.com/NikosEvenTrue/DBMS_course/blob/dev/postgresql/reset_ifmo.ps1

Скрипт для заполнения БД большим кол-вом сгенерированных данных:
https://github.com/NikosEvenTrue/DBMS_course/blob/dev/postgresql/big_fill.sql

Скрипт для очищения схемы:
https://github.com/NikosEvenTrue/DBMS_course/blob/dev/postgresql/clear.sql

4. Создание индексов

Тестирование проходилось на сервере se.ifmo.ru

Память занимая таблицами и их заполненность:

table_name	size	rows
user_settings	8Kb	2
folder_settings	8Kb	2
evaluations	8Kb	3
user_description	3,46Mb	100e3
users	4,22Mb	100e3
user_credentials	5,61Mb	100e3
modules_tags	34,56Mb	1e6
modules	53,13Mb	1e6
folders	57,85Mb	1e6
cards	89,53Mb	1e6
modules_cards	99,52Mb	2e6
users_modules_evaluations	213,12Mb	3e6
tags	579,6MB	11e6

1. Индексы для user_settings, folder_settings, evaluations

   По сути данные таблицы являются enum'ами и не буду содержать в себе большого кол-ва элементов. Поэтому создание индексов в данных таблицах излишне.

2. Индексы для user_credentials

   Наиболее частые операции:
   a. выборка по login:
      1. при входе:

```
SELECT * FROM user_credentials
WHERE user_credentials.login = 'login'
LIMIT 1;
```

   Создание индексов:
   a. по полю login:

      Так как поле login имеет ограничение UNIQUE индекс уже создан.

3. Индексы для user_descriptions

   Наиболее частые операции:
   a. выборка по user_id:
      1. при запросе пользователя:

```
SELECT *  FROM user_descriptions
WHERE user_descriptions.user_id = user_id
LIMIT 1;
```

   Создание индексов:
   a. по полю user_id:

      Так как поле user является PRIMARY KEY индекс уже создан.

4. Индексы для users

Наиболее частые операции:

a. выборка по id:

1. при запросе пользователя:

```
SELECT * FROM users
WHERE users.id = user_id;
```

Создание индексов:

a. по полю login:

Так как поле id является PRIMARY KEY индекс уже создан.

5. Индексы для folders

Наиболее частые операции:

a. выборка по id:

1. при обращении к папке:

```
SELECT * FROM folders WHERE id = fodler_id;
```

b. выпорка по user_id:

1. при каскадном удалении user

```
DELETE FROM folders WHERE user_id = user_id;
```

2. при выборке всех папок

```
SELECT * FROM folders WHERE user_id = user_id;
```

c. выборка по parent_folder_id:

1. поиск вложенных папок (триггер предотвращения циклических ссылок)

```
SELECT id FROM folders WHERE parent_folder_id = parent_folder_id;
```

2. удаление вложенных папок при удалении родительской

```
DELETE FROM folders WHERE parent_folder_id = parent_folder_id;
```

Создание индексов:

a. по полю id:

Так как по id является PRIMARY KEY btree-index уже создан.

b. по полю user_id:

Сравнение индексов:

type	cost (EXPLAIN)	size
btree	8,62	8,87Mb
hash	46,95	32Mb

И хоть единственная операция, которая здесь применяется «сравнивание», результаты команды EXPLAIN ANALYZE преимущество btree индекса, а также его меньший вес на диске.

Конечно, однозначно утверждать, что btree всегда будет лучше hash – неверное. Необходимо провести замеры с разным кол-вом данный и на том сервере где планируется размещать базу.

Но пока остановимся на btree.

c. по полю: parent_folder_id

Сравнение индексов:

type	cost (EXPLAIN)	size
btree	8,44	8,63Mb
hash	8,02	28Mb

Благодаря большей производительности и низким штрафом на операции INSERT, UPDATE, DELETE остановимся на hash-index’e.

6. Индексы для modules

Наиболее частые операции:

a. выборка по id:

1. при обращении к модулю:

```
SELECT * FROM modules WHERE id = module_id;
```

b. выборка по user_id:

1. удаление модуля (при каскадном удалении user):

```
DELETE FROM modules WHERE user_id = user_id;
```

2. выборка всех модулей по пользователя:

```
SELECT * FROM modules WHERE user_id = user_id;
```

Создание индекса:

a. по полю id:

Так как по id является PRIMARY KEY btree-index уже создан.

d. по полю user_id:

Сравнение индексов:

type	cost (EXPLAIN)	size
btree	8,62	8,87Mb
hash	46,91	32Mb

Остановимяс на btree-index

7. Индексы для modules_cards

Наиболее частые операции:

a. выборка по module_id:

1. при удалении модуля:

```
DELETE FROM modules_cards WHERE module_id = module_id;
```

2. при получении id карт модуля в триггере на удаление модуля:

```
SELECT cards.id FROM cards
JOIN modules_cards mc on cards.id = mc.card_id
WHERE module_id = module_id;
```

b. выборка по card_id:

1. выборка всех модудлей связанных с картой (при запуске триггера на удаление модуля):

```
SELECT COUNT(DISTINCT module_id) FROM modules_cards
WHERE card_id = card_id;
```

c. выборка по module_id, card_id:

1. выборка всех карт принадлежащих модулю:

```
SELECT cards.id FROM cards
JOIN modules_cards mc on cards.id = mc.card_id
WHERE module_id = module_id;
```

Создание индекса:

a. по полю module_id:

Так как (module_id, card_id) является составным ключом индекс (module_id, card_id) уже создан. Поиск будет производиться по нему.

b. по полю card_id:

Сравнение индексов:

type	cost (EXPLAIN)	size
btree	11,48	39Mb
hash	12,05	64Mb

остановимся на btree-index. Больше производительность, меньше вес.

c. по полям module_id, card_id:

Как и при module_id индекс будет уже создан.
Очень важно сохранять последовательность полей в составном ключе, так как производительность индексов (card_id, module_id) и module_id меньше, чем производительность (module_id, card_id) и card_id. Так как в последнем случае будет использован Only Index Scan.

8. Индексы для cards.

Так как выборка всегда происходит по id, а id является PK, то индекс уже создан.

9. Индексы для users_modules_evaluations

Наиболее частые операции:

a. выборка по user_id, module_id:

1. при запросе на получение существующей оценки:

```
SELECT * FROM users_modules_evaluations
WHERE user_id = user_id AND module_id = module_id
LIMIT 1;
```

b. выборка по user_id:

1. при каскадном удалении user

```
UPDATE users_modules_evaluations SET user_id = NULL
WHERE user_id = user_id;
```

c. выборка по module_id:

1. при каскадном удалении user:

```
UPDATE users_modules_evaluations SET user_id = NULL
WHERE user_id = user_id;
```

2. при получении оценок модуля:

```
SELECT * FROM users_modules_evaluations WHERE module_id = module_id;
```

Создание индексов:

a. по полю user_id, module_id:

Сравнение индексов:

type	cost (EXPLAIN)	size
btree(user_id), btree(module_id)	13,39	user_id(22Mb), module_id(42Mb)
hash (user_id), hash(module_id)	12,53	user_id(96Mb), module_id(87Mb)
btree(user_id, module_id)	8,45	64Mb
btree(module_id, user_id)	8,45	64Mb

b. по полю user_id:

Сравнение индексов:

type	cost (EXPLAIN)	size
btree	20,50	22Mb
hash	20,07	96Mb
btree(user_id, module_id)	55426,00	64Mb
btree(module_id, user_id)	20,50	64Mb

c. по полю module_id

Сравнение индексов:

type	cost (EXPLAIN)	size
btree	125,92	22Mb
hash	125,49	96Mb
btree(user_id, module_id)	125,92	64Mb
btree(module_id, user_id)	55536,27	64Mb

В качестве компромисса по памяти, с учетом, что у нас нет запросов вовлекающих внешние ключи нашей таблицы, кроме представленных, можно остановиться на btree(module_id, user_id) и btree(user_id). Ведь запрос текущих оценок и оценок модуля происходит чаще удаления.

10. Индексы для modules_tags

Наиболее частые операции:

a. выборка по module_id:

1. при удалении модуля (каскадно):

```
DELETE FROM modules_tags WHERE module_id = module_id;
```

2. при выясьнении публичный ли модуль:

```
SELECT * FROM modules_tags WHERE module_id = module_id LIMIT 1;
```

3. при поиске тегов под которыми опубликован модуль:

```
SELECT tags.*
FROM tags JOIN modules_tags ON tags.id = modules_tags.tag_id
WHERE modules_tags.module_id = module_id;
```

4. при поиске модулей по тегу:

```
SELECT modules.*
FROM modules
JOIN modules_tags ON modules.id = modules_tags.module_id
JOIN tags ON tags.id = modules_tags.tag_id
WHERE tags.name = 'tag_name'
```

b. выборка по tag_id:

1. при поиске наиболее используемых тегов по началу набранного текста:

```
EXPLAIN ANALYSE
SELECT * FROM tags
WHERE id IN (SELECT tag_id
             FROM modules_tags
             WHERE tag_id IN (SELECT id
                              FROM tags
                              WHERE name LIKE 'text%')
             GROUP BY tag_id
             ORDER BY count(tag_id) DESC
             LIMIT limit);
```

Создание индексов:

a. по полю module_id:

Так как (module_id, tag_id) являются PK, индекс уже создан.

b. по полю tag_id:

Сравнение индексов:

type	cost (EXPLAIN)	size

btree	4207,51	19Mb
hash	10805,44	32Mb

B-tree индекс оказался производительнее, так как в запросе используется IN. Также он меньше весит. Остановимся на нем.

11. Индексы для tags

Наиболее частые операции:

a. выборка по id:

1. при соединеии с таблице modules_tags:

```
SELECT tags.*
FROM tags JOIN modules_tags ON tags.id = modules_tags.tag_id
WHERE modules_tags.module_id = module_id;
```

b. выборка по name:

1. при поиске наиболее используемых тегов по началу набранного текста:

```
EXPLAIN ANALYSE
SELECT * FROM tags
WHERE id IN (SELECT tag_id
             FROM modules_tags
             WHERE tag_id IN (SELECT id
                              FROM tags
                              WHERE name LIKE 'text%')
             GROUP BY tag_id
             ORDER BY count(tag_id) DESC
             LIMIT limit);
```

2. при поиске по имени тега:

```
EXPLAIN ANALYSE
SELECT modules.*
FROM modules
JOIN modules_tags ON modules.id = modules_tags.module_id
JOIN tags ON tags.id = modules_tags.tag_id
WHERE tags.name = 'tag_name'
```

3. при выяснении существует ли тег:

```
SELECT * FROM tags
WHERE tags.name IN ('tag_name_1', 'tag_name_2');
```

Создание индексов:

a. по полю id:

Так как id являются PK, индекс уже создан.

b. по полю name:

С одной стороны на поле name есть ограничение UNIQUE, что автоматически создает btree-индекс. Он отлично подходит для операций сравнений или IN, но не используется postgres для поиска по левостороннему паттерну команды LIKE “text%”. В этом случае придется создать дополнительный btree-index с явным указанием Operator class’a, чтобы он мог взять актуальные правила для сравнения строк в текущей локале.

```
CREATE INDEX tags_name_btree ON tags(name varchar_pattern_ops);
```

Полезные ссылки:
https://www.postgresql.org/docs/9.5/indexes-opclass.html
http://www.www-old.bartlettpublishing.com/site/bartpub/blog/3/entry/329

5. Тест индексов

Также ради интереса мной было проведено исследование влияние заполненности таблицы на производительность индексов и их вес.
Замеры проводились на таблице users_modules_evaluations по полю module_id.
Производительность измерялась следующей командой:

```
EXPLAIN ANALYSE
WITH vars (r) as (values(randint(1, N())))
SELECT ume.id, ume.user_id, ume.module_id, ume.comment, ume.evaluation_id
FROM users_modules_evaluations ume, vars
WHERE ume.module_id = r;
```

Размер индекса вычислялся следующей командой:

```
SELECT pg_size_pretty (pg_table_size('index_name'));
```

Размер таблицы:

```
SELECT pg_table_size('users_modules_evaluations_module_id_btree');
```

Бралась средняя производительность для текущего количества рядов, кол-во самих запросов: на моем ПК - 100, на se.ifmo.ru - 1000.
Также постарался отсечь выбросы, которые возникали при первых запусках. Команда vacuum analyze не помогала.

Результаты в виде таблиц:

a. На моем устройстве:

index	rows	size	cost	execution time (ms)
no	10	0 bytes	1,19	0,10
btree	10	16 kB	1,19	0,10
hash	10	32 kB	1,19	0,12
no	100	0 bytes	2,43	0,15
btree	100	16 kB	2,43	0,12
hash	100	32 kB	2,43	0,10
no	1000	0 bytes	23,8	0,29
btree	1000	40 kB	8,33	0,05
hash	1000	48 kB	8,06	0,04
no	10e3	0 bytes	228,55	2,16
btree	10e3	240 kB	8,34	0,06
hash	10e3	528 kB	8,06	0,04
no	100e3	0 bytes	2285,05	20,91
btree	100e3	2184 kB	8,35	0,10
hash	100e3	4112 kB	8,06	0,04
no	250e3	0 bytes	5710,56	51,82
btree	250e3	5360 kB	8,48	0,11
hash	250e3	8200 kB	8,06	0,04
no	500e3	0 bytes	11422,06	103,58
btree	500e3	10 MB	8,48	0,10
hash	500e3	16 MB	9,27	0,05
no	750e3	0 bytes	17132,57	162,69
btree	750e3	15 MB	12,51	0,11
hash	750e3	21 MB	12,09	0,04
no	1e6	0 bytes	22843,07	229,96
btree	1e6	19 MB	12,51	0,12
hash	1e6	32 MB	12,09	0,04
no	1,5e6	0 bytes	34265,06	383,27
btree	1,5e6	26 MB	14,32	0,12
hash	1,5e6	42 MB	13,29	0,07
no	2e6	0 bytes	45685,07	538,44
btree	2e6	32 MB	16,54	0,12
hash	2e6	64 MB	16,12	0,10
no	3e6	0 bytes	68529,08	843,28

btree	3e6	42 MB	20,57	0,15
hash	3e6	87 MB	20,14	0,13
no	4e6	0 bytes	91370,27	1242,80
btree	4e6	49 MB	24,6	0,21
hash	4e6	127 MB	24,17	0,20
no	5e6	0 bytes	114213,71	1469,42
btree	5e6	56 MB	28,63	0,21
hash	5e6	130 MB	28,2	0,21
no	6e6	0 bytes	137055,69	1730,29
btree	6e6	63 MB	32,66	0,23
hash	6e6	180 MB	32,23	0,24
no	7e6	0 bytes	159899,58	2073,20
btree	7e6	69 MB	36,68	0,27
hash	7e6	232 MB	36,25	0,27
no	8e6	0 bytes	182740,72	2559,39
btree	8e6	76 MB	40,71	0,30
hash	8e6	253 MB	40,28	0,31
no	9e6	0 bytes	205584,42	2887,62
btree	9e6	82 MB	44,74	0,33
hash	9e6	258 MB	44,31	0,30
no	10e6	0 bytes	228426,98	2936,80
btree	10e6	89 MB	48,77	0,32
hash	10e6	268 MB	48,34	0,30
btree	30e6	215 MB	129,59	0,68
hash	30e6	947 MB	129,57	0,71
btree	40e6	277 MB	169,26	0,90
hash	40e6	1207 MB	169,03	1,94
btree	50e6	339 MB	209,58	2,12
hash	50e6	1604 MB	209,63	2,94
btree	60e6	401 MB	250,13	4,29
hash	60e6	1936 MB	246,64	4,08

b. на se.ifmo.ru:

index	rows	size	cost	execution time (ms)
no	10	0 bytes	1,19	0,04
btree	10	16 kB	1,19	0,04
hash	10	32 kB	1,19	0,04
no	100	0 bytes	2,43	0,06
btree	100	16 kB	2,43	0,06
hash	100	32 kB	2,43	0,06
no	1000	0 bytes	23,8	0,23
btree	1000	40 kB	8,33	0,03
hash	1000	48 kB	8,06	0,03
no	1900	0 bytes	44,18	0,40
btree	1900	64 kB	8,34	0,03
hash	1900	80 kB	8,06	0,03
no	10900	0 bytes	248,93	2,18
btree	10900	256 kB	8,34	0,04
hash	10900	528 kB	8,06	0,04
no	100e3	0 bytes	2304,43	21,45
btree	100e3	2200 kB	8,35	0,05
hash	100e3	4112 kB	8,06	0,05
no	250e3	0 bytes	5730,93	52,36
btree	250e3	5384 kB	8,48	0,07
hash	250e3	8208 kB	8,06	0,05
no	500e3	0 bytes	11441,43	104,86
btree	500e3	10 MB	9,58	0,07
hash	500e3	16 MB	9,18	0,05
no	750e3	0 bytes	17151,94	156,62
btree	750e3	15 MB	12,51	0,08
hash	750e3	21 MB	12,09	0,05
no	1e6	0 bytes	22862,44	207,72
btree	1e6	19 MB	12,51	0,07
hash	1e6	32 MB	12,09	0,05

btree	1,25e6	23 MB	12,51	0,09
hash	1,25e6	32 MB	12,09	0,05
btree	1,75e6	29 MB	16,54	0,10
hash	1,75e6	59 MB	16,12	0,06
btree	2,25e6	35 MB	16,54	0,08
hash	2,25e6	64 MB	16,12	0,05
btree	3,25e6	44 MB	20,57	0,09
hash	3,25e6	108 MB	20,14	0,06
btree	4,25e6	51 MB	24,6	0,09
hash	4,25e6	128 MB	24,17	0,06
btree	5,25e6	58 MB	28,63	0,09
hash	5,25e6	163 MB	28,2	0,06
btree	6,25e6	65 MB	32,66	0,17
hash	6,25e6	189 MB	32,23	0,07
btree	7,25e6	71 MB	36,68	0,14
hash	7,25e6	238 MB	36,25	0,07

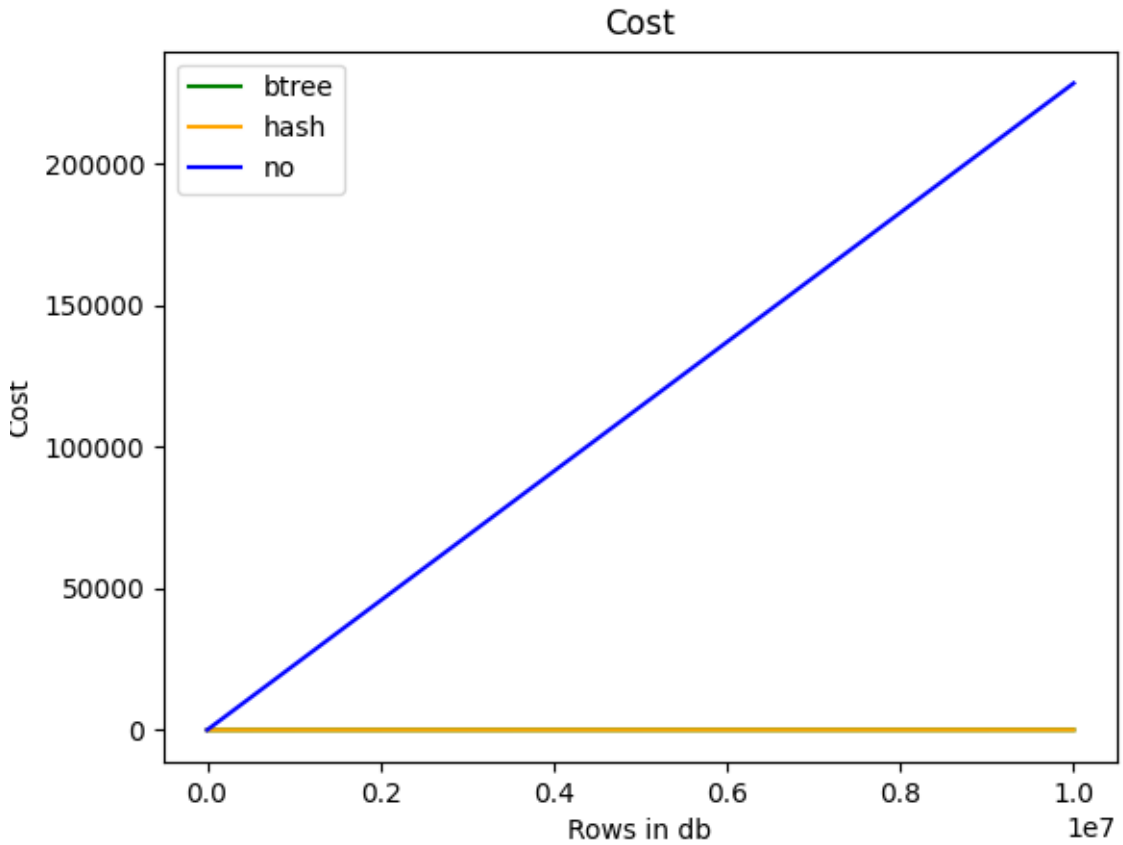
В виде графиков:

no – обозначает измерение без использования индекса

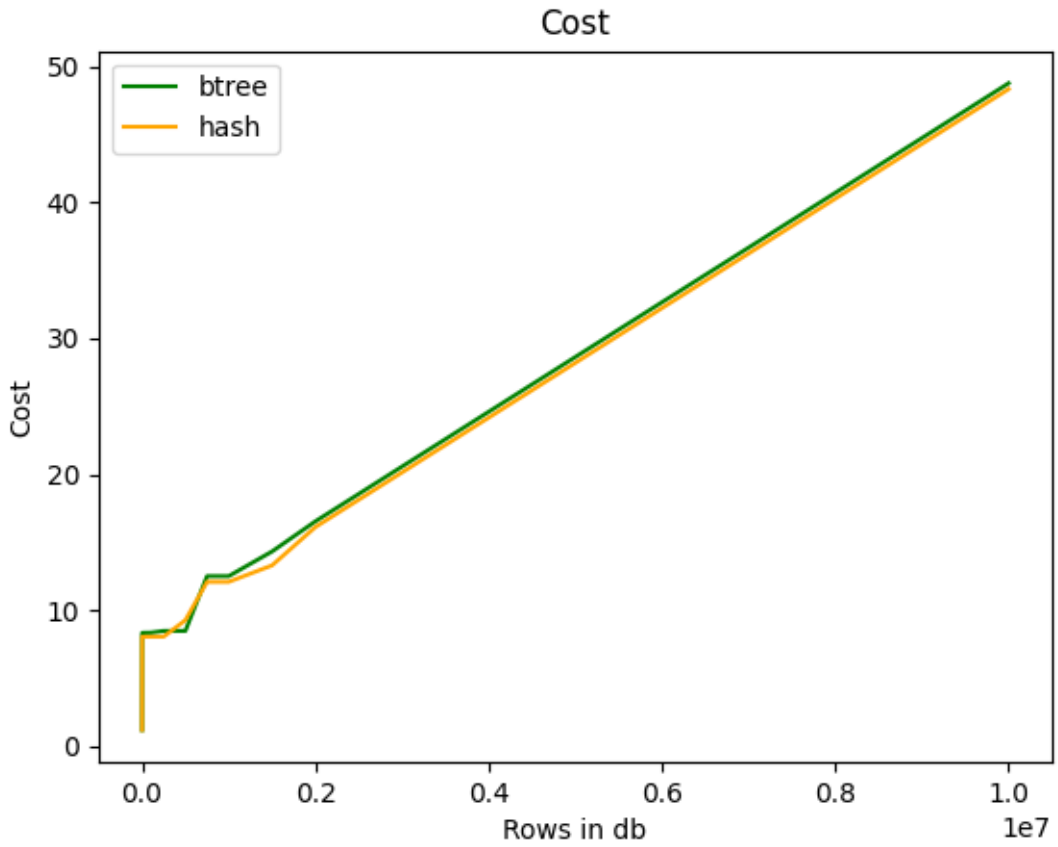
a. Измерения произведенные на моем компьютере:

1. Зависимость по cost:

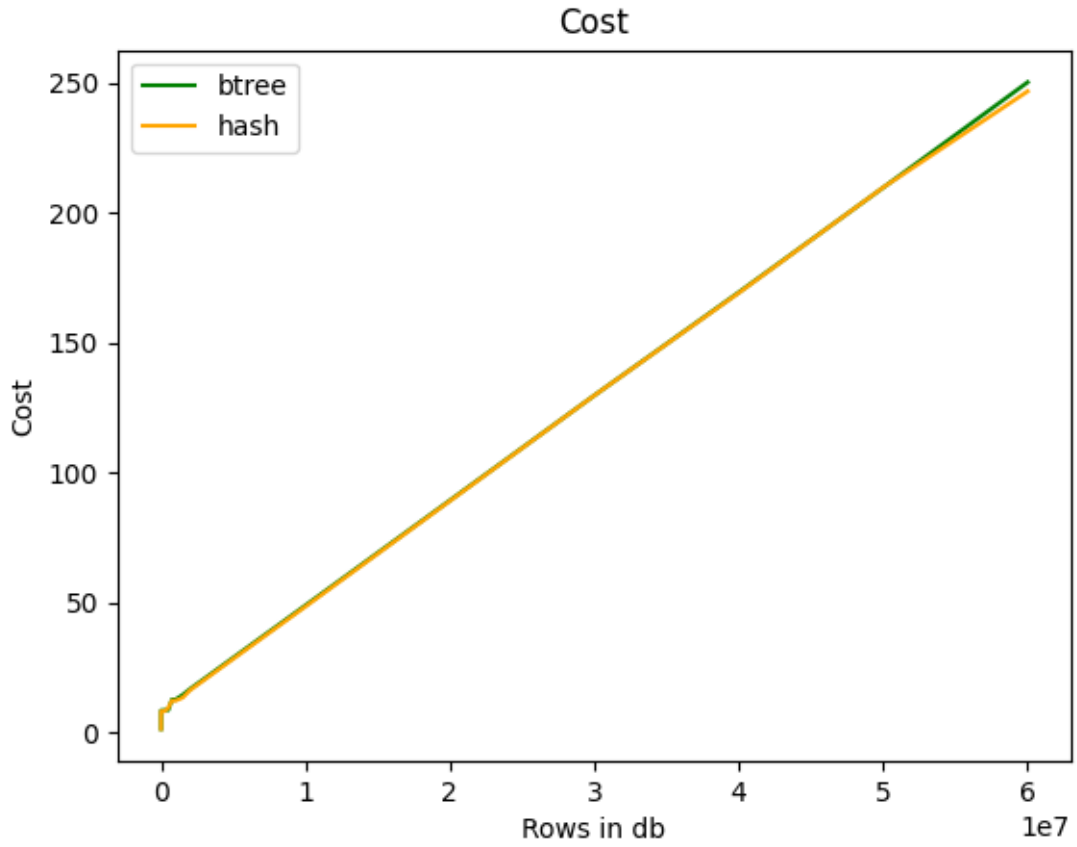
A. от 0 до 10M (с безидексовыми замерами):



B. от 0 до 10M (только и индексами):

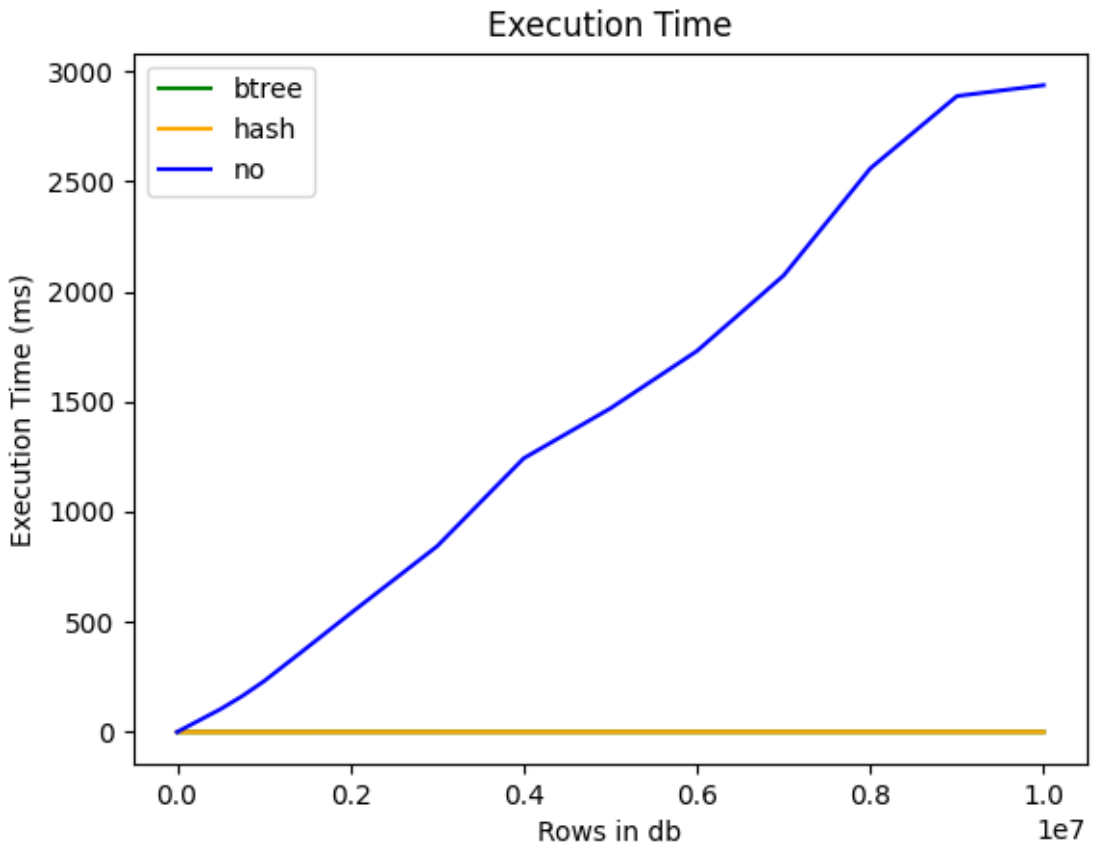


C. от 0 до 60M (только с индексами):

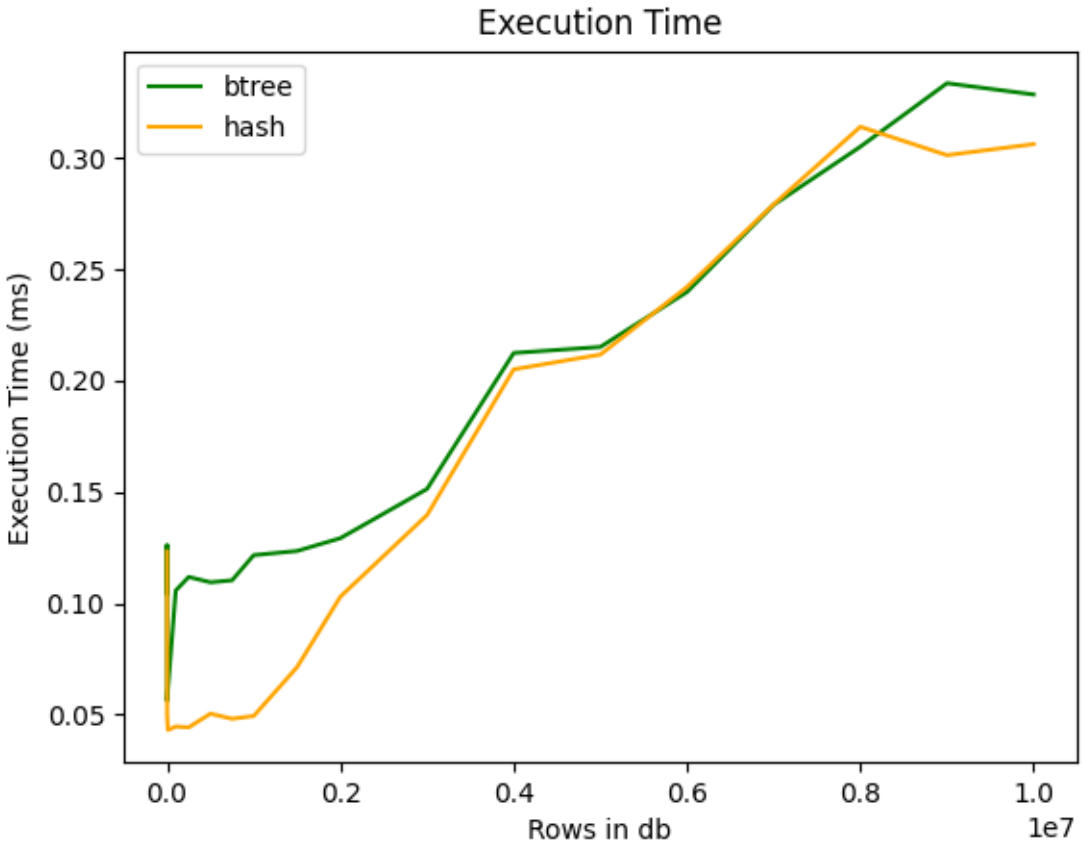


2. Зависимость по execution time:

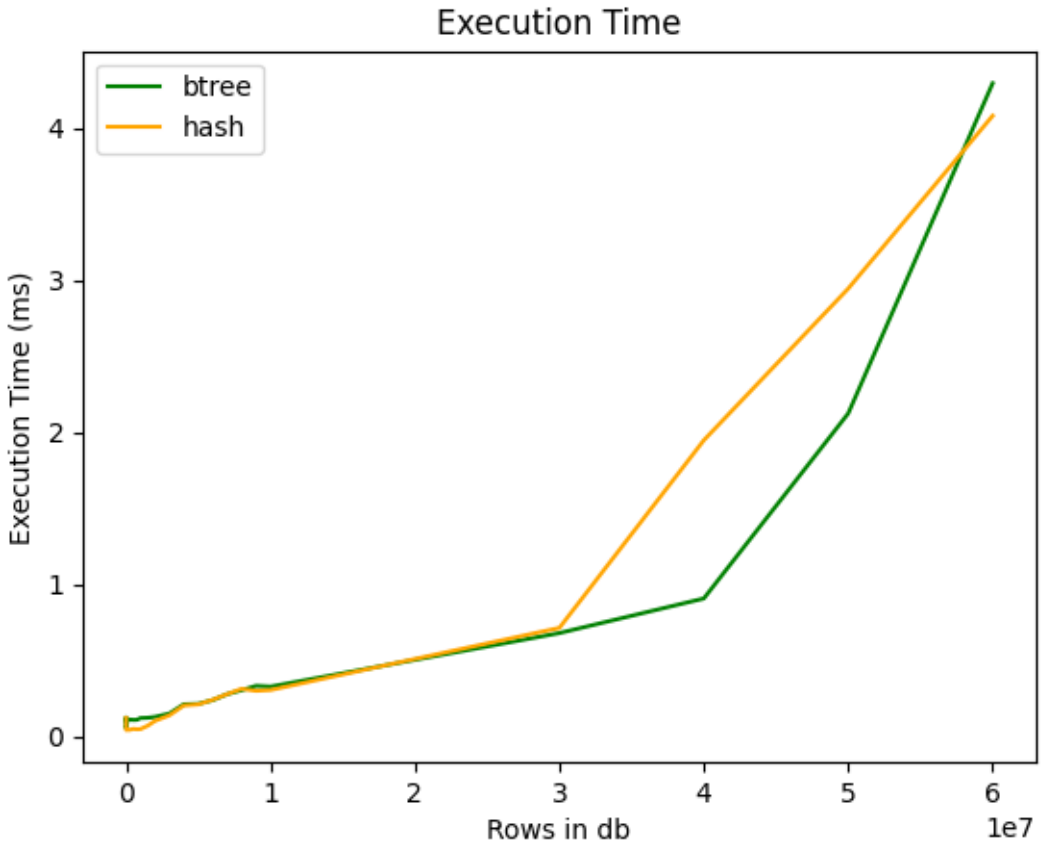
A. от 0 до 10M (с безиндексовыми замерами):



B. от 0 до 10M (только индексы):

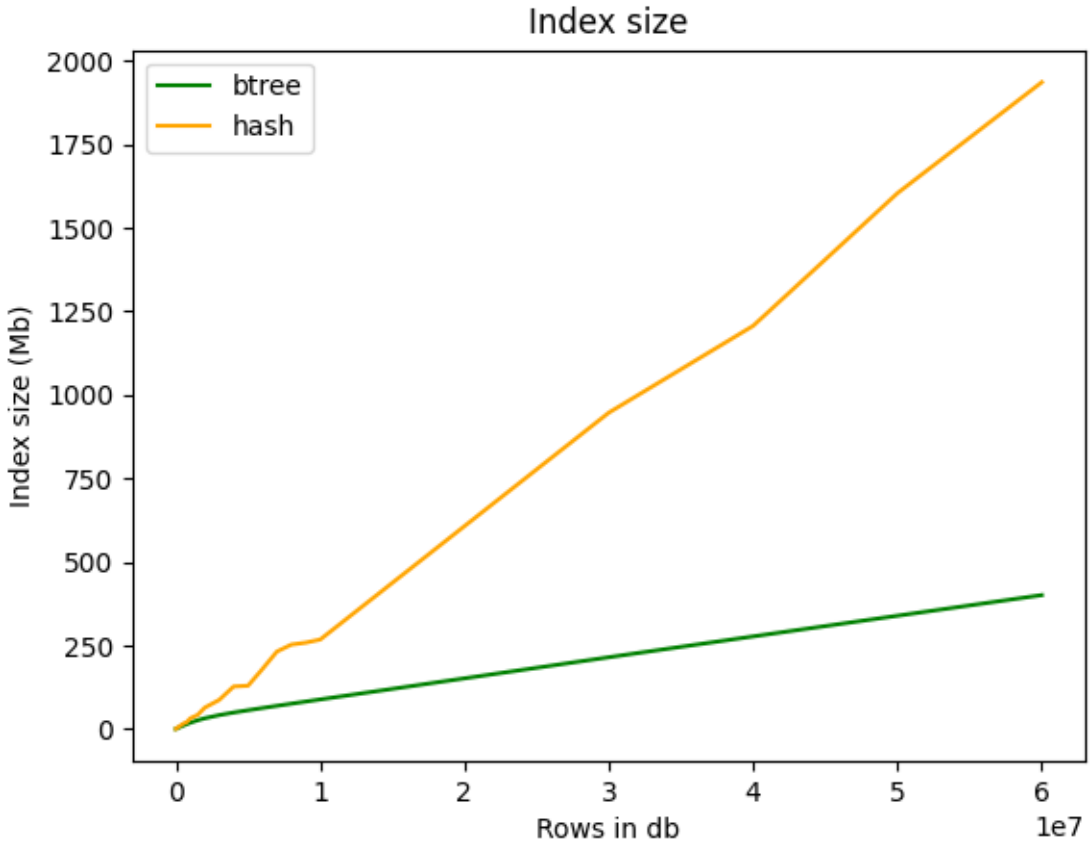


C. от 0 до 60M (только индексы):



3. Зависимость по размеру индексов:

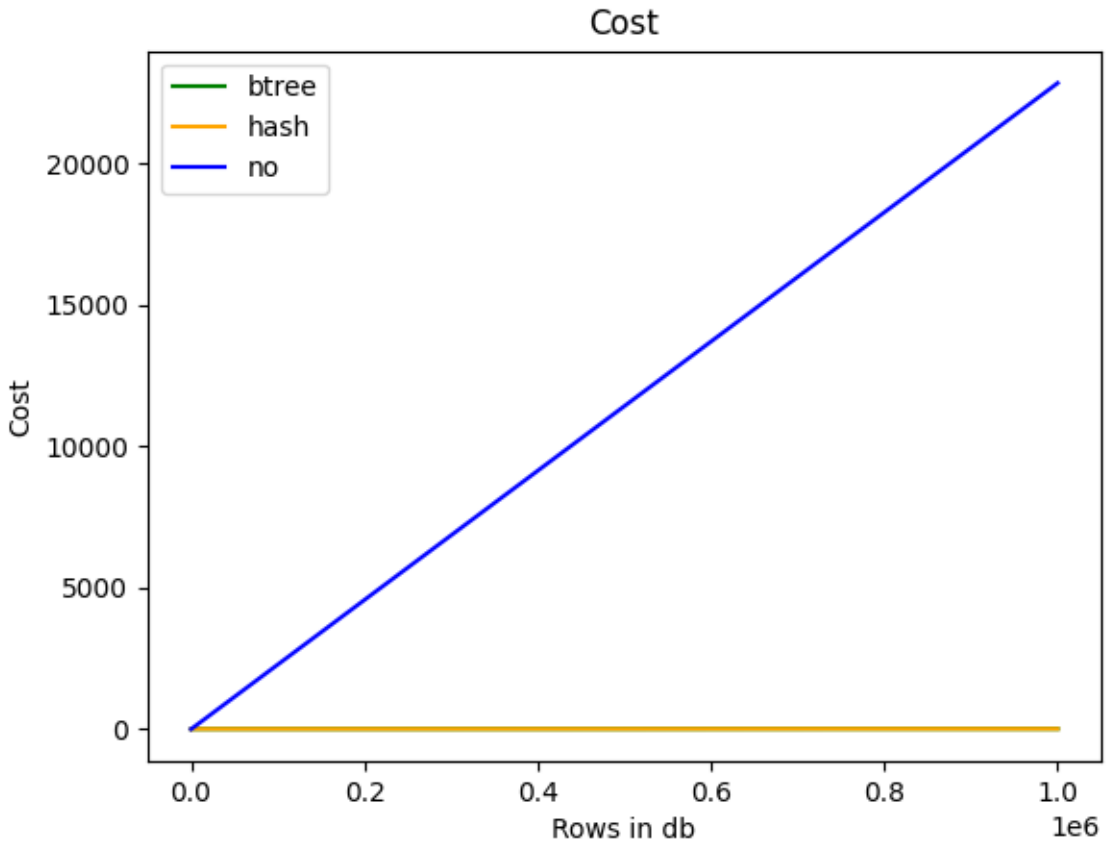
A. от 0 до 60M:



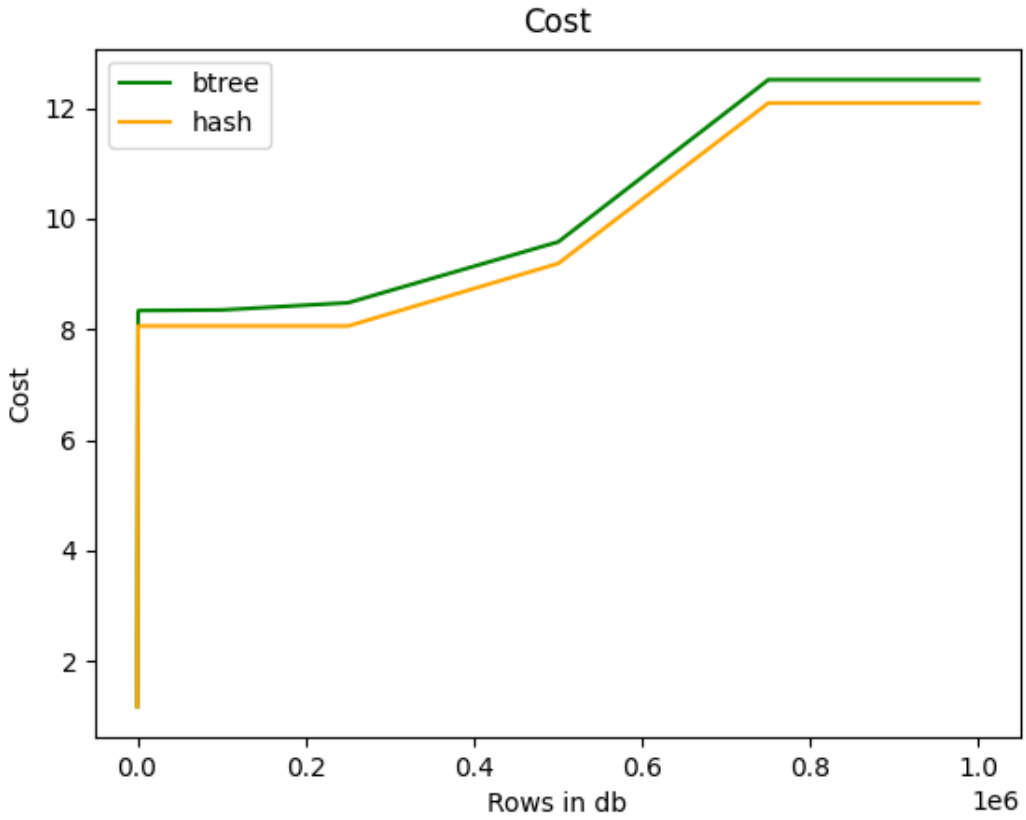
b. Измерения произведенные на se.ifmo.ru:

1. Зависимость по cost:

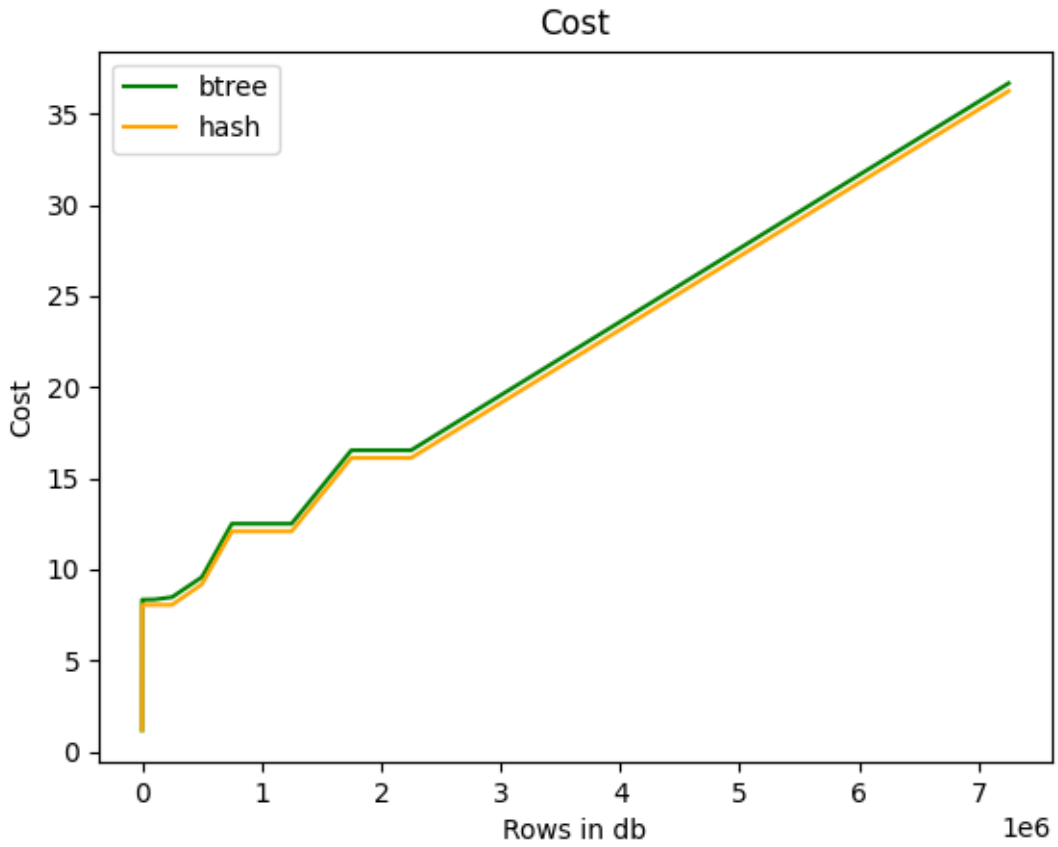
A. от 0 до 1M (с безиндесными замерами):



B. от 0 до 1M (только индексы):

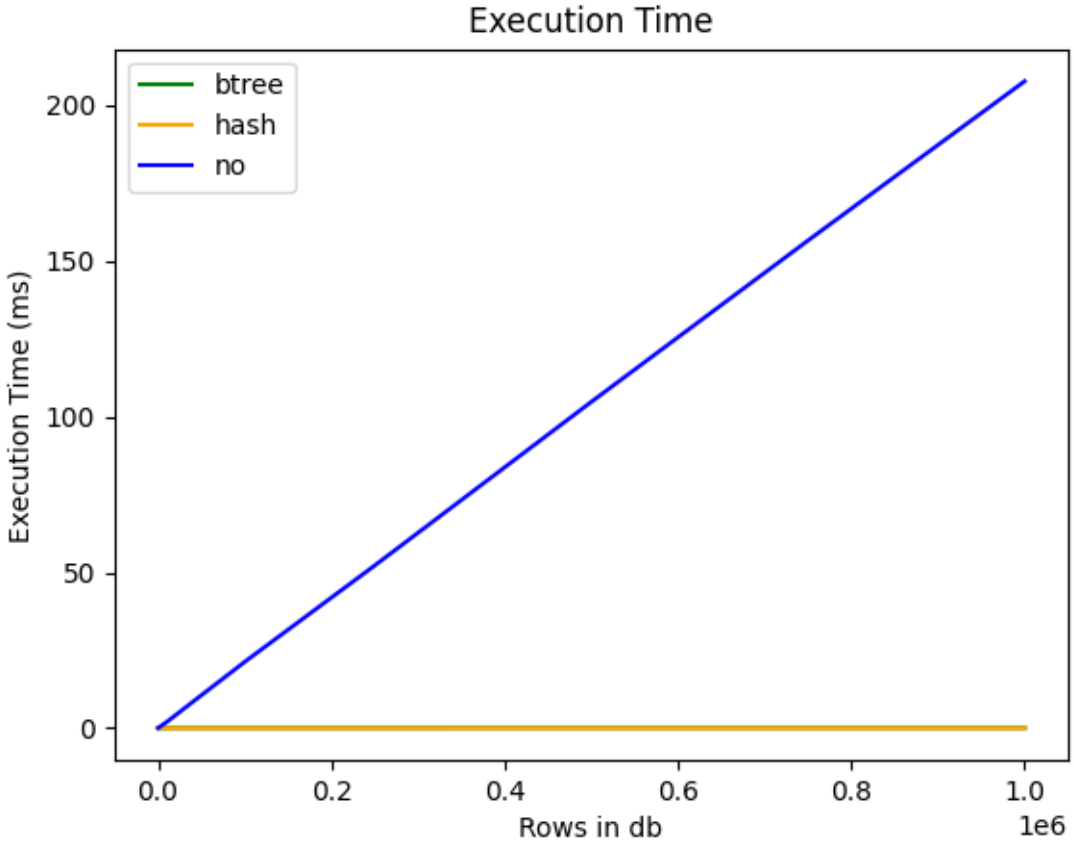


C. от 0 до 7M:

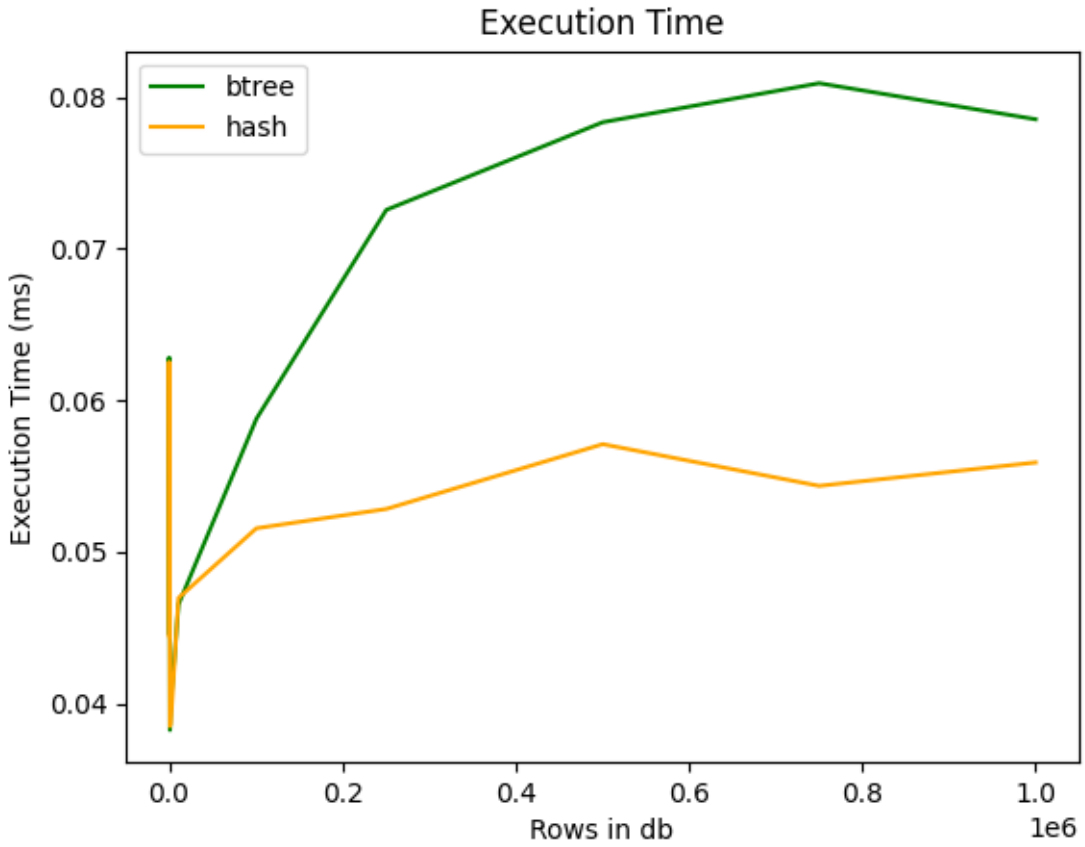


2. Зависимость по execution time:

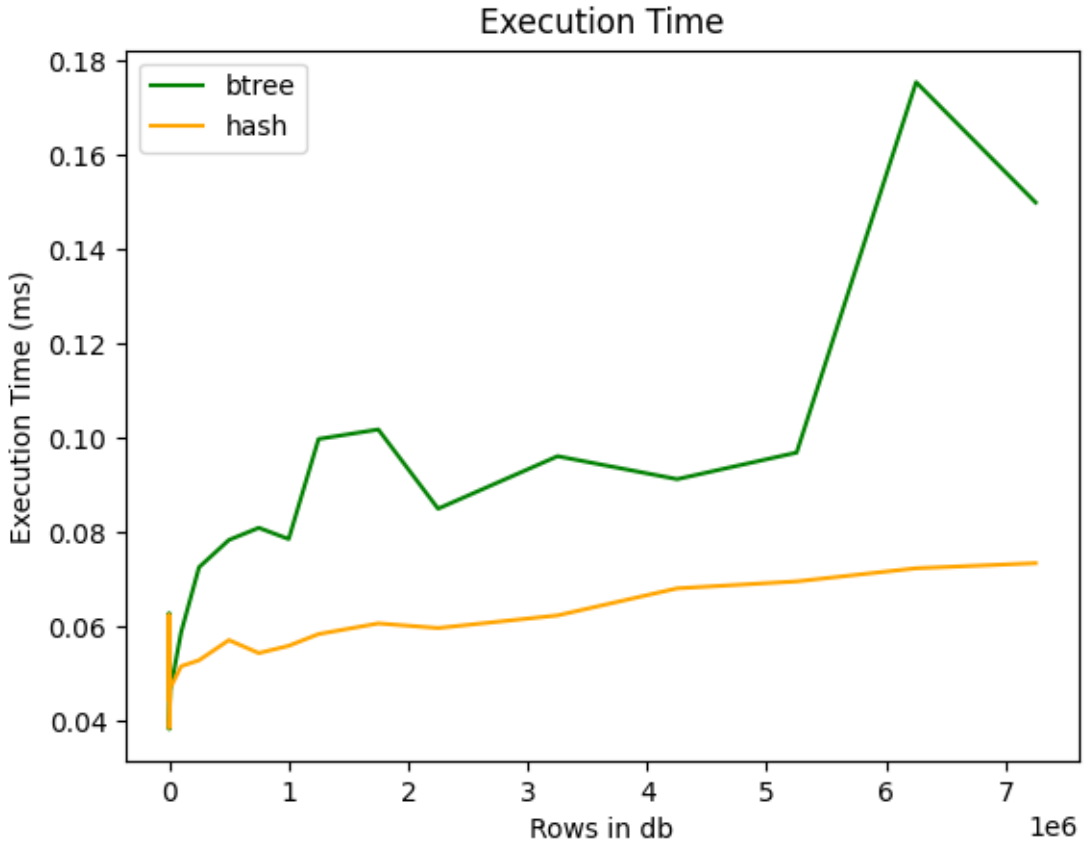
A. от 0 до 1M (с безиндексовыми замерами):



B. от 0 до 1M (только индексы):

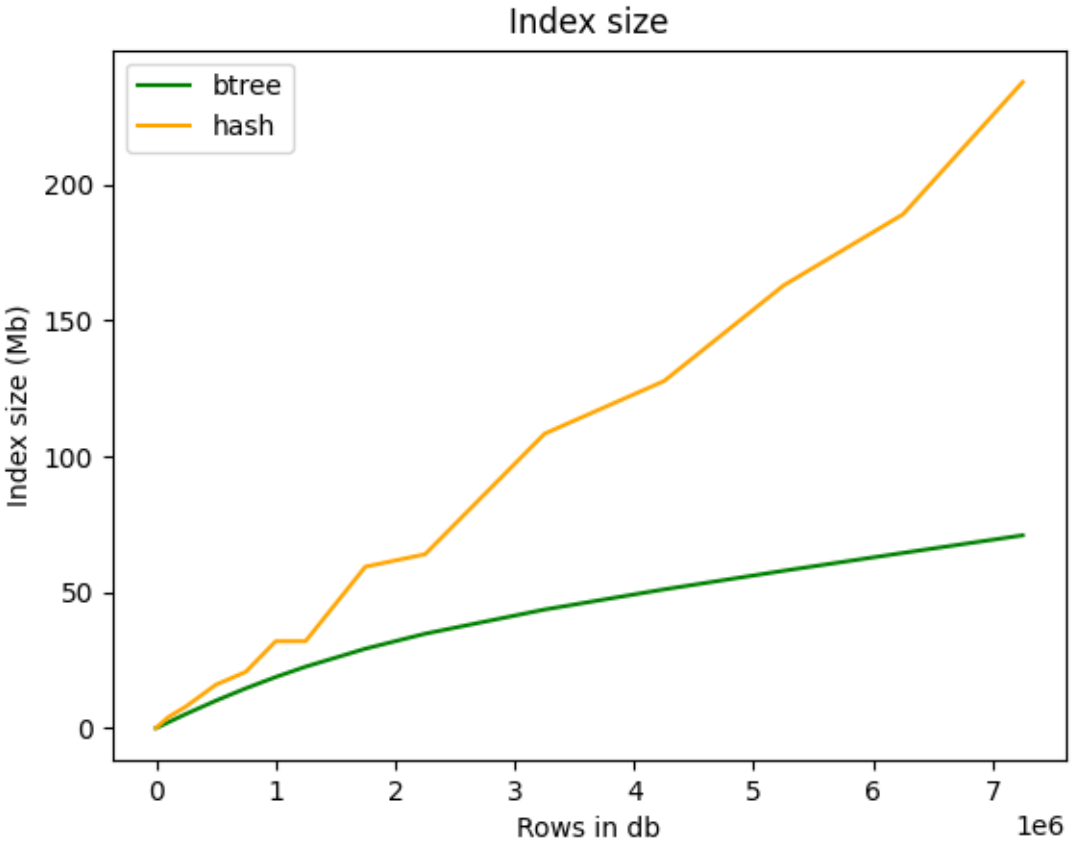


C. от 0 до 7M (только индексы):



3. Зависимость по размеру индексов:

A. от 0 до 7M (только индексы):



Исходя из данных измерений можно сделать вывод, что производительность hash-индекса не всегда выше производительности btree-индекса даже на операции сравнения. Это зависит от сервера, на котором запускается база данных и от заполненности таблицы. Также заметно, что преимущество hash-индекс начинает набирать на больших размерах данных, но вместе с этим растет и его размер при чем линейно в то время, как рост размера btree-индекса больше похож на логарифмический.

Скрипт собирающий данные:
https://github.com/NikosEvenTrue/DBMS_course/blob/dev/postgresql/index/collect.py

Скрипт построения графиков:
https://github.com/NikosEvenTrue/DBMS_course/blob/dev/postgresql/index/plot.py

6. Описание resp api

1. /signin
   a. POST

Точка предназначена для входа в приложение, получения user_id и jwt-токена.

```
Input: {
    "login": string,
    "password": string
}

Output: {
    "access_token": string,
    "user_id": integer
}
```

2. /logout
   a. POST

Точка предназначена для выхода из аккаунта, добавляет jwt-токен в список устаревших.

```
Headers: {
    "Authorization": “Bearer jwt-token”: string
}

Output: {
    "user_id": integer
}
```

2. /signup
   a. POST

Точка предназначена для регистрации нового аккаунта и получения user_id.

```
Input: {
    "login": string,
    "password": string
}

Output: {
    "user_id": integer
}
```

2. /users/<user_id>
   a. GET

Точка предназначена для получения информации о пользователе.

```
path-parameters: {
    "user_id": integer
}
```

```
Output: {
    "user": {
        "name": string,
        "description": string
    }
}
```

b. PATCH

Точка предназначена для модификации существующего пользователя.

```
path-parameters: {
    "user_id": integer
}

Headers: {
    "Authorization": “Bearer jwt-token”: string
}

Input: {
    "name": string,  # optional
    "description": string  # optional
}

Output: {
    "user": {
        "name": string,
        "description": string
    }
}
```

c. DELETE

Точка предназначена для удаления существующего пользователя.

```
path-parameters: {
    "user_id": integer
}

Headers: {
    "Authorization": “Bearer jwt-token”: string
}

Output: {
    "user_id": integer,
    "deleted": boolean
}
```

3. /users/<user_id>/folders

a. GET

Точка предназначена для получения всех папок, принадлежащих указанному пользователю. Пользователь может получить только папки принадлежащие ему.

```
path-parameters: {
    "user_id": integer
}
```

```
Headers: {
    "Authorization": "Bearer jwt-token": string
}

Output: {
    "folders": [
        {
            "id": integer,
            "name": string,
            "parent_folder_id": integer || null,
            "user_id": integer,
            "folder_settings_id": integer
        },
        ...
    ]
}
```

b. POST

```
Точка предназначена создания новых папок. Пользователь может создавать папки только
для себя.

path-parameters: {
    "user_id": integer
}

Headers: {
    "Authorization": "Bearer jwt-token": string
}

Input: {
    "name": string,
    "parent_folder_id": integer || null  # optional
}

Output: {
    "folder": {
        "id": integer,
        "name": string,
        "parent_folder_id": integer || null,
        "user_id": integer,
        "folder_settings_id": integer
    }
}
```

4. /users/<user_id>/folders/<folder_id>

a. GET

```
Точка предназначена для получения указанной папки.
Пользователь может получить только папку принадлежащие ему.

path-parameters: {
    "user_id": integer,
    "folder_id": integer
}

Headers: {
    "Authorization": "Bearer jwt-token": string
}
```

```
Output: {
    "folder": {
        "id": integer,
        "name": string,
        "parent_folder_id": integer || null,
        "user_id": integer,
        "folder_settings_id": integer
    }
}
```

b. PATCH

Точка предназначена для изменения существующих папок. Пользователь может изменять только свои папки.

```
path-parameters: {
    "user_id": integer,
    "folder_id": integer
}

Headers: {
    "Authorization": "Bearer jwt-token": string
}

Input: {
    "name": string,  # optional
    "parent_folder_id": integer || null,  # optional
    "folder_settings_id": integer  # optional
}

Output: {
    "folder": {
        "id": integer,
        "name": string,
        "parent_folder_id": integer || null,
        "user_id": integer,
        "folder_settings_id": integer
    }
}
```

c. DELETE

Точка предназначена для удаления существующей папки. Пользователь может удалять только свои папки.

```
path-parameters: {
    "user_id": integer,
    "folder_id": integer
}

Headers: {
    "Authorization": "Bearer jwt-token": string
}

Output: {
    "folder_id": integer,
    "deleted": boolean
}
```

5. /users/<user_id>/modules

a. GET

Точка предназначена для получения всех модулей, принадлежащих указанному пользователю.
Пользователь может получить все свои модули указав свой user_id, либо все опубликованные модули указав user_id интересующего пользователя.

```
path-parameters: {
    "user_id": integer
}

Headers: {
    "Authorization": “Bearer jwt-token”: string
}

Output: {
    "modules": [
        {
            "id": integer,
            "name": string,
            "user_id": integer,
            "folder_id": integer || null
        },
        ...
    ]
}
```

b. POST

Точка предназначена создания новых модулей. Пользователь может создавать модули только для себя.

```
path-parameters: {
    "user_id": integer
}

Headers: {
    "Authorization": “Bearer jwt-token”: string
}

Input: {
    "name": string,
    "parent_folder_id": integer || null  # optional
}

Output: {
    "module": {
        "id": integer,
        "name": string,
        "user_id": integer,
        "folder_id": integer || null
    }
}
```

6. /users/<user_id>/modules/<module_id>

a. GET

Метод GET носит в себе две функции.

При обращении без указания path-параметра public_info:
Метод будет возвращать указанный модуль.
Пользователь может получить доступ к любому своему или опубликованному модулю.

```
path-parameters: {
    "user_id": integer,
    "module_id": integer
}

Headers: {
    "Authorization": “Bearer jwt-token”: string
}

Output: {
    "module": {
        "id": integer,
        "name": string,
        "user_id": integer,
        "folder_id": integer || null
    }
}
```

При обращении с указания path-параметра public_info:
Метод будет возвращать публичную информацию для указанного модуля (теги и оценки).
Вы можете запросить публичную информацию на любой опубликованный модуль или любой свой модуль, если ваш модуль не опубликован вернется ответ с пустыми массивами.

/users/<user_id>/modules/<module_id>?public_info=<boolean>

```
path-parameters: {
    "user_id": integer,
    "module_id": integer,
    "public_info": boolean
}

Headers: {
    "Authorization": “Bearer jwt-token”: string
}

Output: {
    "tags": [
        {
            "id": integer,
            "name": string
        },
        ...
    ],
    "evaluations": [
        {
            "id": integer,
            "user_id": integer || null,
            "module_id": integer,
            "comment": string,
            "evaluation_id": integer
```

```
        },
        ...
    ]
}
```

b. PATCH

Точка предназначена для изменения существующих модулей. Пользователь может изменять только свои модули.

```
path-parameters: {
    "user_id": integer,
    "module_id": integer
}

Headers: {
    "Authorization": “Bearer jwt-token”: string
}

Input: {
    "name": string,  # optional
    "folder_id": integer || null  # optional
}

Output: {
    "folder": {
        "id": integer,
        "name": string,
        "user_id": integer,
        "folder_id": integer || null
    }
}
```

c. DELETE

Точка предназначена для удаления существующего модуля. Пользователь может удалять только свои модули.

```
path-parameters: {
    "user_id": integer,
    "module_id": integer
}

Headers: {
    "Authorization": “Bearer jwt-token”: string
}

{
    "module_id": integer,
    "deleted": boolean
}
```

d. COPY

Точка предназначена для копирования существующих модулей. Пользователь может копировать как свои, так и чужие опубликованные модули. Вместе с модулем копируется его содержимое (карточки).

```
path-parameters: {
    "user_id": integer,
    "module_id": integer
}

Headers: {
    "Authorization": “Bearer jwt-token”: string
}

Input: {
    "folder_id": integer || null  # optional
}

Output: {
    "module": {
        "id": intger,
        "name": string,
        "user_id": integer,
        "folder_id": integer || null
    }
}
```

e. POST

Метод POST носит в себе две функции.

При обращении к своим модулям вы публикуете его с указанным набором тегов:

```
path-parameters: {
    "user_id": integer,
    "module_id": integer
}


Headers: {
    "Authorization": “Bearer jwt-token”: string
}

Input: {
    "tags": [string, ...]
}

Output: {
    "msg": "tags were added"
}
```

При обращении к чужим модулям вы оставляете на него оценку:

```
path-parameters: {
    "user_id": integer,
    "module_id": integer
}

Headers: {
    "Authorization": “Bearer jwt-token”: string
}
```

```
Input: {
    "comment": string,
    "evaluation_id": integer
}

Output: {
    "evaluation": {
        "id": integer,
        "user_id": integer,
        "module_id": integer,
        "comment": string,
        "evaluation_id": integer
    }
}
```

7. /users/<user_id>/modules/<module_id>/cards

   a. GET

   Точка предназначена для получения всех карт, принадлежащих указанному модулю. Пользователь может запрашивать карты любого своего модуля или чужого опубликованного. Если модуль принадлежит вам ответ вернется с указание времени прошлого и следующего повторения, если не ваш – без. Время приходит в подобном формате "Thu, 16 Feb 2023 09:23:26 GMT".

```
path-parameters: {
    "user_id": integer,
    "module_id": integer
}

Headers: {
    "Authorization": “Bearer jwt-token”: string
}

Output: {
    "cards": [
        {
            "id": integer,
            "face": string,
            "back": string,
            "next_repeat_at": string,  # optional
            "last_repeated_at": string  # optional
        },
        ...
```

   b. POST

   Точка предназначена для создания карт в принадлежащих вам модулях.

```
path-parameters: {
    "user_id": integer,
    "module_id": integer
}

Headers: {
    "Authorization": “Bearer jwt-token”: string
}
```

```
Input: {
    "face": string,
    "back": string
}

Output: {
    "card": {
        "id": integer,
        "face": string,
        "back": string
    }
}
```

8. /users/<user_id>/modules/<module_id>/cards/<card_id>

a. GET

Точка предназначена для получения указанной карты. Если карта из принадлежащего вам модуля, она вернется с указанием времени прошлого и следующего повторения, если нет – без. Время приходит в подобном формате "Thu, 16 Feb 2023 09:23:26 GMT".

```
path-parameters: {
    "user_id": integer,
    "module_id": integer,
    "card_id": integer
}

Headers: {
    "Authorization": “Bearer jwt-token”: string
}

Output: {
    "card": {
        "id": integer,
        "face": string,
        "back": string,
        "next_repeat_at": string,  # optional
        "last_repeated_at": string   # optional
    }
}
```

b. PATCH

Точка предназначена для изменения существующей принадлежащей вам карты. В случае, если содержимое карты было изменено, а сама карта разделяется между несколькими модулями, карта будет пересоздана и возвращена с новым card_id, старая карта продолжит свое существование с прежним card_id и будет доступна из модулей, в которых она лежала.

```
path-parameters: {
    "user_id": integer,
    "module_id": integer,
    "card_id": integer
}

Headers: {
    "Authorization": “Bearer jwt-token”: string
}
```

```
Input: {
    "module_id": 1000011,  # optional
    "face": string,  # optional
    "back": string,    # optional
    "next_repeat_at": string,  # optional
    "last_repeated_at": string   # optional
}

Output: {
    "card": {
        "id": integer,
        "face": string,
        "back": string,
        "next_repeat_at": string,
        "last_repeated_at": string
    }
}
```

c. DELETE

Точка предназначена для удаления указанной принадлежащей вам карты.

```
path-parameters: {
    "user_id": integer,
    "module_id": integer,
    "card_id": integer
}

Headers: {
    "Authorization": “Bearer jwt-token”: string
}

Output: {
    "card_id": 1000008,
    "deleted": true
}
```

d. COPY

Точка предназначена копирования принадлежащей вам карты или карты из опубликованного модуля.

```
path-parameters: {
    "user_id": integer,
    "module_id": integer,
    "card_id": integer
}

Headers: {
    "Authorization": “Bearer jwt-token”: string
}

Input: {
    "module_id": intger
}
```

```
Output: {
    "card": {
        "id": integer,
        "face": string,
        "back": string,
        "next_repeat_at": string,
        "last_repeated_at": string
    }
}
```

9. /tags?start=<string>&limit=<integer>

   e. GET

```
Точка предназначена для получения тегов, которые начинаются с указанного текста и расположены в порядке популярности (кол-во связанных с данным тегов модулей). Предназначено для реализации автодополнения.

path-parameters: {
    "limit": integer,  # optional, default = 10
    "start": string
}

Output: {
    "tags": [
        {
            "id": 106894,
            "name": "abcd9d0b"
        },
        ...
    ]
}
```

10. /search?tag=<string>

    f. GET

```
Точка предназначена для получения всех модулей, которые опубликованы под указанным тегом.

path-parameters: {
    "tag": string
}

Output: {
    "modules": [
        {
            "id": integer,
            "name": string,
            "user_id": integer,
            "folder_id": integer || null
        },
        ...
    ]
}
```

Ссылка на репозиторий с проектом:
https://github.com/NikosEvenTrue/DBMS_course