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

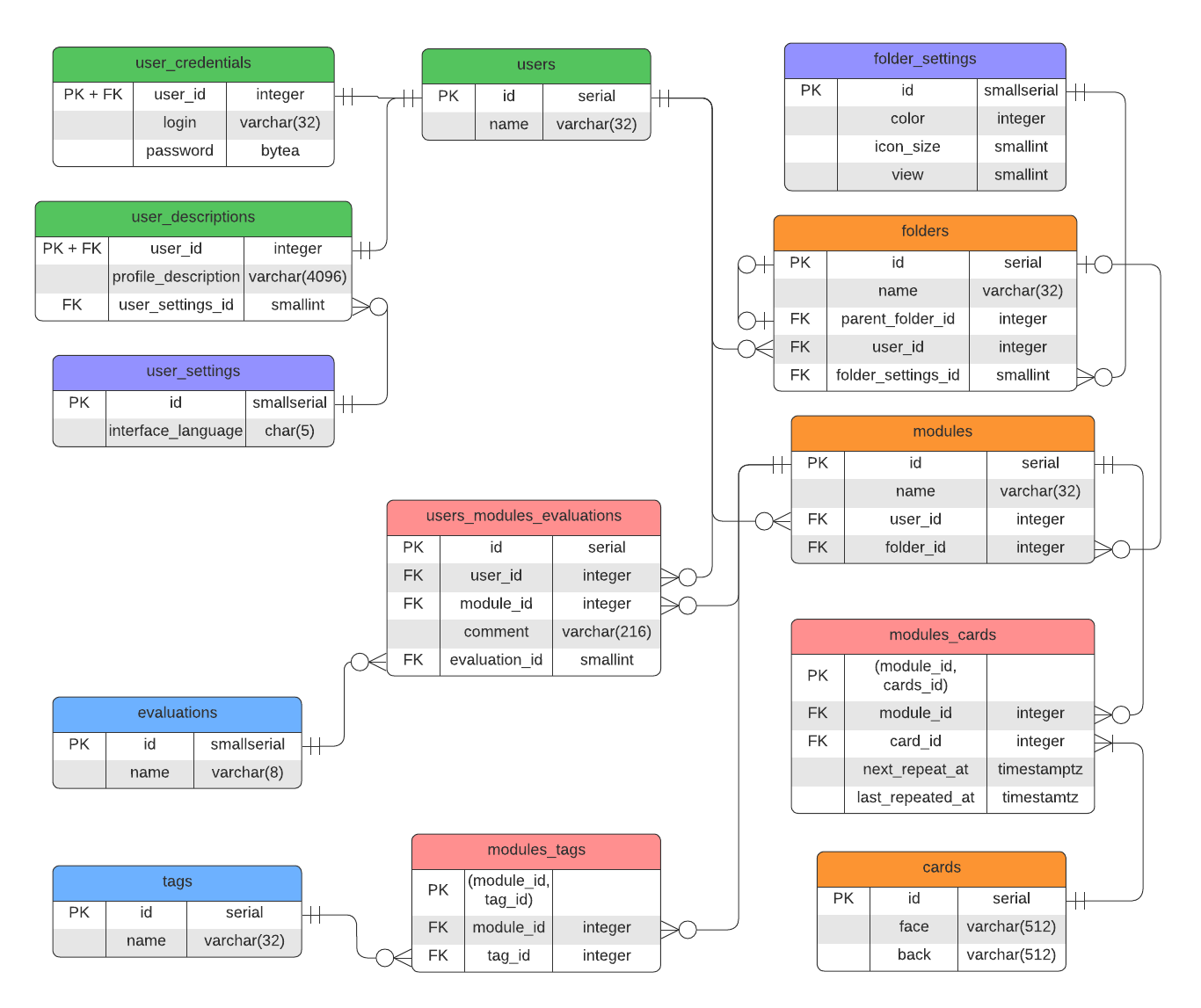
Преподаватели:

Харитонова Анастасия Евгеньевна

Санкт-Петербург

2022

1. Даталогическая модель



1. Реализация таблиц в СУБД PostgreSQL
2. 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 |

1. 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.

1. 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.

1. 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.

1. 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 |

1. 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).

1. 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.

1. 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.

1. 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.

1. 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.

1. 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.

1. 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.

1. 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.

1. Реализация функций и триггеров в СУБД PostgreSQL
2. 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.

1. 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.

1. 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.

1. 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.

1. 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.

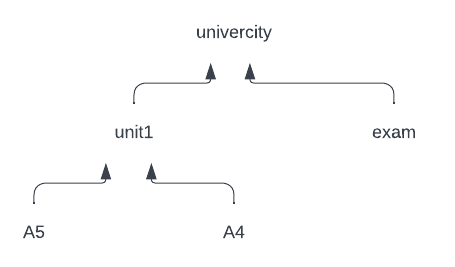
Пример работы:

**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:

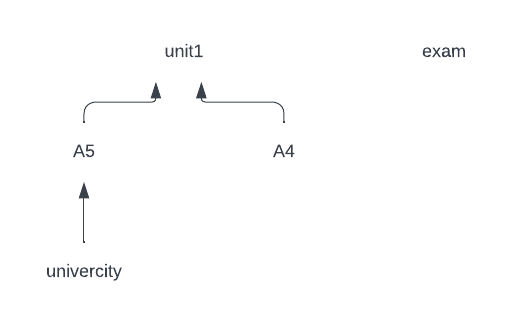


Допустим пользователем драг-энд-дропом перетащил папку univecity в A5:

**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.ps>1

**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>

1. 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.

1. Индексы для user\_credentials

Наиболее частые операции:

* 1. выборка по login:
     1. при входе:

**SELECT** *\** **FROM** user\_credentials  
**WHERE** user\_credentials.**login** = **‘login’  
LIMIT** 1;

Создание индексов:

* 1. по полю login:

Так как поле login имеет ограничение UNIQUE индекс уже создан.

1. Индексы для user\_descriptions

Наиболее частые операции:

* 1. выборка по user\_id:
     1. при запросе пользователя:

**SELECT** *\** **FROM** user\_descriptions  
**WHERE** user\_descriptions.**user\_id** = user\_id  
**LIMIT** 1;

Создание индексов:

* 1. по полю user\_id:

Так как поле user является PRIMARY KEY индекс уже создан.

1. Индексы для users

Наиболее частые операции:

* 1. выборка по id:
     1. при запросе пользователя:

**SELECT** *\** **FROM** users  
**WHERE** users.**id** = user\_id;

Создание индексов:

* 1. по полю login:

Так как поле id является PRIMARY KEY индекс уже создан.

1. Индексы для folders

Наиболее частые операции:

* 1. выборка по id:
     1. при обращении к папке:  
        **SELECT** \* **FROM** folders **WHERE id** = fodler\_id;
  2. выпорка по user\_id:
     1. при каскадном удалении user

**DELETE** **FROM** folders **WHERE** **user\_id** = user\_id;

* + 1. при выборке всех папок

**SELECT** \* **FROM** folders **WHERE** **user\_id** = user\_id;

* 1. выборка по parent\_folder\_id:
     1. поиск вложенных папок (триггер предотвращения циклических ссылок)

**SELECT** id **FROM** folders **WHERE** **parent\_folder\_id** = parent\_folder\_id;

* + 1. удаление вложенных папок при удалении родительской

**DELETE** **FROM** folders **WHERE** **parent\_folder\_id** = parent\_folder\_id;

Создание индексов:

* 1. по полю id:

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

* 1. 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.

* 1. 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’е.

1. Индексы для modules

Наиболее частые операции:

* 1. выборка по id:
     1. при обращении к модулю:

**SELECT** \* **FROM** modules **WHERE** **id** = module\_id;

* 1. выборка по user\_id:
     1. удаление модуля (при каскадном удалении user):

**DELETE** **FROM** modules **WHERE** **user\_id** = user\_id;

* + 1. выборка всех модулей по пользователя:

**SELECT** \* **FROM** modules **WHERE** **user\_id** = user\_id;

Создание индекса:

* 1. по полю id:

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

* 1. по полю user\_id:

Сравнение индексов:

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

Остановимяс на btree-index

1. Индексы для modules\_cards

Наиболее частые операции:

* 1. выборка по module\_id:
     1. при удалении модуля:

**DELETE** **FROM** modules\_cards **WHERE** **module\_id** = module\_id;

* + 1. 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;

* 1. 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;

* 1. 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:

* 1. 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.

* 1. 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.

* 1. 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.

1. Индексы для cards.

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

1. Индексы для users\_modules\_evaluations

Наиболее частые операции:

* 1. 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;
  2. выборка по user\_id:
     1. при каскадном удалении user

**UPDATE** users\_modules\_evaluations **SET** **user\_id** = NULL

**WHERE** **user\_id** = user\_id;

* 1. выборка по module\_id:
     1. при каскадном удалении user:

**UPDATE** users\_modules\_evaluations **SET** **user\_id** = NULL

**WHERE** **user\_id** = user\_id;

* + 1. при получении оценок модуля:

**SELECT** \* **FROM** users\_modules\_evaluations **WHERE** **module\_id** = module\_id;

Создание индексов:

* 1. 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 |

* 1. 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 |

* 1. 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.

1. Индексы для modules\_tags

Наиболее частые операции:

* 1. 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;

* + 1. 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;

* + 1. 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'**

* 1. 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:

* 1. по полю module\_id:

Так как (module\_id, tag\_id) являются PK, индекс уже создан.

* 1. 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.

1. Индексы для tags

Наиболее частые операции:

* 1. 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;

* 1. 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);

* + 1. 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'**

* + 1. при выяснении существует ли тег:

**SELECT** *\** **FROM** tags  
**WHERE** tags.**name IN** (**'tag\_name\_1'**, **'tag\_name\_2'**);

Создание индексов:

* 1. по полю id:

Так как id являются PK, индекс уже создан.

* 1. по полю name:

C 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’а, 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>

1. 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:

* 1. 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 |

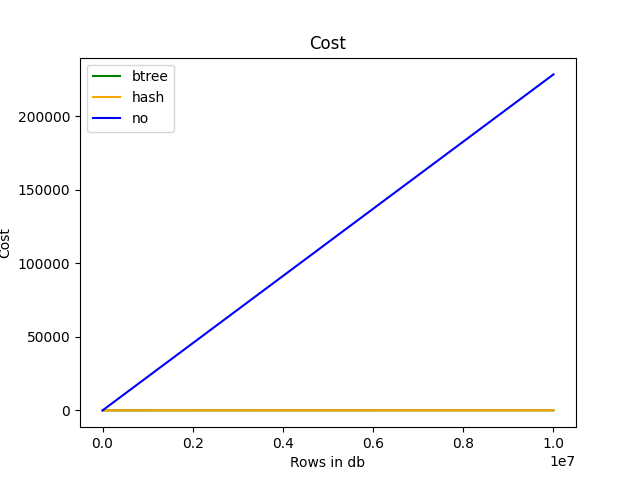
* 1. на se.ifmo.ru:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 |

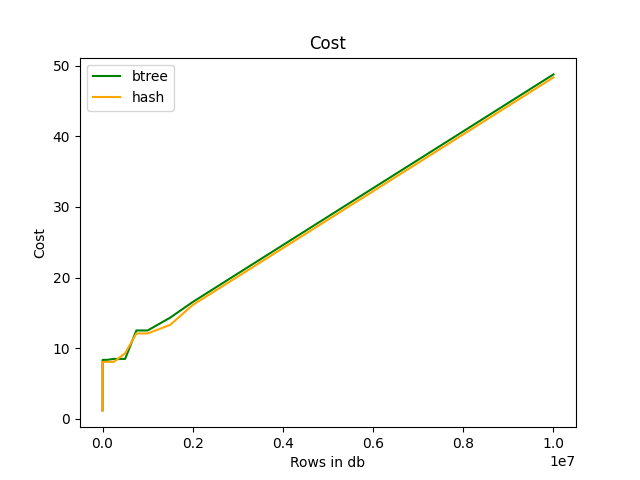
В виде графиков:

no – обозначает измерение без использования индекса

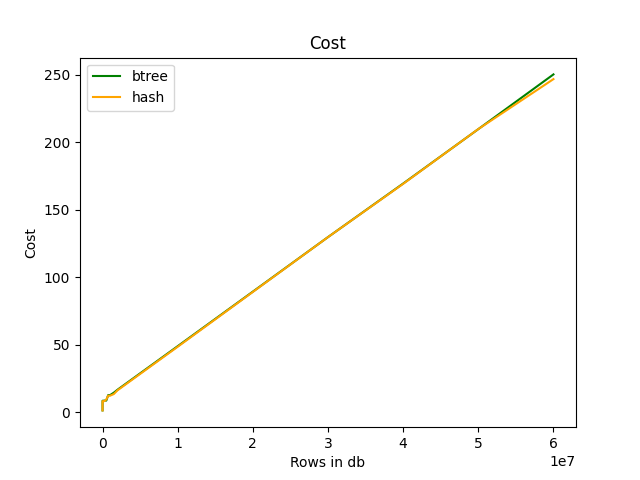
* 1. Измерения произведенные на моем компьютере:
     1. Зависимость по cost:
        1. от 0 до 10М (с безидексовыми замерами):



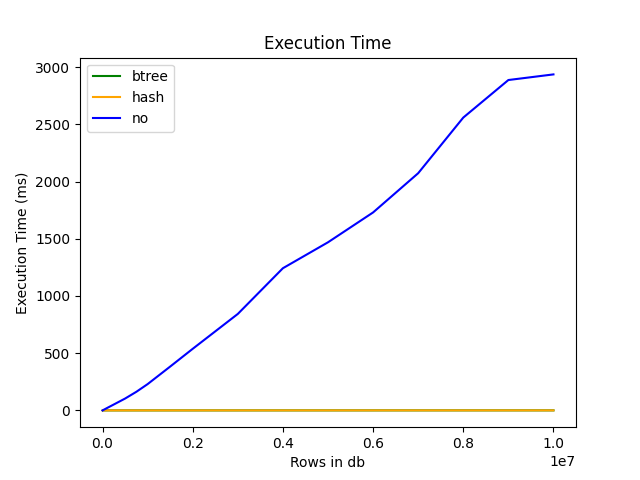
* + - 1. от 0 до 10М (только и индексами):



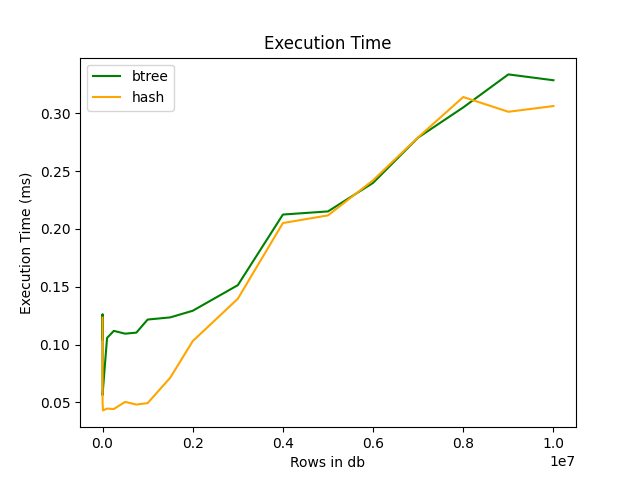
* + - 1. от 0 до 60М (только с индексами):



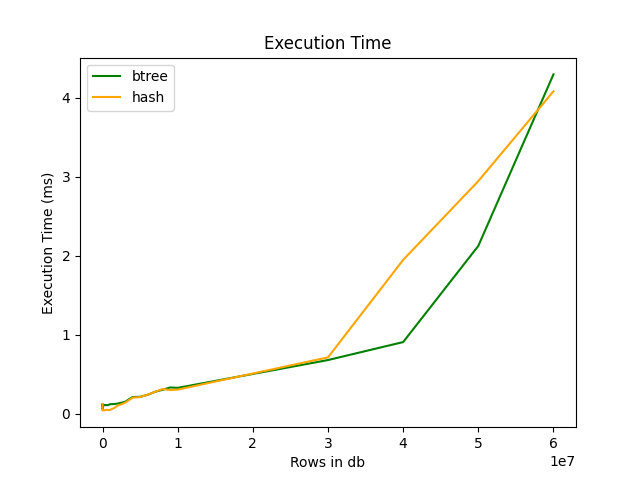
* + 1. Зависимость по execution time:
       1. от 0 до 10М (с безиндексовыми замерами):



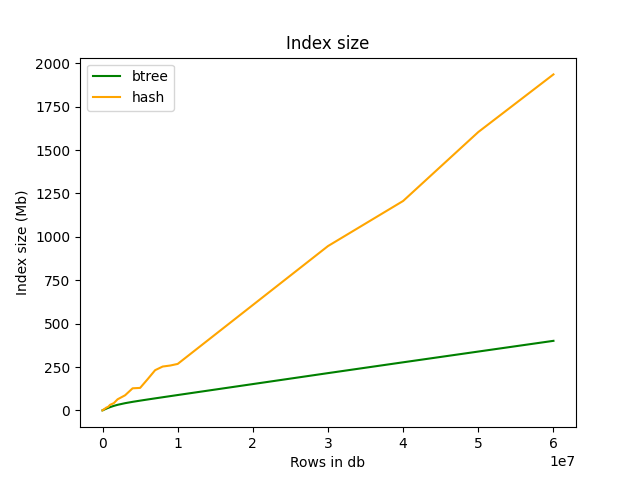
* + - 1. от 0 до 10М (только индексы):



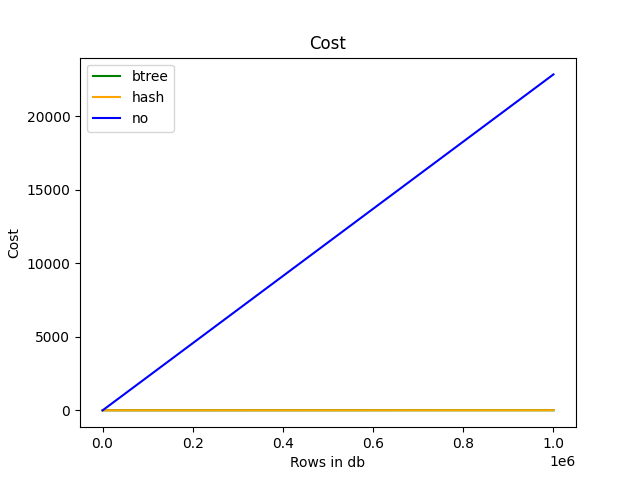
* + - 1. от 0 до 60М (только индексы):

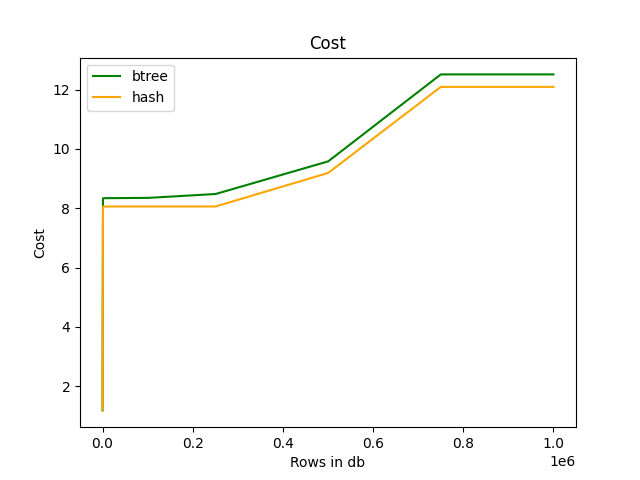


* + 1. Зависимость по размеру индексов:
       1. от 0 до 60М:

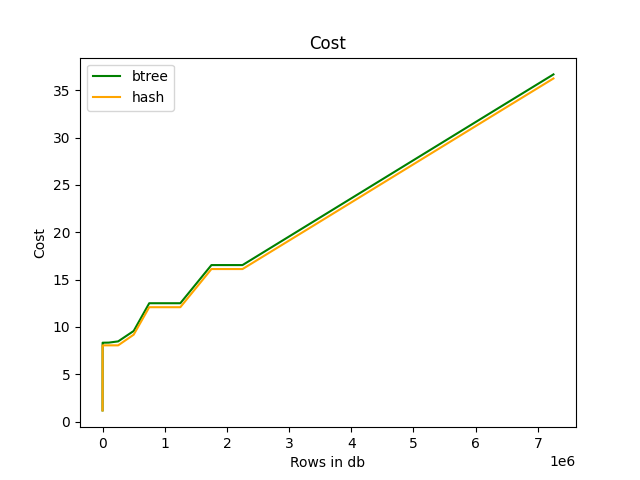


* 1. Измерения произведенные на se.ifmo.ru:
     1. Зависимость по cost:
        1. от 0 до 1М (с безиндесными замерами):

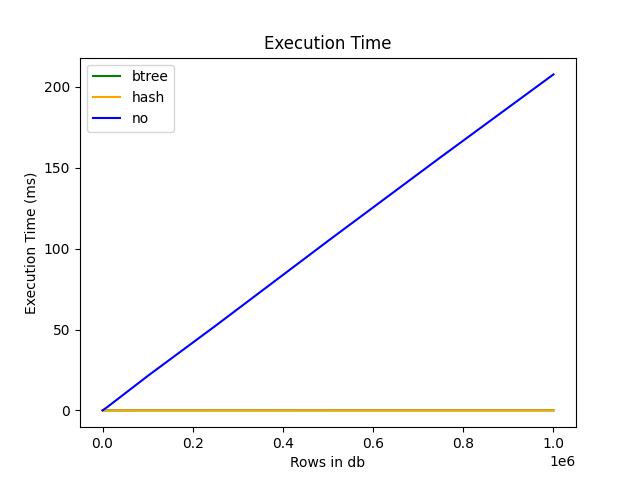


* + - 1. от 0 до 1М (только индексы): 

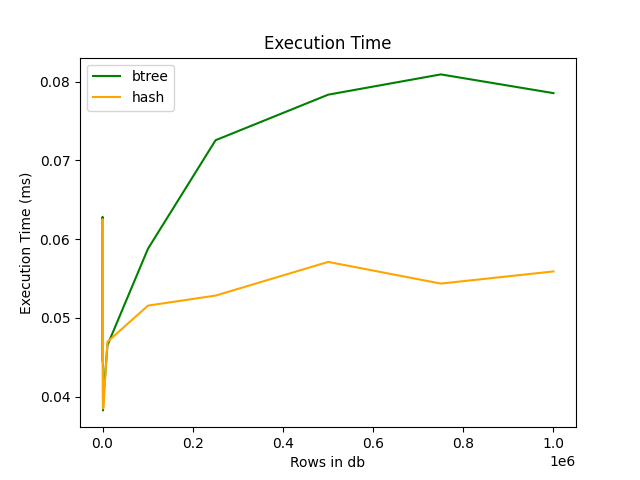
* + - 1. от 0 до 7М:



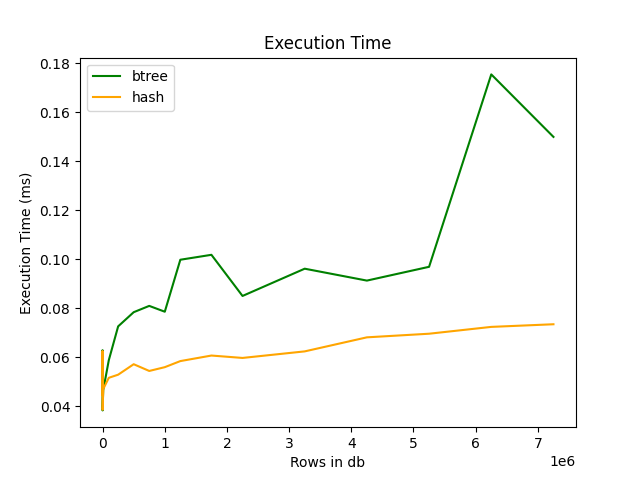
* + 1. Зависимость по execution time:
       1. от 0 до 1М (с безиндексовыми замерами):



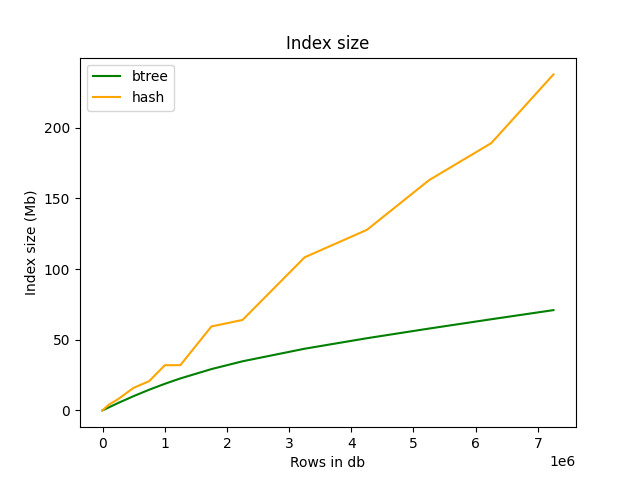
* + - 1. от 0 до 1М (только индексы):



* + - 1. от 0 до 7М (только индексы):



* + 1. Зависимость по размеру индексов:
       1. от 0 до 7М (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>

1. Описание resp api
   * 1. /signin
   1. 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

}

* + 1. /logout
  1. POST

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

Headers: {

    "Authorization": “Bearer jwt-token”: string

}

Output: {

    "user\_id": integer

}

* + 1. /signup
  1. POST

Точка предназначена для регистрации нового аккаунта и получения user\_id.

Input: {

    "login": string,

    "password": string

}

Output: {

    "user\_id": integer

}

* + 1. /users/<user\_id>
  1. GET

Точка предназначена для получения информации о пользователе.

path-parameters: {

    "user\_id": integer

}

Output: {

    "user": {

        "name": string,

        "description": string

    }

}

* 1. 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

    }

}

* 1. DELETE

Точка предназначена для удаления существующего пользователя.

path-parameters: {

    "user\_id": integer

}

Headers: {

    "Authorization": “Bearer jwt-token”: string

}

Output: {

    "user\_id": integer,

    "deleted": **boolean**

}

* + 1. /users/<user\_id>/folders
  1. 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

        },

        ...

    ]

}

* 1. 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

    }

}

* + 1. /users/<user\_id>/folders/<folder\_id>
  1. 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

    }

}

* 1. 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

    }

}

* 1. 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**

}

* + 1. /users/<user\_id>/modules
  1. 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**

        },

        ...

    ]

}

* 1. 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**

    }

}

* + 1. /users/<user\_id>/modules/<module\_id>
  1. 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

        },

        ...

    ]

}

* 1. 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**

    }

}

* 1. 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**

}

* 1. 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**

    }

}

* 1. 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

    }

}

* + 1. /users/<user\_id>/modules/<module\_id>/cards
  1. 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

        },

        ...

* 1. 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

    }

}

* + 1. /users/<user\_id>/modules/<module\_id>/cards/<card\_id>
  1. 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

    }

}

* 1. 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

    }

}

* 1. 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**

}

* 1. 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

    }

}

* + 1. /tags?start=<string>&limit=<integer>
  1. 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"

        },

        ...

    ]

}

* + 1. /search?tag=<string>
  1. 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>