

Университет ИТМО, факультет ПИиКТ

Курсовая работа. Часть №3

Дисциплина: Информационные системы и базы данных

Выполнили: Чангалиди Антон

Чайка Алексей

Группа: Р33113

Преподаватель:
Николаев Владимир Вячеславович

г. Санкт-Петербург

2020 г.

Задание

Реализовать даталогическую модель в реляционной СУБД PostgreSQL:

- Создать необходимые объекты базы данных.
- Заполнить созданные таблицы тестовыми данными.
- Сделать скрипты для:
 - создания/удаления объектов базы данных;
 - заполнения/удаления созданных таблиц.
- Обеспечить целостность данных при помощи средств языка DDL.
- Добавить в базу данных триггеры для обеспечения комплексных ограничений целостности.
- Реализовать функции и процедуры на основе описания бизнес-процессов (из этапа №1).
- Произвести анализ использования созданной базы данных:
 - выявить наиболее часто используемые запросы к объектам базы данных;
 - результаты представить в виде текстового описания.
- Создать индексы и доказать, что они полезны для вашей базы данных:
 - доказательство должно быть приведено в виде текстового описания

Весь код и актуальная версия отчета доступны тут:

https://github.com/TohaRhymes/databases_pharmacy_coursework



Описание предметной области

Фармакологический рынок на данный момент - отдельная ниша как экономики, так и науки любого государства. Поэтому нашей предметной областью мы взяли фармакологию и рассмотрим ее с разных ракурсов.

Существует много ПАТОГЕНОВ (разной биологической природы: вирусы/бактерии/прионы/грибы), которые могут вызывать БОЛЕЗНИ разной степени тяжести; кроме всего прочего существуют ЯДЫ, которые вредны сами по себе.

Все это лечится с помощью ЛЕКАРСТВ (действующих веществ) различных категорий. ФАРМАКОЛОГИЧЕСКИЕ КОМПАНИИ производят их и выпускают под ТОРГОВЫМИ НАЗВАНИЯМИ (одно лекарство может выпускаться под разными торговыми названиями с разной средней ценой, но иметь одно и то же действующее вещество). Кроме всего прочего, можно лечиться ГОМЕОПАТИЕЙ (ее тоже производят компании) и НАРОДНОЙ МЕДИЦИНОЙ разного происхождения. Одно вещество может лечить несколько болезней, как и одну болезнь можно вылечить несколькими веществами.

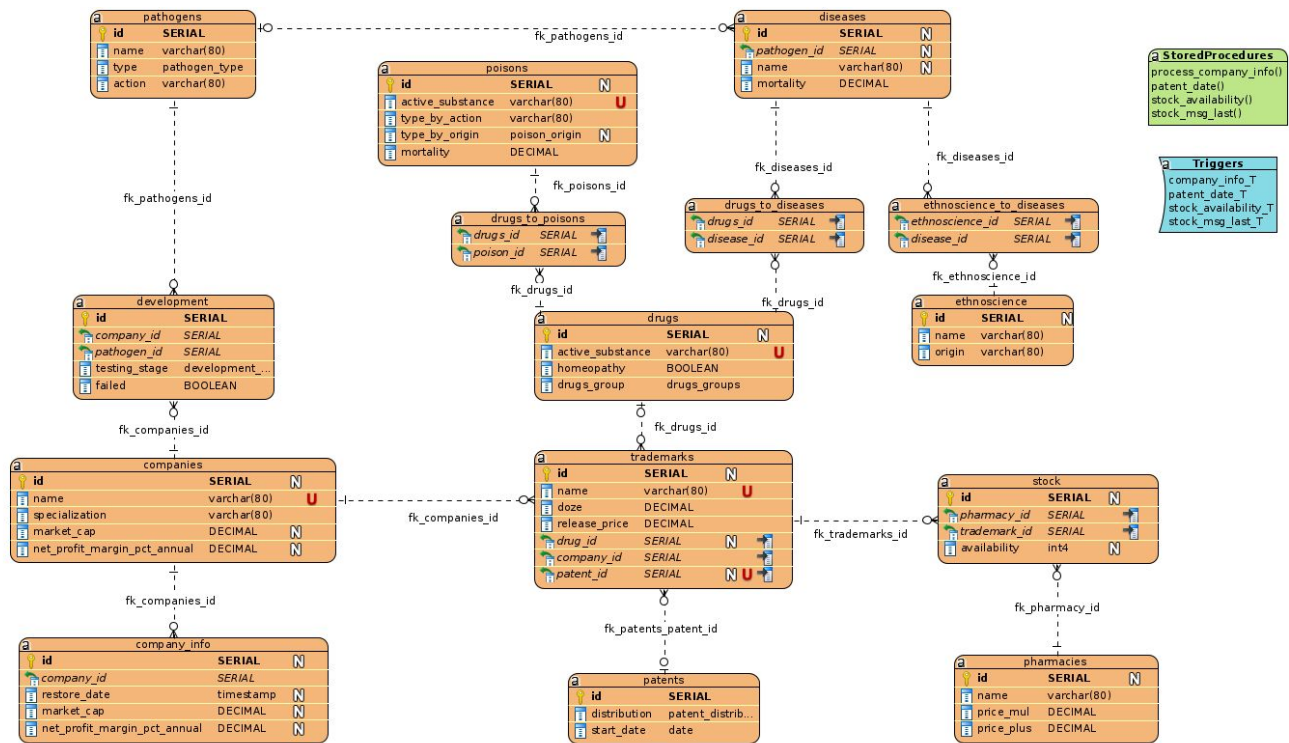
Помимо уже придуманных лекарств, некоторые продвинутые компании ведут РАЗРАБОТКИ новых (как против болезней, так и против ядов): какие-то еще только придумываются, какие-то находятся на различных стадиях испытаний, какие-то уже получили ПАТЕНТЫ и поступили в производство (патент не обязательное условие продажи: лекарства, придуманные давно, могут производиться и без них - но тоже под торговыми марками). Особенно важно вести разработку против тех болезней, лекарство от которых еще не придумано.

Все старые (выпускающиеся давно) и новые (только получившие патент) лекарства могут продаваться в АПТЕКАХ. Если человек хочет вылечиться, он идет туда. Лекарство может быть доступно сейчас, располагаться на складе, или быть вообще недоступным в данный момент - это можно проверить в каждой аптеке отдельно: если оно есть - купить его, если нет - проверить на складе, и если оно есть там - заказать оттуда. При поступлении товара в аптеку - он отмечается, как доступный. Ну а если товар отсутствует и на складе - можно пойти в другую аптеку.

Если же человек выбирает лечение с помощью народных средств, он просто может узнать, чем лечится его болезнь.

Некоторые компании могут торговаться на бирже, поэтому у них есть экономические показатели. Вся история изменения показателей каждой компании хранится и может быть использована.

Наша модель:



Код создания:

Ниже представлен пример создания 2х таблиц и связи М-М, полный код по ссылке:

https://github.com/TohaRhymes/databases_pharmacy_coursework/blob/main/creation.sql

```
CREATE TYPE drugs_groups AS ENUM ('Group A (prohibited substances)', 'Group B (limited turnover)', 'Group C (free circulation)');
```

```
CREATE TABLE diseases
(
    id SERIAL PRIMARY KEY,
    pathogen_id INTEGER
        CONSTRAINT fk_pathogens_id REFERENCES pathogens (id) ON DELETE CASCADE,
    name VARCHAR(80),
    mortality DECIMAL NOT NULL
        DEFAULT 0 CHECK ( mortality >= 0 and mortality <= 1 )
);
```

```
CREATE TABLE drugs
(
    id SERIAL PRIMARY KEY,
    active_substance VARCHAR(80) UNIQUE NOT NULL,
    homeopathy BOOLEAN NOT NULL
        DEFAULT false,
    drugs_group drugs_groups NOT NULL
);
```

```
CREATE TABLE drugs_to_diseases
```

```
(
    drugs_id    INTEGER
        CONSTRAINT fk_drugs_id REFERENCES drugs (id) ON DELETE CASCADE
        NOT NULL,
    disease_id  INTEGER
        CONSTRAINT fk_diseases_id REFERENCES diseases (id) ON DELETE CASCADE
        NOT NULL,
    PRIMARY KEY (drugs_id, disease_id)
);
```

Заполнение БД:

Ниже показаны несколько примеров заполнения, полный пример заполнения может быть рассмотрен по ссылке: https://github.com/TohaRhymes/databases_pharmacy_coursework/blob/main/insertion.sql

```
INSERT INTO pathogens(name, type, action) VALUES ('cov2019', 'virus', 'infect lungs');
INSERT INTO pathogens(name, type, action) VALUES ('Treponema pallidum', 'bacterium', 'violate
skeen');
```

```
INSERT INTO diseases(pathogen_id, name, mortality) VALUES (1, 'loss of smell', 0.001);
INSERT INTO diseases(pathogen_id, name, mortality) VALUES (1, 'coma', 0.95);
INSERT INTO diseases(pathogen_id, name, mortality) VALUES (2, 'Syphilis', 0.23);
```

```
INSERT INTO drugs(active_substance, homeopathy, drugs_group) VALUES ('Echinacea purpurea
D3',true,'Group C (free circulation)');
INSERT INTO drugs(active_substance, homeopathy, drugs_group) VALUES
('interferon',false,'Group B (limited turnover)');
INSERT INTO drugs(active_substance, homeopathy, drugs_group) VALUES ('Phenol',false,'Group A
(prohibited substances)');
```

```
INSERT INTO drugs_to_diseases VALUES (2, 1);
INSERT INTO drugs_to_diseases VALUES (3, 2);
```

Скрипты для массового заполнения:

Также был разработан скрипт для массового заполнения таблиц (настройки выставляются начиная со строки:

```
# settings - amount of rows
```

(номера строк: 99-113)

Скрипт:

https://github.com/TohaRhymes/databases_pharmacy_coursework/blob/main/insertion_generator.py

Сценарии использования:

Основные два сценария использования: это действия от имени компании и действия от имени аптеки. Если пользователь действует от имени аптеки, он может продавать конкретные лекарства, смотреть какие лекарства подходят против конкретной болезни и выводить их цену, произвести закупку лекарства. Если пользователь выполняет действия от имени компании, он может выпустить лекарство, получив при этом патент на него.

Система автоматически раз в определенный промежуток обновляет информацию о компании.

Также существует “режим Бога” (администратор), в котором можно добавить новые патогены, яды, народные средства.

Бизнес-процессы:

- РЕГИСТРАЦИЯ ПОЛЬЗОВАТЕЛЯ

Пользователь может зарегистрироваться как компания и как аптека, в зависимости от этого ему будут доступны разные функции (см. далее). При регистрации вносится новая строка в необходимую таблицу (аптека/компания).

- ВХОД КАК АПТЕКА/КОМПАНИЯ/АДМИН

Посетитель входит на сайт и вводит свои данные, в зависимости от того кем он изначально регистрировался (аптека/пользователь/админ), ему предоставляется разный функционал. Система обращается к БД, и проверяет, есть ли данная аптека/компания в системе.

- РЕЖИМ БОГА

В системе залогинен админ. Может открыть существующий список болезней/ядов/патогенов и поменять их характеристики (смертность - морталити, тип), а также создать новые (используя соответствующие таблицы).

- ПРОДАЖА ЛЕКАРСТВ

В системе залогинена аптека. Покупатель (ирл) покупает лекарство и сотрудник, работающий с системой (залогиненный как аптека) изменяет оставшееся количество данного лекарства на складе (конкретно, количество декрементируется) в таблице stock.

- ЗАКУПКА ЛЕКАРСТВ

В системе залогинена аптека. Сотрудник, закупив лекарства, указывает в системе перечень купленных, что влечет за собой изменения на складе данной аптеки (в таблице stock).

- ВЫПУСК НОВОГО ЛЕКАРСТВА

В системе залогинена компания. Компания получает патент на новую trademark и указывает всю информацию о новом продукте. После коммита транзакции новая trademark становится доступной для закупки аптеками по release_price.

- ОБНОВЛЕНИЕ ИНФОРМАЦИИ О КОМПАНИИ

Актером выступает сама система. Поскольку экономическая информация о компании общедоступна, система будет автоматически обновлять экономические показатели. С каждым

обновлением, будет фиксироваться снимок в соответствующей таблице (company_info - она хранит всю историю изменений).

Триггеры, функции:

Для сохранения целостности введем 4 триггера (сохранение истории изменения капитала компании, сохранение даты при выдаче патента (введенная или текущая), и достижение нуля лекарств - будем извещать пользователя)

Код можно увидеть тут:

https://github.com/TohaRhymes/databases_pharmacy_coursework/blob/main/functions_with_triggers.sql

1. Табличка company_info - хранит историю изменений капитала и оборота денежных средств компании, поэтому ее полностью автоматически будет заполнять функция с триггером:

```
CREATE OR REPLACE FUNCTION process_company_info() RETURNS TRIGGER AS
$company_info$
BEGIN
    --
    -- Добавление строки в company_info, которая отражает новую запись в company;
    --
    INSERT INTO company_info(company_id, restore_date, market_cap,
net_profit_margin_pct_annual)
    SELECT NEW.id, now(), NEW.market_cap, NEW.net_profit_margin_pct_annual;
    RETURN NEW;
END ;
$company_info$ LANGUAGE plpgsql;

drop TRIGGER IF EXISTS company_info_T on companies;

CREATE TRIGGER company_info_T
AFTER INSERT OR UPDATE
on companies
FOR EACH ROW
EXECUTE PROCEDURE process_company_info();
```

2. В таблицу патентов обязательно необходимо внести дату выдачи патента, а если не внеслась - дефолтно будем ставить текущую дату и оповестим об этом.

```
CREATE OR REPLACE FUNCTION patent_date() RETURNS trigger AS
$patent_date$
BEGIN
    -- Проверить, что указана дата
    IF NEW.start_date IS NULL THEN
        NEW.start_date := now();
        RAISE NOTICE 'Patent's date is set as %', NEW.start_date;
    END IF;
    RETURN NEW;
END;
$patent_date$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS patent_date_T ON patents;

CREATE TRIGGER patent_date_T
BEFORE INSERT OR UPDATE
ON patents
FOR EACH ROW
EXECUTE PROCEDURE patent_date();
```

3. Последние два триггера - таблица доступности лекарств в конкретных аптеках, при добавлении налл-значения - будем менять на ноль, а при достижении нуля (когда продали последнее лекарство) - будем извещать, что все распродано.

```
CREATE OR REPLACE FUNCTION stock_availability() RETURNS trigger AS
$stock_availability$
BEGIN
    IF NEW.availability IS NULL THEN
        NEW.availability := 0;
        RAISE NOTICE 'Supposed, that this trademark is empty!';
    END IF;
    RETURN NEW;
END;
$stock_availability$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS stock_availability_T on stock;

CREATE TRIGGER stock_availability_T
    BEFORE INSERT OR UPDATE
    ON stock
    FOR EACH ROW
EXECUTE PROCEDURE stock_availability();

CREATE OR REPLACE FUNCTION stock_msg_last() RETURNS trigger AS
$stock_msg_last$
BEGIN
    IF NEW.availability = 0 THEN
        RAISE NOTICE 'Bought the last pack of treatment!';
    END IF;
    RETURN NEW;
END;
$stock_msg_last$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS stock_msg_last_T on stock;

CREATE TRIGGER stock_msg_last_T
    AFTER UPDATE
    ON stock
    FOR EACH ROW
EXECUTE PROCEDURE stock_msg_last();
```

Методы и процедуры:

Код можно увидеть тут:

https://github.com/TohaRhymes/databases_pharmacy_coursework/blob/main/functions_without_triggers.sql

Изменение экономической информации о компании:

Процедура случайно меняет экономические показатели по запросу, а периодически она будет с сервера вызываться как бы имитируя, что периодически запрос будет идти на сервера.

```
-- Function for changing current economic indicators of the company
-- BUT FIRST returns random in range: 0.95-1.15
DROP FUNCTION IF EXISTS random_changing();

CREATE OR REPLACE FUNCTION random_changing()
    RETURNS DECIMAL AS
$$
BEGIN
    RETURN (SELECT 1 + random() * 0.2 - 0.05 AS random_changing);
END;
$$ LANGUAGE plpgsql VOLATILE;

-- USAGE EXAMPLE;
SELECT *
FROM random_changing();
```



```

--AND now function for changing
DROP FUNCTION IF EXISTS change_economical(_company_id int);

CREATE OR REPLACE FUNCTION change_economical(_company_id int)
RETURNS TABLE
(
    idd INTEGER,
    n VARCHAR(80),
    s VARCHAR(80),
    m_c DECIMAL,
    np DECIMAL
)
-- RETURNS void
AS
$$
DECLARE
    a DECIMAL := (SELECT SUM(random_changing)
FROM random_changing());
    b DECIMAL := (SELECT SUM(random_changing)
FROM random_changing());
    c DECIMAL := (SELECT SUM(random_changing)
FROM random_changing());
    d DECIMAL := (SELECT SUM(random_changing)
FROM random_changing());
BEGIN
    UPDATE companies cc
    SET (market_cap, net_profit_margin_pct_annual) = (mc * a, npmpa * c)
    FROM (SELECT id idd, market_cap mc, net_profit_margin_pct_annual npmpa
FROM companies c
WHERE (c.id = _company_id)
) AS prev_val
WHERE cc.id = _company_id;

    RETURN QUERY (SELECT comp.id idd,
comp.name n,
comp.specialization s,
comp.market_cap m_c,
comp.net_profit_margin_pct_annual np
FROM companies comp
WHERE comp.id = _company_id);
END;
$$ LANGUAGE plpgsql VOLATILE;

-- USAGE
SELECT *
FROM change_economical(4);

```

Закуп и продажа лекарств:

Функция, которая проверяет, есть ли купленное лекарство в таблице stock, если есть - прибавляет к существующему значению купленное число единиц, если нет - создает новую запись. Возвращает суммарное число купленного аптекой лекарства (что было + то, что завезли).

Также, если указать отрицательное число - это будет считаться продажей лекарства (причем, если покупателю хочется купить больше, чем возможно, то покупается только возможное количество и выведется сообщение).

```

-- Function for buying trademarks (new, or existing ones)

DROP FUNCTION IF EXISTS add_to_stock(_pharmacy_id int, _trademark_id int, _availability
int);

CREATE OR REPLACE FUNCTION add_to_stock(_pharmacy_id int,
                                         _trademark_id int,
                                         _availability int)
RETURNS int AS
$$
BEGIN

```

```

    IF EXISTS(SELECT 1 FROM stock s WHERE (s.pharmacy_id, s.trademark_id) =
(_pharmacy_id, _trademark_id)) THEN
        UPDATE stock
        SET availability = prev_val.a + _availability
        FROM (SELECT id, availability a
              FROM stock s
              WHERE ((s.pharmacy_id, s.trademark_id) = (_pharmacy_id, _trademark_id))
              ) AS prev_val
        WHERE stock.id = prev_val.id;
    ELSE
        INSERT INTO stock(pharmacy_id, trademark_id, availability)
        VALUES (_pharmacy_id, _trademark_id, _availability);
    END If;
    RETURN (SELECT availability FROM stock s WHERE ((s.pharmacy_id, s.trademark_id) =
(_pharmacy_id, _trademark_id)));
END;
$$ LANGUAGE plpgsql VOLATILE;

-- USAGE:
SELECT add_to_stock(2, 1, 118);

```

Создание компанией своей трейдмарки:

Функция, которая принимает id компании, id драгс, имя нового лекарства, doze, release_price, способ дистрибуции, создает для него патент и саму трейдмарку.

```

-- A function takes the companies_id, drugs_id, the name of the new drug,
-- doze, release_price, the distribution method,
-- creates a patent for it and the trademark itself.
-- CREATE TYPE patent_distribution AS ENUM ('free-to-use', 'usage with some
constraints', 'restricted-to-use');

DROP FUNCTION IF EXISTS add_trademark(_company_id int, _drug_id int, _name VARCHAR(80),
_doze VARCHAR(80), _release_price VARCHAR(80), _distribution VARCHAR(80));

CREATE OR REPLACE FUNCTION add_trademark(_company_id int,
                                         _drug_id int,
                                         _name VARCHAR(80),
                                         _doze DECIMAL,
                                         _release_price DECIMAL,
                                         _distribution VARCHAR(80))

    RETURNS int AS
$$
DECLARE
    _patent_id INTEGER := (SELECT MAX(id) + 1
                           FROM patents);
BEGIN
    INSERT INTO patents(id, distribution) VALUES (_patent_id,
_distribution::patent_distribution);
--
    INSERT INTO trademarks(name, doze, release_price, drug_id, company_id, patent_id)
    VALUES (_name, _doze, _release_price, _drug_id, _company_id, _patent_id);
--
    RETURN _patent_id;
END;
$$ LANGUAGE plpgsql VOLATILE;

-- USAGE EXAMPLES
SELECT * FROM add_trademark(2,
3,
'Anamorgggen',
0.5,
500.6,
'usage with some constraints');

```

Анализ использования созданной БД и создание индексов:

Большое количество сценариев использования БД - с использованием объединения таблиц (получить по данному лекарству список, чего он лечит, для данной болезни/яда - список лекарств, которыми он

лечится, по аптеке показывать доступные таблетки и для конкретного лекарства - где оно доступно). Все это будет соединяться через primary и foreign_keys с проверкой на равенство. Вводить тут индексы не имеет смысла, так как postgresql автоматически создает индексы для ключей и уникальных значений.

Скорее всего будет фильтрация и сортировка по цене трейдмарки, т.е. запросы такого рода:

```
SELECT name, doze, release_price
FROM trademarks
ORDER BY release_price;
```

Для того, чтобы выполнять более быстро, добавлю BTREE:

```
create index trademark_price on trademarks using btree(release_price);
```

Проверим:

ДО:

QUERY PLAN	
1	Sort (cost=230.21..237.68 rows=2986 width=25) (actual time=8.135..9.543 rows=2986 loops=1)
2	Sort Key: release_price
3	Sort Method: quicksort Memory: 330kB
4	-> Seq Scan on trademarks (cost=0.00..57.86 rows=2986 width=25) (actual time=0.024..2.905 rows=2986 loops=1)
5	Planning time: 0.152 ms
6	Execution time: 10.808 ms

После:

QUERY PLAN	
1	Index Scan using trademark_price on trademarks (cost=0.28..201.03 rows=2986 width=25) (actual time=0.017..2.111 rows=2986 loops=1)
2	Planning time: 0.088 ms
3	Execution time: 2.853 ms

Ученым и простым людям будут интересны запросы в духе “самая убийственная болезнь”:

Через какое-то время в БД (и мире) появится огромное количество новых болезней и ядов (особенно сейчас, потому что мы очень быстро придумываем новые лекарства, а бактерии и вирусы, мутируя, быстро ищут новые способы их обхода и блокировки, обретая новые резистентности (дошло до того, что существует мультирезистентность - конкретный патоген сложно вылечить одним способом) => организму сложнее детектировать и убивать его => он действует на организм сильнее => **важно определять приоритетные направления разработки => необходимо отсортировать болезни по mortality**).

```
SELECT name
FROM diseases
ORDER BY mortality DESC;
```

Для того, чтобы выполнять более быстро, добавлю BTREE - причем нам нужно будет идти от максимума к минимуму, поэтому DESC:

```
create index d_mort on diseases using btree (mortality DESC);
```

Проверим:

ДО:

QUERY PLAN
1 Sort (cost=5569.38..5712.70 rows=57330 width=25) (actual time=106.944..136.120 rows=57330 loops=1)
2 Sort Key: mortality DESC
3 Sort Method: external merge Disk: 2032kB
4 -> Seq Scan on diseases (cost=0.00..1038.30 rows=57330 width=25) (actual time=0.013..24.629 rows=57330 loops=1)
5 Planning time: 0.052 ms
6 Execution time: 157.682 ms

После:

QUERY PLAN
1 Index Scan using d_mort on diseases (cost=0.41..3616.30 rows=57330 width=25) (actual time=0.016..42.353 rows=57330 loops=1)
2 Planning time: 0.062 ms
3 Execution time: 56.975 ms

Аналогично для ядов:

```
create index p_mort on poisons using btree (mortality DESC);
```

Что меняется?

В обоих случаях кардинально меняется план запроса: если сначала каждый раз все сортировалось по ключу (можно увидеть), то теперь, в запросе просто используется индекс, и никакая сортировка не требуется.

Еще деревья?

Последний кейс - когда-нибудь продвинутый пользователь захочет понять: “Какая компания самая крутая?” - то есть отсортировать компании по их экономическим показателям: `market_cap` и `net_profit_margin_pct_annual`.

```
SELECT *
```

```
FROM companies
```

```
ORDER BY market_cap DESC;
```

Или:

```
SELECT *
```

```
FROM companies
```

```
ORDER BY market_cap DESC, net_profit_margin_pct_annual DESC;
```

Очевидно, что для этого будут тоже необходимы двоичные деревья - тут тоже нужно будет идти от максимума к минимуму, поэтому DESC

```
create index c_mc on companies using btree (market_cap DESC);
```

```
create index c_nmpma on companies using btree (net_profit_margin_pct_annual DESC);
```

```
create index c_nmpma on companies using btree (market_cap DESC, net_profit_margin_pct_annual DESC);
```

Однако в данный момент такая оптимизация показалась неэффективной (не была использована) - оптимизатор посчитал, что при относительно небольшом количестве компаний, использовать индексы не надо :(. Возможно, через много лет - когда число компаний вырастет, можно будет думать над такой оптимизацией. Пока - нет.

ДО:

QUERY PLAN	
3	Sort Method: quicksort Memory: 89kB
4	-> Seq Scan on companies (cost=0.00..11.64 rows=564 width=47) (actual time=0.014..0.441 rows=564 loops=1)
5	Planning time: 0.131 ms
6	Execution time: 1.998 ms

После:

QUERY PLAN	
1	Sort (cost=37.41..38.82 rows=564 width=47) (actual time=1.195..1.658 rows=564 loops=1)
2	Sort Key: market_cap DESC
3	Sort Method: quicksort Memory: 89kB
4	-> Seq Scan on companies (cost=0.00..11.64 rows=564 width=47) (actual time=0.012..0.446 rows=564 loops=1)
5	Planning time: 0.346 ms
6	Execution time: 2.067 ms

Кроме всего прочего, будут часто искать таблетки, болезни по именам - добавлю на имена тоже хеши:

Это нужно, например, когда будет необходимо пробить информацию о конкретной болезни, или в аптеке - о конкретном лекарстве.

```
create index company_name on companies using hash(name);
create index trademark_name on trademarks using hash(name);
create index drug_substance on drugs using hash(active_substance);
create index disease_name on diseases using hash(name);
create index poison_substance on poisons using hash(active_substance);
create index ethnoscience_name on ethnoscience using hash(name);
create index pharmacy_name on pharmacies using hash(name);
```

Например,

```
SELECT *
FROM diseases
WHERE name = 'Newton''s kishg';
```

Для того, чтобы выполнять более быстро, добавлю HASH:

```
create index d_name on diseases using hash (name);
```

Проверим:

ДО:

QUERY PLAN	
1	Seq Scan on diseases (cost=0.00..1181.62 rows=1 width=33) (actual time=10.042..19.471 rows=1 loops=1)
2	Filter: ((name)::text = 'Newton''s kishg'::text)
3	Rows Removed by Filter: 57329
4	Planning time: 0.145 ms
5	Execution time: 19.502 ms

После:

■ ■ ■ QUERY PLAN	
1	Index Scan using d_name on diseases (cost=0.00..8.02 rows=1 width=33) (actual time=0.013..0.014 rows=1 loops=1)
2	Index Cond: ((name)::text = 'Newton''s kishg'::text)
3	Planning time: 0.135 ms
4	Execution time: 0.029 ms

Что меняется?

Скорость увеличилась в несколько сотен раз!

Оно и логично, вместо того, чтобы пробегать по всей таблице, мы просто переходим по нужному кэшу - удобно!