Университет ИТМО

Факультет программной инженерии и компьютерной техники

**Курсовая работа**

по «Информационным системам и базам данных»

**Часть №3**

Выполнили:

Студенты группы P33101

Стенина Арина

Максимова Мария

Преподаватель:

Гаврилов А.В.

Санкт-Петербург

2021

**Создание и заполнение базы данных**

В ходе выполнения были созданы следующие скрипты:

* create\_table.sql - создает базу данных
* triggers\_functions.sql - добавляет в нее триггеры и функции
* insert\_values.sql - заполняет данными
* delete\_data.sql - удаляет данные
* drop\_tables.sql - удаляет таблицы базы данных
* indexes.sql – создает индексы

**Функции**

1. **insert\_client** (добавить абсолютно нового клиента с автоматическим заполнением таблицы человек)

Синтаксис: SELECT insert\_client(name, surname, age, gender, profession, police, cash);

Пример: SELECT insert\_client('Mary', 'Rose', 40, true, NULL, false, 0);

1. **insert\_case** (добавить дело без указания даты начала и окончания)

Синтаксис: SELECT insert\_case(client\_id, address\_id, name);

Пример: SELECT insert\_case(11, NULL, 'Pink Karbunkul');

1. **end\_case** (закончить дело: проверка, что дело начато, найден хотя бы один преступник, и у него есть наказание)

Синтаксис: SELECT end\_case(case\_id);

Пример: SELECT end\_case(2);

1. **set\_suspect\_criminal** (установка поля is\_crminal являлся ли подозреваемый преступником раньше)

Синтаксис: SELECT set\_suspect\_criminal(suspect\_id);

Пример: SELECT set\_suspect\_criminal(1);

1. **insert\_suspect** (добавление подозреваемого c установкой поля is\_criminal на основе таблицы преступников)

Синтаксис: SELECT insert\_suspect(appearance, human\_id);

Пример: SELECT insert\_suspect(‘tall’, 2);

**Код:**

CREATE OR REPLACE FUNCTION insert\_suspect(appearance text,

human\_id int) RETURNS integer AS

$$

DECLARE

suspect integer;

BEGIN

INSERT INTO suspect (appearance, human\_id)

VALUES (insert\_suspect.appearance, insert\_suspect.human\_id) RETURNING suspect\_id INTO suspect;

PERFORM set\_suspect\_criminal(suspect);

RETURN suspect;

END;

$$ LANGUAGE plpgSQL;

**Функции проверки комплексных ограничений целостности данных**

* **check\_performer()** - перед добавлением исполнителя проверяет, есть ли он в базе преступников
* **check\_case\_performer()** - перед добавлением исполнителя к делу проверяет, что на дело может быть назначено не более двух исполнителей
* **check\_case\_start()** - перед добавлением улики или преступника проверяет, что дело начато, но еще не закончено
* **check\_case\_end()** - перед добавлением новостного ресурса к делу проверяет, что дело уже завершено
* **start\_case()** - срабатывает после добавления или изменения case, проверяет, если ли клиент, адрес и хотя бы один исполнитель. Если да, автоматически начинает дело ( дата = текущей)
* **start\_case\_with\_perormer()** - срабатывает после добавления или изменения performer, проверяет, если ли клиент, адрес и хотя бы один исполнитель по этому делу. Если да, автоматически начинает дело ( дата = текущей)
* **set\_money()** – срабатывает после добавление или изменения в таблице case\_source, меняет атрибут money в таблице client, связанной с делом, на новый (считается по формуле с учетом среднего рейтинга всех изданий).

**Код:**

CREATE OR REPLACE FUNCTION set\_money() RETURNS trigger AS

$$

DECLARE

cur\_source source%rowtype;

sum int;

counter int;

BEGIN

sum := 0;

counter := 0;

FOR cur\_source IN (SELECT \* FROM source WHERE source.case\_id = NEW.case\_id)

LOOP

IF (cur\_criminal.source\_rating IS NOT NULL) THEN

sum := sum + cur\_criminal.source\_rating;

END IF;

END LOOP;

UPDATE client

SET money = 1000 \* sum / counter WHERE (client.client\_id = (SELECT client\_id FROM "case" WHERE "case".case\_id = NEW.case\_id));

END;

$$ LANGUAGE plpgSQL;

**Триггеры**

* **checkPerformer** - срабатывает при добавлении исполнителя и проверяет, есть ли он в базе преступников
* **checkCasePerformer** - срабатывает при добавлении в таблицу дело-исполнитель и проверяет количество исполнителей для одного дела
* **checkCriminalAdd** - срабатывает при добавлении преступника и проверяет, что дело является открытым и незавершенным
* **checkEvidenceAdd** - срабатывает при добавлении улики и проверяет, что дело является открытым и незавершенным
* **checkSourceAdd** - срабатывает при добавлении новостного источника и проверяет, является ли дело завершенным
* **startCase** – срабатывает после добавления или изменения case, проверяет, если ли клиент, адрес и хотя бы один исполнитель. Если да, автоматически начинает дело ( дата = текущей)
* **startCaseWithPerformer** - срабатывает после добавления или изменения performer, проверяет, если ли клиент, адрес и хотя бы один исполнитель по этому делу. Если да, автоматически начинает дело ( дата = текущей)
* **setMoney** - срабатывает после добавление или изменения в таблице case\_source, меняет атрибут money в таблице client, связанной с делом, на новый (считается по формуле с учетом среднего рейтинга всех изданий). То есть в зависимости от среднего рейтинга новостных источников, где упоминается дело, меняется выплата денег исполнителю.

**Код:**

CREATE TRIGGER setMoney

AFTER INSERT OR UPDATE

ON case\_source

FOR EACH ROW

EXECUTE PROCEDURE set\_money();

**Индексы**

Составим индексы, на основе бизнес-процессов, которые представляют наиболее частые запросы к базе данных.

* **possible\_suspect\_index** - btree индекс на атрибуты age и gender таблицы human (запрос на поиск потенциальных подозреваемых по внешним приметам), используем именно btree, так как в запросе такого типа используются операции сравнения >,<,=.

**Код**:

CREATE INDEX "possible\_suspect\_index" ON human USING btree ("age", "gender");

|  |  |
| --- | --- |
| До создания индекса | После создания индекса |
| Изображение выглядит как текст  Автоматически созданное описание | Изображение выглядит как текст  Автоматически созданное описание |

Время выполнения *уменьшилось* *почти в два раза*

* **suspect\_index** – hash индекс на атрибут human\_id в таблице suspect (запросы на то, был ли человек подозреваемым), используем именно hash, так как в запросе такого типа используется операция на проверку равенства.

**Код:**

CREATE INDEX "suspect\_index" ON suspect USING hash ("human\_id");

|  |  |
| --- | --- |
| До создания индекса | После создания индекса |
| Изображение выглядит как текст  Автоматически созданное описание | Изображение выглядит как текст  Автоматически созданное описание |

Время *уменьшилось* *незначительно*

* **search\_address\_index** – btree индекс на атрибут city и street в таблице address (актуально для запросов поиска адресов в округе, по которым заведены дела). Используется btree, потому что для такого запроса надо индексировать два атрибута, hash не позволяет так делать.

|  |  |
| --- | --- |
| До создания индекса | После создания индекса |
| Изображение выглядит как текст  Автоматически созданное описание | Изображение выглядит как текст  Автоматически созданное описание |

Время выполнения запроса *немного уменьшилось*.

* **search\_date\_case** – btree индекс на атрибут start\_date в таблице case (актуально для запросов, которые направлены на поиск дел, начатых в недавнее время). используем именно btree, так как в запросе такого типа используются операции сравнения >,<,=.

|  |  |
| --- | --- |
| До создания индекса | После создания индекса |
| Изображение выглядит как текст  Автоматически созданное описание | Изображение выглядит как текст  Автоматически созданное описание |

Время выполнения запроса *почти не изменилось*.

* **criminal\_search\_index** – hash индекс на атрибут human\_id в таблице criminal (запросы на поиск человека в базе преступников). Используем именно hash, так как в запросе такого типа используется операция на проверку равенства.

|  |  |
| --- | --- |
| До создания индекса | После создания индекса |
| Изображение выглядит как текст  Автоматически созданное описание | Изображение выглядит как текст  Автоматически созданное описание |

Время запроса *уменьшилось незначительно*.

Время выполнения запросов сокращается при использовании индексов типа hash или btree, так как запросы содержат условия, в которых используются операторы сравнения.

* 1. Для btree вместо перебора всех строк таблиц, что составляет алгоритмическую сложность поиска O(n), производится поиск по листьям сбалансированного дерева со сложностью O(log(n)).
  2. Для hash вместо перебора всех строк таблиц что составляет алгоритмическую сложность поиска O(n), производится переход по нужному хешу со сложностью O(1).

Таким образом, время, необходимое на поиск с помощью индексов, при росте объема базы данных будет расти существенно медленнее, чем при использовании полного перебора.

**Вывод**

В ходе выполнения 3 этапа были созданы необходимые объекты базы данных, скрипты для заполнения ее данными, реализованы функции и триггеры, а также созданы индексы.