**Министерство образования и науки Российской Федерации**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**



Факультет Программной Инженерии и Компьютерной техники

**Отчет**

По дисциплине

**«Информационные системы и базы данных»**

*Курсовая работа на основе произведения Ф. Кафка «Замок»*

Выполнили:

Студенты группы P33111

Мишанин Н.А  
Козлов В.Н

Преподаватель:

Николаев В.В

*Санкт – Петербург, 2021 г.*

# **Описание предметной области** Франц Кафка - Замок

Главному герою произведения необходимо автоматизировать процесс попадания в Замок. Его путь тернист и содержит большое количество комитетов, домов жителей, дорог и встреч. Главная цель пройти весь маршрут собирая все необходимые документы и попасть в замок к бургомистру, где главному герою предстоит получить последний документ. Благодаря этому герой сможет вырваться из порочного круга бюрократии и уехать из злополучной деревни и замка. Процесс достаточно сложный и запутанный, нам нужно избавляться от большого количества очередей и пытаться сделать все быстро.  
  
Герой романа К. приезжает в Деревню, управляемую Замком, К. работает землемером. Ему нужно получить необходимые документы, чтобы выполнить работу и уехать обратно домой. Чтобы получить необходимые документы ему надо попасть в Замок. Для получения пропуска в замок нужно пройти несколько инстанций управления, которое подчиняется Замком. В этом управлении существует множество людей разных должностей, которые сидят в различных кабинетах и подчиняются Наместнику из Замка. Таким образом нужно пройти путь по всему маршруту, который ему должен объяснить Деревенский староста. Для того чтобы попасть к старосте, нужно выяснить, где староста живет и получить документ, говорящий о том, что староста готов его принять. Это документ состоит из подписей определенных людей в деревне, которые подчиняются Старосте. Но как только К. приехал в Деревню, он понимает, что у него нет распоряжения из комитета деревни по приезде.

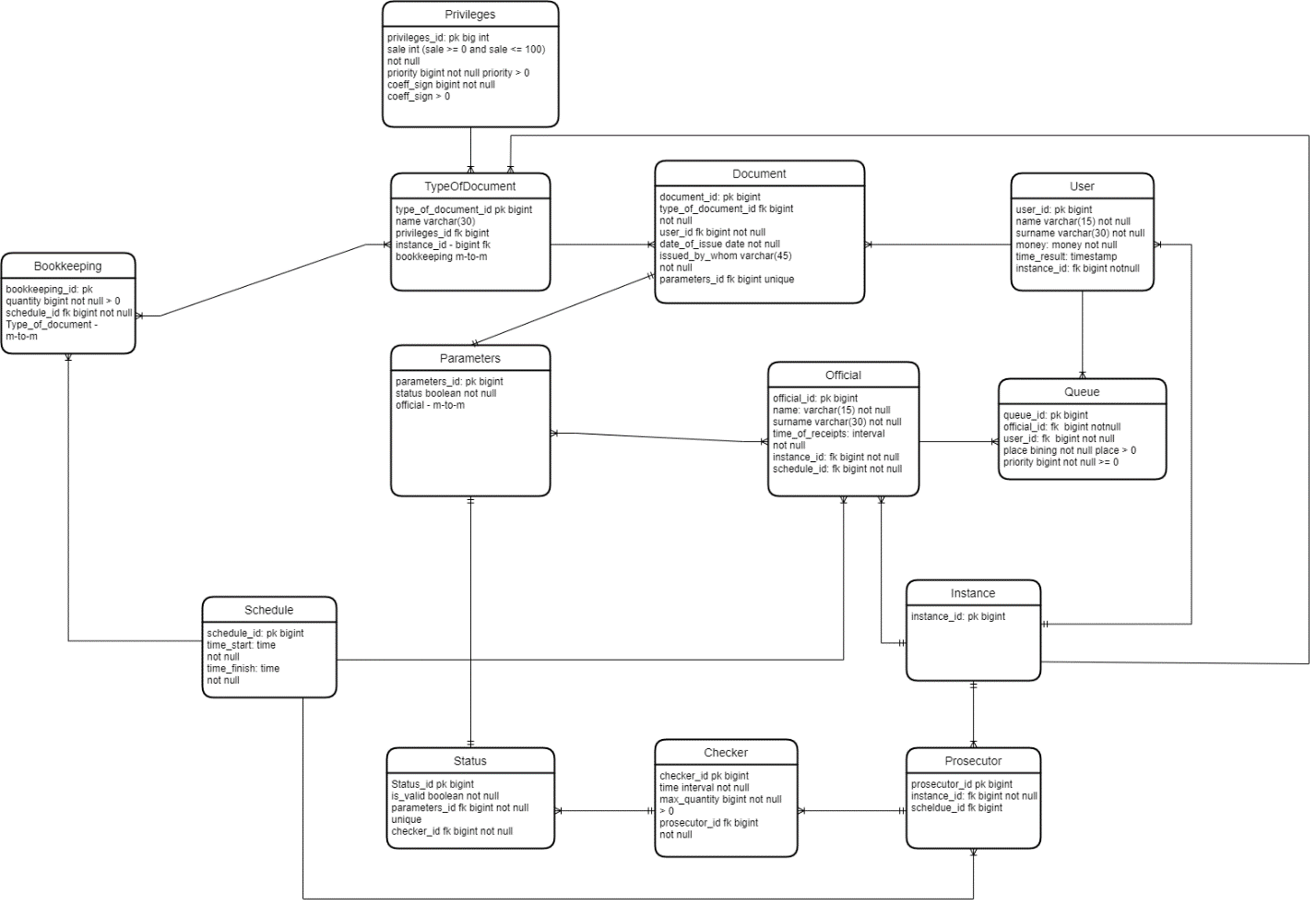
Таким образом нужно пройти процесс регистрации и предъявить благие намерения для работы землемером на территории, которая подчиняется Замком. Чтобы найти всех людей и получить подписи для допуска к старостам, нужно походить по деревне и получить все необходимые сведения, чтобы найти людей. Процесс очень затягивается, и помимо героя К. есть и другие работники из дальних мест. Нужно всех обслужить как можно быстрее. При этом существуют дополнительные опции сокращать маршрут при необходимости - льготы, оказание дополнительных услуг, законная покупка справок. За всем этим активно следить комитет Замка - прокуратура. Но если находятся несоответствия, то придется вернуться и все исправлять, в зависимости от нарушения, что затруднит прохождение к замку.

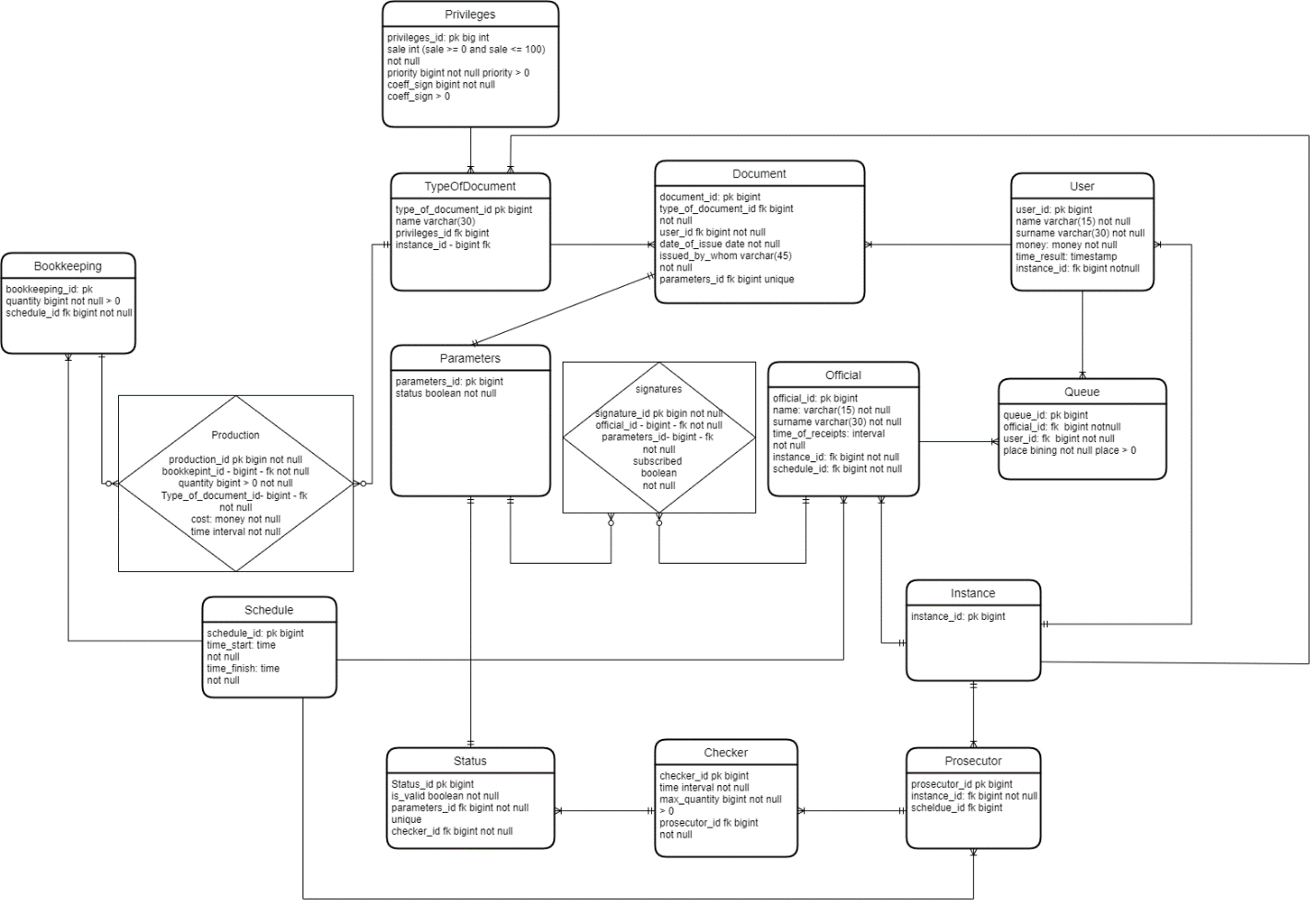
Бизнес-процессы

1. Пользователь попадает в город с изначальным для себя набором документов, а также денежным эквивалентом, т. е. суммой денег.
2. Виды документов:
   1. Справки с места прошлой работы, как доказательство, что человек приехал работать
   2. Льготные справки
   3. Подтверждающие личность:
      * 1. Паспорт
        2. Свидетельство о рождении
        3. Права
3. У пользователя есть цель на получение справки, а также дата, до которой он должен эту справку получить.
4. В зависимости от начального набора документов, пользователю предоставляется к какой дате пользователь получит свою справку.
5. Выпуском справок занимается бухгалтерия.
6. У каждой справки есть параметры:
   1. Время выпуска
   2. Тираж
   3. Стоимость покупки справки, можно не дожидаясь времени выпуска, получить справку.
   4. Графа подписей, которые нужны, чтобы справка стала действующей
7. Так же бухгалтерия поставляет справки на уровни инстанций и физическим лицам.
8. Чтобы получить подписи, нужно найти нужных людей.
9. У каждого нужно человека есть:
   1. График работы
   2. Время приема одного человека
   3. Очередь людей, которые хотят получить у него справку.
10. В зависимости от потребностей пользователя, ему предоставляются следующие пути решения:  
    a. Если срок приемлемый, то выполняются операции на уровне инстанции.

b. Если срок не приемлемый, то пользователю предоставляются опции, ускоряющие процесс.

1. Виды опций:  
   a. Собрать справки за денежный эквивалент.  
   b. Предоставить льготу. Виды льгот:  
    1. Инвалидность  
    2. Ветеран боевых действий  
    3. Почетный гражданин страны  
   с. Найти человека, чья подпись может заменить n других.
2. У каждой опции есть сложность, которая измеряется во времени и денежном эквиваленте.
3. Льготы влияют на:
   * + 1. Получение скидки на покупку справок
       2. Приоритет в очереди
       3. Сокращение количество требуемых подписей на справке
4. За некоторыми процессом наблюдает определенная инстанция, называемая прокуратура.
5. При обнаружении несоответствия прокуратура может отправить пользователя на исправление несоответствий.
6. Прокуратура занимается проверкой документов, чтобы пользователь мог перейти на новый уровень  
   Виды несоответствий:
   1. Проверка подлинности документов.
   2. Проверка соответствия всех подписей.
   3. Проверка действующих дат.
   4. Законность льгот
7. При переходе на новый уровень, человек переносится на новую инстанцию в зависимости от его положения в системе.
8. При достижении желаемого результата пользователь отправляется на следующий уровень.
9. Когда пользователь получил нужную справку, пользователь считается обработанным.

**Инфологическая и даталогическая модели**  
1. ER-диаграмма предметной области  


2. Даталогическая модель  


3. Ограничения  
USER: Time\_result не может быть раньше, чем нынешняя\* дата

USER: не может попасть в базу без документа, подтверждающего личность (паспорт и т.д.)

USER: не может быть несколько льгот, документов, подтверждающих личность одного типа (несколько ветеранств, несколько свидетельств о рождении и т.д.)

Parameter: не может быть куплена, если её количество ноль

SCHEDULE: рабочих часов в расписании не может превышать 8 в день

OFFICIAL, Prosecutor, Bookkeping: не может принимать посетителей после 22:00 и не может начинать принимать посетителей раньше 10:00

CHECKER: Количество проверяемых справок и документов не может превышать количество работников в прокуратуре

BOOKEPING: Количество производимых бумаг не может превышать лимита

Parameters: присущ только справкам

**Описание бизнес-процессов, создание функций, генераторов, добавление индексов и анализ запросов**Бизнес-процессы (основная последовательность с указанием сущностей):

1. Инициализация данных, пользователь попадает в город с изначальным для себя набором документов, а также денежным эквивалентом, т. е. суммой денег, также по умолчанию все пользователи оказываются на первой инстанции. У самого пользователя есть атрибут, указывающий дату приемлемого получения справки.  
   а) Самого пользователя отражает таблица User  
   б) Набор документов пользователя отражает таблица Document  
   с) Список всех инстанций, на которые может попадать пользователь отражен в таблице Instance.
2. Пользователь выбирает конечную цель (документ) после получения которого он может закончить процесс. Документ также отражен в таблице Document. Расширяет документ таблица TypeOfDocument, в которой отражены доступные и необходимые для определённых инстанций документы. Помимо этого, для документов существуют законные привилегии, которые могут ускорить процесс, привилегии отражены в таблице Privileges, но базовый набор документов каждого пользователя не дает никаких привилегий.
3. Выпуском справок занимается бухгалтерия, которая, как и все инстанции имеет расписание времени работы что отражает таблица Schedule, а саму бухгалтерию и количество выпускаемых справок отражает таблица Bookkeeping. Артикулы справок, их цены, количество для печати и время изготовления доступны в таблице Production  
   а) Процесс покупки справки: пользователь смотрит на ту инстанцию, для которой нужны справки для покупки  
   б) Далее выбирается тип справок, который присущ этой инстанции, далее достаются данные из таблицы Production и с учетом всех параметров происходит процесс изготовления и покупки справки
4. Каждая справка нуждается в определенном количестве подписей, которые позволят применить её как действительный документ.  
   Процесс подписей осуществляется следующим образом:  
   а) Пользователю необходимо записаться в очереди к каждому работнику, который может подписать конкретную справку или справки. Работник инстанций отражен в таблице Official  
   б) Запись в очередь отражает таблица Queue  
   в) Каждый документ из таблицы Document связан с конкретными подписями через таблицу Parameters, запись в которой в конечном счете может определить все ли подписи собраны на конкретном документе.
5. После подписания справки она нуждается в проверке специальной прокуратурой и её отражает таблица Prosecutor. Процесс проверки происходит так:  
   а) Каждая прокуратура связана с определенной инстанцией и для каждой инстанции существует система проверки, которая отражена в таблице Checker, также разные статусы подтверждения для конкретного параметра отражает таблица Status  
   б) В зависимости от времени и загруженности проверочной системы будут подтверждаться конкретные документы.
6. Когда все справки для конкретной инстанции подписаны, проверены и одобрены, мы можем перевести пользователя на следующий этап в зависимости от конкретной цели, либо закончить его пребывание в системе и сделать пользователя обработанным.

Код:  
DDL

create table "Privileges"

(

privileges\_id bigserial primary key,

sale integer not null check (sale >= 0 and sale <= 100),

priority bigint not null check ( priority >= 0 ),

coeff\_sign bigint not null check ( coeff\_sign >= 0 )

);

create table "Parameters"

(

parameters\_id bigserial primary key ,

status boolean not null

);

create table "Instance"

(

instance\_id bigserial primary key,

name text not null

);

create table "TypeOfDocument"

(

type\_of\_document\_id bigserial primary key,

name varchar(30) not null ,

privileges\_id bigint,

instance\_id bigint not null,

foreign key (instance\_id) references "Instance" (instance\_id) on delete restrict ,

foreign key (privileges\_id) references "Privileges" (privileges\_id)

);

create table "User"

(

user\_id bigserial primary key,

name varchar(15) not null,

surname varchar(30) not null,

money money not null,

time\_result timestamp not null,

instance\_id bigint not null,

foreign key (instance\_id) references "Instance" (instance\_id)

);

create table "Document"

(

document\_id bigserial primary key,

type\_of\_document\_id bigint not null,

user\_id bigint not null,

date\_of\_issue date not null,

validity date,

issued\_by\_whom varchar(45) not null,

parameters\_id bigint unique,

foreign key (user\_id) references "User" (user\_id) on delete restrict ,

foreign key (type\_of\_document\_id) references "TypeOfDocument" (type\_of\_document\_id),

foreign key (parameters\_id) references "Parameters" (parameters\_id) on delete cascade,

check ( validity is null or validity >= date\_of\_issue)

);

create table "Schedule"

(

schedule\_id bigserial primary key,

time\_start time not null check (time\_start >= '10:00:00'::time),

time\_finish time not null check (time\_finish <= '22:00:00'::time),

check ( time\_start < time\_finish )

);

create table "Prosecutor"

(

prosecutor\_id bigserial primary key ,

instance\_id bigint not null,

schedule\_id bigint,

foreign key (instance\_id) references "Instance" (instance\_id) on delete restrict ,

foreign key (schedule\_id) references "Schedule" (schedule\_id) on delete set null

);

create table "Checker"

(

checker\_id bigserial primary key,

time interval not null,

max\_quantity bigint not null check ( max\_quantity > 0 ),

prosecutor\_id bigint not null,

foreign key (prosecutor\_id) references "Prosecutor" (prosecutor\_id) on delete restrict

);

create table "Status"

(

status\_id bigserial primary key,

is\_valid boolean not null,

parameter\_id bigint not null unique,

checker\_id bigint not null,

foreign key (parameter\_id) references "Parameters" (parameters\_id) on delete cascade,

foreign key (checker\_id) references "Checker" (checker\_id)

);

create table "Bookkeeping"

(

bookkeeping\_id bigserial primary key,

quantity bigint not null check (quantity > 0),

schedule\_id bigint,

foreign key (schedule\_id) references "Schedule" (schedule\_id) on delete set null

);

create table "Production"

(

production\_id bigserial primary key,

bookkeeping\_id bigint not null,

quantity bigint not null check ( quantity >= 0 ),

type\_of\_document\_id bigint not null,

cost money not null ,

time interval not null,

foreign key (bookkeeping\_id) references "Bookkeeping" (bookkeeping\_id),

foreign key (type\_of\_document\_id) references "TypeOfDocument" (type\_of\_document\_id)

);

create table "Official"

(

official\_id bigserial primary key,

name varchar(15) not null,

surname varchar(30) not null,

time\_of\_receipts interval not null,

instance\_id bigint,

schedule\_id bigint,

foreign key (instance\_id) references "Instance" (instance\_id) on delete restrict ,

foreign key (schedule\_id) references "Schedule" (schedule\_id) on delete set null

);

create table "Queue"

(

queue\_id bigserial primary key,

official\_id bigint not null,

user\_id bigint not null,

place bigint not null check ( place > 0 ),

priority bigint not null,

foreign key (official\_id) references "Official" (official\_id),

foreign key (user\_id) references "User" (user\_id)

);

create table "Signatures"

(

signature\_id bigserial primary key,

official\_id bigint not null,

parameters\_id bigint not null,

is\_subscribed boolean not null,

foreign key (official\_id) references "Official" (official\_id),

foreign key (parameters\_id) references "Parameters" (parameters\_id) on delete cascade

);

# Functions and Triggers

CREATE or REPLACE function check\_time\_result() returns trigger as

$$

BEGIN

IF NEW.time\_result <= (SELECT NOW() + interval '3 hours') THEN

raise exception '% this time cannot be used as the result time', NEW.time\_result;

ELSE

return NEW;

END IF;

end;

$$ LANGUAGE plpgsql;

CREATE TRIGGER tr\_check\_time\_result

BEFORE INSERT

ON "User"

FOR EACH ROW

EXECUTE PROCEDURE check\_time\_result();

CREATE or REPLACE function check\_docs\_type() returns trigger as

$$

BEGIN

IF (Select count(\*) from "Document"

where user\_id = NEW.user\_id and type\_of\_document\_id = NEW.type\_of\_document\_id) > 0 THEN

raise exception '% such a document already exists', NEW.user\_id;

Else

return NEW;

end if;

end;

$$ LANGUAGE plpgsql;

CREATE TRIGGER tr\_check\_docs\_type

BEFORE INSERT

ON "Document"

FOR EACH ROW

EXECUTE PROCEDURE check\_docs\_type();

CREATE or REPLACE function check\_office\_hours() returns trigger as

$$

BEGIN

if NEW.time\_finish - interval '8 hours' > NEW.time\_start then

raise exception '% the number of working hours cannot exceed 8', NEW;

ELSE

return NEW;

end if;

end;

$$LANGUAGE plpgsql;

CREATE TRIGGER tr\_check\_office\_hours

BEFORE INSERT

ON "Schedule"

FOR EACH ROW

EXECUTE PROCEDURE check\_office\_hours();

CREATE or REPLACE function check\_parametrs() returns trigger as

$$

BEGIN

if NEW.type\_of\_document\_id > 3 and NEW.parameters\_id is null then

raise exception '% reference parameter cannot be null', NEW;

ELSE

if new.type\_of\_document\_id <= 3 and New.parameters\_id is not null then

raise exception '% document cannot have parameters', NEW;

Else

return NEW;

end if;

end if;

end;

$$LANGUAGE plpgsql;

CREATE TRIGGER tr\_check\_parametrs

BEFORE INSERT

ON "Document"

FOR EACH ROW

EXECUTE PROCEDURE check\_parametrs();

CREATE or REPLACE function check\_max\_status() returns trigger as

$$

BEGIN

IF (SELECT max\_quantity from "Checker" where checker\_id = NEW.checker\_id) < (SELECT count(\*) from "Status"

where checker\_id = NEW.checker\_id and is\_valid = false) + 1 then

raise exception '% search of the checked references', NEW.checker\_id;

ELSE

return NEW;

end if;

end;

$$LANGUAGE plpgsql;

CREATE TRIGGER tr\_check\_max\_status

BEFORE INSERT

ON "Status"

FOR EACH ROW

EXECUTE PROCEDURE check\_max\_status();

CREATE or replace function verification\_of\_signatures(bigint) returns void as

$$BEGIN

if((SELECT count(\*) from "Signatures" where parameters\_id = $1 and is\_subscribed = false) = 0) then

update "Parameters"

set status = true

where parameters\_id = $1;

Raise notice 'successfully updated';

else

raise exception '% not enough signatures', $1;

end if;

end;

$$LANGUAGE plpgsql;

CREATE or replace function add\_for\_review(bigint) returns void as

$$

declare

flag boolean[];

inst bigint;

pull\_checker bigint[];

lenght2 bigint;

ch\_id bigint;

col bigint;

status bigint;

begin

col = (SELECT count(\*) from "Parameters" where parameters\_id = $1)::bigint;

if(col = 0) then

raise exception 'there is no parameter %', $1;

else

flag = ARRAY(SELECT "Parameters".status from "Parameters" where parameters\_id = $1);

if(NOT flag[1]) then

raise exception '% update the status of signatures', $1;

else

status = (select count(\*) from "Status" join "Parameters" P on P.parameters\_id = "Status".parameter\_id where parameters\_id = $1)::bigint;

if(status = 0) then

inst = (select instance\_id from "Document"

join "TypeOfDocument" TOD on "Document".type\_of\_document\_id = TOD.type\_of\_document\_id where parameters\_id = $1)::bigint;

pull\_checker = ARRAY(select checker\_id from "Checker" join "Prosecutor" P on "Checker".prosecutor\_id = P.prosecutor\_id where instance\_id = inst);

lenght2 = (array\_length(pull\_checker, 1))::bigint;

ch\_id = 0;

for j in 1..lenght2

loop

if((select count(\*) from "Status" where is\_valid = false and checker\_id = pull\_checker[j]) < (select max\_quantity from "Checker" where checker\_id = pull\_checker[j])) then

ch\_id = pull\_checker[j];

end if;

end loop;

if(NOT(ch\_id = 0)) then

insert into "Status" (is\_valid, parameter\_id, checker\_id) VALUES (false, $1, ch\_id);

raise notice 'successfully added for verification';

else

raise exception 'there are no available employees at this number %', inst;

end if;

else

raise exception '% already under review', $1;

end if;

end if;

end if;

end;

$$LANGUAGE plpgsql;

create or replace function transfer\_to\_the\_next\_level(bigint) returns bool as

$$

DECLARE

pull\_parameters bigint;

pull\_status bigint;

begin

pull\_parameters = (select count(\*) from "Parameters"

join "Document" D on "Parameters".parameters\_id = D.parameters\_id

join "User" U on U.user\_id = D.user\_id

join "TypeOfDocument" TOD on D.type\_of\_document\_id = TOD.type\_of\_document\_id

where D.user\_id = $1 and TOD.instance\_id = (Select instance\_id from "User" where "User".user\_id = $1) and "Parameters".status = false);

if(pull\_parameters = 0) then

pull\_status = (select count(\*) from "Parameters"

join "Document" D on "Parameters".parameters\_id = D.parameters\_id

join "User" U on U.user\_id = D.user\_id

join "TypeOfDocument" TOD on D.type\_of\_document\_id = TOD.type\_of\_document\_id

join "Status" S on "Parameters".parameters\_id = S.parameter\_id

where D.user\_id = $1 and TOD.instance\_id = (Select instance\_id from "User" where "User".user\_id = $1) and S.is\_valid = false);

if(pull\_status = 0) then

raise notice 'you can transfer to the next instance';

return true;

else

raise notice 'not all references have been checked yet';

return false;

end if;

else

raise notice 'not all certificates are signed';

return false;

end if;

end;

$$LANGUAGE plpgsql;

create or replace function advance\_the\_queue(bigint) returns bool as

$$BEGIN

Delete from "Queue"

where place = 1;

UPDATE "Queue"

SET place = place - 1

where place > 0 and official\_id = $1;

RAISE NOTICE 'queue moved';

return true;

end;

$$ LANGUAGE plpgsql;

create or replace function calculate\_priority(bigint) returns bigint as

$$

declare

pr bigint;

begin

pr = (Select SUM(priv.priority) from "Document" docs

join "TypeOfDocument" typed on typed.type\_of\_document\_id = docs.type\_of\_document\_id

join "Privileges" priv on priv.privileges\_id = typed.privileges\_id

where docs.user\_id = $1)::bigint;

return pr;

end;

$$ LANGUAGE plpgsql;

create or replace function put\_in\_queue(bigint, bigint)returns bigint as

$$

DECLARE

prior bigint;

plac bigint;

u\_id bigint;

of\_id bigint;

BEGIN

u\_id = $1;

of\_id = $2;

prior = calculate\_priority(u\_id);

plac = (select min(place) from "Queue" where prior > priority and official\_id = of\_id);

if(plac is not null) then

UPDATE "Queue"

SET place = place + 1

where place >= plac and official\_id = of\_id;

Insert INTO "Queue" (official\_id, user\_id, place, priority) values(of\_id, u\_id, plac, prior);

RAISE NOTICE 'queue updated';

else

plac = (select max(place) from "Queue" where official\_id = of\_id);

if(plac is not null) then

plac = plac + 1;

else

plac = 1;

end if;

Insert INTO "Queue" (official\_id, user\_id, place, priority) values(of\_id, u\_id, plac, prior);

RAISE NOTICE 'queue updated';

end if;

return plac;

end;

$$LANGUAGE plpgsql;

# Generators

CREATE OR REPLACE FUNCTION generate\_privileges() RETURNS VOID AS

$$

declare

sale\_possible\_new int = 0;

priority\_new bigint = 0;

coefficient\_sign\_new bigint = 0;

begin

sale\_possible\_new = round(random() \* 100);

priority\_new = round(random() \* 1000);

coefficient\_sign\_new = round(random() \* 1000);

INSERT INTO "Privileges" (sale, priority, coeff\_sign)

VALUES (sale\_possible\_new, priority\_new, coefficient\_sign\_new);

end;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION generate\_privileges\_default() RETURNS VOID AS

$$

declare

sale\_possible\_new int = 0;

priority\_new bigint = 0;

coefficient\_sign\_new bigint = 0;

begin

INSERT INTO "Privileges" (sale, priority, coeff\_sign)

VALUES (sale\_possible\_new, priority\_new, coefficient\_sign\_new);

end;

$$ LANGUAGE plpgsql;

select generate\_privileges\_default(); -- work

select generate\_privileges() FROM generate\_series(1, 500); -- work

CREATE OR REPLACE FUNCTION generate\_instance() RETURNS VOID AS

$$

declare

brands\_of\_instance varchar[] = '{Староста, Отделение управления деревни, Психиатрическая лечебница, Полиция, Братство,' ||

' Регулятор, Поликлиника, Почта, Отделение управления замка, Юридическая контора, Церковь}';

index int = 0;

new\_name varchar;

begin

index = floor(random() \* array\_length(brands\_of\_instance, 1) + 1)::int;

new\_name = brands\_of\_instance[index];

insert into "Instance" (name) values (new\_name);

end;

$$ LANGUAGE plpgsql;

select generate\_instance() FROM generate\_series(1, 500); -- work

CREATE OR REPLACE FUNCTION generate\_type\_of\_document() RETURNS VOID AS

$$

declare

type\_doc varchar[] = '{Справка разрешения, Договор, Подписной лист, Медецинская справка, Юридическая справка}';

index bigint = 0;

new\_name varchar;

inst\_id bigint[];

length int = 0;

length2 bigint = 0;

some\_ind\_1 bigint = 1;

some\_char\_1 varchar = '1';

some\_ind\_2 bigint = 1;

some\_char\_2 varchar = '1';

some\_ind\_3 bigint = 1;

some\_char\_3 varchar = '1';

some\_ind\_4 bigint = 1;

some\_char\_4 varchar = '1';

some\_ind\_5 bigint = 1;

some\_char\_5 varchar = '1';

begin

inst\_id = ARRAY(select instance\_id from "Instance")::bigint[];

length2 = (array\_length(inst\_id, 1))::bigint;

for i in 1..length2

loop

length = round(random() \* 4) + 1;

for j in 1..length

loop

index = floor(random() \* array\_length(type\_doc, 1) + 1)::int;

if index = 1 then

new\_name = type\_doc[index] || ' ' || some\_char\_1;

some\_ind\_1 = some\_ind\_1 + 1;

some\_char\_1 = some\_ind\_1::varchar;

else if index = 2 then

new\_name = type\_doc[index] || ' ' || some\_char\_2;

some\_ind\_2 = some\_ind\_2 + 1;

some\_char\_2 = some\_ind\_2::varchar;

else if index = 3 then

new\_name = type\_doc[index] || ' ' || some\_char\_3;

some\_ind\_3 = some\_ind\_3 + 1;

some\_char\_3 = some\_ind\_3::varchar;

else if index = 4 then

new\_name = type\_doc[index] || ' ' || some\_char\_4;

some\_ind\_4 = some\_ind\_4 + 1;

some\_char\_4 = some\_ind\_4::varchar;

else

new\_name = type\_doc[index] || ' ' || some\_char\_5;

some\_ind\_5 = some\_ind\_5 + 1;

some\_char\_5 = some\_ind\_5::varchar;

end if;

end if;

end if;

end if;

insert into "TypeOfDocument" (name, privileges\_id, instance\_id)

values (new\_name, 1, inst\_id[i]);

end loop;

end loop;

end;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION generate\_type\_of\_document\_default() RETURNS VOID AS

$$

begin

insert into "TypeOfDocument" (name, privileges\_id, instance\_id) values ('Пасспорт', 1, 1);

insert into "TypeOfDocument" (name, privileges\_id, instance\_id) values ('Свидетельство о рождении', 1, 1);

insert into "TypeOfDocument" (name, privileges\_id, instance\_id) values ('Права на вождение автомобиля', 1, 1);

insert into "TypeOfDocument" (name, privileges\_id, instance\_id) values ('Справка с места работы', 1, 1);

end;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION generate\_type\_of\_document\_benefits() RETURNS VOID AS

$$

declare

type\_doc varchar[] = '{Почетный гражданин страны, Инвалидность, Ветеран боевых действий}';

index bigint = 0;

new\_name varchar;

p\_id bigint[];

index\_p bigint;

some\_ind\_1 bigint = 1;

some\_char\_1 varchar = '1';

some\_ind\_2 bigint = 1;

some\_char\_2 varchar = '1';

some\_ind\_3 bigint = 1;

some\_char\_3 varchar = '1';

length int = 0;

begin

length = round(random() \* 100);

for i in 1..length

loop

index = floor(random() \* array\_length(type\_doc, 1) + 1)::int;

if index = 1 then

new\_name = type\_doc[index] || ' ' || some\_char\_1;

some\_ind\_1 = some\_ind\_1 + 1;

some\_char\_1 = some\_ind\_1::varchar;

else if index = 2 then

new\_name = type\_doc[index] || ' ' || some\_char\_2;

some\_ind\_2 = some\_ind\_2 + 1;

some\_char\_2 = some\_ind\_2::varchar;

else

new\_name = type\_doc[index] || ' ' || some\_char\_3;

some\_ind\_3 = some\_ind\_3 + 1;

some\_char\_3 = some\_ind\_3::varchar;

end if;

end if;

p\_id = ARRAY(select privileges\_id from "Privileges")::int[];

index\_p = floor(random() \* array\_length(p\_id, 1) + 1)::int;

index\_p = p\_id[index\_p];

insert into "TypeOfDocument" (name, privileges\_id, instance\_id)

values (new\_name, index\_p, 1);

end loop;

end;

$$ LANGUAGE plpgsql;

select generate\_type\_of\_document\_default(); --work

select generate\_type\_of\_document\_benefits(); --work

select generate\_type\_of\_document(); --work

CREATE or replace function generate\_schedule() returns void as

$$

declare

time\_startt time;

time\_finissh time;

time\_of\_work interval;

index int = 1;

pool\_times time[] = '{10:00:00, 10:30:00, 11:00:00, 11:30:00, 12:00:00, 12:30:00, 13:00:00, 13:30:00, 14:00:00, 14:30:00, ' ||

'15:00:00, 15:30:00, 16:00:00, 16:30:00, 17:00:00, 17:30:00, 18:00:00, 18:30:00, 19:00:00, 19:30:00, 20:00:00, 20:30:00, ' ||

'21:00:00, 21:30:00}';

BEGIN

time\_of\_work = random() \* (interval '7 hours' + interval '59 minutes') + interval '1 minute';

index = floor(random() \* array\_length(pool\_times, 1) + 1)::int;

time\_startt = pool\_times[index];

if(time\_startt + time\_of\_work > '22:00:00' or time\_startt + time\_of\_work < '10:00:00') then

time\_finissh = '22:00:00';

else

time\_finissh = time\_startt + time\_of\_work;

end if;

insert into "Schedule" (time\_start, time\_finish) values (time\_startt, time\_finissh);

end;

$$ LANGUAGE plpgsql;

select generate\_schedule() FROM generate\_series(1, 500); -- work

CREATE or replace function generate\_officials() returns void as

$$

declare

name\_u varchar(15);

pull\_names varchar[] = '{Михаил, Марк, Мирон, Андрей, Павел, Анатолий, Николай, Виктор, Кирилл, Богдан, Никита, Владимир, ' ||

'Парфен, Карл}';

surname\_u varchar(30);

pull\_surnames varchar[] = '{Козлов, Мишанин, Николаев, Попов, Троцкий, Маркс, Капронов, Ксенофонтов, Иванов, Лазарев, ' ||

'Пересвятов, Краснов, Серов, Голубков, Перенов, Датагрипов, Хохлов, Серов, Салов, Пис, Работов, Роботов, ' ||

'Умнов, Поляков, Коков, Мореходов, Перескопов}';

pull\_names\_w varchar[] = '{Елена, Мария, Светлана, Михалина, Юлия, Анастасия, Елизавета, Софья, Соня, София, Анна, Анет, Ким}';

pull\_surnames\_w varchar[] = '{Камова, Пулева, Булева, Комарова, Тамова, Тарханова, Мирославова, Лютикова, Козлова, Мишанина, ' ||

'Лампова, Постгресова, Механова, Плеханова, Сын Зю, Перефракторава, Рефакторова, Мониторова, Усатова, Нью Белосова, Найкова, Таймстампова}';

sex int;

inst\_id bigint[];

length bigint;

length2 bigint;

index\_n int;

index\_s int;

pull\_sch bigint[];

index\_sch bigint;

time\_r interval;

sch\_id bigint;

begin

inst\_id = ARRAY(select instance\_id from "Instance")::bigint[];

pull\_sch = ARRAY(select schedule\_id from "Schedule")::bigint[];

length = (array\_length(inst\_id, 1))::bigint;

for i in 1..length

loop

length2 = round(random() \* 2) + 3;

for j in 1..length2

loop

sex = round(random()) + 1;

if(sex = 1) then

index\_n = floor(random() \* array\_length(pull\_names, 1) + 1)::int;

index\_s = floor(random() \* array\_length(pull\_surnames, 1) + 1)::int;

name\_u = pull\_names[index\_n];

surname\_u = pull\_surnames[index\_s];

else

index\_n = floor(random() \* array\_length(pull\_names\_w, 1) + 1)::int;

index\_s = floor(random() \* array\_length(pull\_surnames\_w, 1) + 1)::int;

name\_u = pull\_names\_w[index\_n];

surname\_u = pull\_surnames\_w[index\_s];

end if;

index\_sch = floor(random() \* array\_length(pull\_sch, 1) + 1)::bigint;

sch\_id = pull\_sch[index\_sch];

time\_r = random() \* interval '59 minutes' + interval '1 minutes';

insert into "Official" (name, surname, time\_of\_receipts, instance\_id, schedule\_id) values

(name\_u, surname\_u, time\_r, inst\_id[i], sch\_id);

end loop;

end loop;

end;

$$ LANGUAGE plpgsql;

select generate\_officials(); --work

CREATE or replace function generate\_prosecutor() returns void as

$$

declare

inst\_id bigint[];

pull\_sch bigint[];

sch\_id bigint;

length bigint;

index\_sch bigint;

BEGIN

inst\_id = ARRAY(select instance\_id from "Instance")::bigint[];

pull\_sch = ARRAY(select schedule\_id from "Schedule")::bigint[];

length = (array\_length(inst\_id, 1))::bigint;

for i in 1..length

loop

index\_sch = floor(random() \* array\_length(pull\_sch, 1) + 1)::bigint;

sch\_id = pull\_sch[index\_sch];

insert into "Prosecutor" (instance\_id, schedule\_id) VALUES (inst\_id[i], sch\_id);

end loop;

end;

$$ LANGUAGE plpgsql;

select generate\_prosecutor(); -- work

CREATE or replace function generate\_checker() returns void as

$$

declare

time\_i interval;

qual bigint;

process\_id bigint[];

length bigint;

length2 int;

BEGIN

process\_id = ARRAY(select prosecutor\_id from "Prosecutor")::bigint[];

length = (array\_length(process\_id, 1))::bigint;

for i in 1..length

loop

length2 = round(random() \* 9) + 1;

for j in 1..length2

loop

qual = floor(random() \* 10) + 1;

time\_i = random() \* interval '23 hours' + interval '1 hours';

insert into "Checker" (time, max\_quantity, prosecutor\_id) values (time\_i, qual, process\_id[i]);

end loop;

end loop;

end;

$$ LANGUAGE plpgsql;

select generate\_checker(); --work

CREATE or replace function generate\_bookkeeping() returns void as

$$

declare

qual bigint;

pull\_sch bigint[];

index\_sch bigint;

sch\_id bigint;

begin

pull\_sch = ARRAY(select schedule\_id from "Schedule")::bigint[];

qual = floor(random() \* 10) + 1;

index\_sch = floor(random() \* array\_length(pull\_sch, 1) + 1)::bigint;

sch\_id = pull\_sch[index\_sch];

insert into "Bookkeeping" (quantity, schedule\_id) VALUES (qual, sch\_id);

end;

$$ LANGUAGE plpgsql;

select generate\_bookkeeping() FROM generate\_series(1, 500); --work

CREATE or replace function generate\_production() returns void as

$$

declare

pull\_bk bigint[];

quanlity bigint;

q bigint;

type\_of\_doc\_pull bigint[];

costt money;

time\_int interval;

length bigint;

type\_id bigint;

BEGIN

pull\_bk = ARRAY(select bookkeeping\_id from "Bookkeeping")::bigint[];

type\_of\_doc\_pull = ARRAY(select type\_of\_document\_id from "TypeOfDocument" where (privileges\_id = 1

and name != 'Пасспорт' and name != 'Свидетельство о рождении' and name != 'Права на вождение автомобиля'

and name != 'Справка с места работы'))::bigint[];

length = (array\_length(pull\_bk, 1))::bigint;

for i in 1..length

loop

q = (select quantity from "Bookkeeping" where bookkeeping\_id = pull\_bk[i])::bigint;

for j in 1..q

loop

quanlity = round(random() \* 100);

time\_int = random() \* interval '1 days' + interval '10 minutes';

costt = round(random() \* 1000);

type\_id = floor(random() \* array\_length(type\_of\_doc\_pull, 1) + 1)::bigint;

if((select count(\*) from "Production" where (bookkeeping\_id = pull\_bk[i] and type\_of\_document\_id = type\_of\_doc\_pull[type\_id])) = 0) then

insert into "Production" (bookkeeping\_id, quantity, type\_of\_document\_id, cost, time) VALUES (pull\_bk[i], quanlity, type\_of\_doc\_pull[type\_id], costt, time\_int);

end if;

end loop;

end loop;

end;

$$ LANGUAGE plpgsql;

select generate\_production();

create or replace function generate\_basic\_user() returns void as

$$

declare

name\_u varchar(15);

pull\_names varchar[] = '{Михаил, Марк, Мирон, Андрей, Павел, Анатолий, Николай, Виктор, Кирилл, Богдан, Никита, Владимир, ' ||

'Парфен, Карл}';

surname\_u varchar(30);

pull\_surnames varchar[] = '{Козлов, Мишанин, Николаев, Попов, Троцкий, Маркс, Капронов, Ксенофонтов, Иванов, Лазарев, ' ||

'Пересвятов, Краснов, Серов, Голубков, Перенов, Датагрипов, Хохлов, Серов, Салов, Пис, Работов, Роботов, ' ||

'Умнов, Поляков, Коков, Мореходов, Перескопов}';

pull\_names\_w varchar[] = '{Елена, Мария, Светлана, Михалина, Юлия, Анастасия, Елизавета, Софья, Соня, София, Анна, Анет, Ким}';

pull\_surnames\_w varchar[] = '{Камова, Пулева, Булева, Комарова, Тамова, Тарханова, Мирославова, Лютикова, Козлова, Мишанина, ' ||

'Лампова, Постгресова, Механова, Плеханова, Сын Зю, Перефракторава, Рефакторова, Мониторова, Усатова, Нью Белосова, Найкова, Таймстампова}';

sex int;

mon money;

time\_res timestamp;

index\_n int;

index\_s int;

BEGIN

mon = round(random() \* 10000);

time\_res = now() + random()\*interval '1 years' + interval '3 hours' + interval '1 hours';

sex = round(random()) + 1;

if(sex = 1) then

index\_n = floor(random() \* array\_length(pull\_names, 1) + 1)::int;

index\_s = floor(random() \* array\_length(pull\_surnames, 1) + 1)::int;

name\_u = pull\_names[index\_n];

surname\_u = pull\_surnames[index\_s];

else

index\_n = floor(random() \* array\_length(pull\_names\_w, 1) + 1)::int;

index\_s = floor(random() \* array\_length(pull\_surnames\_w, 1) + 1)::int;

name\_u = pull\_names\_w[index\_n];

surname\_u = pull\_surnames\_w[index\_s];

end if;

insert into "User" (name, surname, money, time\_result, instance\_id) values

(name\_u, surname\_u, mon, time\_res, 1);

end;

$$ LANGUAGE plpgsql;

select generate\_basic\_user() from generate\_series(1, 2000);

create or replace function generate\_document\_basic() returns void as

$$

declare

type\_pull\_basic bigint[];

type\_pull\_priveleges bigint[];

basic bigint;

priv bigint;

us bigint;

date\_of\_i1 date;

date\_of\_i2 date;

valid date;

name\_d bigint;

pull\_names varchar[] = '{Москва, Самара, Владивосток, Владикавказ, Париж, Вашингтон, Санкт-Петербург, Киев, Брест,' ||

'Белград, Казань}';

pull\_user bigint[];

som bigint;

som2 bigint;

som3 bigint;

length bigint;

begin

type\_pull\_basic = ARRAY(select type\_of\_document\_id from "TypeOfDocument" where type\_of\_document\_id < 4)::bigint[];

type\_pull\_priveleges = ARRAY(select type\_of\_document\_id from "TypeOfDocument" where privileges\_id > 1)::bigint[];

pull\_user = ARRAY(select user\_id from "User")::bigint[];

length = (array\_length(pull\_user, 1))::bigint;

for i in 1..length

loop

som = round(random()) + 1;

som2 = floor(random() \* array\_length(type\_pull\_basic, 1) + 1)::bigint;

som3 = floor(random() \* array\_length(type\_pull\_priveleges, 1) + 1)::bigint;

date\_of\_i1 = '2014-01-01'::date + random() \* interval '8 years';

date\_of\_i2 = '2014-01-01'::date + random() \* interval '60 years';

name\_d = floor(random() \* array\_length(pull\_names, 1) + 1)::bigint;

valid = now()::date + random() \* interval '10 years' + interval '1 year';

us = pull\_user[i];

basic = type\_pull\_basic[som2];

priv = type\_pull\_priveleges[som3];

insert into "Document" (type\_of\_document\_id, user\_id, date\_of\_issue, validity, issued\_by\_whom, parameters\_id)

values (basic, us, date\_of\_i1, valid, pull\_names[name\_d], null);

if(som = 1) then

insert into "Document" (type\_of\_document\_id, user\_id, date\_of\_issue, validity, issued\_by\_whom, parameters\_id)

values (priv, us, date\_of\_i2, null, pull\_names[name\_d], null);

end if;

end loop;

end;

$$ LANGUAGE plpgsql;

select generate\_document\_basic(); --work

create or replace function distribution\_of\_documents() returns void as

$$

declare

instance\_array bigint;

instance\_random\_index bigint;

instance\_random bigint;

user\_array\_id bigint[];

type\_of\_documentId bigint[];

length bigint;

length2 bigint;

date\_of\_i1 date;

parameter bigint;

valid date;

name\_d bigint;

begin

instance\_array = (select count(\*) from "Instance");

user\_array\_id = ARRAY(select user\_id from "User");

length = (array\_length(user\_array\_id, 1))::bigint;

for i in 1..length

loop

instance\_random\_index = floor(random() \* instance\_array + 1)::bigint;

instance\_random = instance\_random\_index;

type\_of\_documentId = ARRAY(select type\_of\_document\_id from "TypeOfDocument" where instance\_id = instance\_random and type\_of\_document\_id > 4 and privileges\_id = 1);

length2 = (array\_length(type\_of\_documentId, 1))::bigint;

update "User" set instance\_id = instance\_random where user\_id = user\_array\_id[i];

for j in 1..length2

loop

date\_of\_i1 = '2014-01-01'::date + random() \* interval '8 years';

insert into "Parameters" (status) values (false);

parameter = (select max(parameters\_id) from "Parameters");

insert into "Document" (type\_of\_document\_id,

user\_id,

date\_of\_issue, validity, issued\_by\_whom, parameters\_id)

values (type\_of\_documentId[j], user\_array\_id[i], date\_of\_i1, null, 'Замок', parameter);

end loop;

end loop;

end

$$ LANGUAGE plpgsql;

select distribution\_of\_documents(); --work

create or replace function official\_some\_function() returns void as

$$

declare

officialId bigint[];

rnd bigint;

parametersId bigint[];

length bigint;

flag bool;

begin

parametersId = ARRAY(select parameters\_id from "Parameters");

length = (array\_length(parametersId, 1))::bigint;

for i in 1..length

loop

officialId = ARRAY(select official\_id from "Parameters"

join "Document" on "Parameters".parameters\_id = "Document".parameters\_id

join "TypeOfDocument" on "Document".type\_of\_document\_id = "TypeOfDocument".type\_of\_document\_id

join "Official" on "Official".instance\_id = "TypeOfDocument".instance\_id

where "Parameters".parameters\_id = parametersId[i]);

rnd = floor(random() \* 3 + 1)::bigint;

for j in 1..rnd

loop

flag = random() > 0.5;

insert into "Signatures" (official\_id, parameters\_id, is\_subscribed)

values (officialId[j], parametersId[i], flag);

end loop;

end loop;

end

$$ LANGUAGE plpgsql;

select official\_some\_function(); --work

create or replace function generate\_queue() returns void as

$$

declare

pull\_users bigint[];

plac bigint;

pull\_parameters bigint[];

pull\_officcials bigint [];

lenght bigint;

lenght2 bigint;

lenght3 bigint;

begin

pull\_users = ARRAY(select user\_id from "User");

lenght = (array\_length(pull\_users, 1))::bigint;

for i in 1..lenght

loop

pull\_parameters = ARRAY(select "Parameters".parameters\_id from "Parameters" join "Document" D on "Parameters".parameters\_id = D.parameters\_id

where user\_id = pull\_users[i]);

lenght2 = (array\_length(pull\_parameters, 1))::bigint;

for j in 1 ..lenght2

loop

pull\_officcials = ARRAY(SELECT official\_id from "Signatures" where parameters\_id = pull\_parameters[j] and is\_subscribed = false);

lenght3 = (array\_length(pull\_officcials, 1))::bigint;

if(lenght3 is null) then

Update "Parameters"

set status = true

where parameters\_id = pull\_parameters[j];

else

for k in 1..lenght3

loop

plac = (select put\_in\_queue(pull\_users[i], pull\_officcials[k]));

end loop;

end if;

end loop;

end loop;

end;

$$ LANGUAGE plpgsql;

select generate\_queue(); --work

create or replace function generate\_status() returns void as

$$

DECLARE

pull\_checker bigint[];

pull\_param bigint[];

lenght bigint;

lenght2 bigint;

inst bigint;

ch\_id bigint;

flag boolean;

Begin

pull\_param = ARRAY(select parameters\_id from "Parameters" where status = true);

lenght = (array\_length(pull\_param, 1))::bigint;

for i in 1..lenght

loop

inst = (select instance\_id from "Document"

join "TypeOfDocument" TOD on "Document".type\_of\_document\_id = TOD.type\_of\_document\_id where parameters\_id = pull\_param[i])::bigint;

pull\_checker = ARRAY(select checker\_id from "Checker" join "Prosecutor" P on "Checker".prosecutor\_id = P.prosecutor\_id where instance\_id = inst);

lenght2 = (array\_length(pull\_checker, 1))::bigint;

ch\_id = 0;

for j in 1..lenght2

loop

if((select count(\*) from "Status" where is\_valid = false and checker\_id = pull\_checker[j]) < (select max\_quantity from "Checker" where checker\_id = pull\_checker[j])) then

ch\_id = pull\_checker[j];

end if;

end loop;

if(NOT(ch\_id = 0)) then

flag = random() > 0.5;

insert into "Status" (is\_valid, parameter\_id, checker\_id) VALUES (flag, pull\_param[i], ch\_id);

end if;

end loop;

end;

$$ LANGUAGE plpgsql;

select generate\_status();

# Удаление

Очищение таблиц

truncate "Bookkeeping" cascade;

truncate "Checker" cascade;

truncate "Document" cascade;

truncate "Instance" cascade;

truncate "Official" cascade;

truncate "Parameters" cascade;

truncate "Privileges" cascade;

truncate "Production" cascade;

truncate "Prosecutor" cascade;

truncate "Queue" cascade;

truncate "Schedule" cascade;

truncate "Signatures" cascade;

truncate "Status" cascade;

truncate "TypeOfDocument" cascade;

truncate "User" cascade;

Удаление БД  
DROP SCHEMA public CASCADE; CREATE SCHEMA public;

Наиболее частые запросы:  
Так как одной из ключевых точек бизнес-процесса является добавление пользователя в конкретную очередь и обновление приоритета, то, например тяжелый запрос обновления приоритета является одним из самых частых:  
Изображение выглядит как текст

Автоматически созданное описание

Еще одним частым запросом будет проверка соответствующего количества подписей на справках, для этого будет выполняться запрос вида:  
Изображение выглядит как текст

Автоматически созданное описание

А также аналогичный запрос на проверку уже подписанных справок в прокуратуре:

Изображение выглядит как текст

Автоматически созданное описание  
  
Эти два запроса способствуют проверке, можно ли пользователю отправляться на следующий уровень или нет.

Описание индексов и обоснование их использования:  
Для запроса (1), который осуществляет обновление приоритета в очереди, мы предлагаем следующие индексы:  
Изображение выглядит как текст, табло, закрыть

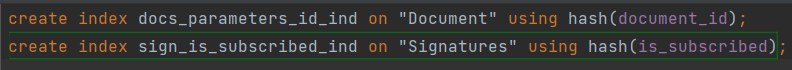
Автоматически созданное описание  
Так как btree индекс хорошо подходит для тех условий, в которых происходит сравнение на больше или меньше, внутренняя реализация дерева может позволить ускорить время работы, а hash индекс на official\_id поможет в проверке на равенство для поиска подходящего значения.  
  
Вид анализа до индексов:  
Изображение выглядит как текст

Автоматически созданное описание

После:  
Изображение выглядит как текст

Автоматически созданное описание  
  
Больше чем в два раза уменьшилась стадия планирования, небольшой прирост показала стадия выполнения, возможно стоит убрать один из индексов.

Для запроса (2), который осуществляет проверку соответствия необходимого количества подписей, мы предлагаем использовать индексы на document\_id таблицы Document и is\_subscribed таблицы Signature, так как сам запрос содержит условия на равенство и в этом случае лучше использовать hash index, добавлять индекс на поле, по которому происходит соединение, не имеет большого смысла.



Вид анализа до индексов:  
Изображение выглядит как текст

Автоматически созданное описание  
  
После:  
Изображение выглядит как текст

Автоматически созданное описание

Как мы видим, наблюдается прирост производительности стадии планирования и стадии выполнения, общая оценка прироста около 1.5x

Для запроса (3), который проверяет можно ли пользователю отправляться на следующий уровень, мы предлагаем hash индекс на instance\_id таблицы TypeOfDocument  


Выполнение до:  
Изображение выглядит как текст

Автоматически созданное описание  
  
Выполнение после:  
Изображение выглядит как текст

Автоматически созданное описание  
  
Как мы видим стадия планирования и выполнения также уменьшили свое время, скорость выполнения возросла в 3 раза.