Курсовая работа 2 этап
Модель ИС «Регистратор доменных имён»

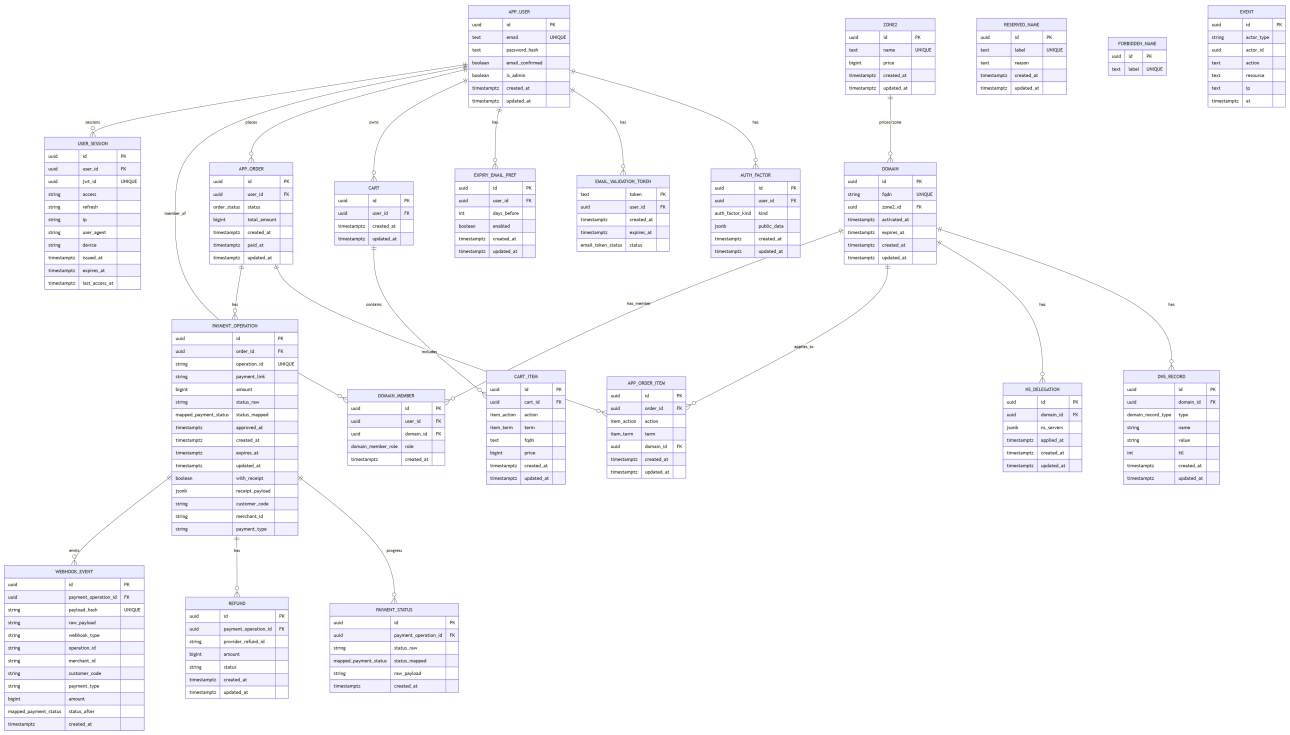
Выполнили:
Малышев Никита Александрович (409067),
Зинченко Иван Николаевич (408657)

Хеш коммита с SRS(ветка srs): `a95e42613364b1a6e1d34e7f11b58675ca561a25`
Хеш(sha1) файла SRS: `46a069b66f43797ac2afc61b9693cfdd1b7c6ffe`

Содержание

1. Инфологическая модель

Ссылка



2. Создание БД

```
CREATE TYPE auth_factor_kind AS ENUM ('TOTP', 'WebAuthn');

CREATE TYPE email_token_status AS ENUM ('VERIFY_EMAIL', 'RESET_PASSWORD');

CREATE TYPE domain_record_type AS ENUM ('A','AAAA','CNAME','TXT','MX','SRV','CAA');

CREATE TYPE item_action AS ENUM ('register','renew');
CREATE TYPE item_term   AS ENUM ('monthly','yearly');

CREATE TYPE order_status AS ENUM
('created','pending_payment','paid','cancelled','failed');

CREATE TYPE mapped_payment_status AS ENUM
('CREATED','APPROVED','ON_REFUND','REFUNDED','EXPIRED');

CREATE TYPE domain_member_role AS ENUM ('OWNER','USER');

CREATE TABLE app_user (
    id              uuid PRIMARY KEY DEFAULT gen_random_uuid(),
    email           text NOT NULL UNIQUE,
    password_hash   text NOT NULL,
    email_confirmed boolean NOT NULL DEFAULT false,
    is_admin        boolean NOT NULL DEFAULT false,
    created_at      timestamptz NOT NULL DEFAULT now(),
    updated_at      timestamptz NOT NULL DEFAULT now()
);
```

```
CREATE TABLE auth_factor (
    id          uuid PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id     uuid NOT NULL REFERENCES app_user(id) ON DELETE CASCADE,
    kind        auth_factor_kind NOT NULL,
    public_data jsonb NOT NULL,
    created_at  timestamptz NOT NULL DEFAULT now(),
    updated_at  timestamptz NOT NULL DEFAULT now()
);
CREATE INDEX auth_factor_user_id_idx ON auth_factor(user_id);

CREATE TABLE email_validation_token (
    token      text PRIMARY KEY,
    user_id    uuid NOT NULL REFERENCES app_user(id) ON DELETE CASCADE,
    created_at timestamptz NOT NULL DEFAULT now(),
    expires_at timestamptz NOT NULL,
    status     email_token_status NOT NULL
);
CREATE INDEX email_validation_token_expires_at_idx ON email_validation_token(expires_at);

CREATE TABLE expiry_email_pref (
    id          uuid PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id     uuid NOT NULL REFERENCES app_user(id) ON DELETE CASCADE,
    days_before integer NOT NULL CHECK (days_before >= 0),
    enabled     boolean NOT NULL DEFAULT true,
    created_at  timestamptz NOT NULL DEFAULT now(),
    updated_at  timestamptz NOT NULL DEFAULT now(),
    UNIQUE (user_id, days_before)
);

CREATE TABLE zone2 (
    id         uuid PRIMARY KEY DEFAULT gen_random_uuid(),
    name       text NOT NULL UNIQUE,
    price      bigint NOT NULL CHECK (price >= 0),
    created_at timestamptz NOT NULL DEFAULT now(),
    updated_at timestamptz NOT NULL DEFAULT now()
);

CREATE TABLE domain (
    id           uuid PRIMARY KEY DEFAULT gen_random_uuid(),
    fqdn         varchar(63) NOT NULL UNIQUE,
    zone2_id     uuid NOT NULL REFERENCES zone2(id) ON DELETE RESTRICT,
    activated_at timestamptz,            -- NULL = не активирован
    expires_at   timestamptz NOT NULL,
    created_at   timestamptz NOT NULL DEFAULT now(),
    updated_at   timestamptz NOT NULL DEFAULT now()
);
CREATE INDEX domain_expires_at_idx ON domain(expires_at);

CREATE TABLE domain_member (
    id         uuid PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id    uuid NOT NULL REFERENCES app_user(id) ON DELETE CASCADE,
    domain_id  uuid NOT NULL REFERENCES domain(id) ON DELETE CASCADE,
    role       domain_member_role NOT NULL,
    created_at timestamptz NOT NULL DEFAULT now(),
    UNIQUE (user_id, domain_id)
);

CREATE TABLE dns_record (
```

```
    id         uuid PRIMARY KEY DEFAULT gen_random_uuid(),
    domain_id  uuid NOT NULL REFERENCES domain(id) ON DELETE CASCADE,
    type       domain_record_type NOT NULL,
    name       varchar(255) NOT NULL,
    value      varchar(1024) NOT NULL,
    ttl        integer NOT NULL CHECK (ttl > 0),
    created_at timestamptz NOT NULL DEFAULT now(),
    updated_at timestamptz NOT NULL DEFAULT now()
);
CREATE INDEX dns_record_domain_id_idx ON dns_record(domain_id);

CREATE TABLE ns_delegation (
    id         uuid PRIMARY KEY DEFAULT gen_random_uuid(),
    domain_id  uuid NOT NULL UNIQUE REFERENCES domain(id) ON DELETE CASCADE,
    ns_servers jsonb NOT NULL,
    applied_at timestamptz,
    created_at timestamptz NOT NULL DEFAULT now(),
    updated_at timestamptz NOT NULL DEFAULT now()
);

CREATE TABLE cart (
    id         uuid PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id    uuid NOT NULL UNIQUE REFERENCES app_user(id) ON DELETE CASCADE,
    created_at timestamptz NOT NULL DEFAULT now(),
    updated_at timestamptz NOT NULL DEFAULT now()
);

CREATE TABLE cart_item (
    id         uuid PRIMARY KEY DEFAULT gen_random_uuid(),
    cart_id    uuid NOT NULL REFERENCES cart(id) ON DELETE CASCADE,
    action     item_action NOT NULL,
    term       item_term NOT NULL,
    fqdn       text NOT NULL,
    price      bigint NOT NULL CHECK (price >= 0),
    created_at timestamptz NOT NULL DEFAULT now(),
    updated_at timestamptz NOT NULL DEFAULT now(),
    UNIQUE (cart_id, fqdn, action, term)
);

CREATE TABLE app_order (
    id           uuid PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id      uuid NOT NULL REFERENCES app_user(id) ON DELETE RESTRICT,
    status       order_status NOT NULL DEFAULT 'created',
    total_amount bigint NOT NULL CHECK (total_amount >= 0),
    created_at   timestamptz NOT NULL DEFAULT now(),
    paid_at      timestamptz,
    updated_at   timestamptz NOT NULL DEFAULT now()
);
CREATE INDEX app_order_user_id_idx ON app_order(user_id);
CREATE INDEX app_order_created_at_paid_at_idx ON app_order(created_at, paid_at) WHERE
paid_at IS NULL;

CREATE TABLE app_order_item (
    id         uuid PRIMARY KEY DEFAULT gen_random_uuid(),
    order_id   uuid NOT NULL REFERENCES app_order(id) ON DELETE CASCADE,
    action     item_action NOT NULL,
    term       item_term   NOT NULL,
    domain_id  uuid NOT NULL REFERENCES domain(id) ON DELETE RESTRICT,
    created_at timestamptz NOT NULL DEFAULT now(),
```

```
    updated_at timestamptz NOT NULL DEFAULT now()
);
CREATE INDEX app_order_item_order_id_idx ON app_order_item(order_id);

CREATE TABLE payment_operation (
    id              uuid PRIMARY KEY DEFAULT gen_random_uuid(),
    order_id        uuid NOT NULL REFERENCES app_order(id) ON DELETE CASCADE,
    operation_id    varchar(255) UNIQUE,
    payment_link    varchar(255),
    amount          bigint NOT NULL CHECK (amount >= 0),
    status_raw      varchar(255) NOT NULL,
    status_mapped   mapped_payment_status NOT NULL,
    approved_at     timestamptz,
    created_at      timestamptz NOT NULL DEFAULT now(),
    expires_at      timestamptz,
    updated_at      timestamptz NOT NULL DEFAULT now(),
    with_receipt    boolean NOT NULL DEFAULT false,
    receipt_payload jsonb,

    customer_code varchar(255),
    merchant_id   varchar(255),
    payment_type  varchar(255)
);
CREATE INDEX payment_operation_order_id_idx ON payment_operation(order_id);

CREATE TABLE payment_status (
    id                  uuid PRIMARY KEY DEFAULT gen_random_uuid(),
    payment_operation_id uuid NOT NULL REFERENCES payment_operation(id) ON DELETE CASCADE,
    status_raw          varchar(255) NOT NULL,
    status_mapped       mapped_payment_status NOT NULL,
    raw_payload         varchar(255),
    created_at          timestamptz NOT NULL DEFAULT now()
);
CREATE INDEX payment_status_op_id_idx ON payment_status(payment_operation_id);

CREATE TABLE refund (
    id                  uuid PRIMARY KEY DEFAULT gen_random_uuid(),
    payment_operation_id uuid NOT NULL REFERENCES payment_operation(id) ON DELETE CASCADE,
    provider_refund_id  varchar(255),
    amount              bigint NOT NULL CHECK (amount >= 0),
    status              varchar(255) NOT NULL,
    created_at          timestamptz NOT NULL DEFAULT now(),
    updated_at          timestamptz NOT NULL DEFAULT now()
);
CREATE INDEX refund_op_id_idx ON refund(payment_operation_id);

CREATE TABLE webhook_event (
    id                  uuid PRIMARY KEY DEFAULT gen_random_uuid(),
    payment_operation_id uuid REFERENCES payment_operation(id) ON DELETE SET NULL,
    payload_hash        varchar(255),
    raw_payload         varchar(255),

    webhook_type        varchar(255),
    operation_id        varchar(255),
    merchant_id         varchar(255),
    customer_code       varchar(255),
    payment_type        varchar(255),
    amount              bigint,
    status_after        mapped_payment_status,
```

```
    created_at           timestamptz NOT NULL DEFAULT now()
);

CREATE TABLE reserved_name (
    id         uuid PRIMARY KEY DEFAULT gen_random_uuid(),
    label      text NOT NULL UNIQUE,
    reason     text,
    created_at timestamptz NOT NULL DEFAULT now(),
    updated_at timestamptz NOT NULL DEFAULT now()
);

CREATE TABLE forbidden_name (
    id    uuid PRIMARY KEY DEFAULT gen_random_uuid(),
    label text NOT NULL UNIQUE
);

CREATE TABLE event (
    id         uuid PRIMARY KEY DEFAULT gen_random_uuid(),
    actor_type varchar(64) NOT NULL,  -- 'user' | 'admin' | 'system'
    actor_id   uuid,
    action     text NOT NULL,
    resource   text,
    ip         text,
    at         timestamptz NOT NULL
);
CREATE INDEX event_at_idx ON event USING BRIN(at);
CREATE INDEX event_actor_idx ON event(actor_type, actor_id);

CREATE TABLE user_session (
    id             uuid PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id        uuid NOT NULL REFERENCES app_user(id) ON DELETE CASCADE,
    jwt_id         uuid NOT NULL UNIQUE,  -- jti
    access         varchar(255) NOT NULL,
    refresh        varchar(255) NOT NULL,
    ip             varchar(40),
    user_agent     varchar(255),
    device         varchar(255),
    issued_at      timestamptz NOT NULL,
    expires_at     timestamptz NOT NULL,
    last_access_at timestamptz
);
CREATE INDEX user_session_user_id_idx ON user_session(user_id);
CREATE INDEX user_session_expires_at_idx ON user_session(expires_at);


CREATE OR REPLACE FUNCTION set_updated_at() RETURNS trigger AS $$
BEGIN
  NEW.updated_at := now();
  RETURN NEW;
END; $$ LANGUAGE plpgsql;

DO $$
DECLARE
  r record;
BEGIN
  FOR r IN
    SELECT
 t.table_name
```

```
    FROM information_schema.tables AS t
    JOIN information_schema.columns AS c
      ON c.table_schema = t.table_schema
        AND c.table_name = t.table_name
        AND c.column_name = 'updated_at'
    WHERE t.table_schema NOT IN ('pg_catalog', 'information_schema')
 AND t.table_type = 'BASE TABLE'
  LOOP
    EXECUTE format(
      'CREATE TRIGGER %I_set_updated_at
         BEFORE UPDATE ON %I
         FOR EACH ROW
         EXECUTE FUNCTION set_updated_at()',
      r.table_name, r.table_name
    );
    END LOOP;
END $$;
```

3. Удаление БД

```
DO $$
DECLARE
  r record;
BEGIN
  FOR r IN
 SELECT t.table_name
    FROM information_schema.tables  AS t
    JOIN information_schema.columns AS c
      ON c.table_schema = t.table_schema
      AND c.table_name = t.table_name
      AND c.column_name = 'updated_at'
    WHERE t.table_schema NOT IN ('pg_catalog', 'information_schema')
      AND t.table_type = 'BASE TABLE'
  LOOP
    EXECUTE format('DROP TRIGGER IF EXISTS %I_set_updated_at ON %I;', r.table_name,
r.table_name);
  END LOOP;
END $$;

DROP FUNCTION IF EXISTS set_updated_at() CASCADE;

DROP TABLE IF EXISTS
  user_session,
  domain_member,
  event,
  forbidden_name,
  reserved_name,
  webhook_event,
  refund,
  payment_status,
  payment_operation,
  app_order_item,
  app_order,
  cart_item,
  cart,
  ns_delegation,
  dns_record,
  domain,
  zone2,
```

```
  expiry_email_pref,
  email_validation_token,
  auth_factor,
  app_user
CASCADE;

DROP TYPE IF EXISTS
  domain_member_role,
  mapped_payment_status,
  order_status,
  item_term,
  item_action,
  domain_record_type,
  email_token_status,
  auth_factor_kind
CASCADE;
```

4. Генерация данных

```
BEGIN;

CREATE OR REPLACE FUNCTION generate_ipv4()
 RETURNS text
 LANGUAGE plpgsql
AS $$
DECLARE
 ip text;
BEGIN
 SELECT (trunc(50+random()*200))::int || '.' || (trunc(1+random()*250))::int || '.' ||
(trunc(1+random()*250))::int || '.' || (trunc(1+random()*250))::int
  INTO ip;

  RETURN ip;
END $$;

DO $$
DECLARE
  users_cnt   int := 100_000;
  domains_cnt int := 1_000_000;
  carts_cnt   int := 42_000;
  orders_cnt  int := 1_100_000;
BEGIN

-- Зоны
INSERT INTO zone2(name, price)
SELECT name, price FROM (
  VALUES
    ('com',   150000),
    ('net',   130000),
    ('org',   120000),
    ('io',    390000),
    ('app',   250000),
    ('dev',   220000),
    ('ai',    590000),
    ('ru',     70000),
    ('by',     80000),
    ('site',  110000)
) AS t(name, price)
ORDER BY price DESC
```

```
ON CONFLICT DO NOTHING;

-- Пользователи
INSERT INTO app_user(email, password_hash, email_confirmed, is_admin, created_at)
SELECT
  'user' || gs || '@example.com',
  md5(random()::text),
  random() < 0.95,
  random() < 0.01,
  now() - random() * interval '120 days'
FROM generate_series(1, users_cnt) gs;

-- 2FA
INSERT INTO auth_factor(user_id, kind, public_data)
SELECT id, 'TOTP', '{"secret":"test"}'
FROM app_user
WHERE random() < 0.7;

-- Email tokens
INSERT INTO email_validation_token(token, user_id, created_at, expires_at, status)
SELECT
  md5(random()::text),
  id,
  now() - random() * interval '10 days',
  now() + (1 + random() * 10)::int * interval '1 day',
  (ARRAY['VERIFY_EMAIL','RESET_PASSWORD'])[1 + (random()*1)::int]::email_token_status
FROM app_user
WHERE random() < 0.3;

-- Зоны готовы, теперь домены
INSERT INTO domain(fqdn, zone2_id, activated_at, expires_at, created_at)
SELECT
  left(md5(random()::text), 8) || '.' || z.name,
  z.id,
  CASE WHEN random() < 0.95 THEN now() - random() * interval '90 days' END,
  now() + (10 + random() * 700)::int * interval '1 day',
  now() - random() * interval '200 days'
FROM zone2 z
CROSS JOIN generate_series(1, domains_cnt) gs
LIMIT domains_cnt
ON CONFLICT DO NOTHING;

-- DNS записи
INSERT INTO dns_record(domain_id, type, name, value, ttl)
SELECT
  d.id,
  'A',
  '@',
  generate_ipv4(),
  3600
FROM domain d
WHERE random() < 0.9;

-- NS
INSERT INTO ns_delegation(domain_id, ns_servers)
SELECT d.id,
       jsonb_build_array('ns1.example.net','ns2.example.net')
FROM domain d;
```

```sql
-- Корзины
INSERT INTO cart(user_id)
SELECT id FROM app_user ORDER BY random() LIMIT carts_cnt;

-- Товары в корзине
INSERT INTO cart_item(cart_id, action, term, fqdn, price)
SELECT
  c.id,
  (ARRAY['register','renew'])[1 + (random()*1)::int]::item_action,
  (ARRAY['monthly','yearly'])[1 + (random()*1)::int]::item_term,
  (SELECT fqdn FROM domain ORDER BY random() LIMIT 1),
  (10000 + (random()*600000))::bigint
FROM cart c
JOIN generate_series(1,2) g ON TRUE
ON CONFLICT DO NOTHING;

-- Заказы
INSERT INTO app_order(user_id, status, total_amount, created_at, paid_at)
SELECT
  id,
  (ARRAY['created','pending_payment','paid','cancelled','failed'])[1 +
(random()*4)::int]::order_status,
  (100000 + random()*1000000)::bigint,
  now() - random() * interval '60 days',
  CASE WHEN random() < 0.5 THEN now() - random() * interval '30 days' END
FROM app_user
ORDER BY random()
LIMIT domains_cnt
ON CONFLICT DO NOTHING;

-- Позиции заказа
INSERT INTO app_order_item(order_id, action, term, domain_id)
SELECT
  o.id,
  (ARRAY['register','renew'])[1 + (random()*1)::int]::item_action,
  (ARRAY['monthly','yearly'])[1 + (random()*1)::int]::item_term,
  (SELECT id FROM domain ORDER BY random() LIMIT 1)
FROM app_order o
JOIN generate_series(1,2) g ON true;

-- Платёжные операции
INSERT INTO payment_operation(order_id, operation_id, payment_link, amount, status_raw,
status_mapped, with_receipt)
SELECT
  o.id,
  'op_' || left(md5(random()::text), 8),
  'https://enter.tochka.com/'||o.id,
  o.total_amount,
  'CREATED',
  'CREATED',
  random() < 0.5
FROM app_order o
WHERE random() < 0.9
ON CONFLICT DO NOTHING;

-- Платёжные статусы
INSERT INTO payment_status(payment_operation_id, status_raw, status_mapped)
SELECT id, 'CREATED', 'CREATED' FROM payment_operation;
```

```sql
-- Возвраты
INSERT INTO refund(payment_operation_id, provider_refund_id, amount, status)
SELECT id, 'rf_'||left(md5(random()::text), 8), amount, 'SUCCESS'
FROM payment_operation
WHERE random() < 0.2;

-- Вебхуки
INSERT INTO webhook_event(payload_hash, raw_payload, webhook_type, amount, status_after)
SELECT
  md5(random()::text),
  'payload',
  'payment',
  100000 + random()*500000,
  'APPROVED'
FROM generate_series(1,orders_cnt*0.7);

-- Сессии
INSERT INTO user_session(user_id, jwt_id, access, refresh, issued_at, expires_at)
SELECT
  id,
  gen_random_uuid(),
  md5(random()::text),
  md5(random()::text),
  now() - random() * interval '10 days',
  now() + random() * interval '30 days'
FROM app_user;

INSERT INTO domain_member(user_id, domain_id, role, created_at)
SELECT
  u.id,
  d.id,
  'OWNER',
  now() - random() * interval '100 days'
FROM app_user u
JOIN domain d ON random() < 0.05
LIMIT users_cnt*0.8;

-- Запрещённые имена
INSERT INTO forbidden_name(label)
SELECT
  md5(random()::text)
FROM generate_series(1, 5000) g;

-- Зарезервированные имена
INSERT INTO reserved_name(label)
SELECT
  md5(random()::text)
FROM generate_series(1, 5000) g;

-- Настройки уведомлений о продлении доменов
INSERT INTO expiry_email_pref(user_id, days_before, enabled, created_at, updated_at)
SELECT
  id,
  (ARRAY[1, 3, 7, 14, 30])[1 + (random()*4)::int],
  random() < 0.9,
  now() - random() * interval '30 days',
  now() - random() * interval '10 days'
FROM app_user
WHERE random() < 0.3
```

```
ON CONFLICT (user_id, days_before) DO NOTHING;

-- email_validation_token
INSERT INTO email_validation_token(token, user_id, created_at, expires_at, status)
SELECT
  md5(random()::text),
  id,
  now() - random() * interval '5 days',
  now() + (1 + random() * 10) * interval '1 day',
  (ARRAY['VERIFY_EMAIL','RESET_PASSWORD'])[1 + (random()*1)::int]::email_token_status
FROM app_user
WHERE random() < 0.2
ON CONFLICT DO NOTHING;

BEGIN
  DECLARE
    uids uuid[];
  BEGIN
    SELECT array_agg(id) INTO uids FROM app_user;

    FOR months in 0..9 LOOP
      FOR seconds in 1..10000 LOOP
        INSERT INTO event(actor_type, actor_id, action, resource, ip, at)
        SELECT
          (ARRAY['user','admin','system'])[1 + (random()*2)::int],
          uids[months + seconds],
          (ARRAY['login', 'logout', 'create_payment', 'delete_dns_record'])[1 +
(random()*3)::int],
          (ARRAY[uids[months + seconds]::text, 'example.com'])[1 + (random()*1)::int],
          generate_ipv4(),
          now() + seconds * '1 second'::interval + months * '1 month'::interval;
      END LOOP;
    END LOOP;
  END;
END;

END $$;

DROP FUNCTION IF EXISTS generate_ipv4;

COMMIT;

ANALYZE;

VACUUM;
```

5. Доказательство необходимости индексов

Докажем, что индексы, которые созданы через `CREATE INDEX` повышают производительность системы. Имеются следующие индексы:

1. auth_factor_user_id_idx
2. email_validation_token_expires_at_idx
3. domain_expires_at_idx
4. dns_record_domain_id_idx
5. app_order_user_id_idx
6. app_order_created_at_paid_at_idx
7. app_order_item_order_id_idx

8. payment_operation_order_id_idx
9. payment_status_op_id_idx
10. refund_op_id_idx
11. event_at_idx
12. event_actor_idx
13. user_session_user_id_idx
14. user_session_expires_at_idx

Теоретически можно объяснить применение индексов:

1. auth_factor_user_id_idx
   Индекс по user_id ускоряет выборку всех способов 2FA, принадлежащих конкретному пользователю.
2. email_validation_token_expires_at_idx
   Позволяет быстро находить просроченные или активные токены, например при очистке старых записей.
3. domain_expires_at_idx
   Ускоряет поиск доменов, срок действия которых подходит к концу (для напоминаний и продлений).
4. dns_record_domain_id_idx
   Позволяет быстро получить все DNS-записи конкретного домена при открытии панели управления DNS.
5. app_order_user_id_idx
   Ускоряет выборку всех заказов пользователя.
6. app_order_created_at_paid_at_idx
   Это частичный составной индекс нужен, чтобы быстро находить неоплаченные заказы.
7. app_order_item_order_id_idx
   Позволяет быстро получить все позиции внутри конкретного заказа.
8. payment_operation_order_id_idx
   Ускоряет выборку операций оплаты, связанных с определённым заказом.
9. payment_status_op_id_idx
   Помогает быстро находить историю изменения статусов для конкретной операции.
10. refund_op_id_idx
    Ускоряет поиск возвратов по операции оплаты.
11. event_at_idx
    Ускоряет поиск по временным меткам событий.
12. event_actor_idx
    Позволяет быстро найти все действия конкретного пользователя, администратора или системы.
13. user_session_user_id_idx
    Ускоряет выборку всех активных сессий пользователя.
14. user_session_expires_at_idx
    Позволяет быстро очистить просроченные сессии.

Для проверки необходимости индексов будем сравнивать планы выполнения SQL запросов при помощи команды `EXPLAIN ANALYZE`. Сначала выполним её без индексов, затем после. Выполнение будет на тестовых данных, которые сгенерированы скриптом выше (Раздел 4).

Рассмотрим прецеденты из SRS, которые выполняют операции чтения из базы данных и обращаются к полям, отличным от первичных или уникальных.

Удалим индексы из БД, выполнив команду:

```
DROP INDEX auth_factor_user_id_idx;
DROP INDEX email_validation_token_expires_at_idx;
DROP INDEX domain_expires_at_idx;
DROP INDEX dns_record_domain_id_idx;
DROP INDEX app_order_user_id_idx;
DROP INDEX app_order_created_at_paid_at_idx;
DROP INDEX app_order_item_order_id_idx;
DROP INDEX payment_operation_order_id_idx;
DROP INDEX payment_status_op_id_idx;
DROP INDEX refund_op_id_idx;
DROP INDEX event_at_idx;
DROP INDEX event_actor_idx;
DROP INDEX user_session_user_id_idx;
DROP INDEX user_session_expires_at_idx;
```

Не забудем выполнить `ANALYZE`. Проведём анализ планов и затем вернём индексы обратно.

```
CREATE INDEX auth_factor_user_id_idx ON auth_factor(user_id);
CREATE INDEX email_validation_token_expires_at_idx ON email_validation_token(expires_at);
CREATE INDEX domain_expires_at_idx ON domain(expires_at);
CREATE INDEX dns_record_domain_id_idx ON dns_record(domain_id);
CREATE INDEX app_order_user_id_idx ON app_order(user_id);
CREATE INDEX app_order_created_at_paid_at_idx ON app_order(created_at, paid_at) WHERE
paid_at IS NULL;
CREATE INDEX app_order_item_order_id_idx ON app_order_item(order_id);
CREATE INDEX payment_operation_order_id_idx ON payment_operation(order_id);
CREATE INDEX payment_status_op_id_idx ON payment_status(payment_operation_id);
CREATE INDEX refund_op_id_idx ON refund(payment_operation_id);
CREATE INDEX event_at_idx ON event USING BRIN(at);
CREATE INDEX event_actor_idx ON event(actor_type, actor_id);
CREATE INDEX user_session_user_id_idx ON user_session(user_id);
CREATE INDEX user_session_expires_at_idx ON user_session(expires_at);
```

Проведём сравнение планов выполнения запросов.

5.1. R2 - Аутентификация пользователя

Примерный SQL запрос:

```
-- Получить 2FA-параметры
SELECT id, kind, public_data
FROM auth_factor
WHERE user_id = $1;
```

План выполнения без индексов:

```
Seq Scan on auth_factor  (cost=0.00..1813.40 rows=1 width=43) (actual time=26.360..26.361
rows=0 loops=1)
  Filter: (user_id = '88b30e7d-8475-4bad-aeaa-6a11e33c5921'::uuid)
  Rows Removed by Filter: 70192
Planning Time: 0.525 ms
Execution Time: 26.393 ms
```

План выполнения с индексами:

```
Index Scan using auth_factor_user_id_idx on auth_factor  (cost=0.42..8.44 rows=1 width=43)
(actual time=0.137..0.137 rows=0 loops=1)
  Index Cond: (user_id = '88b30e7d-8475-4bad-aeaa-6a11e33c5921'::uuid)
```

```
Planning Time: 0.488 ms
Execution Time: 0.159 ms
```

Максимальный cost параметр уменьшился в 214.9 раз с 1813.4 до 8.44.

Время исполнения запроса уменьшилось в 166 раз с 26.393 мс до 0.159 мс.

Индекс помогает и существенно ускоряет систему.

5.2. CA2/B1 - Оформление заказа и Оплата заказа

Примерный SQL запрос:

```sql
-- Ожидающие оплаты заказы пользователя за последние 24 часа
SELECT id, status, total_amount, created_at
FROM app_order
WHERE user_id = $1
  AND paid_at IS NULL
  AND created_at >= now() - INTERVAL '10 minutes'
ORDER BY created_at DESC;
```

План выполнения без индексов:

```
Sort  (cost=3186.01..3186.02 rows=1 width=36) (actual time=31.416..31.418 rows=0 loops=1)
  Sort Key: created_at DESC
  Sort Method: quicksort  Memory: 25kB
  ->  Seq Scan on app_order  (cost=0.00..3186.00 rows=1 width=36) (actual
time=31.370..31.371 rows=0 loops=1)
        Filter: ((paid_at IS NULL) AND (user_id = '5109bc19-175d-441a-9da7-
c2ac7e92f708'::uuid) AND (created_at >= (now() - '00:10:00'::interval)))
        Rows Removed by Filter: 100000
Planning Time: 1.208 ms
Execution Time: 31.628 ms
```

План выполнения с индексами:

```
Sort  (cost=8.45..8.46 rows=1 width=36) (actual time=0.859..0.860 rows=0 loops=1)
  Sort Key: created_at DESC
  Sort Method: quicksort  Memory: 25kB
  ->  Index Scan using app_order_user_id_idx on app_order  (cost=0.42..8.44 rows=1
width=36) (actual time=0.847..0.847 rows=0 loops=1)
        Index Cond: (user_id = '8b0102f1-40ca-42a3-8dc0-e57fc1df777c'::uuid)
        Filter: ((paid_at IS NULL) AND (created_at >= (now() - '00:10:00'::interval)))
Planning Time: 1.694 ms
Execution Time: 0.909 ms
```

Максимальный cost параметр уменьшился в 376.6 раз с 3186.02 до 8.46.

Время исполнения запроса уменьшилось в 34.8 раз с 31.628 мс до 0.909 мс.

Индекс помогает и существенно ускоряет систему.

Примерный SQL запрос:

```sql
-- Получить историю статусов заказа
SELECT status_mapped, status_raw, created_at
FROM payment_status
WHERE payment_operation_id = $1
ORDER BY created_at DESC;
```

План выполнения без индексов:

```
Sort  (cost=2052.50..2052.50 rows=1 width=20) (actual time=24.104..24.105 rows=0 loops=1)
  Sort Key: created_at DESC
  Sort Method: quicksort  Memory: 25kB
```

```
  ->  Seq Scan on payment_status  (cost=0.00..2052.49 rows=1 width=20) (actual
time=24.092..24.092 rows=0 loops=1)
        Filter: (payment_operation_id = '5109bc19-175d-441a-9da7-c2ac7e92f708'::uuid)
        Rows Removed by Filter: 89959
Planning Time: 0.613 ms
Execution Time: 24.176 ms
```

План выполнения с индексами:

```
Sort  (cost=8.45..8.45 rows=1 width=20) (actual time=0.986..0.987 rows=0 loops=1)
  Sort Key: created_at DESC
  Sort Method: quicksort  Memory: 25kB
  ->  Index Scan using payment_status_op_id_idx on payment_status  (cost=0.42..8.44 rows=1
width=20) (actual time=0.975..0.975 rows=0 loops=1)
        Index Cond: (payment_operation_id = '8b0102f1-40ca-42a3-8dc0-e57fc1df777c'::uuid)
Planning Time: 0.702 ms
Execution Time: 1.031 ms
```

Максимальный cost параметр уменьшился в 242.9 раз с 2052.5 до 8.45.

Время исполнения запроса уменьшилось в 23.4 раз с 24.176 мс до 1.031 мс.

Индекс помогает и существенно ускоряет систему.

Примерный SQL запрос:

```sql
-- Получить историю возвратов заказа
SELECT provider_refund_id, amount, status, created_at
FROM refund
WHERE payment_operation_id = $1
ORDER BY created_at DESC;
```

План выполнения без индексов:

```
Sort  (cost=466.95..466.95 rows=1 width=36) (actual time=7.627..7.630 rows=0 loops=1)
  Sort Key: created_at DESC
  Sort Method: quicksort  Memory: 25kB
  ->  Seq Scan on refund  (cost=0.00..466.94 rows=1 width=36) (actual time=7.613..7.614
rows=0 loops=1)
        Filter: (payment_operation_id = '5109bc19-175d-441a-9da7-c2ac7e92f708'::uuid)
        Rows Removed by Filter: 18075
Planning Time: 0.580 ms
Execution Time: 7.721 ms
```

План выполнения с индексами:

```
Sort  (cost=8.31..8.32 rows=1 width=36) (actual time=0.582..0.582 rows=0 loops=1)
  Sort Key: created_at DESC
  Sort Method: quicksort  Memory: 25kB
  ->  Index Scan using refund_op_id_idx on refund  (cost=0.29..8.30 rows=1 width=36)
(actual time=0.569..0.569 rows=0 loops=1)
        Index Cond: (payment_operation_id = '8b0102f1-40ca-42a3-8dc0-e57fc1df777c'::uuid)
Planning Time: 0.915 ms
Execution Time: 0.632 ms
```

Максимальный cost параметр уменьшился в 56.1 раз с 466.95 до 8.32.

Время исполнения запроса уменьшилось в 12.2 раз с 7.721 мс до 0.632 мс.

Индекс помогает и существенно ускоряет систему.

5.3. DNS1 - Управление DNS

Примерный SQL запрос:

```sql
-- Список DNS-записей домена
SELECT id, type, name, value, ttl, updated_at
FROM dns_record
WHERE domain_id = $1
ORDER BY created_at;
```

План выполнения без индексов:

```
Sort  (cost=17219.48..17219.48 rows=1 width=56) (actual time=124.010..126.412 rows=0
loops=1)
  Sort Key: created_at
  Sort Method: quicksort  Memory: 25kB
  ->  Gather  (cost=1000.00..17219.47 rows=1 width=56) (actual time=123.999..126.400
rows=0 loops=1)
        Workers Planned: 2
        Workers Launched: 2
        ->  Parallel Seq Scan on dns_record  (cost=0.00..16219.37 rows=1 width=56) (actual
time=114.080..114.081 rows=0 loops=3)
              Filter: (domain_id = '5109bc19-175d-441a-9da7-c2ac7e92f708'::uuid)
              Rows Removed by Filter: 299864
Planning Time: 0.710 ms
Execution Time: 126.532 ms
```

План выполнения с индексами:

```
Sort  (cost=8.45..8.46 rows=1 width=56) (actual time=1.078..1.080 rows=0 loops=1)
  Sort Key: created_at
  Sort Method: quicksort  Memory: 25kB
  ->  Index Scan using dns_record_domain_id_idx on dns_record  (cost=0.42..8.44 rows=1
width=56) (actual time=1.066..1.067 rows=0 loops=1)
        Index Cond: (domain_id = '8b0102f1-40ca-42a3-8dc0-e57fc1df777c'::uuid)
Planning Time: 1.263 ms
Execution Time: 1.135 ms
```

Максимальный cost параметр уменьшился в 2035.4 раз с 17219.48 до 8.46.

Время исполнения запроса уменьшилось в 111.5 раз с 126.532 мс до 1.135 мс.

Индекс помогает и существенно ускоряет систему.

5.4. PR2 - Напоминания о продлении

Примерный SQL запрос:

```sql
-- Рассылки о скором истечении: домены, истекающие в указанное окно
SELECT d.id, d.fqdn, d.expires_at
FROM domain AS d
WHERE d.expires_at >= $1
  AND d.expires_at < $2
ORDER BY d.expires_at;
```

План выполнения без индексов:

```
Gather Merge  (cost=20963.67..22589.42 rows=13934 width=36) (actual time=145.551..150.478
rows=19341 loops=1)
  Workers Planned: 2
  Workers Launched: 2
  ->  Sort  (cost=19963.65..19981.06 rows=6967 width=36) (actual time=138.779..139.308
rows=6447 loops=3)
        Sort Key: expires_at
        Sort Method: quicksort  Memory: 611kB
        Worker 0:  Sort Method: quicksort  Memory: 580kB
        Worker 1:  Sort Method: quicksort  Memory: 596kB
```

```
        ->  Parallel Seq Scan on domain d  (cost=0.00..19518.93 rows=6967 width=36)
(actual time=0.103..135.043 rows=6447 loops=3)
              Filter: ((expires_at >= '2025-10-17 00:00:00+03'::timestamp with time zone)
AND (expires_at < '2025-11-17 00:00:00+03'::timestamp with time zone))
              Rows Removed by Filter: 326883
Planning Time: 0.131 ms
Execution Time: 151.154 ms
```

План выполнения с индексами:

```
Index Scan using domain_expires_at_idx on domain d  (cost=0.42..8.45 rows=1 width=36)
(actual time=0.022..0.022 rows=0 loops=1)
  Index Cond: ((expires_at >= '2025-09-17 00:00:00+03'::timestamp with time zone) AND
(expires_at < '2025-10-17 00:00:00+03'::timestamp with time zone))
Planning Time: 1.213 ms
Execution Time: 0.048 ms
```

Максимальный cost параметр уменьшился в 2673.3 раз с 22589.42 до 8.45.

Время исполнения запроса уменьшилось в 3149 раз с 151.154 мс до 0.048 мс.

Индекс помогает и существенно ускоряет систему.

5.5. LK1/ADM1 - Личный кабинет

Примерный SQL запрос:

```sql
-- Активные сессии пользователя
SELECT id, issued_at, expires_at, ip, user_agent, device, last_access_at
FROM user_session
WHERE user_id = $1
ORDER BY issued_at DESC;
```

План выполнения без индексов:

```
Sort  (cost=3378.01..3378.02 rows=1 width=1170) (actual time=61.844..61.846 rows=0
loops=1)
  Sort Key: issued_at DESC
  Sort Method: quicksort  Memory: 25kB
  ->  Seq Scan on user_session  (cost=0.00..3378.00 rows=1 width=1170) (actual
time=61.832..61.833 rows=0 loops=1)
        Filter: (user_id = '5109bc19-175d-441a-9da7-c2ac7e92f708'::uuid)
        Rows Removed by Filter: 100000
Planning Time: 0.979 ms
Execution Time: 61.907 ms
```

План выполнения с индексами:

```
Sort  (cost=8.45..8.45 rows=1 width=1170) (actual time=0.913..0.914 rows=0 loops=1)
  Sort Key: issued_at DESC
  Sort Method: quicksort  Memory: 25kB
  ->  Index Scan using user_session_user_id_idx on user_session  (cost=0.42..8.44 rows=1
width=1170) (actual time=0.898..0.899 rows=0 loops=1)
        Index Cond: (user_id = '8b0102f1-40ca-42a3-8dc0-e57fc1df777c'::uuid)
Planning Time: 2.060 ms
Execution Time: 0.988 ms
```

Максимальный cost параметр уменьшился в 399.8 раз с 3378.02 до 8.45.

Время исполнения запроса уменьшилось в 62.7 раз с 61.907 мс до 0.988 мс.

Индекс помогает и существенно ускоряет систему.

5.6. LOG1 - Аудит действий

Примерный SQL запрос:

```
-- Аудит по актёру
SELECT at, actor_type, actor_id, action, resource, ip
FROM event
WHERE actor_type = $1
  AND actor_id = $2
ORDER BY at DESC
LIMIT 200;
```

План выполнения без индексов:

```
Limit  (cost=274.01..274.01 rows=1 width=54) (actual time=3.404..3.405 rows=0 loops=1)
  ->  Sort  (cost=274.01..274.01 rows=1 width=54) (actual time=3.403..3.404 rows=0
loops=1)
        Sort Key: at DESC
        Sort Method: quicksort  Memory: 25kB
        ->  Seq Scan on event  (cost=0.00..274.00 rows=1 width=54) (actual
time=3.391..3.391 rows=0 loops=1)
              Filter: (((actor_type)::text = 'user'::text) AND (actor_id =
'5109bc19-175d-441a-9da7-c2ac7e92f708'::uuid))
              Rows Removed by Filter: 10000
Planning Time: 0.677 ms
Execution Time: 3.469 ms
```

План выполнения с индексами:

```
Limit  (cost=8.31..8.32 rows=1 width=54) (actual time=0.039..0.039 rows=0 loops=1)
  ->  Sort  (cost=8.31..8.32 rows=1 width=54) (actual time=0.038..0.039 rows=0 loops=1)
        Sort Key: at DESC
        Sort Method: quicksort  Memory: 25kB
        ->  Index Scan using event_actor_idx on event  (cost=0.29..8.30 rows=1 width=54)
(actual time=0.023..0.024 rows=0 loops=1)
              Index Cond: (((actor_type)::text = 'user'::text) AND (actor_id =
'489bb75e-9327-44e1-a644-0cfeffc787e3'::uuid))
Planning Time: 0.085 ms
Execution Time: 0.104 ms
```

Максимальный cost параметр уменьшился в 32.9 раз с 274.01 до 8.32.

Время исполнения запроса уменьшилось в 33.4 раз с 3.469 мс до 0.104 мс.

Индекс помогает и существенно ускоряет систему.

Примерный SQL запрос:

```
-- Аудит за временной интервал
SELECT at, actor_type, actor_id, action, resource, ip
FROM event
WHERE at >= $1 AND at < $2
ORDER BY at DESC
LIMIT 1000;
```

План выполнения без индексов:

```
Limit  (cost=3602.02..3604.52 rows=1000 width=79) (actual time=9.159..9.240 rows=1000
loops=1)
  ->  Sort  (cost=3602.02..3626.23 rows=9685 width=79) (actual time=9.158..9.190 rows=1000
loops=1)
        Sort Key: at DESC
        Sort Method: top-N heapsort  Memory: 314kB
        ->  Seq Scan on event  (cost=0.00..3071.00 rows=9685 width=79) (actual
```

```
time=0.006..6.383 rows=10000 loops=1)
              Filter: ((at >= '2025-11-16 00:00:00+03'::timestamp with time zone) AND (at
< '2025-12-18 00:00:00+03'::timestamp with time zone))
              Rows Removed by Filter: 90000
Planning Time: 0.284 ms
Execution Time: 9.425 ms
```

План выполнения с индексами:

```
Limit  (cost=2402.36..2404.86 rows=1000 width=78) (actual time=4.807..4.889 rows=1000
loops=1)
  ->  Sort  (cost=2402.36..2429.07 rows=10685 width=78) (actual time=4.806..4.839
rows=1000 loops=1)
        Sort Key: at DESC
        Sort Method: top-N heapsort  Memory: 314kB
        ->  Bitmap Heap Scan on event  (cost=14.74..1816.51 rows=10685 width=78) (actual
time=0.221..2.192 rows=10000 loops=1)
              Recheck Cond: ((at >= '2025-11-16 00:00:00+03'::timestamp with time zone)
AND (at < '2025-12-18 00:00:00+03'::timestamp with time zone))
              Rows Removed by Index Recheck: 6305
              Heap Blocks: lossy=256
              ->  Bitmap Index Scan on event_at_idx  (cost=0.00..12.06 rows=15385 width=0)
(actual time=0.162..0.163 rows=2560 loops=1)
                    Index Cond: ((at >= '2025-11-16 00:00:00+03'::timestamp with time
zone) AND (at < '2025-12-18 00:00:00+03'::timestamp with time zone))
Planning Time: 1.605 ms
Execution Time: 5.148 ms
```

Максимальный cost параметр уменьшился в 1.5 раз с 3604.52 до 2404.86.

Время исполнения запроса уменьшилось в 1.8 раз с 9.425 мс до 5.148 мс.

Индекс помогает и ускоряет систему.

5.7. Использования во функциях

Примерный SQL запрос:

```sql
-- домены, срок которых истёк
SELECT id, fqdn, expires_at
FROM domain
WHERE expires_at < now()
ORDER BY expires_at ASC
LIMIT 1000;
```

План выполнения без индексов:

```
Limit  (cost=20520.05..20529.62 rows=82 width=36) (actual time=43.427..45.906 rows=0
loops=1)
  ->  Gather Merge  (cost=20520.05..20529.62 rows=82 width=36) (actual time=43.416..45.895
rows=0 loops=1)
        Workers Planned: 2
        Workers Launched: 2
        ->  Sort  (cost=19520.03..19520.13 rows=41 width=36) (actual time=37.791..37.792
rows=0 loops=3)
              Sort Key: expires_at
              Sort Method: quicksort  Memory: 25kB
              Worker 0:  Sort Method: quicksort  Memory: 25kB
              Worker 1:  Sort Method: quicksort  Memory: 25kB
              ->  Parallel Seq Scan on domain  (cost=0.00..19518.93 rows=41 width=36)
(actual time=37.508..37.508 rows=0 loops=3)
                    Filter: (expires_at < now())
```

```
                    Rows Removed by Filter: 333330
Planning Time: 0.531 ms
Execution Time: 45.978 ms
```

План выполнения с индексами:

```
Limit  (cost=0.43..8.44 rows=1 width=36) (actual time=0.020..0.021 rows=0 loops=1)
  ->  Index Scan using domain_expires_at_idx on domain  (cost=0.43..8.44 rows=1 width=36)
(actual time=0.019..0.019 rows=0 loops=1)
        Index Cond: (expires_at < now())
Planning Time: 0.147 ms
Execution Time: 0.050 ms
```

Максимальный cost параметр уменьшился в 2432.4 раз с 20529.62 до 8.44.

Время исполнения запроса уменьшилось в 919.6 раз с 45.978 мс до 0.05 мс.

Индекс помогает и существенно ускоряет систему.

Примерный SQL запрос:

```sql
-- очистка просроченных сессий
DELETE FROM user_session
WHERE expires_at < now();
```

План выполнения без индексов:

```
Delete on user_session  (cost=0.00..3881.67 rows=0 width=0) (actual time=12.467..12.468
rows=0 loops=1)
  ->  Seq Scan on user_session  (cost=0.00..3881.67 rows=26 width=6) (actual
time=0.624..10.381 rows=7031 loops=1)
        Filter: (expires_at < now())
        Rows Removed by Filter: 92947
Planning Time: 0.123 ms
Execution Time: 12.484 ms
```

План выполнения с индексами:

```
Delete on user_session  (cost=24.13..1853.39 rows=0 width=0) (actual time=3.552..3.553
rows=0 loops=1)
  ->  Bitmap Heap Scan on user_session  (cost=24.13..1853.39 rows=1011 width=6) (actual
time=0.348..1.195 rows=7018 loops=1)
        Recheck Cond: (expires_at < now())
        Heap Blocks: exact=192
        ->  Bitmap Index Scan on user_session_expires_at_idx  (cost=0.00..23.88 rows=1011
width=0) (actual time=0.326..0.327 rows=7043 loops=1)
              Index Cond: (expires_at < now())
Planning Time: 0.121 ms
Execution Time: 3.600 ms
```

Максимальный cost параметр уменьшился в 2.1 раз с 3881.67 до 1853.39.

Время исполнения запроса уменьшилось в 3.5 раз с 12.484 мс до 3.6 мс.

Индекс помогает и ускоряет систему.

6. Функции

6.1. Удалить все просроченные заказы

Функция возвращает количество удалённых записей заказов.

```sql
CREATE OR REPLACE FUNCTION maintenance_delete_stale_orders()
RETURNS BIGINT
LANGUAGE plpgsql
```

```
AS $$
DECLARE
  v_deleted bigint;
BEGIN
  DELETE FROM app_order o
  WHERE o.paid_at IS NULL
    AND now() - o.created_at >= interval '10 minutes'
    AND o.status in ('created','pending_payment')
  returning COUNT(*) INTO v_deleted;

  RETURN coalesce(v_deleted, 0);
END;
$$;
```

6.2. Удалить все просроченные email_tokens

Функция возвращает количество удалённых email_tokens записей.

```
CREATE OR REPLACE FUNCTION maintenance_delete_expired_email_tokens()
RETURNS bigint
LANGUAGE plpgsql
AS $$
DECLARE
  v_deleted bigint;
BEGIN
  DELETE FROM email_validation_token t
  WHERE t.expires_at < now()
  RETURNING count(*) INTO v_deleted;

  RETURN coalesce(v_deleted, 0);
END;
$$;
```

6.3. Административный отчёт за месяц (пользователи, домены, сумма оплат)

Функция возвращает таблицу с колонками:

- Начало месяца
- Конец месяца
- Кол-во созданных пользователей
- Кол-во созданных доменов
- Кол-во оплаченных заказов
- Сумма оплаченных заказов
- Дата первого платежа
- Дата последнего платежа

```
CREATE OR REPLACE FUNCTION
 admin_monthly_report(p_month date default date_trunc('month', now())::date)
RETURNS table (
  month_start date,
  month_end date,
  users_created bigint,
  domains_created bigint,
  paid_orders_count bigint,
  paid_amount bigint,
  first_paid_at timestamptz,
  last_paid_at timestamptz
)
LANGUAGE plpgsql
```

```
AS $$
DECLARE
  v_start timestamp := date_trunc('month', p_month);
  v_end   timestamp := (date_trunc('month', p_month) + interval '1 month');
BEGIN
  RETURN query
  with users_cte AS (
    select count(*)::bigint cnt
    from app_user
    WHERE created_at >= v_start and created_at < v_end
  ),
  domains_cte AS (
    select count(*)::bigint cnt
    from domain
    WHERE created_at >= v_start and created_at < v_end
  ),
  paid_orders AS (
    select o.id, o.total_amount, o.paid_at
    from app_order o
    WHERE o.paid_at is not null
      and o.paid_at >= v_start
      and o.paid_at <  v_end
      and o.status = 'paid'
  )
  select
    v_start::date                             AS month_start,
    (v_end - interval '1 day')::date          AS month_end,
    (select cnt from users_cte)               AS users_created,
    (select cnt from domains_cte)             AS domains_created,
    count(po.id)::bigint                      AS paid_orders_count,
    coalesce(sum(po.total_amount),0)::bigint AS paid_amount,
    min(po.paid_at)                           AS first_paid_at,
    max(po.paid_at)                           AS last_paid_at
  from paid_orders po;
END;
$$;
```

6.4. Формирование списка всех просроченных доменов

Функция возвращает таблицу с колонками:

- Идентификатор домена
- Полное имя домена
- Дата просрочки домена
- Дата создания домена
- Дата активации домена

```
CREATE OR REPLACE FUNCTION
  get_expired_domains()
RETURNS table (
  domain_id uuid,
  fqdn text,
  created_at timestamptz,
  activated_at timestamptz,
  expires_at timestamptz
)
LANGUAGE plpgsql
AS $$
BEGIN
  RETURN query
```

```
  select d.id, d.fqdn, d.expires_at, d.activated_at, d.created_at
  from domain d
  WHERE d.expires_at < now()
  order by d.expires_at asc;
END;
$$;
```