МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Национальный исследовательский университет ИТМО»

ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И
КОМПЬЮТЕРНОЙ ТЕХНИКИ

# Создание информационной системы для генерации дизайнов футболок «Тиишка»

по дисциплине

«Информационные системы»

Курсовая работа, этап №2

Вариант № 49858

***Выполнили:***

Студенты группы P3307

Чусовлянов Максим Сергеевич

Мартышов Данила Викторович

Санкт-Петербург, 2025

# Содержание

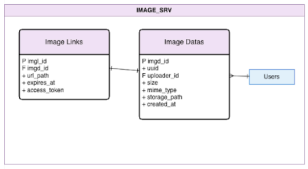
Задание 3
ER-модель 4
**писание программы** **5**
Таблица трассировки 6
Заключение 9

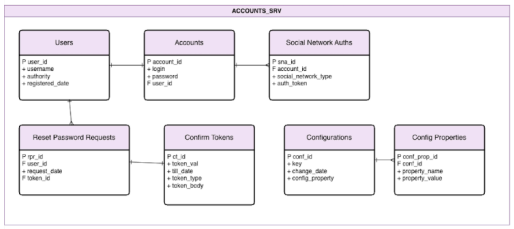
# Задание

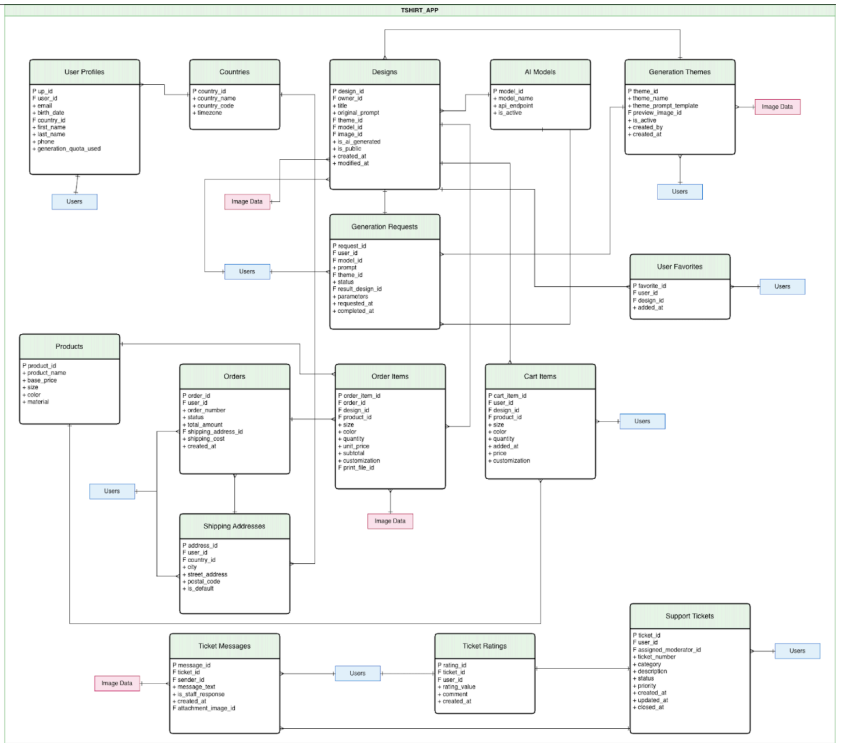
### Этап 2:

1. Сформировать ER-модель базы данных (на основе описаний предметной области и прецедентов из предыдущего этапа).
   ER-модель должна:
   a. включать в себя не менее 10 сущностей;
   b. содержать хотя бы одно отношение вида «многие-ко-многим».
2. Согласовать ER-модель с преподавателем. На основе ER-модели построить даталогическую модель.
3. Реализовать даталогическую модель в реляционной СУБД PostgreSQL.
4. Обеспечить целостность данных при помощи средств языка DDL и триггеров.
5. Реализовать скрипты для создания, удаления базы данных, заполнения базы тестовыми данными.
6. Предложить pl/pgsql-функции и процедуры, для выполнения критически важных запросов (которые потребуются при последующей реализации прецедентов).
7. Создать индексы на основе анализа использования базы данных в контексте описанных на первом этапе прецедентов. Обосновать полезность созданных индексов для реализации представленных на первом
   бизнес-процессов.
8. Составить отчет.

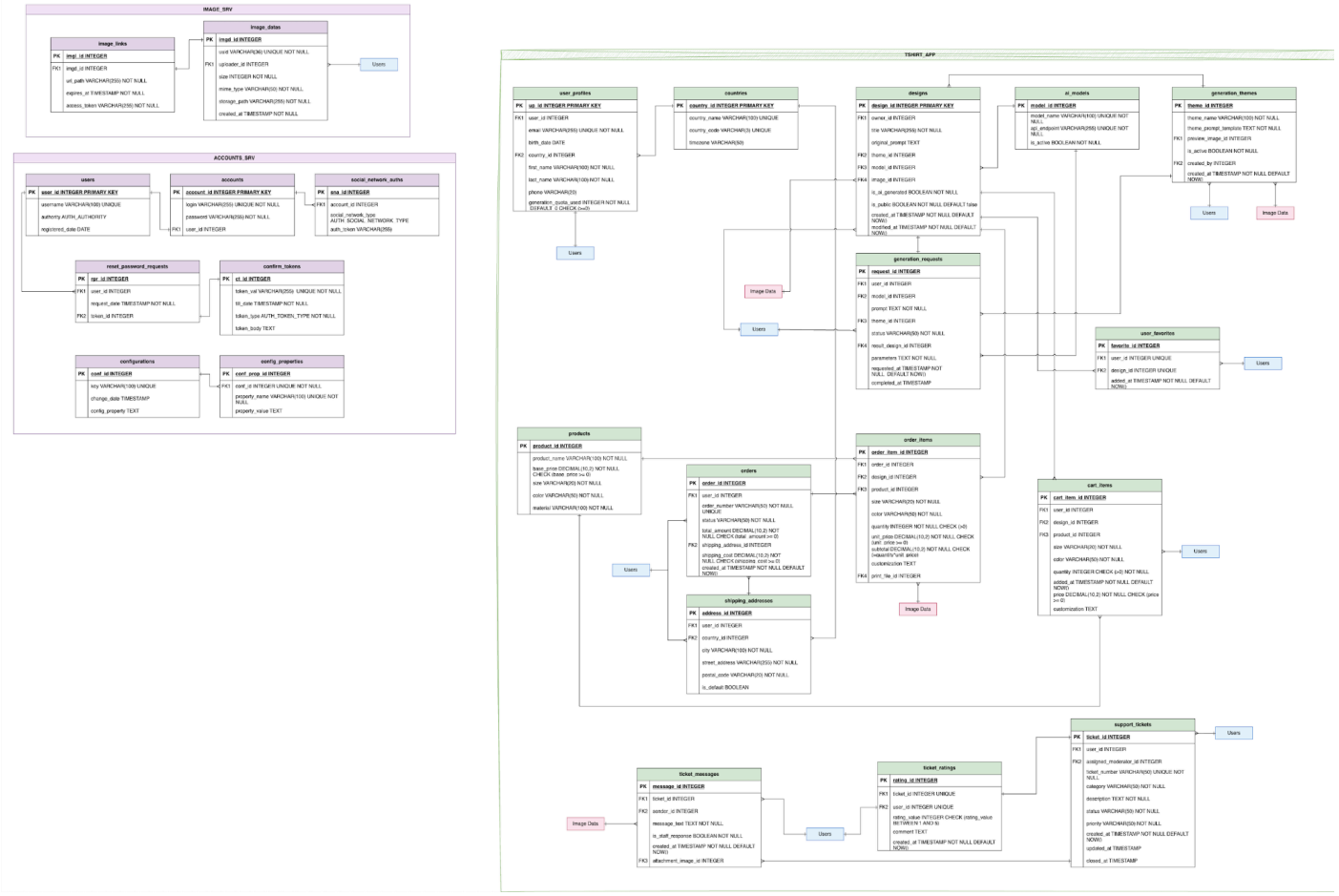
# ER-модель


IMAGE_SRV
Image Links
Image Datas
Users


ACCOUNTS_SRV
Users
Accounts
Social Network Auths
Reset Password Requests
Confirm Tokens
Configurations
Config Properties


TSHIRT_APP
User Profiles
Countries
Designs
AI Models
Generation Themes
Image Data
Users
Generation Requests
User Favorites
Products
Orders
Order Items
Cart Items
Shipping Addresses
Support Tickets
Ticket Messages
Ticket Ratings

# Даталогическая модель



# Реализация даталогической модели в реляционной СУБД PostgreSQL

База данных разделена на 4 микросервиса: accounts_srv, image_srv, config_srv, tiishka_backend:

```
CREATE TYPE AUTH_AUTHORITY AS ENUM ('ADMIN', 'USER', 'MODERATOR');

CREATE TYPE AUTH_SOCIAL_NETWORK_TYPE AS ENUM ('TELEGRAM', 'VK', 'GOOGLE', 'YANDEX');

CREATE TYPE AUTH_TOKEN_TYPE AS ENUM ('RESET_PASSWORD', 'CONFIRM_ACCOUNT', 'OTHER');

CREATE TABLE users (
    user_id SERIAL PRIMARY KEY,
    username VARCHAR(100) UNIQUE,
    authority AUTH_AUTHORITY NOT NULL DEFAULT 'USER',
    registered_date DATE NOT NULL DEFAULT CURRENT_DATE
);

CREATE TABLE accounts (
    account_id SERIAL PRIMARY KEY,
    login VARCHAR(255) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    user_id INTEGER,
    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE
);

CREATE TABLE social_network_auths (
    sna_id SERIAL PRIMARY KEY,
    account_id INTEGER,
    social_network_type AUTH_SOCIAL_NETWORK_TYPE NOT NULL,
    auth_token VARCHAR(255) NOT NULL,
    FOREIGN KEY (account_id) REFERENCES accounts(account_id) ON DELETE CASCADE
);

CREATE TABLE confirm_tokens (
    ct_id SERIAL PRIMARY KEY,
    token_val VARCHAR(255) NOT NULL UNIQUE,
    till_date TIMESTAMP NOT NULL,
    token_type AUTH_TOKEN_TYPE NOT NULL,
```

```
    token_body TEXT
);
CREATE TABLE reset_password_requests (
    rpr_id SERIAL PRIMARY KEY,
    user_id INTEGER,
    request_date TIMESTAMP NOT NULL DEFAULT NOW(),
    token_id INTEGER,
    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE,
    FOREIGN KEY (token_id) REFERENCES confirm_tokens(ct_id) ON DELETE CASCADE
);
```

image_srv:

```
CREATE TABLE image_datas (
    imgd_id SERIAL PRIMARY KEY,
    uuid VARCHAR(36) NOT NULL UNIQUE,
    uploader_id INTEGER,
    size INTEGER NOT NULL,
    mime_type VARCHAR(50) NOT NULL,
    storage_path VARCHAR(255) NOT NULL,
    created_at TIMESTAMP NOT NULL DEFAULT NOW(),
    FOREIGN KEY (uploader_id) REFERENCES users(user_id) ON DELETE SET NULL
);
CREATE TABLE image_links (
    imgl_id SERIAL PRIMARY KEY,
    imgd_id INTEGER,
    url_path VARCHAR(255) NOT NULL,
    expires_at TIMESTAMP NOT NULL,
    access_token VARCHAR(255) NOT NULL,
    FOREIGN KEY (imgd_id) REFERENCES image_datas(imgd_id) ON DELETE CASCADE
);
```

config_srv:

```
CREATE TABLE configurations (
    conf_id SERIAL PRIMARY KEY,
    key VARCHAR(100) UNIQUE,
```

```
    change_date TIMESTAMP NOT NULL DEFAULT NOW(),
    config_property TEXT
);
CREATE TABLE config_properties (
    conf_prop_id SERIAL PRIMARY KEY,
    conf_id INTEGER NOT NULL UNIQUE,
    property_name VARCHAR(100) NOT NULL UNIQUE,
    property_value TEXT,
    FOREIGN KEY (conf_id) REFERENCES configurations(conf_id) ON DELETE CASCADE
);
```

tiishka_backend :

```
CREATE TYPE generation_status AS ENUM ('PENDING', 'PROCESSING', 'COMPLETED', 'DECLINED');
CREATE TYPE order_status AS ENUM ('PENDING', 'PAID', 'PROCESSING', 'SHIPPED', 'DELIVERED',
'CANCELLED', 'REFUNDED');
CREATE TYPE ticket_priority AS ENUM ('LOW', 'MEDIUM', 'HIGH', 'URGENT');
CREATE TYPE ticket_category AS ENUM ('TECHNICAL', 'BILLING', 'ACCOUNT', 'OTHER');
CREATE TYPE ticket_status AS ENUM ('OPEN', 'IN_PROGRESS', 'WAITING_ON_USER', 'RESOLVED',
'CLOSED', 'AUTO_CLOSED');
CREATE TABLE countries (
    country_id SERIAL PRIMARY KEY,
    country_name VARCHAR(100) UNIQUE,
    country_code VARCHAR(3) UNIQUE,
    timezone VARCHAR(50)
);
CREATE TABLE products (
    product_id SERIAL PRIMARY KEY,
    product_name VARCHAR(100) NOT NULL,
    base_price DECIMAL(10, 2) NOT NULL CHECK (base_price >= 0),
    size VARCHAR(20) NOT NULL,
    color VARCHAR(50) NOT NULL,
    material VARCHAR(100) NOT NULL
);
CREATE TABLE user_profiles (
```

```
    up_id SERIAL PRIMARY KEY,
    user_id INTEGER,
    email VARCHAR(255) NOT NULL UNIQUE,
    birth_date DATE,
    country_id INTEGER,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    phone VARCHAR(20),
    generation_quota_used INTEGER NOT NULL DEFAULT 0 CHECK (generation_quota_used >= 0),
    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE,
    FOREIGN KEY (country_id) REFERENCES countries(country_id) ON DELETE SET NULL
);
CREATE TABLE designs (
    design_id SERIAL PRIMARY KEY,
    owner_id INTEGER,
    title VARCHAR(255) NOT NULL,
    original_prompt TEXT,
    theme_id INTEGER,
    model_id INTEGER,
    image_id INTEGER,
    is_ai_generated BOOLEAN NOT NULL,
    is_public BOOLEAN NOT NULL DEFAULT false,
    created_at TIMESTAMP NOT NULL DEFAULT NOW(),
    modified_at TIMESTAMP NOT NULL DEFAULT NOW(),
    FOREIGN KEY (owner_id) REFERENCES users(user_id) ON DELETE CASCADE
);
CREATE TABLE user_favorites (
    favorite_id SERIAL PRIMARY KEY,
    user_id INTEGER UNIQUE,
    design_id INTEGER UNIQUE,
    added_at TIMESTAMP NOT NULL DEFAULT NOW(),
    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE,
    FOREIGN KEY (design_id) REFERENCES designs(design_id) ON DELETE CASCADE
```

```
);

CREATE TABLE orders (
    order_id SERIAL PRIMARY KEY,
    user_id INTEGER,
    order_number VARCHAR(50) NOT NULL UNIQUE,
    status order_status NOT NULL,
    total_amount DECIMAL(10, 2) NOT NULL CHECK (total_amount >= 0),
    shipping_address_id INTEGER,
    shipping_cost DECIMAL(10, 2) NOT NULL CHECK (shipping_cost >= 0),
    created_at TIMESTAMP NOT NULL DEFAULT NOW()
);

CREATE TABLE support_tickets (
    ticket_id SERIAL PRIMARY KEY,
    user_id INTEGER,
    assigned_moderator_id INTEGER,
    ticket_number VARCHAR(50) NOT NULL UNIQUE,
    category ticket_category NOT NULL,
    description TEXT NOT NULL,
    status ticket_status NOT NULL,
    priority ticket_priority NOT NULL,
    created_at TIMESTAMP NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMP,
    closed_at TIMESTAMP,
    CHECK (closed_at IS NULL OR closed_at >= created_at)
);
```

# Триггеры

```
-- Очистка просроченных токенов подтверждения (accounts_srv)
CREATE OR REPLACE FUNCTION cleanup_expired_tokens()
RETURNS TRIGGER AS $$
BEGIN
   DELETE FROM confirm_tokens WHERE till_date < NOW();
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_cleanup_expired_tokens
   AFTER INSERT ON confirm_tokens
   FOR EACH ROW
   EXECUTE FUNCTION cleanup_expired_tokens();

-- Автообновление modified_at при изменении дизайна (tiishka_backend)
CREATE OR REPLACE FUNCTION update_design_modified_at()
RETURNS TRIGGER AS $$
BEGIN
   NEW.modified_at = NOW();
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_design_modified_at
   BEFORE UPDATE ON designs
   FOR EACH ROW
   EXECUTE FUNCTION update_design_modified_at();

-- Автообновление updated_at и closed_at для тикетов (tiishka_backend)
CREATE OR REPLACE FUNCTION update_ticket_updated_at()
RETURNS TRIGGER AS $$
BEGIN
   NEW.updated_at = NOW();
   IF NEW.status IN ('RESOLVED', 'CLOSED', 'AUTO_CLOSED')
      AND (OLD.status IS NULL OR OLD.status NOT IN ('RESOLVED', 'CLOSED', 'AUTO_CLOSED')) THEN
       NEW.closed_at = NOW();
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_ticket_updated_at
   BEFORE UPDATE ON support_tickets
   FOR EACH ROW
   EXECUTE FUNCTION update_ticket_updated_at();
```

# Скрипты для создания, удаления БД, заполнения базы тестовыми данными

Скрипт для создания БД (create_all.sql):

```
\i accounts_srv/create.sql

\i accounts_srv/triggers.sql

\i image_srv/create.sql

\i config_srv/create.sql

\i tiishka_backend/create.sql

\i tiishka_backend/triggers.sql

\i tiishka_backend/indexes.sql

\i tiishka_backend/functions.sql
```

Скрипт для удаления БД (delete_all.sql):

```
\i tiishka_backend/delete.sql

\i config_srv/delete.sql

\i image_srv/delete.sql

\i accounts_srv/delete.sql
```

Скрипт для заполнения тестовыми данными (seed.sql):

```
INSERT INTO countries (country_name, country_code, timezone) VALUES

('Россия', 'RU', 'Europe/Moscow'),

('Беларусь', 'BY', 'Europe/Minsk'),

('Казахстан', 'KZ', 'Asia/Almaty');

INSERT INTO ai_models (model_name, api_endpoint, is_active) VALUES

('DALL-E 3', 'https://api.openai.com/v1/images/generations', true),

('Stable Diffusion XL', 'https://api.stability.ai/v1/generation', true);

INSERT INTO products (product_name, base_price, size, color, material) VALUES

('Футболка базовая', 1500.00, 'M', 'Белый', 'Хлопок 100%'),

('Футболка базовая', 1500.00, 'L', 'Черный', 'Хлопок 100%'),

('Худи', 3500.00, 'L', 'Серый', 'Хлопок/полиэстер');

INSERT INTO users (username, authority) VALUES

('admin', 'ADMIN'),

('сергей_бессмертный_забрал_кофе', 'MODERATOR'),
```

```
('втшник_страдалец', 'USER');

INSERT INTO accounts (login, password, user_id) VALUES

('admin@tiishka.ru', '$2a$10$hashed', 1),

('s.bessmertny@itmo.ru', '$2a$10$hashed', 2),

('student@niuitmo.ru', '$2a$10$hashed', 3);

INSERT INTO user_profiles (user_id, email, first_name, last_name, country_id,
generation_quota_used) VALUES

(1, 'admin@tiishka.ru', 'Админ', 'Системы', 1, 0),

(2, 's.bessmertny@itmo.ru', 'Сергей', 'Бессмертный', 1, 0),

(3, 'student@niuitmo.ru', 'Ваня', 'Курсач', 1, 9);
```

# PL/pgSQL-функции и процедуры для выполнения критически важных запросов

```
-- Атомарная регистрация пользователя
CREATE OR REPLACE FUNCTION register_user(
    p_username VARCHAR(100),
    p_login VARCHAR(255),
    p_password VARCHAR(255),
    p_email VARCHAR(255),
    p_first_name VARCHAR(100),
    p_last_name VARCHAR(100)
) RETURNS INTEGER AS $$
DECLARE
    v_user_id INTEGER;
BEGIN
    INSERT INTO users (username, authority)
    VALUES (p_username, 'USER')
    RETURNING user_id INTO v_user_id;

    INSERT INTO accounts (login, password, user_id)
    VALUES (p_login, p_password, v_user_id);

    INSERT INTO user_profiles (user_id, email, first_name, last_name)
    VALUES (v_user_id, p_email, p_first_name, p_last_name);

    RETURN v_user_id;
END;
$$ LANGUAGE plpgsql;

-- Атомарное завершение генерации
CREATE OR REPLACE FUNCTION complete_generation(
    p_request_id INTEGER,
    p_image_id INTEGER,
    p_title VARCHAR(255) DEFAULT ''
) RETURNS INTEGER AS $$
DECLARE
    v_design_id INTEGER;
    v_user_id INTEGER;
    v_theme_id INTEGER;
    v_model_id INTEGER;
    v_prompt TEXT;
BEGIN
    SELECT user_id, theme_id, model_id, prompt
    INTO v_user_id, v_theme_id, v_model_id, v_prompt
    FROM generation_requests
    WHERE request_id = p_request_id;

    INSERT INTO designs (owner_id, title, original_prompt, theme_id, model_id, image_id,
```

```
is_ai_generated, is_public)
    VALUES (v_user_id, p_title, v_prompt, v_theme_id, v_model_id, p_image_id, true, false)
    RETURNING design_id INTO v_design_id;

    UPDATE generation_requests
    SET status = 'COMPLETED', result_design_id = v_design_id, completed_at = NOW()
    WHERE request_id = p_request_id;

    RETURN v_design_id;
END;
$$ LANGUAGE plpgsql;

-- Расчёт стоимости корзины
CREATE OR REPLACE FUNCTION calculate_cart_total(p_user_id INTEGER)
RETURNS DECIMAL(10,2) AS $$
DECLARE
    v_total DECIMAL(10,2);
BEGIN
    SELECT COALESCE(SUM(quantity * price), 0)
    INTO v_total
    FROM cart_items
    WHERE user_id = p_user_id;

    RETURN v_total;
END;
$$ LANGUAGE plpgsql;

-- Атомарное оформление заказа
CREATE OR REPLACE FUNCTION checkout_cart(
    p_user_id INTEGER,
    p_shipping_address_id INTEGER,
    p_shipping_cost DECIMAL(10,2) DEFAULT 0
) RETURNS INTEGER AS $$
DECLARE
    v_order_id INTEGER;
    v_total DECIMAL(10,2);
    v_cart_item RECORD;
    v_order_number VARCHAR(50);
BEGIN
    v_total := calculate_cart_total(p_user_id);

    IF v_total = 0 THEN
        RAISE EXCEPTION 'Cart is empty';
    END IF;

    v_order_number := 'ORD-' || TO_CHAR(NOW(), 'YYYYMMDD') || '-' ||
                      LPAD(nextval('orders_order_id_seq')::TEXT, 6, '0');

    INSERT INTO orders (user_id, order_number, status, total_amount, shipping_address_id,
shipping_cost)
```

```
    VALUES (p_user_id, v_order_number, 'PENDING', v_total + p_shipping_cost,
p_shipping_address_id, p_shipping_cost)
    RETURNING order_id INTO v_order_id;

    FOR v_cart_item IN
        SELECT design_id, product_id, size, color, quantity, price, customization
        FROM cart_items
        WHERE user_id = p_user_id
    LOOP
        INSERT INTO order_items (order_id, design_id, product_id, size, color, quantity,
unit_price, subtotal, customization)
        VALUES (v_order_id, v_cart_item.design_id, v_cart_item.product_id, v_cart_item.size,
                v_cart_item.color, v_cart_item.quantity, v_cart_item.price,
                v_cart_item.quantity * v_cart_item.price, v_cart_item.customization);
    END LOOP;

    DELETE FROM cart_items WHERE user_id = p_user_id;

    RETURN v_order_id;
END;
$$ LANGUAGE plpgsql;

-- Проверка популярности дизайна
CREATE OR REPLACE FUNCTION is_popular_design(p_design_id INTEGER)
RETURNS BOOLEAN AS $$
DECLARE
    v_favorites_count INTEGER;
    v_orders_count INTEGER;
BEGIN
    SELECT COUNT(*) INTO v_favorites_count
    FROM user_favorites
    WHERE design_id = p_design_id;

    SELECT COUNT(*) INTO v_orders_count
    FROM order_items
    WHERE design_id = p_design_id;

    IF v_favorites_count >= 5 OR v_orders_count >= 3 THEN
        RETURN TRUE;
    END IF;

    RETURN FALSE;
END;
$$ LANGUAGE plpgsql;

-- Получение популярных дизайнов
CREATE OR REPLACE FUNCTION get_popular_designs(p_limit INTEGER DEFAULT 10)
RETURNS TABLE(
    design_id INTEGER,
    title VARCHAR(255),
```

```
    favorites_count BIGINT,
    orders_count BIGINT
) AS $$
BEGIN
    RETURN QUERY
    SELECT
        d.design_id,
        d.title,
        COUNT(DISTINCT uf.favorite_id) as favorites_count,
        COUNT(DISTINCT oi.order_item_id) as orders_count
    FROM designs d
    LEFT JOIN user_favorites uf ON d.design_id = uf.design_id
    LEFT JOIN order_items oi ON d.design_id = oi.design_id
    WHERE d.is_public = true
    GROUP BY d.design_id, d.title
    ORDER BY (COUNT(DISTINCT uf.favorite_id) + COUNT(DISTINCT oi.order_item_id)) DESC
    LIMIT p_limit;
END;
$$ LANGUAGE plpgsql;
```

# Индексы

```sql
-- B-tree индексы для поиска по FK
CREATE INDEX idx_user_profiles_user_id ON user_profiles(user_id);
CREATE INDEX idx_generation_requests_user_id ON generation_requests(user_id);
CREATE INDEX idx_designs_owner_id ON designs(owner_id);
CREATE INDEX idx_cart_items_user_id ON cart_items(user_id);
CREATE INDEX idx_orders_user_id ON orders(user_id);
CREATE INDEX idx_order_items_order_id ON order_items(order_id);
CREATE INDEX idx_support_tickets_user_id ON support_tickets(user_id);
CREATE INDEX idx_ticket_messages_ticket_id ON ticket_messages(ticket_id);

-- Partial B-tree индексы
CREATE INDEX idx_generation_requests_status ON generation_requests(status)
   WHERE status IN ('PENDING', 'PROCESSING');
CREATE INDEX idx_designs_public ON designs(is_public) WHERE is_public = true;

-- GIN индексы для полнотекстового поиска
CREATE INDEX idx_designs_title_gin ON designs USING GIN(to_tsvector('russian', title));
CREATE INDEX idx_designs_prompt_gin ON designs USING GIN(to_tsvector('russian',
original_prompt))
   WHERE original_prompt IS NOT NULL;
```

| **Индекс** | **Тип** | **Прецедент** | **Обоснование** |
|---|---|---|---|
| idx_user_profiles_user_id | B-tree | Авторизация | Поиск профиля при каждом запросе |
| idx_generation_requests_user_id | B-tree | История генераций | Запросы пользователя |
| idx_generation_requests_status | B-tree (partial) | Обработка генерации | Очередь для фонового воркера |
| idx_designs_public | B-tree (partial) | Галерея | Фильтрация публичных дизайнов |
| idx_designs_owner_id | B-tree | Библиотека | Дизайны пользователя |
| idx_cart_items_user_id | B-tree | Корзина | Содержимое корзины |
| idx_orders_user_id | B-tree | История заказов | Заказы пользователя |
| idx_order_items_order_id | B-tree | Детали заказа | Позиции заказа |
| idx_support_tickets_user_id | B-tree | Техподдержка | Тикеты пользователя |
| idx_ticket_messages_ticket_id | B-tree | Переписка | Сообщения тикета |
| idx_designs_title_gin | GIN | Поиск дизайнов | Полнотекстовый поиск по названию |
| idx_designs_prompt_gin | GIN | Поиск дизайнов | Полнотекстовый поиск по промпту |

# Вывод

В результате выполнения второго этапа курсовой работы, была разработана реляционная база данных на основе предварительно сформулированной ER- и даталогической модели базы данных для информационной системы «Тиишка». При реализации функций и создании индексов в реляционной БД PostgreSQL была несколько раз пересмотрена даталогическая модель, добавлены недостающие ограничения целостности, CHECK constraints для валидации данных, а также реализованы триггеры для автоматического поддержания консистентности. Реализованная система может быть переработана в дальнейшем с учетом новых требований.