МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«Национальный исследовательский университет ИТМО»

ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ

**КУРСОВАЯ РАБОТА Ч2**

по дисциплине

«ИНФОРМАЦИОННЫЕ СИСТЕМЫ»

***Выполнил:***

Студент группы P3318

Рамеев Тимур

Ильгизович

***Преподаватель:***

Николаев Владимир

Вячеславович

Оглавление

[Задание 3](#_Toc191921010)

[ER-модель 3](#_Toc191921011)

[Даталогическая модель 3](#_Toc191921012)

[Реализация датологической модели в PostgeSQL 3](#_Toc191921013)

[Функции/процедуры и триггеры 3](#_Toc191921014)

[Скрипты 3](#_Toc191921015)

[Добавление индексов 3](#_Toc191921016)

# Задание

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

# ER-модель

Изображение выглядит как снимок экрана, дизайн

Контент, сгенерированный ИИ, может содержать ошибки.

# Даталогическая модель

Изображение выглядит как снимок экрана, дизайн

Контент, сгенерированный ИИ, может содержать ошибки.

# Реализация датологической модели в PostgeSQL

## Создание отношений

|  |
| --- |
| CREATE TABLE users (      id SERIAL PRIMARY KEY,      name TEXT NOT NULL,      surname TEXT NOT NULL,      email TEXT NOT NULL UNIQUE,      password TEXT NOT NULL,      role TEXT NOT NULL CHECK (role IN ('medical\_employee', 'bank\_employee', 'admin'))  );  CREATE TABLE medical\_institutions (      id SERIAL PRIMARY KEY,      name TEXT NOT NULL,      address TEXT NOT NULL  );  CREATE TABLE medical\_employees (      user\_id INTEGER PRIMARY KEY REFERENCES users(id) ON DELETE CASCADE,      institution\_id INTEGER REFERENCES medical\_institutions(id) ON DELETE CASCADE,      post TEXT NOT NULL  );  CREATE TABLE blood\_banks (      id SERIAL PRIMARY KEY,      name TEXT NOT NULL,      address TEXT NOT NULL,      work\_time TEXT NOT NULL  );  CREATE TABLE bank\_employees (      user\_id INTEGER PRIMARY KEY REFERENCES users(id) ON DELETE CASCADE,      bank\_id INTEGER REFERENCES blood\_banks(id) ON DELETE CASCADE,      post TEXT NOT NULL  );  CREATE TABLE blood\_requests (      id SERIAL PRIMARY KEY,      blood\_type TEXT CHECK (blood\_type IN ('A', 'B', 'AB', 'O')),      rh\_factor TEXT CHECK (rh\_factor IN ('+', '-')),      status TEXT CHECK (status IN ('pending', 'approved', 'rejected', 'completed')),      quantity INTEGER CHECK (quantity > 0),      priority TEXT CHECK (priority IN ('low', 'medium', 'high')),      created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,      approved\_at TIMESTAMP NULL,      rejected\_at TIMESTAMP NULL,      completed\_at TIMESTAMP NULL,      rejection\_reason TEXT NULL  );  CREATE TABLE blood\_units (      id SERIAL PRIMARY KEY,      bank\_id INTEGER REFERENCES blood\_banks(id) ON DELETE CASCADE,      blood\_type TEXT CHECK (blood\_type IN ('A', 'B', 'AB', 'O')),      rh\_factor TEXT CHECK (rh\_factor IN ('+', '-')),      expiration\_date DATE NOT NULL,      quantity INTEGER CHECK (quantity > 0)  );  CREATE TABLE blood\_inventories (      bank\_id INTEGER REFERENCES blood\_banks(id) ON DELETE CASCADE,      blood\_type TEXT CHECK (blood\_type IN ('A', 'B', 'AB', 'O')),      rh\_factor TEXT CHECK (rh\_factor IN ('+', '-')),      total\_quantity INTEGER CHECK (total\_quantity > 0),      PRIMARY KEY (bank\_id, blood\_type, rh\_factor)  );  CREATE TABLE bank\_request (      bank\_id INTEGER REFERENCES blood\_banks(id) ON DELETE CASCADE,      blood\_request\_id INTEGER REFERENCES blood\_requests(id) ON DELETE CASCADE,      PRIMARY KEY (bank\_id, blood\_request\_id)  );  CREATE TABLE registration\_requests (      id SERIAL PRIMARY KEY,      name TEXT NOT NULL,      surname TEXT NOT NULL,      email TEXT NOT NULL UNIQUE,      password TEXT NOT NULL,      role TEXT CHECK (role IN ('medical\_employee', 'bank\_employee', 'admin')),      institution\_id INTEGER REFERENCES medical\_institutions(id) ON DELETE CASCADE,      bank\_id INTEGER REFERENCES blood\_banks(id) ON DELETE CASCADE,      created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  );  CREATE TABLE change\_status\_notifications (      id SERIAL PRIMARY KEY,      blood\_request\_id INTEGER REFERENCES blood\_requests(id) ON DELETE CASCADE,      medical\_employee\_id INTEGER REFERENCES medical\_employees(user\_id) ON DELETE CASCADE,      is\_read BOOLEAN DEFAULT FALSE,      created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,      new\_status TEXT CHECK (new\_status IN ('approved', 'rejected'))  );  CREATE TABLE expiration\_notifications (      id SERIAL PRIMARY KEY,      blood\_unit\_id INTEGER REFERENCES blood\_units(id) ON DELETE CASCADE,      bank\_employee\_id INTEGER REFERENCES bank\_employees(user\_id) ON DELETE CASCADE,      is\_read BOOLEAN DEFAULT FALSE,      created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  );  CREATE TABLE new\_request\_notifications (      id SERIAL PRIMARY KEY,      blood\_request\_id INTEGER REFERENCES blood\_requests(id) ON DELETE CASCADE,      bank\_employee\_id INTEGER REFERENCES bank\_employees(user\_id) ON DELETE CASCADE,      is\_read BOOLEAN DEFAULT FALSE,      created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  ); |

## Удаление сущностей

|  |
| --- |
| DROP TABLE IF EXISTS  new\_request\_notifications,  expiration\_notifications,  change\_status\_notifications,  bank\_request,  blood\_inventories,  blood\_requests,  blood\_units,  bank\_employees,  medical\_employees,  blood\_banks,  medical\_institutions,  registration\_requests,  users  CASCADE; |

## Генерация данных

|  |
| --- |
| INSERT INTO users (name, surname, email, password, role)  SELECT      'Name' || id,      'Surname' || id,      'user' || id || '@example.com',      'password123',      CASE WHEN id % 3 = 0 THEN 'medical\_employee'           WHEN id % 3 = 1 THEN 'bank\_employee'           ELSE 'admin'      END  FROM generate\_series(1, 1000) AS id;  INSERT INTO medical\_institutions (name, address)  SELECT 'Hospital ' || id, 'Medicinskaya st., ' || (id % 50 + 1)  FROM generate\_series(1, 100) AS id;  INSERT INTO blood\_banks (name, address, work\_time)  SELECT 'Blood bank ' || id, 'Medicinskaya st., ' || (id % 30 + 1), '08:00-18:00'  FROM generate\_series(1, 50) AS id;  INSERT INTO medical\_employees (user\_id, institution\_id, post)  SELECT id, (id % 100) + 1, 'Doctor'  FROM generate\_series(1, 500) AS id;  INSERT INTO bank\_employees (user\_id, bank\_id, post)  SELECT id, (id % 50) + 1, 'Laboratory assistant'  FROM generate\_series(501, 1000) AS id;  INSERT INTO blood\_units (bank\_id, blood\_type, rh\_factor, expiration\_date, quantity)  SELECT      (id % 50) + 1,      CASE WHEN id % 4 = 0 THEN 'A' WHEN id % 4 = 1 THEN 'B' WHEN id % 4 = 2 THEN 'AB' ELSE 'O' END,      CASE WHEN id % 2 = 0 THEN '+' ELSE '-' END,      NOW() - INTERVAL '1 day' \* (id % 365),      (id % 5) + 1  FROM generate\_series(1, 10000) AS id;  INSERT INTO blood\_requests (blood\_type, rh\_factor, status, quantity, priority, created\_at)  SELECT      CASE WHEN id % 4 = 0 THEN 'A' WHEN id % 4 = 1 THEN 'B' WHEN id % 4 = 2 THEN 'AB' ELSE 'O' END,      CASE WHEN id % 2 = 0 THEN '+' ELSE '-' END,      CASE WHEN id % 3 = 0 THEN 'pending' WHEN id % 3 = 1 THEN 'approved' ELSE 'rejected' END,      (id % 3) + 1,      CASE WHEN id % 3 = 0 THEN 'low' WHEN id % 3 = 1 THEN 'medium' ELSE 'high' END,      NOW() - INTERVAL '1 day' \* (id % 30)  FROM generate\_series(1, 5000) AS id;  INSERT INTO registration\_requests (name, surname, email, password, role, institution\_id, created\_at)  SELECT      'Name' || id,      'Surname' || id,      'request' || id || '@example.com',      'password123',      CASE WHEN id % 3 = 0 THEN 'medical\_employee'           WHEN id % 3 = 1 THEN 'bank\_employee'           ELSE 'admin'      END,      (id % 100) + 1,      NOW() - INTERVAL '1 day' \* (id % 30)  FROM generate\_series(1, 200) AS id;  INSERT INTO change\_status\_notifications (blood\_request\_id, medical\_employee\_id, is\_read, created\_at, new\_status)  SELECT      (id % 5000) + 1,      (id % 500) + 1,      FALSE,      NOW() - INTERVAL '1 day' \* (id % 30),      CASE WHEN id % 3 = 0 THEN 'approved' ELSE 'rejected' END  FROM generate\_series(1, 500) AS id;  INSERT INTO expiration\_notifications (blood\_unit\_id, bank\_employee\_id, is\_read, created\_at)  SELECT      (id % 10000) + 1,      (id % 500) + 501,      FALSE,      NOW() - INTERVAL '1 day' \* (id % 30)  FROM generate\_series(1, 500) AS id;  INSERT INTO new\_request\_notifications (blood\_request\_id, bank\_employee\_id, is\_read, created\_at)  SELECT      (id % 5000) + 1,      (id % 500) + 501,      FALSE,      NOW() - INTERVAL '1 day' \* (id % 30)  FROM generate\_series(1, 500) AS id;  INSERT INTO bank\_request (bank\_id, blood\_request\_id)  SELECT      (id % 50) + 1,      (id % 5000) + 1  FROM generate\_series(1, 1000) AS id;  INSERT INTO blood\_inventories (bank\_id, blood\_type, rh\_factor, total\_quantity)  SELECT      bank\_id,      blood\_type,      rh\_factor,      (random() \* 50 + 1)::int -- случайное количество от 1 до 50  FROM (      SELECT DISTINCT ON (bank\_id, blood\_type, rh\_factor)          bank\_id,          blood\_type,          rh\_factor      FROM (          SELECT              (b % 50) + 1 AS bank\_id,              blood\_type,              rh\_factor          FROM generate\_series(1, 50) b          CROSS JOIN (VALUES ('A', '+'), ('A', '-'), ('B', '+'), ('B', '-'), ('AB', '+'), ('AB', '-'), ('O', '+'), ('O', '-')) AS blood\_types(blood\_type, rh\_factor)      ) AS unique\_combinations  ) AS final\_selection; |

# Функции/процедуры и триггеры

## Триггеры

|  |
| --- |
| -- добавление данных в total\_sum для каждого банка при insert нового blood\_item  CREATE OR REPLACE FUNCTION update\_inventory\_on\_insert()  RETURNS TRIGGER AS $$  BEGIN      INSERT INTO blood\_inventories (bank\_id, blood\_type, rh\_factor, total\_quantity)      VALUES (NEW.bank\_id, NEW.blood\_type, NEW.rh\_factor, NEW.quantity)      ON CONFLICT (bank\_id, blood\_type, rh\_factor)      DO UPDATE SET total\_quantity = blood\_inventories.total\_quantity + NEW.quantity;        RETURN NEW;  END;  $$ LANGUAGE plpgsql;  CREATE TRIGGER trigger\_update\_inventory  AFTER INSERT ON blood\_units  FOR EACH ROW  EXECUTE FUNCTION update\_inventory\_on\_insert();  -- тоже самое для удаления  CREATE OR REPLACE FUNCTION update\_inventory\_on\_delete()  RETURNS TRIGGER AS $$  BEGIN      UPDATE blood\_inventories      SET total\_quantity = GREATEST(total\_quantity - OLD.quantity, 0)      WHERE bank\_id = OLD.bank\_id        AND blood\_type = OLD.blood\_type        AND rh\_factor = OLD.rh\_factor;      RETURN OLD;  END;  $$ LANGUAGE plpgsql;  CREATE TRIGGER trigger\_update\_inventory\_on\_delete  AFTER DELETE ON blood\_units  FOR EACH ROW  EXECUTE FUNCTION update\_inventory\_on\_delete();  -- Проверка на наличие крови для запроса  CREATE OR REPLACE FUNCTION check\_blood\_availability()  RETURNS TRIGGER AS $$  DECLARE      available\_quantity INTEGER;  BEGIN      SELECT total\_quantity INTO available\_quantity      FROM blood\_inventories      WHERE bank\_id = NEW.bank\_id        AND blood\_type = NEW.blood\_type        AND rh\_factor = NEW.rh\_factor;      IF available\_quantity IS NULL OR available\_quantity < NEW.quantity THEN          RAISE EXCEPTION 'Недостаточное количество крови в банке!';      END IF;      RETURN NEW;  END;  $$ LANGUAGE plpgsql;  CREATE TRIGGER trigger\_check\_blood\_availability  BEFORE INSERT ON blood\_requests  FOR EACH ROW  EXECUTE FUNCTION check\_blood\_availability(); |

## Процедуры и функции

|  |
| --- |
| -- текущий запас крови определенного банка определенной группы и rh фактора  CREATE OR REPLACE FUNCTION get\_blood\_stock(bank\_id INT, blood\_type TEXT, rh\_factor TEXT)  RETURNS INT AS $$  DECLARE      blood\_count INT;  BEGIN      SELECT COALESCE(total\_quantity, 0)      INTO blood\_count      FROM blood\_inventories bi      WHERE bi.bank\_id = get\_blood\_stock.bank\_id        AND bi.blood\_type = get\_blood\_stock.blood\_type        AND bi.rh\_factor = get\_blood\_stock.rh\_factor;      RETURN blood\_count;  END;  $$ LANGUAGE plpgsql;  -- использование  SELECT get\_blood\_stock(3, 'A', '+');  -- текущий запас всех видов крови для банка  CREATE OR REPLACE FUNCTION get\_total\_blood\_in\_bank(bank\_id INT)  RETURNS INT AS $$  DECLARE      total\_blood INT;  BEGIN      SELECT COALESCE(SUM(total\_quantity), 0)      INTO total\_blood      FROM blood\_inventories bi      WHERE bi.bank\_id = get\_total\_blood\_in\_bank.bank\_id;      RETURN total\_blood;  END;  $$ LANGUAGE plpgsql;  -- использование  SELECT get\_total\_blood\_in\_bank(2);  -- количество заявок в ожидании обработки для банка  CREATE OR REPLACE FUNCTION get\_pending\_requests\_count(bank\_id INT)  RETURNS INT AS $$  DECLARE      request\_count INT;  BEGIN      SELECT COUNT(\*) INTO request\_count      FROM blood\_requests br      JOIN bank\_request brq ON br.id = brq.blood\_request\_id      WHERE brq.bank\_id = get\_pending\_requests\_count.bank\_id      AND br.status = 'pending';      RETURN request\_count;  END;  $$ LANGUAGE plpgsql;  -- использование  SELECT get\_pending\_requests\_count(2);  --ПРОЦЕДУРЫ  -- добавляет кровь в таблицу blood\_inventories  CREATE OR REPLACE PROCEDURE add\_blood\_to\_inventory(      p\_bank\_id INT,      p\_blood\_type TEXT,      p\_rh\_factor TEXT,      p\_quantity INT  )  LANGUAGE plpgsql AS $$  BEGIN      INSERT INTO blood\_inventories (bank\_id, blood\_type, rh\_factor, total\_quantity)      VALUES (p\_bank\_id, p\_blood\_type, p\_rh\_factor, p\_quantity)      ON CONFLICT (bank\_id, blood\_type, rh\_factor)      DO UPDATE SET total\_quantity = blood\_inventories.total\_quantity + p\_quantity;  END;  $$;  -- использование  CALL add\_blood\_to\_inventory(1, 'O', '+', 10);  -- пересчет blood\_inventory  CREATE OR REPLACE PROCEDURE recalculate\_blood\_inventory()  LANGUAGE plpgsql  AS $$  BEGIN      -- Обновляем существующие записи      UPDATE blood\_inventories bi      SET total\_quantity = COALESCE((          SELECT SUM(quantity)          FROM blood\_units bu          WHERE bu.bank\_id = bi.bank\_id            AND bu.blood\_type = bi.blood\_type            AND bu.rh\_factor = bi.rh\_factor      ), 0); -- Если нет записей, ставим 0  END;  $$;  -- использование  CALL recalculate\_blood\_inventory(); |

# Добавление индексов

1. **Индекс для поиска заявок по статусу**

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

|  |
| --- |
| CREATE INDEX idx\_blood\_requests\_status ON blood\_requests(status); |

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

1. **Составной индекс для поиска заявок по bank\_id**

**Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.**

|  |
| --- |
| CREATE INDEX idx\_bank\_request\_bank\_id ON bank\_request(bank\_id, blood\_request\_id); |

**Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.**

1. **Составной индекс для поиска подходящих запасов**

**Изображение выглядит как снимок экрана, текст, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.**

|  |
| --- |
| CREATE INDEX idx\_blood\_inventories\_filter ON blood\_inventories (blood\_type, rh\_factor); |

**Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.**