Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский университет ИТМО»
Факультет программной инженерии и компьютерной техники

Вариант: Пасека
Курсовая работа этап №4
по дисциплине
Информационные системы

Выполнил студент группы P3312
Соколов Анатолий Владимирович
Пархоменко Кирилл Александрович
Преподаватель:
Бострикова Дарья Константиновна

г. Санкт-Петербург
2024г.

Содержание

1 Отчет четвертой части

1.1 Реализовать уровень представления приложения для осуществления описанных на первом этапе бизнес-процессов.

Смотрите приложение 1

1.2 Сформировать итоговый отчет, содержащий все предыдущие этапы

1.3 Отчет первой части

1.3.1 Подробное текстовое описание предметной области

Предметная область: гео-распределенная пасека

1.3.2 Подробное текстовое описание предметной области

Улей

- Характеристики: Номер улья, тип улья (например, лежак, многокорпусный), дата установки.
- Источник: Основы пчеловодства, где рассматриваются различные типы ульев и их использование.

Пчелосемья

- Характеристики: Номер семьи, количество пчел, состояние (здоровая, больная).
- Источник: Статья о контроле летной активности пчел и их состоянии.

Датчик температуры и влажности

- Характеристики: Идентификатор датчика, значения температуры и влажности, дата и время измерения.
- Источник: Описание систем мониторинга в пчеловодстве, где используются датчики для контроля условий в улье.

Запись о медосборе

- Характеристики: Дата сбора меда, количество собранного меда, качество.
- Источник: Основы пчеловодства и практики сбора меда.

Журнал наблюдений

- Характеристики: Дата записи, описание наблюдений (поведение пчел, состояние улья), рекомендации.
- Источник: Методические рекомендации по ведению журнала наблюдений за пчелами.

Ветеринарный паспорт

- Характеристики: Номер паспорта, дата выдачи, состояние здоровья пчелосемьи.
- Источник: Ветеринарные документы для учета здоровья животных на пасеке.

Система управления

- Характеристики: Название системы, версия программного обеспечения, дата установки.
- Источник: Описание программных решений для управления пасеками.

План обслуживания

- Характеристики: Дата планового обслуживания, виды работ (например, осмотр ульев), ответственный за выполнение.
- Источник: Рекомендации по техническому обслуживанию ульев и оборудования.

Инциденты

- Характеристики: Дата инцидента, описание (например, болезни пчел), принятые меры.
- Источник: Нормативные акты по регистрации инцидентов на пасеке.

Отчетность по производству

- Характеристики: Период отчета, количество произведенного меда, расходы на содержание пасеки.
- Источник: Статья о ведении отчетности в пчеловодстве.

1.3.3 Зачем нужна информационная система

Было объяснено лично.

1.3.4 Функциональные/нефункциональные требования

Функциональные требования

1. Система должна обеспечивать добавление ульев, редактирование и удаление информации об ульях, включая номер, тип и дату установки.
2. Система должна вести записи о датах сбора меда, количестве собранного меда и качестве меда.
3. Система должна вести записи о датах проведения плановых обслуживаний, видах работ и ответственных за выполнение.
4. Система должна вести записи о датах и описании инцидентов, принятых мерах.
5. Система должна обеспечивать уведомления о критических изменениях в состоянии ульев или при возникновении инцидентов.
6. Система должна поддерживать экспорт данных в формате CSV.

Нефункциональные требования

1. Система должна быть доступна 99.9% времени.
2. Система должна защищать данные от несанкционированного доступа, включая аутентификацию и авторизацию пользователей.
3. Система должна быть устойчивой к сбоям и отказам, обеспечивая сохранность данных.
4. Система должна обеспечивать масштабируемость для поддержки роста числа ульев и обработки большего объема данных.
5. Система должна обеспечивать быстрый отклик при выполнении операций, не превышающий 2 секунд.

1.3.5 Модель основных прецедентов

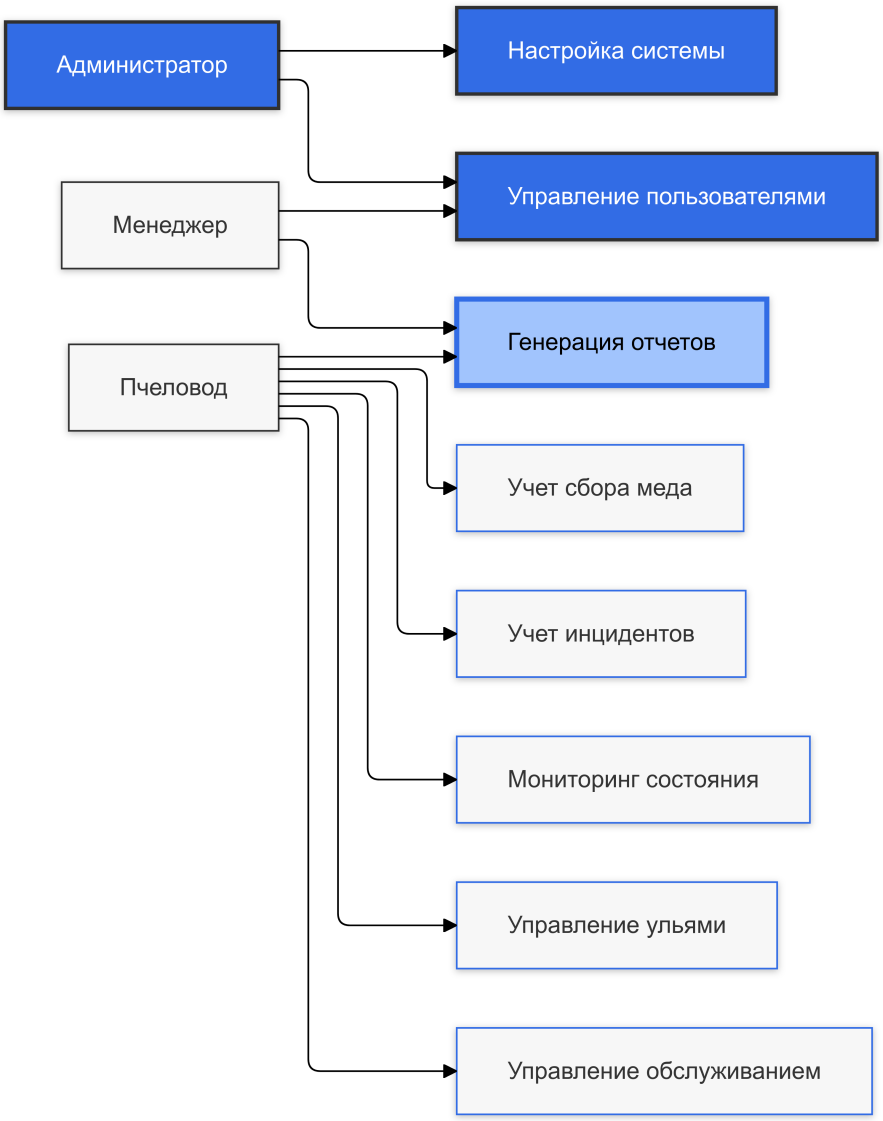

Рис. 1: использование системы разными пользователями

1.3.6 Архитектура будующей системы

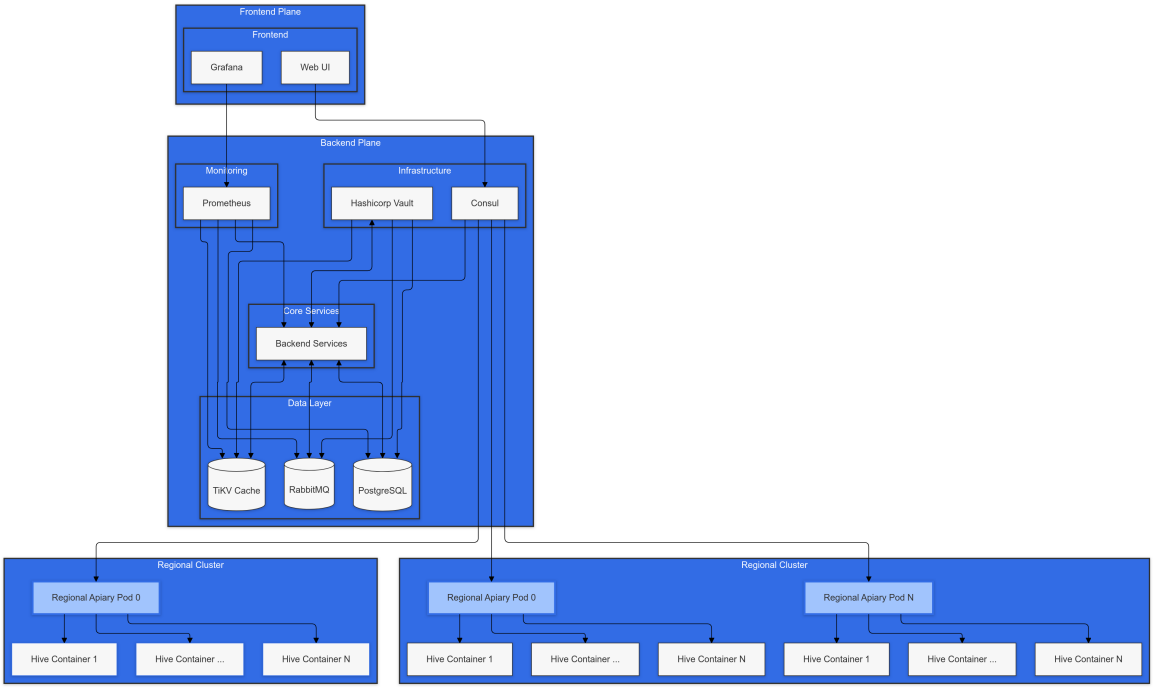

Рис. 2: архитектура

1.4 Отчет второй части

Предметная область: гео-распределенная пасека

1.4.1 Подробное текстовое описание предметной области

Улей

- **Характеристики**: Номер улья, тип улья (например, лежак, многокорпусный), дата установки.
- **Источник**: Основы пчеловодства, где рассматриваются различные типы ульев и их использование [8].

Пчелосемья

- **Характеристики**: Номер семьи, количество пчел, состояние (здоровая, больная).
- **Источник**: Статья о контроле летной активности пчел и их состоянии [7].

Датчик температуры и влажности

- **Характеристики**: Идентификатор датчика, значения температуры и влажности, дата и время измерения.
- **Источник**: Описание систем мониторинга в пчеловодстве, где используются датчики для контроля условий в улье [7].

Запись о медосборе

- **Характеристики**: Дата сбора меда, количество собранного меда, качество.
- **Источник**: Основы пчеловодства и практики сбора меда [8].

Журнал наблюдений

- **Характеристики**: Дата записи, описание наблюдений (поведение пчел, состояние улья), рекомендации.
- **Источник**: Методические рекомендации по ведению журнала наблюдений за пчелами [8].

Ветеринарный паспорт

- **Характеристики**: Номер паспорта, дата выдачи, состояние здоровья пчелосемьи.
- **Источник**: Ветеринарные документы для учета здоровья животных на пасеке [8].

Система управления

- **Характеристики**: Название системы, версия программного обеспечения, дата установки.
- **Источник**: Описание программных решений для управления пасеками [7].

План обслуживания

- **Характеристики**: Дата планового обслуживания, виды работ (например, осмотр ульев), ответственный за выполнение.
- **Источник**: Рекомендации по техническому обслуживанию ульев и оборудования [7].

Инциденты

- **Характеристики**: Дата инцидента, описание (например, болезни пчел), принятые меры.
- **Источник**: Нормативные акты по регистрации инцидентов на пасеке [8].

Отчетность по производству

- **Характеристики**: Период отчета, количество произведенного меда, расходы на содержание пасеки.
- **Источник**: Статья о ведении отчетности в пчеловодстве [8].
- **Характеристики**: Период отчета, количество произведенного меда, расходы на содержание пасеки.
- **Источник**: Статья о ведении отчетности в пчеловодстве [8].

1.4.2 Сформировать ER-модель базы данных (на основе описаний предметной области и прецедентов из предыдущего этапа).

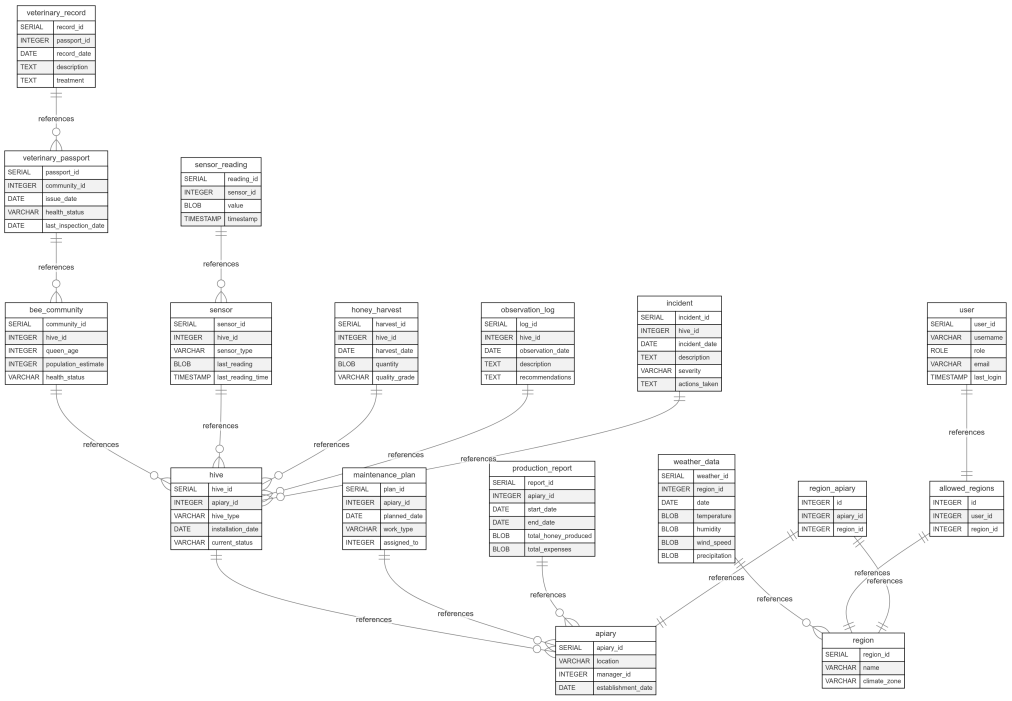

Рис. 3: ER диаграмма

1.4.3 Реализовать даталогическую модель в реляционной СУБД PostgreSQL

```
CREATE OR REPLACE FUNCTION add_constraint_if_not_exists(
   p_table_name TEXT,
   p_constraint_name TEXT,
   p_constraint_sql TEXT
) RETURNS VOID AS $$
BEGIN
   EXECUTE format('ALTER TABLE %I ADD CONSTRAINT %I %s', p_table_name,
      ↪ p_constraint_name, p_constraint_sql);
EXCEPTION
   WHEN duplicate_table THEN
      RAISE NOTICE 'Table constraint %.% already exists', p_table_name,
         ↪ p_constraint_name;
   WHEN duplicate_object THEN
      RAISE NOTICE 'Table constraint %.% already exists', p_table_name,
         ↪ p_constraint_name;
END;
$$ LANGUAGE plpgsql;

DO $$
BEGIN
   PERFORM add_constraint_if_not_exists('apiary', 'check_establishment_date', '
      ↪ CHECK ("establishment_date" ⩽ CURRENT_DATE)');
   PERFORM add_constraint_if_not_exists('hive', 'check_installation_date', '
      ↪ CHECK ("installation_date" ⩽ CURRENT_DATE)');
   PERFORM add_constraint_if_not_exists('bee_community', 'check_queen_age', '
      ↪ CHECK ("queen_age" ⩾ 0)');
   PERFORM add_constraint_if_not_exists('bee_community', '
      ↪ check_population_estimate', 'CHECK ("population_estimate" ⩾ 0)');
   PERFORM add_constraint_if_not_exists('veterinary_passport', 'check_issue_date
      ↪ ', 'CHECK ("issue_date" ⩽ CURRENT_DATE)');
   PERFORM add_constraint_if_not_exists('veterinary_passport', '
      ↪ check_last_inspection_date', 'CHECK ("last_inspection_date" ⩽
      ↪ CURRENT_DATE)');
   PERFORM add_constraint_if_not_exists('veterinary_record', 'check_record_date'
      ↪ , 'CHECK ("record_date" ⩽ CURRENT_DATE)');
   PERFORM add_constraint_if_not_exists('honey_harvest', 'check_harvest_date', '
      ↪ CHECK ("harvest_date" ⩽ CURRENT_DATE)');
   PERFORM add_constraint_if_not_exists('honey_harvest', 'check_quantity', '
      ↪ CHECK ("quantity" ⩾ 0)');
   PERFORM add_constraint_if_not_exists('observation_log', '
      ↪ check_observation_date', 'CHECK ("observation_date" ⩽ CURRENT_DATE)');
   PERFORM add_constraint_if_not_exists('maintenance_plan', 'check_planned_date'
      ↪ , 'CHECK ("planned_date" ⩾ CURRENT_DATE)');
   PERFORM add_constraint_if_not_exists('incident', 'check_incident_date', '
      ↪ CHECK ("incident_date" ⩽ CURRENT_DATE)');
   PERFORM add_constraint_if_not_exists('production_report', 'check_date_range',
      ↪  'CHECK ("start_date" ⩽ "end_date")');
   PERFORM add_constraint_if_not_exists('production_report', '
      ↪ check_total_honey_produced', 'CHECK ("total_honey_produced" ⩾ 0)');
   PERFORM add_constraint_if_not_exists('weather_data', 'check_weather_date', '
      ↪ CHECK ("date" ⩽ CURRENT_DATE)');
   PERFORM add_constraint_if_not_exists('bee_community', 'unique_hive_id', '
      ↪ UNIQUE ("hive_id")');
END $$;


DO $$ BEGIN IF NOT EXISTS (
      SELECT
            1
      FROM
```

```
            pg_type typ
            INNER JOIN pg_namespace nsp ON nsp.oid = typ.typnamespace
      WHERE
            nsp.nspname = current_schema()
            AND typ.typname = 'role'
) THEN CREATE TYPE role AS ENUM ('ADMIN', 'WORKER', 'MANAGER');

END IF;

END;

$$ LANGUAGE plpgsql;

CREATE TABLE IF NOT EXISTS "region" (
      "region_id" SERIAL,
      "name" VARCHAR NOT NULL,
      "climate_zone" VARCHAR,
      PRIMARY KEY("region_id")
);

CREATE TABLE IF NOT EXISTS "apiary" (
      "apiary_id" SERIAL,
      "location" VARCHAR NOT NULL,
      "manager_id" INTEGER,
      "establishment_date" DATE,
      PRIMARY KEY("apiary_id")
);

CREATE TABLE IF NOT EXISTS "hive" (
      "hive_id" SERIAL,
      "apiary_id" INTEGER,
      "hive_type" VARCHAR,
      "installation_date" DATE,
      "current_status" VARCHAR,
      PRIMARY KEY("hive_id")
);

CREATE TABLE IF NOT EXISTS "bee_community" (
      "community_id" SERIAL,
      "hive_id" INTEGER,
      "queen_age" INTEGER,
      "population_estimate" INTEGER,
      "health_status" VARCHAR,
      PRIMARY KEY("community_id")
);

CREATE TABLE IF NOT EXISTS "veterinary_passport" (
      "passport_id" SERIAL,
      "bee_community_id" INTEGER,
      "issue_date" DATE,
      "health_status" VARCHAR,
      "last_inspection_date" DATE,
      PRIMARY KEY("passport_id")
);

CREATE TABLE IF NOT EXISTS "veterinary_record" (
      "record_id" SERIAL,
      "passport_id" INTEGER,
      "record_date" DATE,
      "description" TEXT,
      "treatment" TEXT,
```

```
        PRIMARY KEY("record_id")
);

CREATE TABLE IF NOT EXISTS "sensor" (
        "sensor_id" SERIAL,
        "hive_id" INTEGER,
        "sensor_type" VARCHAR,
        "last_reading" BYTEA,
        "last_reading_time" TIMESTAMP,
        PRIMARY KEY("sensor_id")
);

CREATE TABLE IF NOT EXISTS "sensor_reading" (
        "reading_id" SERIAL,
        "sensor_id" INTEGER,
        "value" BYTEA,
        "timestamp" TIMESTAMP,
        PRIMARY KEY("reading_id")
);

CREATE TABLE IF NOT EXISTS "honey_harvest" (
        "harvest_id" SERIAL,
        "hive_id" INTEGER,
        "harvest_date" DATE,
        "quantity" FLOAT,
        "quality_grade" VARCHAR,
        PRIMARY KEY("harvest_id")
);

CREATE TABLE IF NOT EXISTS "observation_log" (
        "log_id" SERIAL,
        "hive_id" INTEGER,
        "observation_date" DATE,
        "description" TEXT,
        "recommendations" TEXT,
        PRIMARY KEY("log_id")
);

CREATE TABLE IF NOT EXISTS "maintenance_plan" (
        "plan_id" SERIAL,
        "apiary_id" INTEGER,
        "planned_date" DATE,
        "work_type" VARCHAR,
        "assigned_to" INTEGER,
        "status" VARCHAR(50) NOT NULL DEFAULT 'pending',
        PRIMARY KEY("plan_id")
);

CREATE TABLE IF NOT EXISTS "incident" (
        "incident_id" SERIAL,
        "hive_id" INTEGER,
        "incident_date" DATE,
        "description" TEXT,
        "severity" VARCHAR,
        "actions_taken" TEXT,
        PRIMARY KEY("incident_id")
);

CREATE TABLE IF NOT EXISTS "production_report" (
        "report_id" SERIAL,
        "apiary_id" INTEGER,
```

```
163       "start_date" DATE,
164       "end_date" DATE,
165       "total_honey_produced" FLOAT,
166       "total_expenses" FLOAT,
167       "curated_by" INTEGER,
168       PRIMARY KEY("report_id"),
169       UNIQUE("apiary_id", "start_date", "end_date")
170 );
171
172 CREATE TABLE IF NOT EXISTS "weather_data" (
173       "weather_id" SERIAL,
174       "region_id" INTEGER,
175       "date" DATE,
176       "temperature" FLOAT,
177       "humidity" FLOAT,
178       "wind_speed" FLOAT,
179       "precipitation" FLOAT,
180       PRIMARY KEY("weather_id")
181 );
182
183 CREATE TABLE IF NOT EXISTS "user" (
184       "user_id" SERIAL,
185       "username" VARCHAR NOT NULL UNIQUE,
186       "full_name" VARCHAR NOT NULL,
187       "role" ROLE,
188       "email" VARCHAR NOT NULL UNIQUE,
189       "password" VARCHAR NOT NULL,
190       "last_login" TIMESTAMP,
191       PRIMARY KEY("user_id")
192 );
193
194 CREATE TABLE IF NOT EXISTS "allowed_region" (
195       "id" SERIAL NOT NULL,
196       "user_id" INTEGER,
197       "region_id" INTEGER,
198       PRIMARY KEY("id"),
199       UNIQUE ("user_id", "region_id")
200 );
201
202 CREATE TABLE IF NOT EXISTS "region_apiary" (
203       "id" SERIAL NOT NULL UNIQUE,
204       "apiary_id" INTEGER,
205       "region_id" INTEGER,
206       PRIMARY KEY("id")
207 );
208
209 CREATE TABLE IF NOT EXISTS "worker_group" (
210    "group_id" SERIAL PRIMARY KEY,
211       "manager_id" INTEGER NOT NULL,
212    "group_name" VARCHAR NOT NULL,
213    "created_at" TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
214    "updated_at" TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
215 UNIQUE ("manager_id", "group_name")
216 );
217
218 CREATE TABLE IF NOT EXISTS "worker_group_member" (
219       "id" SERIAL PRIMARY KEY,
220       "group_id" INTEGER NOT NULL,
221       "worker_id" INTEGER NOT NULL,
222       "joined_at" TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
223       UNIQUE ("group_id", "worker_id")
```

```
);

DO $$
BEGIN
   IF NOT EXISTS (
      SELECT 1 FROM information_schema.columns
      WHERE table_name='production_report' AND column_name='curated_by'
   ) THEN
      ALTER TABLE "production_report" ADD COLUMN "curated_by" INTEGER;
   END IF;
END $$;

ALTER TABLE
      "worker_group"
ADD
FOREIGN KEY ("manager_id") REFERENCES "user"("user_id") ON UPDATE NO ACTION ON
   ↪ DELETE CASCADE;

ALTER TABLE
      "hive"
ADD
      FOREIGN KEY("apiary_id") REFERENCES "apiary"("apiary_id") ON UPDATE NO
         ↪ ACTION ON DELETE CASCADE;

ALTER TABLE
      "bee_community"
ADD
      FOREIGN KEY("hive_id") REFERENCES "hive"("hive_id") ON UPDATE NO ACTION ON
         ↪  DELETE CASCADE;

ALTER TABLE
      "veterinary_passport"
ADD
      FOREIGN KEY("bee_community_id") REFERENCES "bee_community"("community_id")
         ↪  ON UPDATE NO ACTION ON DELETE CASCADE;

ALTER TABLE
      "veterinary_record"
ADD
      FOREIGN KEY("passport_id") REFERENCES "veterinary_passport"("passport_id")
         ↪  ON UPDATE NO ACTION ON DELETE CASCADE;

ALTER TABLE
      "sensor"
ADD
      FOREIGN KEY("hive_id") REFERENCES "hive"("hive_id") ON UPDATE NO ACTION ON
         ↪  DELETE CASCADE;

ALTER TABLE
      "sensor_reading"
ADD
      FOREIGN KEY("sensor_id") REFERENCES "sensor"("sensor_id") ON UPDATE NO
         ↪ ACTION ON DELETE CASCADE;

ALTER TABLE
      "honey_harvest"
ADD
      FOREIGN KEY("hive_id") REFERENCES "hive"("hive_id") ON UPDATE NO ACTION ON
         ↪  DELETE CASCADE;

ALTER TABLE
```

```
277         "observation_log"
278 ADD
279         FOREIGN KEY("hive_id") REFERENCES "hive"("hive_id") ON UPDATE NO ACTION ON
                ↪  DELETE CASCADE;
280
281 ALTER TABLE
282         "maintenance_plan"
283 ADD
284         FOREIGN KEY("apiary_id") REFERENCES "apiary"("apiary_id") ON UPDATE NO
                ↪ ACTION ON DELETE CASCADE;
285
286 ALTER TABLE
287         "incident"
288 ADD
289         FOREIGN KEY("hive_id") REFERENCES "hive"("hive_id") ON UPDATE NO ACTION ON
                ↪  DELETE CASCADE;
290
291 ALTER TABLE
292         "production_report"
293 ADD
294         FOREIGN KEY("apiary_id") REFERENCES "apiary"("apiary_id") ON UPDATE NO
                ↪ ACTION ON DELETE CASCADE;
295
296 ALTER TABLE
297         "production_report"
298 ADD
299         FOREIGN KEY("curated_by") REFERENCES "user"("user_id") ON UPDATE NO ACTION
                ↪  ON DELETE CASCADE;
300
301 ALTER TABLE
302         "weather_data"
303 ADD
304         FOREIGN KEY("region_id") REFERENCES "region"("region_id") ON UPDATE NO
                ↪ ACTION ON DELETE CASCADE;
305
306 ALTER TABLE
307         "allowed_region"
308 ADD
309         FOREIGN KEY("region_id") REFERENCES "region"("region_id") ON UPDATE NO
                ↪ ACTION ON DELETE NO ACTION;
310
311 ALTER TABLE
312         "region_apiary"
313 ADD
314         FOREIGN KEY("apiary_id") REFERENCES "apiary"("apiary_id") ON UPDATE NO
                ↪ ACTION ON DELETE NO ACTION;
315
316 ALTER TABLE
317         "region_apiary"
318 ADD
319         FOREIGN KEY("region_id") REFERENCES "region"("region_id") ON UPDATE NO
                ↪ ACTION ON DELETE NO ACTION;
320
321 ALTER TABLE
322         "allowed_region"
323 ADD
324         FOREIGN KEY("user_id") REFERENCES "user"("user_id") ON UPDATE NO ACTION ON
                ↪  DELETE CASCADE;
325
326 ALTER TABLE
327         "sensor"
```

```
ALTER COLUMN
      "last_reading_time"
SET
      DEFAULT now();

ALTER TABLE
      "sensor_reading"
ALTER COLUMN
      "timestamp"
SET
      DEFAULT now();
ALTER TABLE
      "worker_group_member"
ADD
      FOREIGN KEY ("group_id") REFERENCES "worker_group"("group_id") ON UPDATE
         ↪ NO ACTION ON DELETE CASCADE;

ALTER TABLE
      "worker_group_member"
ADD
      FOREIGN KEY ("worker_id") REFERENCES "user"("user_id") ON UPDATE NO ACTION
         ↪  ON DELETE CASCADE;
```

1.4.4 Обеспечить целостность данных при помощи средств языка DDL и триггеров

```
CREATE OR REPLACE FUNCTION add_constraint_if_not_exists(
   p_table_name TEXT,
   p_constraint_name TEXT,
   p_constraint_sql TEXT
) RETURNS VOID AS $$
BEGIN
   EXECUTE format('ALTER TABLE %I ADD CONSTRAINT %I %s', p_table_name,
      ↪ p_constraint_name, p_constraint_sql);
EXCEPTION
   WHEN duplicate_table THEN
      RAISE NOTICE 'Table constraint %.% already exists', p_table_name,
         ↪ p_constraint_name;
   WHEN duplicate_object THEN
      RAISE NOTICE 'Table constraint %.% already exists', p_table_name,
         ↪ p_constraint_name;
END;
$$ LANGUAGE plpgsql;

DO $$
BEGIN
   PERFORM add_constraint_if_not_exists('apiary', 'check_establishment_date', '
      ↪ CHECK ("establishment_date" ⩽ CURRENT_DATE)');
   PERFORM add_constraint_if_not_exists('hive', 'check_installation_date', '
      ↪ CHECK ("installation_date" ⩽ CURRENT_DATE)');
   PERFORM add_constraint_if_not_exists('bee_community', 'check_queen_age', '
      ↪ CHECK ("queen_age" ⩾ 0)');
   PERFORM add_constraint_if_not_exists('bee_community', '
      ↪ check_population_estimate', 'CHECK ("population_estimate" ⩾ 0)');
   PERFORM add_constraint_if_not_exists('veterinary_passport', 'check_issue_date
      ↪ ', 'CHECK ("issue_date" ⩽ CURRENT_DATE)');
   PERFORM add_constraint_if_not_exists('veterinary_passport', '
      ↪ check_last_inspection_date', 'CHECK ("last_inspection_date" ⩽
      ↪ CURRENT_DATE)');
   PERFORM add_constraint_if_not_exists('veterinary_record', 'check_record_date'
      ↪ , 'CHECK ("record_date" ⩽ CURRENT_DATE)');
```

```
    PERFORM add_constraint_if_not_exists('honey_harvest', 'check_harvest_date', '
        ↪ CHECK ("harvest_date" ⩽ CURRENT_DATE)');
    PERFORM add_constraint_if_not_exists('honey_harvest', 'check_quantity', '
        ↪ CHECK ("quantity" ⩾ 0)');
    PERFORM add_constraint_if_not_exists('observation_log', '
        ↪ check_observation_date', 'CHECK ("observation_date" ⩽ CURRENT_DATE)');
    PERFORM add_constraint_if_not_exists('maintenance_plan', 'check_planned_date'
        ↪ , 'CHECK ("planned_date" ⩾ CURRENT_DATE)');
    PERFORM add_constraint_if_not_exists('incident', 'check_incident_date', '
        ↪ CHECK ("incident_date" ⩽ CURRENT_DATE)');
    PERFORM add_constraint_if_not_exists('production_report', 'check_date_range',
        ↪  'CHECK ("start_date" ⩽ "end_date")');
    PERFORM add_constraint_if_not_exists('production_report', '
        ↪ check_total_honey_produced', 'CHECK ("total_honey_produced" ⩾ 0)');
    PERFORM add_constraint_if_not_exists('weather_data', 'check_weather_date', '
        ↪ CHECK ("date" ⩽ CURRENT_DATE)');
    PERFORM add_constraint_if_not_exists('bee_community', 'unique_hive_id', '
        ↪ UNIQUE ("hive_id")');
END $$;
```

1.4.5 Реализовать скрипты для создания, удаления базы данных, заполнения базы тестовыми данными.

```
1
2  -- Function to populate the database with test data
3  CREATE OR REPLACE FUNCTION populate_test_data()
4  RETURNS VOID AS $$
5  BEGIN
6      -- Insert test data for Region
7      INSERT INTO Region (name, climate_zone) VALUES
8      ('North', 'Temperate'),
9      ('South', 'Mediterranean'),
10     ('East', 'Continental'),
11     ('West', 'Oceanic');
12
13     -- Insert test data for Apiary
14     INSERT INTO Apiary (location, establishment_date, region_id) VALUES
15     ('Forest Edge', '2020-05-15', 1),
16     ('Meadow', '2019-07-20', 2),
17     ('Mountain Side', '2021-03-10', 3),
18     ('Coastal Area', '2018-09-01', 4);
19
20     -- Insert test data for Hive
21     INSERT INTO Hive (apiary_id, installation_date) VALUES
22     (1, '2020-06-01'),
23     (1, '2020-06-02'),
24     (2, '2019-08-01'),
25     (3, '2021-04-01'),
26     (4, '2018-10-01');
27
28     -- Insert test data for BeeCommunity
29     INSERT INTO BeeCommunity (hive_id, queen_age, population_estimate) VALUES
30     (1, 2, 50000),
31     (2, 1, 40000),
32     (3, 3, 60000),
33     (4, 1, 45000),
34     (5, 2, 55000);
35
36     -- Insert test data for User
37     INSERT INTO "User" (username, password_hash, role) VALUES
```

```
    ('admin', 'hashed_password', 'admin'),
    ('user1', 'hashed_password', 'user'),
    ('user2', 'hashed_password', 'user');

    -- Insert test data for UserRegionAccess
    INSERT INTO UserRegionAccess (user_id, region_id) VALUES
    (2, 1),
    (2, 2),
    (3, 3),
    (3, 4);

    -- Insert test data for HoneyHarvest
    INSERT INTO HoneyHarvest (hive_id, harvest_date, quantity) VALUES
    (1, '2021-08-15', 25.5),
    (2, '2021-08-16', 22.0),
    (3, '2021-08-20', 30.5),
    (4, '2021-09-01', 28.0),
    (5, '2021-09-05', 26.5);

    -- Insert test data for ObservationLog
    INSERT INTO ObservationLog (hive_id, user_id, observation_date, notes) VALUES
    (1, 2, '2021-07-01', 'Bees appear healthy and active'),
    (2, 2, '2021-07-02', 'Queen spotted, laying eggs'),
    (3, 3, '2021-07-10', 'Hive population growing steadily'),
    (4, 3, '2021-07-15', 'Some signs of possible mite infestation'),
    (5, 2, '2021-07-20', 'Honey production looks promising');

    -- Insert test data for MaintenancePlan
    INSERT INTO MaintenancePlan (hive_id, planned_date, description) VALUES
    (1, '2021-10-01', 'Winter preparation'),
    (2, '2021-10-02', 'Winter preparation'),
    (3, '2021-10-05', 'Winter preparation'),
    (4, '2021-10-10', 'Mite treatment'),
    (5, '2021-10-15', 'Winter preparation');

    -- Insert test data for Incident
    INSERT INTO Incident (hive_id, incident_date, description) VALUES
    (4, '2021-07-20', 'Mite infestation detected');

    -- Insert test data for WeatherData
    INSERT INTO WeatherData (region_id, date, temperature, humidity,
       ↪ precipitation) VALUES
    (1, '2021-08-01', 25.5, 60.0, 0.0),
    (2, '2021-08-01', 28.0, 55.0, 0.0),
    (3, '2021-08-01', 23.0, 65.0, 5.0),
    (4, '2021-08-01', 22.0, 70.0, 2.0);

END;
$$ LANGUAGE plpgsql;

-- Execute the function to populate the database
SELECT populate_test_data();

-- Confirm population
SELECT 'Test data inserted successfully!' AS result;
```

1.4.6 Предложить pl/pgsql-функции и процедуры, для выполнения критически важных запросов (которые потребуются при последующей реализации прецедентов).

```
    -- Create function to create triggers if they don't exist
```

```
CREATE OR REPLACE FUNCTION create_trigger_if_not_exists(
   p_trigger_name TEXT,
   p_table_name TEXT,
   p_trigger_timing TEXT,
   p_trigger_event TEXT,
   p_trigger_function TEXT
) RETURNS VOID AS $$
BEGIN
   -- Check if the trigger already exists
   IF NOT EXISTS (
      SELECT 1
      FROM information_schema.triggers
      WHERE trigger_name = p_trigger_name
        AND event_object_table = p_table_name
   ) THEN
      -- Create the trigger if it doesn't exist
      EXECUTE format(
         'CREATE TRIGGER %I
          %s %s ON %I
          FOR EACH ROW
          EXECUTE FUNCTION %s()',
         p_trigger_name,
         p_trigger_timing,
         p_trigger_event,
         p_table_name,
         p_trigger_function
      );
      RAISE NOTICE 'Trigger % created on table %', p_trigger_name, p_table_name;
   ELSE
      RAISE NOTICE 'Trigger % already exists on table %', p_trigger_name,
         ↪ p_table_name;
   END IF;
END;
$$ LANGUAGE plpgsql;

-- Create triggers
CREATE OR REPLACE FUNCTION update_last_login()
RETURNS TRIGGER AS $$
BEGIN
   NEW.last_login = CURRENT_TIMESTAMP;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

SELECT create_trigger_if_not_exists(
   'update_user_last_login',
   'user',
   'BEFORE',
   'UPDATE',
   'update_last_login'
);

-- Create a function to grant access to all regions for admin users
CREATE OR REPLACE FUNCTION grant_admin_access()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.role = 'ADMIN' THEN
      INSERT INTO "allowed_region" (user_id, region_id)
      SELECT NEW.user_id, r.region_id
      FROM "region" r
```

```
61        ON CONFLICT ("user_id", "region_id") DO NOTHING;
62     END IF;
63     RETURN NEW;
64  END;
65  $$ LANGUAGE plpgsql;
66
67  SELECT create_trigger_if_not_exists(
68     'admin_access_trigger',
69     'user',
70     'AFTER',
71     'INSERT OR UPDATE',
72     'grant_admin_access'
73  );
74
75  CREATE OR REPLACE FUNCTION update_population_estimate()
76  RETURNS TRIGGER AS $$
77  BEGIN
78     IF TG_OP = 'INSERT' THEN
79        UPDATE "bee_community"
80        SET population_estimate = population_estimate + NEW.quantity
81        WHERE community_id = (SELECT community_id FROM bee_community WHERE hive_id
              ↪  = NEW.hive_id);
82     ELSIF TG_OP = 'DELETE' THEN
83        UPDATE "bee_community"
84        SET population_estimate = population_estimate - OLD.quantity
85        WHERE community_id = (SELECT community_id FROM bee_community WHERE hive_id
              ↪  = OLD.hive_id);
86     END IF;
87     RETURN NULL;
88  END;
89  $$ LANGUAGE plpgsql;
90
91  SELECT create_trigger_if_not_exists(
92     'update_bee_population',
93     'honey_harvest',
94     'AFTER',
95     'INSERT OR DELETE',
96     'update_population_estimate'
97  );
98
99  -- Trigger to automatically create a veterinary passport for new bee communities
100 CREATE OR REPLACE FUNCTION create_veterinary_passport()
101 RETURNS TRIGGER AS $$
102 BEGIN
103    INSERT INTO "veterinary_passport" (bee_community_id, issue_date,
          ↪ last_inspection_date)
104    VALUES (NEW.community_id, CURRENT_DATE, CURRENT_DATE);
105    RETURN NEW;
106 END;
107 $$ LANGUAGE plpgsql;
108
109 SELECT create_trigger_if_not_exists(
110    'new_bee_community_passport',
111    'bee_community',
112    'AFTER',
113    'INSERT',
114    'create_veterinary_passport'
115 );
116
117 -- Trigger to update production report when new honey harvest is added
```

```sql
CREATE OR REPLACE FUNCTION update_production_report()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO "production_report" (apiary_id, start_date, end_date,
        ↪ total_honey_produced)
    VALUES (
        (SELECT apiary_id FROM "hive" WHERE hive_id = NEW.hive_id),
        DATE_TRUNC('month', NEW.harvest_date),
        DATE_TRUNC('month', NEW.harvest_date) + INTERVAL '1 month' - INTERVAL '1
            ↪ day',
        NEW.quantity
    )
    ON CONFLICT (apiary_id, start_date, end_date)
    DO UPDATE SET total_honey_produced = "production_report".total_honey_produced
        ↪  + NEW.quantity;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

SELECT create_trigger_if_not_exists(
    'update_honey_production',
    'honey_harvest',
    'AFTER',
    'INSERT',
    'update_production_report'
);


-- 1. Функциядляполученияобщегоколичествамеда,
    ↪ собранноговконкретномульезаопределенныйпериод
CREATE OR REPLACE FUNCTION get_total_honey_harvested(
    p_hive_id INTEGER,
    p_start_date DATE,
    p_end_date DATE
) RETURNS DECIMAL AS $$
DECLARE
    total_honey DECIMAL;
BEGIN
    SELECT COALESCE(SUM(quantity::DECIMAL), 0)
    INTO total_honey
    FROM honey_harvest
    WHERE hive_id = p_hive_id
    AND harvest_date BETWEEN p_start_date AND p_end_date;

    RETURN total_honey;
END;
$$ LANGUAGE plpgsql;

-- 2. Процедурадлядобавленияновогонаблюдениявжурнал
CREATE OR REPLACE PROCEDURE add_observation(
    p_hive_id INTEGER,
    p_observation_date DATE,
    p_description TEXT,
    p_recommendations TEXT
) AS $$
BEGIN
    INSERT INTO observation_log (hive_id, observation_date, description,
        ↪ recommendations)
    VALUES (p_hive_id, p_observation_date, p_description, p_recommendations);
END;
```

```
$$ LANGUAGE plpgsql;

-- 3. Функциядляполучениятекущегосостоянияздоровьяпчелинойобщины
CREATE OR REPLACE FUNCTION get_community_health_status(p_community_id INTEGER)
RETURNS VARCHAR AS $$
DECLARE
    health_status VARCHAR;
BEGIN
    SELECT vp.health_status
    INTO health_status
    FROM veterinary_passport vp
    WHERE vp.bee_community_id = p_community_id
    ORDER BY vp.last_inspection_date DESC
    LIMIT 1;

    RETURN COALESCE(health_status, 'Unknown');
END;
$$ LANGUAGE plpgsql;

-- 4. Процедурадляобновлениястатусаулья
CREATE OR REPLACE PROCEDURE update_hive_status(
    p_hive_id INTEGER,
    p_new_status VARCHAR
) AS $$
BEGIN
    UPDATE hive
    SET current_status = p_new_status
    WHERE hive_id = p_hive_id;
END;
$$ LANGUAGE plpgsql;

-- 5. Функциядлярасчетасреднейтемпературыврегионезапоследние N дней
CREATE OR REPLACE FUNCTION get_avg_temperature(
    p_region_id INTEGER,
    p_days INTEGER
) RETURNS DECIMAL AS $$
DECLARE
    avg_temp DECIMAL;
BEGIN
    SELECT AVG(temperature::DECIMAL)
    INTO avg_temp
    FROM weather_data
    WHERE region_id = p_region_id
    AND date ⩾ CURRENT_DATE - p_days;

    RETURN COALESCE(avg_temp, 0);
END;
$$ LANGUAGE plpgsql;

-- 6. Процедурадляназначенияработниканапланобслуживания
CREATE OR REPLACE PROCEDURE assign_maintenance_plan(
    p_plan_id INTEGER,
    p_user_id INTEGER
) AS $$
BEGIN
    UPDATE maintenance_plan
    SET assigned_to = p_user_id
    WHERE plan_id = p_plan_id;
END;
$$ LANGUAGE plpgsql;
```

```
233
234 -- 7. Функциядляпроверкидоступапользователякрегиону
235 CREATE OR REPLACE FUNCTION has_region_access(
236     p_user_id INTEGER,
237     p_region_id INTEGER
238 ) RETURNS BOOLEAN AS $$
239 DECLARE
240     has_access BOOLEAN;
241 BEGIN
242     SELECT EXISTS (
243         SELECT 1
244         FROM allowed_region
245         WHERE user_id = p_user_id AND region_id = p_region_id
246     ) INTO has_access;
247
248     RETURN has_access;
249 END;
250 $$ LANGUAGE plpgsql;
251
252 -- 8. Процедурадлярегистрацииинцидента
253 CREATE OR REPLACE PROCEDURE register_incident(
254     p_hive_id INTEGER,
255     p_incident_date DATE,
256     p_description TEXT,
257     p_severity VARCHAR
258 ) AS $$
259 BEGIN
260     INSERT INTO incident (hive_id, incident_date, description, severity)
261     VALUES (p_hive_id, p_incident_date, p_description, p_severity);
262 END;
263 $$ LANGUAGE plpgsql;
264
265 -- 9. Функциядляполученияпоследнихпоказанийдатчика
266 CREATE OR REPLACE FUNCTION get_latest_sensor_reading(p_hive_id INTEGER,
       ↪ p_sensor_type VARCHAR)
267 RETURNS TABLE (value BYTEA, reading_timestamp TIMESTAMP) AS $$
268 BEGIN
269     RETURN QUERY
270     SELECT sr.value, sr.timestamp
271     FROM sensor s
272     JOIN sensor_reading sr ON s.sensor_id = sr.sensor_id
273     WHERE s.hive_id = p_hive_id AND s.sensor_type = p_sensor_type
274     ORDER BY sr.timestamp DESC
275     LIMIT 1;
276 END;
277 $$ LANGUAGE plpgsql;
278
279 -- 10. Процедурадлясозданияотчетаопроизводстве
280 CREATE OR REPLACE PROCEDURE create_production_report(
281     p_apiary_id INTEGER,
282     p_start_date DATE,
283     p_end_date DATE
284 ) AS $$
285 DECLARE
286     total_honey DECIMAL;
287     total_expenses DECIMAL;
288 BEGIN
289     -- Расчетобщегоколичествасобранногомеда
290     SELECT COALESCE(SUM(quantity::DECIMAL), 0)
291     INTO total_honey
```

```
292     FROM honey_harvest hh
293     JOIN hive h ON hh.hive_id = h.hive_id
294     WHERE h.apiary_id = p_apiary_id
295     AND hh.harvest_date BETWEEN p_start_date AND p_end_date;
296
297     -- Здесьдолженбытьрасчетобщихрасходовпример()
298     total_expenses := 1000.00;
299
300     -- Созданиеотчета
301     INSERT INTO production_report (apiary_id, start_date, end_date,
            ↪ total_honey_produced, total_expenses)
302     VALUES (p_apiary_id, p_start_date, p_end_date, total_honey, total_expenses);
303 END;
304 $$ LANGUAGE plpgsql;
```

1.4.7 Создать индексы на основе анализа использования базы данных в контексте описанных на первом этапе прецедентов. Обосновать полезность созданных индексов для реализации представленных на первом этапе бизнес-процессов.

```
1     -- Add indexes for performance
2   CREATE INDEX IF NOT EXISTS idx_apiary_manager ON "apiary"(manager_id);
3
4   CREATE INDEX IF NOT EXISTS idx_hive_apiary ON "hive"(apiary_id);
5
6   CREATE INDEX IF NOT EXISTS idx_bee_community_hive ON "bee_community"(hive_id);
7
8   CREATE INDEX IF NOT EXISTS idx_veterinary_passport_community ON "
        ↪ veterinary_passport"(bee_community_id);
9
10  CREATE INDEX IF NOT EXISTS idx_veterinary_record_passport ON "veterinary_record"
        ↪ (passport_id);
11
12  CREATE INDEX IF NOT EXISTS idx_sensor_hive ON "sensor"(hive_id);
13
14  CREATE INDEX IF NOT EXISTS idx_sensor_reading_sensor ON "sensor_reading"(
        ↪ sensor_id);
15
16  CREATE INDEX IF NOT EXISTS idx_honey_harvest_hive ON "honey_harvest"(hive_id);
17
18  CREATE INDEX IF NOT EXISTS idx_observation_log_hive ON "observation_log"(hive_id
        ↪ );
19
20  CREATE INDEX IF NOT EXISTS idx_maintenance_plan_apiary ON "maintenance_plan"(
        ↪ apiary_id);
21
22  CREATE INDEX IF NOT EXISTS idx_incident_hive ON "incident"(hive_id);
23
24  CREATE INDEX IF NOT EXISTS idx_production_report_apiary ON "production_report"(
        ↪ apiary_id);
25
26  CREATE INDEX IF NOT EXISTS idx_weather_data_region ON "weather_data"(region_id);
27
28  CREATE UNIQUE INDEX IF NOT EXISTS idx_allowed_region_user_region
29  ON "allowed_region" (user_id, region_id);
30
31  CREATE INDEX IF NOT EXISTS idx_allowed_region_region ON "allowed_region"(
        ↪ region_id);
32
33  CREATE INDEX IF NOT EXISTS idx_region_apiary_apiary ON "region_apiary"(apiary_id
        ↪ );
```

```
34
35 CREATE INDEX IF NOT EXISTS idx_region_apiary_region ON "region_apiary"(region_id
       ↪ );
```

1.5 Отчет третьей части

1.5.1 Изобразить диаграмму классов, представляющую общую архитектуру системы.

Рис. 4: ER диаграмма

На языке go нет классов.

Реализовать уровень хранения информационной системы на основе разработанной на предыдущем этапе базы данных.

Makefile

```
# Variables
NAMESPACE := beesbiz-data # Default namespace
TIKV_NAMESPACE := beesbiz-tikv
MONITORING_NAMESPACE := beesbiz-monitoring
CLOUDNATIVEPG_NAMESPACE := beesbiz-data
RABBITMQ_NAMESPACE := beesbiz-rabbitmq
RUNTIME_NAMESPACE := beesbiz-runtime
TIKV_RUNTIME := beesbiz-server

RELEASE_NAME := beesbiz
CHART_FILE := BeesBizData-0.1.0.tgz
KUBECTL := kubectl
HELM := helm
MINIKUBE := minikube

.PHONY: start uninstall template lint package upgrade list get_all port-forward
    ↪ stop-port-forward recreate pause resume pubsub postgresql uninstall
    ↪ uninstall-all upgrade upgrade-all recreate recreate-all install-tikv
    ↪ install-monitoring install-postgresql install-rabbitmq uninstall-tikv
    ↪ uninstall-monitoring uninstall-postgresql uninstall-rabbitmq docker-build
    ↪ update-deps setup-monitoring setup-servicemonitors patch-postgresql-
    ↪ monitoring verify-monitoring monitoring-dashboard prometheus-dashboard

uninstall:
	$(HELM) uninstall $(RELEASE_NAME) --namespace $(NAMESPACE)

upgrade:
	$(HELM) upgrade --install $(RELEASE_NAME) ./$(CHART_FILE) --namespace $(
	    ↪ NAMESPACE)

package:
	$(HELM) package .

lint:
	$(HELM) lint .

template:
	$(HELM) template ./$(CHART_FILE)

recreate: package upgrade

pubsub:
	$(KUBECTL) apply -f ./components/pubsub.yaml -n $(NAMESPACE);

docker-build:
	docker build -t $(TIKV_RUNTIME):latest .

load:
	$(MINIKUBE) image load $(TIKV_RUNTIME):latest

start:
	$(MINIKUBE) start --cpus 8 --memory 10244 --disk-size 20g
	$(MINIKUBE) addons enable storage-provisioner
	$(MINIKUBE) addons enable default-storageclass
	$(KUBECTL) create namespace $(NAMESPACE)
	$(KUBECTL) create namespace $(RABBITMQ_NAMESPACE)
	$(KUBECTL) create namespace $(MONITORING_NAMESPACE)
```

```
51        $(KUBECTL) create namespace $(RUNTIME_NAMESPACE)
52        # $(KUBECTL) apply -f persistent-volumes.yaml -n $(NAMESPACE) --wait
53        $(HELM) repo add cnpg https://cloudnative-pg.github.io/charts
54        $(HELM) repo add pingcap https://charts.pingcap.org/
55        $(HELM) repo update
56        $(KUBECTL) apply --server-side=true -f https://raw.githubusercontent.com/
              ↪ pingcap/tidb-operator/v1.6.0/manifests/crd.yaml
57        $(KUBECTL) apply --server-side=true -f https://github.com/rabbitmq/cluster
              ↪ -operator/releases/latest/download/cluster-operator.yml
58        $(HELM) install cnpg cnpg/cloudnative-pg --namespace $(
              ↪ CLOUDNATIVEPG_NAMESPACE) --create-namespace
59        $(HELM) install tidb-operator pingcap/tidb-operator --namespace $(
              ↪ TIKV_NAMESPACE) --create-namespace -f ./values-tidb-operator.yaml
60        # $(HELM) install tidb-operator pingcap/tidb-operator --namespace $(
              ↪ TIKV_NAMESPACE) --create-namespace
61        $(KUBECTL) create secret generic postgresql-superuser -n $(NAMESPACE) \
62      --from-literal=username=postgres \
63      --from-literal=password=postgres
64        $(MAKE) install-monitoring
65        $(MAKE) setup-servicemonitors
66        $(MAKE) patch-postgresql-monitoring
67
68  pause:
69        $(MINIKUBE) pause
70
71  resume:
72        $(MINIKUBE) start
73
74  install-monitoring:
75        $(HELM) repo add prometheus-community https://prometheus-community.github.
              ↪ io/helm-charts
76        $(HELM) repo add grafana https://grafana.github.io/helm-charts
77        $(HELM) repo update
78        $(HELM) dependency update
79        $(HELM) upgrade --install monitoring prometheus-community/kube-prometheus-
              ↪ stack \
80              --namespace $(MONITORING_NAMESPACE) \
81              --create-namespace \
82              -f ./values.yaml
83        @echo "Waiting for Prometheus operator to be ready..."
84        sleep 30
85        $(MAKE) setup-servicemonitors
86        $(MAKE) patch-postgresql-monitoring
87
88  uninstall-monitoring:
89        $(HELM) uninstall monitoring --namespace $(MONITORING_NAMESPACE)
90
91  update-deps:
92        $(HELM) repo add prometheus-community https://prometheus-community.github.
              ↪ io/helm-charts
93        $(HELM) repo add grafana https://grafana.github.io/helm-charts
94        $(HELM) repo add cnpg https://cloudnative-pg.github.io/charts
95        $(HELM) repo add pingcap https://charts.pingcap.org/
96        $(HELM) repo update
97        $(HELM) dependency update
98
99  setup-users: setup-rabbitmq-user setup-postgresql-user
100
101 setup-postgresql-user:
```

```
102       $(KUBECTL) exec -it -n $(CLOUDNATIVEPG_NAMESPACE) $$($(KUBECTL) get pods -
              ↪ n $(CLOUDNATIVEPG_NAMESPACE) | grep postgresql | awk '{print $$1}')
              ↪ -- psql -U postgres -c "DO \$$\$$ BEGIN IF NOT EXISTS (SELECT FROM
              ↪ pg_user WHERE usename = 'user') THEN CREATE USER \"user\" WITH
              ↪ PASSWORD 'postgres'; GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA
              ↪ public TO \"user\"; ALTER USER \"user\" CREATEDB; GRANT ALL
              ↪ PRIVILEGES ON DATABASE postgres TO \"user\"; END IF; END \$$\$$;"
103
104   setup-rabbitmq-user:
105       $(KUBECTL) exec -it -n $(RABBITMQ_NAMESPACE) $$($(KUBECTL) get pods -n $(
              ↪ RABBITMQ_NAMESPACE) | grep rabbitmq-cluster | awk '{print $$1}') --
              ↪ /bin/bash -c "rabbitmqctl delete_user guest || true && rabbitmqctl
              ↪ add_user guest guest && rabbitmqctl set_user_tags guest
              ↪ administrator && rabbitmqctl set_permissions -p / guest '.*' '.*'
              ↪ '.*'"
106
107   setup-monitoring: setup-rabbit-monitoring setup-postgresql-monitoring setup-
          ↪ grafana
108
109   setup-rabbit-monitoring:
110       sh rabbitmq-monitoring-setup.sh
111
112   setup-postgresql-monitoring:
113       sh postgresql-monitoring-setup.sh
114
115   setup-grafana:
116       $(KUBECTL) exec -it -n $(MONITORING_NAMESPACE) $$($(KUBECTL) get pods -n $
              ↪ (MONITORING_NAMESPACE) | grep grafana | awk '{print $$1}') --
              ↪ grafana cli admin reset-admin-password admin
117
118   setup-servicemonitors:
119       @echo "Applying monitoring configurations..."
120       $(HELM) upgrade --install $(RELEASE_NAME) ./$(CHART_FILE) --namespace $(
              ↪ NAMESPACE)
121
122   patch-postgresql-monitoring:
123       kubectl patch cluster postgresql -n $(CLOUDNATIVEPG_NAMESPACE) --type
              ↪ merge -p '{"spec":{"monitoring":{"enablePodMonitor":true,"
              ↪ customQueriesConfigMap":[{"name":"postgres-metrics-config","key":"
              ↪ queries.yaml"}]}}}'
124
125   verify-monitoring:
126       @echo "Verifying ServiceMonitors..."
127       $(KUBECTL) get servicemonitor -n $(MONITORING_NAMESPACE)
128       @echo "\nVerifying PostgreSQL metrics..."
129       $(KUBECTL) port-forward -n $(CLOUDNATIVEPG_NAMESPACE) postgresql-1
              ↪ 9187:9187 & \
130       sleep 5 && curl http://localhost:9187/metrics && kill %%
131       @echo "\nVerifying RabbitMQ metrics..."
132       $(KUBECTL) port-forward svc/rabbitmq-cluster -n $(RABBITMQ_NAMESPACE)
              ↪ 15692:15692 & \
133       sleep 5 && curl http://localhost:15692/metrics && kill %%
134
135   monitoring-dashboard:
136       $(KUBECTL) port-forward svc/monitoring-grafana -n $(MONITORING_NAMESPACE)
              ↪ 3000:80
137
138   prometheus-dashboard:
139       $(KUBECTL) port-forward svc/monitoring-kube-prometheus-prometheus -n $(
              ↪ MONITORING_NAMESPACE) 9090:9090
```

```
140
141 lens:
142       @echo "Downloading K8s Lens..."
143       curl -L -o lens.AppImage https://api.k8slens.dev/binaries/Lens
          ↪ -2024.11.261604-latest.x86_64.AppImage
144       @echo "Making AppImage executable..."
145       chmod +x lens.AppImage
146       @echo "K8s Lens downloaded successfully. You can run it with: ./lens.
          ↪ AppImage"
```

Конфигурация postgresql оператора

```
 1 apiVersion: postgresql.cnpg.io/v1
 2 kind: Cluster
 3 metadata:
 4   name: postgresql
 5   namespace: { { .Values.cloudnativepg.namespace } }
 6 spec:
 7   instances: { { .Values.cloudnativepg.instances } }
 8   imageName: { { .Values.cloudnativepg.imageName } }
 9   imagePullPolicy: { { .Values.cloudnativepg.imagePullPolicy } }
10   primaryUpdateStrategy: unsupervised
11   storage:
12     size: { { .Values.cloudnativepg.storage.size } }
13     storageClass: { { .Values.cloudnativepg.storage.storageClass } }
14   superuserSecret:
15     name: { { .Values.cloudnativepg.superuserSecret.name } }
16   bootstrap:
17     initdb:
18       database: { { .Values.cloudnativepg.bootstrap.initdb.database } }
19       owner: { { .Values.cloudnativepg.bootstrap.initdb.owner } }
20   postgresql:
21     parameters:
22       max_connections: "1000"
23       shared_buffers: 256MB
24   resources:
25     requests:
26       cpu: { { .Values.cloudnativepg.resources.requests.cpu } }
27       memory: { { .Values.cloudnativepg.resources.requests.memory } }
28     limits:
29       cpu: { { .Values.cloudnativepg.resources.limits.cpu } }
30       memory: { { .Values.cloudnativepg.resources.limits.memory } }
```

Значения для helm оператора

Chart.yaml

```
 1 apiVersion: v2
 2 name: BeesBizData
 3 description: A Helm chart for deploying BeesBiz data layer on Kubernetes
 4 version: 0.1.0
 5 appVersion: "1.0"
 6
 7 dependencies:
 8   - name: kube-prometheus-stack
 9     version: 45.9.1
10     repository: https://prometheus-community.github.io/helm-charts
11   - name: grafana-operator
12     version: 5.15.1
13     repository: https://grafana.github.io/helm-charts
```

persistent-volumes.yaml

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  namespace: beesbiz-data
  name: pv-postgresql
spec:
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: standard
  hostPath:
    path: "/mnt/data/postgresql"

---
apiVersion: v1
kind: PersistentVolume
metadata:
  namespace: beesbiz-data
  name: pv-rabbitmq
spec:
  capacity:
    storage: 512Mi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: standard
  hostPath:
    path: "/mnt/data/rabbitmq"

---
apiVersion: v1
kind: PersistentVolume
metadata:
  namespace: beesbiz-data
  name: pv-tikv
spec:
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: standard
  hostPath:
    path: "/mnt/data/tikv"

---
apiVersion: v1
kind: PersistentVolume
metadata:
  namespace: beesbiz-data
  name: pv-pd
spec:
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: standard
```

```
  hostPath:
    path: "/mnt/data/pd"
```

persistent-volumes.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: beesbiz-runtime
  name: beesbiz-server
spec:
  replicas: 1
  selector:
    matchLabels:
      app: beesbiz-server
  template:
    metadata:
      labels:
        app: beesbiz-server
    spec:
      containers:
        - name: beesbiz-server
          image: beesbiz-server:latest
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 4040
          volumeMounts:
            - name: config-volume
              mountPath: /app/config.env
              subPath: config.env
          resources:
            limits:
              cpu: 1
              memory: 1Gi
            requests:
              cpu: 500m
              memory: 512Mi
      volumes:
        - name: config-volume
          secret:
            secretName: config-env
```

Values.yaml

```yaml
namespace: beesbiz-data
clusterScoped: false

tikvCluster:
  namespace: beesbiz-tikv
  name: tikv-cluster
  pd:
    baseImage: pingcap/pd
    maxFailoverCount: 3
    replicas: 3
    storageClassName: "standard"
    resources:
      requests:
        storage: "2Gi"
        cpu: "500m"
        memory: "1Gi"
      limits:
        cpu: "2"
```

```
      memory: "2Gi"

  tikv:
    baseImage: pingcap/tikv
    maxFailoverCount: 3
    replicas: 3
    storageClassName: "standard"
    resources:
      requests:
        storage: "2Gi"
        cpu: "500m"
        memory: "1Gi"
      limits:
        cpu: "2"
        memory: "2Gi"

cloudnativepg:
  namespace: beesbiz-data
  instances: 3
  imageName: ghcr.io/cloudnative-pg/postgresql:14.7
  imagePullPolicy: IfNotPresent
  resources:
    requests:
      cpu: "500m"
      memory: "1Gi"
    limits:
      cpu: "2"
      memory: "2Gi"
  storage:
    size: 2Gi
    storageClass: "standard"
  superuserSecret:
    name: postgresql-superuser
    namespace: beesbiz-data
  bootstrap:
    initdb:
      database: postgres
      owner: postgres
  monitoring:
    enablePodMonitor: true
    customQueriesConfigMap:
      - name: postgres-metrics-config
        key: queries.yaml
    customMetrics:
      - name: pg_database_size
        query: "SELECT pg_database_size(current_database()) as size"
        metrics:
          - size:
              usage: "GAUGE"
              description: "Database size in bytes"
    metrics:
      enabled: true
      serviceMonitor:
        enabled: true
      prometheusRule:
        enabled: true

rabbitmq:
  namespace: beesbiz-rabbitmq
  name: rabbitmq-cluster
  replicas: 1
```

```
  image: rabbitmq:4-management
  imagePullPolicy: IfNotPresent
  persistence:
    size: 2Gi
    storageClass: "standard"
  resources:
    requests:
      cpu: "500m"
      memory: "1Gi"
      ephemeralStorage: "2Gi"
    limits:
      cpu: "2"
      memory: "2Gi"
      ephemeralStorage: "2Gi"
  auth:
    username: "beesbiz"
    password: "beesbiz"
  plugins:
    - rabbitmq_prometheus
  extraConfig: |
    management.load_definitions = /etc/rabbitmq/definitions.json
    prometheus.return_per_object_metrics = true
  service:
    type: ClusterIP
    ports:
      - name: prometheus
        port: 15692
        targetPort: 15692
  containers:
    ports:
      - name: epmd
        containerPort: 4369
      - name: amqp
        containerPort: 5672
      - name: management
        containerPort: 15672
      - name: prometheus
        containerPort: 15692

monitoring:
  namespace: beesbiz-monitoring
  prometheus:
    replicas: 1
    retention: 15d
    resources:
      requests:
        cpu: "500m"
        memory: "1Gi"
      limits:
        cpu: "1"
        memory: "2Gi"
    storage:
      size: 10Gi
      storageClass: "standard"
  grafana:
    replicas: 1
    resources:
      requests:
        cpu: "200m"
        memory: "512Mi"
      limits:
```

```
      cpu: "500m"
      memory: "1Gi"
    adminPassword: "admin"
```

psql-monitor.yaml

```yaml
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: postgresql-monitor
  namespace: beesbiz-monitoring
  labels:
    release: monitoring
spec:
  endpoints:
  - interval: 30s
    port: metrics
  selector:
    matchLabels:
      cnpg.io/cluster: cloudnative-pg # Adjust this to match your PostgreSQL service labels
  namespaceSelector:
    matchNames:
    - beesbiz-data
```

postgres-cluster.yaml

```yaml
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  name: postgresql
  namespace: {{ .Values.cloudnativepg.namespace }}
spec:
  instances: {{ .Values.cloudnativepg.instances }}
  imageName: {{ .Values.cloudnativepg.imageName }}
  imagePullPolicy: {{ .Values.cloudnativepg.imagePullPolicy }}
  primaryUpdateStrategy: unsupervised
  storage:
    size: {{ .Values.cloudnativepg.storage.size }}
    storageClass: {{ .Values.cloudnativepg.storage.storageClass }}
  superuserSecret:
    name: {{ .Values.cloudnativepg.superuserSecret.name }}
  bootstrap:
    initdb:
      database: {{ .Values.cloudnativepg.bootstrap.initdb.database }}
      owner: {{ .Values.cloudnativepg.bootstrap.initdb.owner }}
  postgresql:
    parameters:
      max_connections: "1000"
      shared_buffers: 256MB
  resources:
    requests:
      cpu: {{ .Values.cloudnativepg.resources.requests.cpu }}
      memory: {{ .Values.cloudnativepg.resources.requests.memory }}
    limits:
      cpu: {{ .Values.cloudnativepg.resources.limits.cpu }}
      memory: {{ .Values.cloudnativepg.resources.limits.memory }}
```

monitoring.yaml

```yaml
apiVersion: monitoring.coreos.com/v1
kind: Prometheus
```

```
metadata:
  name: prometheus
  namespace: {{ .Values.monitoring.namespace }}
spec:
  replicas: {{ .Values.monitoring.prometheus.replicas }}
  retention: {{ .Values.monitoring.prometheus.retention }}
  storage:
    volumeClaimTemplate:
      spec:
        storageClassName: {{ .Values.monitoring.prometheus.storage.storageClass }}
        resources:
          requests:
            storage: {{ .Values.monitoring.prometheus.storage.size }}
  resources:
    requests:
      cpu: {{ .Values.monitoring.prometheus.resources.requests.cpu }}
      memory: {{ .Values.monitoring.prometheus.resources.requests.memory }}
    limits:
      cpu: {{ .Values.monitoring.prometheus.resources.limits.cpu }}
      memory: {{ .Values.monitoring.prometheus.resources.limits.memory }}
  serviceMonitorSelector: {}
  serviceMonitorNamespaceSelector: {}

---
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: rabbitmq-monitor
  namespace: {{ .Values.monitoring.namespace }}
spec:
  endpoints:
  - interval: 30s
    port: prometheus
    path: /metrics
  selector:
    matchLabels:
      app: rabbitmq
  namespaceSelector:
    matchNames:
    - {{ .Values.rabbitmq.namespace }}

---
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: postgresql-monitor
  namespace: {{ .Values.monitoring.namespace }}
spec:
  endpoints:
  - interval: 30s
    port: metrics
  selector:
    matchLabels:
      postgresql: {{ .Values.cloudnativepg.name }}
  namespaceSelector:
    matchNames:
    - {{ .Values.cloudnativepg.namespace }}

---
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
```

```yaml
metadata:
  name: cnpg-monitor
  namespace: {{ .Values.monitoring.namespace }}
spec:
  endpoints:
  - interval: 30s
    port: metrics
  selector:
    matchLabels:
      postgresql: {{ .Values.cloudnativepg.name }}
  namespaceSelector:
    matchNames:
    - {{ .Values.cloudnativepg.namespace }}

---
apiVersion: integreatly.org/v1alpha1
kind: Grafana
metadata:
  name: grafana
  namespace: {{ .Values.monitoring.namespace }}
spec:
  deployment:
    replicas: {{ .Values.monitoring.grafana.replicas }}
    resources:
      requests:
        cpu: {{ .Values.monitoring.grafana.resources.requests.cpu }}
        memory: {{ .Values.monitoring.grafana.resources.requests.memory }}
      limits:
        cpu: {{ .Values.monitoring.grafana.resources.limits.cpu }}
        memory: {{ .Values.monitoring.grafana.resources.limits.memory }}
  config:
    security:
      admin_user: admin
      admin_password: {{ .Values.monitoring.grafana.adminPassword }}
    auth:
      disable_login_form: false
  dashboardLabelSelector:
    - matchExpressions:
        - key: app
          operator: In
          values:
            - grafana
  dataSource:
    - name: Prometheus
      type: prometheus
      url: http://prometheus-operated:9090
      isDefault: true
```

monitoring-namespace.yaml

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: {{ .Values.monitoring.namespace }}
  labels:
    name: {{ .Values.monitoring.namespace }}
```

grafana-dashboard

```yaml
apiVersion: integreatly.org/v1alpha1
kind: GrafanaDashboard
metadata:
```

```
  name: rabbitmq-overview
  namespace: {{ .Values.monitoring.namespace }}
  labels:
    app: grafana
spec:
  json: |
    {
      "title": "RabbitMQ Overview",
      "uid": "rabbitmq-overview",
      "datasource": "Prometheus",
      "panels": [
      ]
    }
---
apiVersion: integreatly.org/v1alpha1
kind: GrafanaDashboard
metadata:
  name: postgresql-overview
  namespace: {{ .Values.monitoring.namespace }}
  labels:
    app: grafana
spec:
  json: |
    {
      "title": "PostgreSQL Overview",
      "uid": "postgresql-overview",
      "datasource": "Prometheus",
      "panels": [
      ]
    }
```

prometheus-rules

```
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  name: rabbitmq-alerts
  namespace: {{ .Values.monitoring.namespace }}
spec:
  groups:
    - name: rabbitmq
      rules:
        - alert: RabbitmqNodeDown
          expr: rabbitmq_up == 0
          for: 5m
          labels:
            severity: critical
          annotations:
            summary: "RabbitMQ node down"
            description: "RabbitMQ node has been down for more than 5 minutes"
        - alert: RabbitmqHighMemory
          expr: rabbitmq_process_resident_memory_bytes / 1024 / 1024 > 1024
          for: 5m
          labels:
            severity: warning
          annotations:
            summary: "RabbitMQ high memory usage"
            description: "RabbitMQ is using more than 1GB of memory"

    - name: postgresql
      rules:
```

```
      - alert: PostgresqlDown
        expr: pg_up == 0
        for: 5m
        labels:
          severity: critical
        annotations:
          summary: "PostgreSQL instance down"
          description: "PostgreSQL instance has been down for more than 5 minutes
            ↪ "
      - alert: PostgresqlHighCPU
        expr: rate(pg_cpu_user_seconds_total[5m]) > 0.8
        for: 5m
        labels:
          severity: warning
        annotations:
          summary: "PostgreSQL high CPU usage"
          description: "PostgreSQL instance is using more than 80% CPU"
```

rabbitmq-cluster.yaml

```
apiVersion: rabbitmq.com/v1beta1
kind: RabbitmqCluster
metadata:
  name: {{ .Values.rabbitmq.name }}
  namespace: {{ .Values.rabbitmq.namespace }}
spec:
  image: {{ .Values.rabbitmq.image }}
  replicas: {{ .Values.rabbitmq.replicas }}
  persistence:
   storageClassName: {{ .Values.rabbitmq.persistence.storageClass }}
   storage: {{ .Values.rabbitmq.persistence.size }}
  resources:
    requests:
      cpu: {{ .Values.rabbitmq.resources.requests.cpu }}
      memory: {{ .Values.rabbitmq.resources.requests.memory }}
    limits:
      cpu: {{ .Values.rabbitmq.resources.limits.cpu }}
      memory: {{ .Values.rabbitmq.resources.limits.memory }}
  rabbitmq:
    additionalConfig: |
      default_user = {{ .Values.rabbitmq.auth.username }}
      default_pass = {{ .Values.rabbitmq.auth.password }}
```

tidb-cluster-auto-scaler.yaml

```
apiVersion: pingcap.com/v1alpha1
kind: TidbClusterAutoScaler

metadata:
  name: {{ .Values.tikvCluster.name }}
  namespace: {{ .Values.tikvCluster.namespace }}

spec:
  cluster:
    name: {{ .Values.tikvCluster.name }}
  tikv:
    resources:
      storage_small:
        cpu: 1000m
        memory: 2Gi
        storage: 1Gi
        count: 3
```

```
    rules:
      cpu:
        max_threshold: 0.8
        min_threshold: 0.2
        resource_types:
          - storage_small
```

tidb-cluster.yaml

```yaml
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster

metadata:
  name: {{ .Values.tikvCluster.name }}
  namespace: {{ .Values.tikvCluster.namespace }}

spec:
  version: v8.1.0
  timezone: UTC
  pvReclaimPolicy: Retain
  enableDynamicConfiguration: true
  configUpdateStrategy: RollingUpdate
  discovery: {}
  helper:
    image: alpine:3.16.0

  pd:
    baseImage: {{ .Values.tikvCluster.pd.baseImage }}
    maxFailoverCount: {{ .Values.tikvCluster.pd.maxFailoverCount }}
    replicas: {{ .Values.tikvCluster.pd.replicas }}
    storageClassName: "{{ .Values.tikvCluster.pd.storageClassName }}"
    readinessProbe:
      type: "tcp"
    requests:
      cpu: {{ .Values.tikvCluster.pd.resources.requests.cpu }}
      memory: {{ .Values.tikvCluster.pd.resources.requests.memory }}
      storage: {{ .Values.tikvCluster.pd.resources.requests.storage }}
    limits:
      cpu: {{ .Values.tikvCluster.pd.resources.limits.cpu }}
      memory: {{ .Values.tikvCluster.pd.resources.limits.memory }}
    config: |
      [keyspace]
      pre-alloc = ["a", "b", "c"]
      [pd-server]
      initial-cluster-state = "existing"
      [log]
      level = "warn"

  tikv:
    baseImage: {{ .Values.tikvCluster.tikv.baseImage }}
    maxFailoverCount: {{ .Values.tikvCluster.tikv.maxFailoverCount }}
    replicas: {{ .Values.tikvCluster.tikv.replicas }}
    storageClassName: "{{ .Values.tikvCluster.tikv.storageClassName }}"
    requests:
      storage: {{ .Values.tikvCluster.tikv.resources.requests.storage }}
      cpu: {{ .Values.tikvCluster.tikv.resources.requests.cpu }}
      memory: {{ .Values.tikvCluster.tikv.resources.requests.memory }}
    limits:
      cpu: {{ .Values.tikvCluster.tikv.resources.limits.cpu }}
      memory: {{ .Values.tikvCluster.tikv.resources.limits.memory }}
    config: |
```

```
    [log]
    level = "warn"

    [storage]
    api-version = 2
    enable-ttl = true
    reserve-space = "1GB"

    [rocksdb]
    max-open-files = 1024

    [raftdb]
    max-open-files = 1024
```

psql-monitoring.sh

```
cat << EOF | kubectl apply -f -
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: postgresql-monitor
  namespace: beesbiz-monitoring
  labels:
    release: monitoring # This must match Prometheus's serviceMonitorSelector
spec:
  endpoints:
  - interval: 30s
    port: metrics
    path: /metrics
  selector:
    matchLabels:
      cnpg.io/cluster: postgresql
  namespaceSelector:
    matchNames:
    - beesbiz-data
EOF


cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Service
metadata:
  name: postgresql-metrics
  namespace: beesbiz-data
  labels:
    cnpg.io/cluster: postgresql
spec:
  ports:
  - name: metrics
    port: 9187
    targetPort: metrics
  selector:
    cnpg.io/cluster: postgresql
EOF

kubectl patch cluster postgresql -n beesbiz-data --type merge -p '
{
  "spec": {
    "monitoring": {
      "enablePodMonitor": true,
```

```
    "customQueriesConfigMap": [
      {
        "name": "cnpg-default-monitoring",
        "key": "queries"
      }
    ],
    "disableDefaultQueries": false
  }
 }
}'


cat << EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: postgres-metrics-config
  namespace: beesbiz-data
data:
  queries.yaml: |
    pg_database:
      query: "SELECT pg_database.datname, pg_database_size(pg_database.datname) as
          ↪ size_bytes FROM pg_database"
      metrics:
        - datname:
            usage: "LABEL"
            description: "Name of the database"
        - size_bytes:
            usage: "GAUGE"
            description: "Disk space used by the database"
EOF
```

rabbitmq-monitoring.sh

```
cat << EOF | kubectl apply -f -
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: rabbitmq-monitor
  namespace: beesbiz-monitoring
  labels:
    release: monitoring
spec:
  endpoints:
  - interval: 30s
    port: prometheus
    path: /metrics
  selector:
    matchLabels:
      app.kubernetes.io/name: rabbitmq-cluster
  namespaceSelector:
    matchNames:
    - beesbiz-rabbitmq
EOF
```

1.5.2 При реализации уровня хранения должны использоваться функции/процедуры, созданные на втором этапе с помощью pl/pgsql. Нельзя замещать их использование альтернативной реализацией аналогичных запросов на уровне хранения информационной системы.

```
func (db *DB) InitSchema(pathToScripts string, sqlFiles []string) error {
	for _, file := range sqlFiles {
		filePath := filepath.Join(pathToScripts, file)
		zap.L().Info("Loading SQL file", zap.String("file", file))
		if err := db.executeSQLFile(filePath); err != nil {
			zap.L().Error("Failed to execute SQL file", zap.String("file",
				↪ file), zap.Error(err))
			return fmt.Errorf("error executing SQL file %s: %w", file, err
				↪ )
		}
		zap.L().Info("Successfully executed SQL file", zap.String("file",
			↪ file))
	}

	zap.L().Info("All SQL files executed successfully")
	return nil
}

func (db *DB) executeSQLFile(filePath string) error {
	content, err := os.ReadFile(filePath)
	if err != nil {
		return fmt.Errorf("error reading SQL file: %w", err)
	}

	_, err = db.Exec(string(content))
	if err != nil {
		return fmt.Errorf("error executing SQL: %w", err)
	}
	return nil
}

func (db *DB) ExecuteSQL(sql string) error {
	_, err := db.Exec(sql)
	if err != nil {
		return fmt.Errorf("error executing SQL: %w", err)
	}
	return nil
}
```

1.5.3 Использование функций/процедур

```
syntax = "proto3";

package bee_management;

import "google/protobuf/empty.proto";

option go_package = "github.com/orientallines/beesbiz/bee_management";

// Service Definition
service BeeManagementService {
  // 1. Get Total Honey Harvested
  rpc GetTotalHoneyHarvested(GetTotalHoneyHarvestedRequest)
     returns (GetTotalHoneyHarvestedResponse) {}

  // 2. Add Observation
  rpc AddObservation(AddObservationRequest) returns (google.protobuf.Empty) {}

```

```
  // 3. Get Community Health Status
  rpc GetCommunityHealthStatus(GetCommunityHealthStatusRequest)
     returns (GetCommunityHealthStatusResponse) {}

  // 4. Update Hive Status
  rpc UpdateHiveStatus(UpdateHiveStatusRequest)
     returns (google.protobuf.Empty) {}

  // 5. Get Average Temperature
  rpc GetAvgTemperature(GetAvgTemperatureRequest)
     returns (GetAvgTemperatureResponse) {}

  // 6. Assign Maintenance Plan
  rpc AssignMaintenancePlan(AssignMaintenancePlanRequest)
     returns (google.protobuf.Empty) {}

  // 7. Check Region Access
  rpc HasRegionAccess(HasRegionAccessRequest)
     returns (HasRegionAccessResponse) {}

  // 8. Register Incident
  rpc RegisterIncident(RegisterIncidentRequest)
     returns (google.protobuf.Empty) {}

  // 9. Get Latest Sensor Reading
  rpc GetLatestSensorReading(GetLatestSensorReadingRequest)
     returns (GetLatestSensorReadingResponse) {}

  // 10. Create Production Report
  rpc CreateProductionReport(CreateProductionReportRequest)
     returns (google.protobuf.Empty) {}

  // 11. Set Region Access
  rpc SetRegionAccess(SetRegionAccessRequest) returns (google.protobuf.Empty) {}
}

// Message Definitions

// 1. GetTotalHoneyHarvested
message GetTotalHoneyHarvestedRequest {
  int32 hive_id = 1;
  string start_date = 2; // Format: YYYY-MM-DD
  string end_date = 3; // Format: YYYY-MM-DD
}

message GetTotalHoneyHarvestedResponse { double total_honey = 1; }

// 2. AddObservation
message AddObservationRequest {
  int32 hive_id = 1;
  string observation_date = 2; // Format: YYYY-MM-DD
  string description = 3;
  string recommendations = 4;
}

// 3. GetCommunityHealthStatus
message GetCommunityHealthStatusRequest { int32 community_id = 1; }

message GetCommunityHealthStatusResponse { string health_status = 1; }

// 4. UpdateHiveStatus
```

```proto
message UpdateHiveStatusRequest {
  int32 hive_id = 1;
  string new_status = 2;
}

// 5. GetAvgTemperature
message GetAvgTemperatureRequest {
  int32 region_id = 1;
  int32 days = 2;
}

message GetAvgTemperatureResponse { double avg_temperature = 1; }

// 6. AssignMaintenancePlan
message AssignMaintenancePlanRequest {
  int32 plan_id = 1;
  int32 user_id = 2;
}

// 7. HasRegionAccess
message HasRegionAccessRequest {
  int32 user_id = 1;
  int32 region_id = 2;
}

message HasRegionAccessResponse { bool has_access = 1; }

// 8. RegisterIncident
message RegisterIncidentRequest {
  int32 hive_id = 1;
  string incident_date = 2; // Format: YYYY-MM-DD
  string description = 3;
  string severity = 4;
}

// 9. GetLatestSensorReading
message GetLatestSensorReadingRequest {
  int32 hive_id = 1;
  string sensor_type = 2;
}

message GetLatestSensorReadingResponse {
  bytes value = 1;
  string timestamp = 2; // ISO 8601 format
}

// 10. CreateProductionReport
message CreateProductionReportRequest {
  int32 apiary_id = 1;
  string start_date = 2; // Format: YYYY-MM-DD
  string end_date = 3; // Format: YYYY-MM-DD
}

// 11. SetRegionAccess
message SetRegionAccessRequest {
  int32 user_id = 1;
  int32 region_id = 2;
}
```

1.5.4 Реализация уровеня бизнес-логики

```
   type Server struct {
      app *fiber.App
      db *database.DB
      jwtKey []byte
}

// NewServer creates a new Server
func NewServer(db *database.DB) *Server {
      return &Server{
            app: fiber.New(),
            db: db,
            jwtKey: []byte(config.GlobalConfig.JwtSecret),
      }
}

// SetupRoutes sets up the routes for the server
func (s *Server) SetupRoutes() {
      s.app.Use(requestid.New())
      s.app.Use(logger.New(logger.Config{
            Format: "[${time}] ${status} - ${method} ${path}\n",
      }))
      s.app.Use(healthcheck.New(healthcheck.Config{
            LivenessProbe: func(c *fiber.Ctx) bool {
                  return true
            },
            LivenessEndpoint: "/livez",
            ReadinessProbe: func(c *fiber.Ctx) bool {
                  return true
            },
            ReadinessEndpoint: "/readyz",
      }))

      prometheus := fiberprometheus.New("beesbiz")
      prometheus.RegisterAt(s.app, "/metrics")
      prometheus.SetSkipPaths([]string{"/ping", "/livez", "/readyz"})

      s.app.Use(prometheus.Middleware)

      s.app.Use(cors.New(cors.Config{
            AllowOrigins: "*", // Your frontend URL
            AllowHeaders: "Origin, Content-Type, Accept, Authorization",
            AllowMethods: "GET,POST,HEAD,PUT,DELETE,PATCH",
            AllowCredentials: false,
      }))

      auth := s.app.Group("/auth")

      auth.Post("/login", handlers.Login(s.db, s.jwtKey))
      auth.Post("/register", handlers.Register(s.db))

      api := s.app.Group("/api", jwtMiddleware(s.jwtKey))

      // Apiary routes
      apiary := api.Group("/apiary", roleMiddleware(types.Worker, types.Manager,
         ↪ types.Admin))

      apiary.Get("/:id", handlers.GetApiary(s.db))
      apiary.Post("/", handlers.CreateApiary(s.db))
      apiary.Put("/", handlers.UpdateApiary(s.db))
```

```
59          apiary.Delete("/:id", handlers.DeleteApiary(s.db))
60          apiary.Get("/", handlers.GetAllApiaries(s.db))
61
62          // Hive routes
63          hive := api.Group("/hive", roleMiddleware(types.Worker, types.Manager,
                ↪ types.Admin))
64
65          hive.Get("/", handlers.GetAllHives(s.db))
66          hive.Post("/", handlers.CreateHive(s.db))
67          hive.Put("/", handlers.UpdateHive(s.db))
68          hive.Delete("/:id", handlers.DeleteHive(s.db, s.rmq))
69          hive.Get("/:apiaryID/hives", handlers.GetAllHivesByApiaryID(s.db))
70
71          // BeeCommunity routes
72          beeCommunity := api.Group("/bee-community", roleMiddleware(types.Worker,
                ↪ types.Manager, types.Admin))
73
74          beeCommunity.Get("/", handlers.GetAllBeeCommunities(s.db))
75          beeCommunity.Post("/", handlers.CreateBeeCommunity(s.db))
76          beeCommunity.Put("/", handlers.UpdateBeeCommunity(s.db))
77          beeCommunity.Delete("/:id", handlers.DeleteBeeCommunity(s.db))
78          beeCommunity.Get("/:hiveID/bee-communities", handlers.
                ↪ GetAllBeeCommunitiesByHiveID(s.db))
79
80          // HoneyHarvest routes
81          honeyHarvest := api.Group("/honey-harvest", roleMiddleware(types.Worker,
                ↪ types.Manager, types.Admin))
82
83          honeyHarvest.Get("/:id", handlers.GetHoneyHarvest(s.db))
84          honeyHarvest.Post("/", handlers.CreateHoneyHarvest(s.db))
85          honeyHarvest.Put("/", handlers.UpdateHoneyHarvest(s.db))
86          honeyHarvest.Delete("/:id", handlers.DeleteHoneyHarvest(s.db))
87          honeyHarvest.Get("/", handlers.GetAllHoneyHarvests(s.db))
88
89          // Region routes
90          region := api.Group("/region", roleMiddleware(types.Worker, types.Manager,
                ↪  types.Admin))
91
92          region.Get("/:id", handlers.GetRegion(s.db))
93          region.Post("/", handlers.CreateRegion(s.db))
94          region.Put("/", handlers.UpdateRegion(s.db))
95          region.Delete("/:id", handlers.DeleteRegion(s.db))
96          region.Get("/", handlers.GetAllRegions(s.db))
97
98          // AllowedRegion routes
99          allowedRegion := api.Group("/allowed-region", roleMiddleware(types.Manager
                ↪ , types.Admin))
100
101         allowedRegion.Get("/user/:id", handlers.GetAllowedRegionsForUser(s.db))
102         allowedRegion.Post("/", handlers.CreateAllowedRegion(s.db))
103         allowedRegion.Put("/", handlers.UpdateAllowedRegion(s.db))
104         allowedRegion.Delete("/:id", handlers.DeleteAllowedRegion(s.db))
105         allowedRegion.Get("/", handlers.GetAllAllowedRegions(s.db))
106
107         // RegionApiary routes
108         regionApiary := api.Group("/region-apiary", roleMiddleware(types.Manager,
                ↪ types.Admin))
109
110         regionApiary.Get("/:id", handlers.GetRegionApiary(s.db))
111         regionApiary.Post("/", handlers.CreateRegionApiary(s.db))
112         regionApiary.Put("/", handlers.UpdateRegionApiary(s.db))
```

```
113         regionApiary.Delete("/:id", handlers.DeleteRegionApiary(s.db))
114         regionApiary.Get("/", handlers.GetAllRegionApiaries(s.db))
115
116         // User routes
117         user := api.Group("/user", roleMiddleware(types.Admin, types.Manager))
118
119         user.Get("/:id", handlers.GetUser(s.db))
120         user.Post("/", handlers.CreateUser(s.db))
121         user.Put("/", handlers.UpdateUser(s.db))
122         user.Delete("/:id", handlers.DeleteUser(s.db))
123         user.Get("/", handlers.GetAllUsers(s.db))
124         user.Get("/:id/allowed-regions", handlers.GetUserAllowedRegions(s.db))
125         user.Put("/role", handlers.ModifyUserRole(s.db))
126         user.Put("/allowed-regions", handlers.ModifyUserAllowedRegions(s.db))
127
128         // WorkerGroup routes
129         workerGroup := api.Group("/worker-group", roleMiddleware(types.Admin,
                ↪ types.Manager))
130
131         workerGroup.Get("/", handlers.GetAllWorkerGroups(s.db))
132         workerGroup.Get("/:id", handlers.GetWorkerGroup(s.db))
133         workerGroup.Post("/", handlers.CreateWorkerGroup(s.db))
134         workerGroup.Get("/manager/:manager_id", handlers.GetWorkerGroupsByManager(
                ↪ s.db))
135         workerGroup.Post("/:group_id/members", handlers.AddGroupMember(s.db))
136         workerGroup.Delete("/:group_id/members/:worker_id", handlers.
                ↪ RemoveGroupMember(s.db))
137         workerGroup.Get("/:group_id/members", handlers.GetGroupMembers(s.db))
138         workerGroup.Get("/worker/:worker_id/groups", handlers.GetWorkerGroups(s.db
                ↪ ))
139         workerGroup.Delete("/:id", handlers.DeleteWorkerGroup(s.db))
140         workerGroup.Put("/:id", handlers.UpdateWorkerGroup(s.db))
141
142         // ProductionReport routes
143         productionReport := api.Group("/production-report", roleMiddleware(types.
                ↪ Manager, types.Worker, types.Admin))
144
145         productionReport.Get("/:id", handlers.GetProductionReport(s.db))
146         productionReport.Post("/", handlers.CreateProductionReport(s.db))
147         productionReport.Put("/", handlers.UpdateProductionReport(s.db))
148         productionReport.Delete("/:id", handlers.DeleteProductionReport(s.db))
149         productionReport.Get("/", handlers.GetAllProductionReports(s.db))
150         productionReport.Get("/curated/:id", handlers.
                ↪ GetCuratedProductionReportsByUser(s.db))
151
152         // Sensor routes
153         sensor := api.Group("/sensor", roleMiddleware(types.Admin, types.Manager,
                ↪ types.Worker))
154
155         sensor.Get("/:id", handlers.GetSensor(s.db))
156         sensor.Post("/", handlers.CreateSensor(s.db, s.rmq))
157         sensor.Put("/", handlers.UpdateSensor(s.db))
158         sensor.Delete("/:id", handlers.DeleteSensor(s.db, s.rmq))
159         sensor.Get("/", handlers.GetAllSensors(s.db))
160
161         // SensorReading routes
162         sensorReading := api.Group("/sensor-reading", roleMiddleware(types.Admin,
                ↪ types.Manager, types.Worker))
163
164         sensorReading.Get("/:id", handlers.GetSensorReading(s.db))
165         sensorReading.Post("/", handlers.CreateSensorReading(s.db))
```

```
166         sensorReading.Put("/", handlers.UpdateSensorReading(s.db))
167         sensorReading.Delete("/:id", handlers.DeleteSensorReading(s.db))
168         sensorReading.Get("/", handlers.GetAllSensorReadings(s.db))
169 
170         // WeatherData routes
171         weatherData := api.Group("/weather-data", roleMiddleware(types.Admin,
                ↪ types.Manager, types.Worker))
172 
173         weatherData.Get("/:id", handlers.GetWeatherData(s.db))
174         weatherData.Post("/", handlers.CreateWeatherData(s.db))
175         weatherData.Put("/", handlers.UpdateWeatherData(s.db))
176         weatherData.Delete("/:id", handlers.DeleteWeatherData(s.db))
177         weatherData.Get("/", handlers.GetAllWeatherData(s.db))
178 
179         // Incident routes
180         incident := api.Group("/incident", roleMiddleware(types.Worker, types.
                ↪ Manager, types.Admin))
181 
182         incident.Get("/:id", handlers.GetIncident(s.db))
183         incident.Post("/", handlers.CreateIncident(s.db, s.rmq))
184         incident.Put("/", handlers.UpdateIncident(s.db))
185         incident.Delete("/:id", handlers.DeleteIncident(s.db))
186         incident.Get("/", handlers.GetAllIncidents(s.db))
187         incident.Put("/:id/status", handlers.UpdateIncidentStatus(s.db))
188 
189         // Observation routes
190         observation := api.Group("/observation", roleMiddleware(types.Worker,
                ↪ types.Manager, types.Admin))
191 
192         observation.Get("/:id", handlers.GetObservationLog(s.db))
193         observation.Post("/", handlers.CreateObservationLog(s.db))
194         observation.Put("/", handlers.UpdateObservationLog(s.db))
195         observation.Delete("/:id", handlers.DeleteObservationLog(s.db))
196         observation.Get("/", handlers.GetAllObservationLogs(s.db))
197 
198         // Maintenance routes
199         maintenance := api.Group("/maintenance", roleMiddleware(types.Worker,
                ↪ types.Manager, types.Admin))
200 
201         maintenance.Get("/:id", handlers.GetMaintenancePlan(s.db))
202         maintenance.Post("/", handlers.CreateMaintenancePlan(s.db))
203         maintenance.Put("/", handlers.UpdateMaintenancePlan(s.db))
204         maintenance.Delete("/:id", handlers.DeleteMaintenancePlan(s.db))
205         maintenance.Get("/", handlers.GetAllMaintenancePlans(s.db))
206         maintenance.Put("/:id/status", handlers.UpdateMaintenancePlanStatus(s.db))
207 
208 }
```

1.5.5 Пример авторизации

```
1 BASE_URL="http://localhost:4040"
2 API_URL="${BASE_URL}/api"
3 
4 curl -X POST "${BASE_URL}/auth/login" -H "Content-Type: application/json" -d '{"
      ↪ email_or_username": "john@example.com", "password": "password"}'
5 
6 # Response example
7 # {"token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
      ↪ eyJleHAiOjE3MzAwMjgzOTQsInJvbGUiOiJXT1JLRVIiLCJ1c2VyX2lkIjoxMDF9.8rim7ZBdT
      ↪ -o8K1PpPqpg5obK3is1U30nSa2dB52bqRM"}%
```

```
curl -X POST "${API_URL}/apiary" \
  -H "Content-Type: application/json" \
  -d '{"location": "Test Location", "manager_id": 1, "establishment_date":
     ↪ "2023-01-01T15:04:05Z"}' \·
 -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
     ↪ eyJleHAiOjE3MzAwMjEzNjYsInJvbGUiOiJXT1JLRVIiLCJ1c2VyX2lkIjoxMDF9.
     ↪ JP_KvMOAyivc2rJQnhC_ajgrwy9cJPjfdynLC6KbNSk"
# {"apiary_id":136,"location":"Test Location","manager_id":1,"establishment_date
    ↪ ":"2023-01-01T00:00:00Z"}
```

1.5.6 Вызов функций внутри psql

```
   import * as grpc from "@grpc/grpc-js";
import * as protoLoader from "@grpc/proto-loader";
import path from "node:path";

// Define the path to the proto file
const PROTO_PATH = path.join(__dirname, "../../proto/bee_management.proto");

// Load the protobuf
const packageDefinition = protoLoader.loadSync(PROTO_PATH, {
  keepCase: true,
  longs: String,
  enums: String,
  defaults: true,
  oneofs: true,
});

// Load the package definition
const protoDescriptor = grpc.loadPackageDefinition(packageDefinition) as any;

// Get the BeeManagementService
const beeManagement = protoDescriptor.bee_management.BeeManagementService;

// Create a client instance
const client = new beeManagement("localhost:50051", grpc.credentials.
    ↪ createInsecure());

// Helper function to promisify client methods
function promisifyClientMethod(method: Function) {
  return (...args: any[]) => {
    return new Promise((resolve, reject) => {
      method(...args, (error: any, response: any) => {
        if (error) {
          reject(error);
        } else {
          resolve(response);
        }
      });
    });
  };
}

// Promisified client methods
const getTotalHoneyHarvested = promisifyClientMethod(client.
    ↪ GetTotalHoneyHarvested.bind(client));
const addObservation = promisifyClientMethod(client.AddObservation.bind(client))
    ↪ ;
const getCommunityHealthStatus = promisifyClientMethod(
  client.GetCommunityHealthStatus.bind(client),
```

```
);
const updateHiveStatus = promisifyClientMethod(client.UpdateHiveStatus.bind(
    ↪ client));
const getAvgTemperature = promisifyClientMethod(client.GetAvgTemperature.bind(
    ↪ client));
const assignMaintenancePlan = promisifyClientMethod(client.AssignMaintenancePlan
    ↪ .bind(client));
const hasRegionAccess = promisifyClientMethod(client.HasRegionAccess.bind(client
    ↪ ));
const registerIncident = promisifyClientMethod(client.RegisterIncident.bind(
    ↪ client));
const getLatestSensorReading = promisifyClientMethod(client.
    ↪ GetLatestSensorReading.bind(client));
const createProductionReport = promisifyClientMethod(client.
    ↪ CreateProductionReport.bind(client));

async function main() {
  try {
    // 1. Get Total Honey Harvested
    const totalHoney = await getTotalHoneyHarvested({
      hive_id: 1,
      start_date: "2023-01-01",
      end_date: "2024-12-31",
    });
    console.log("Total Honey Harvested:", totalHoney.total_honey);

    // 2. Add Observation
    const addObsResponse = await addObservation({
      hive_id: 1,
      observation_date: "2023-04-15",
      description: "Queen is healthy.",
      recommendations: "Continue current beekeeping practices.",
    });
    console.log("Add Observation Response:", addObsResponse);

    // 3. Get Community Health Status
    const communityHealth = await getCommunityHealthStatus({
      community_id: 1,
    });
    console.log("Community Health Status:", communityHealth.health_status);

    // 4. Update Hive Status
    const updateStatusResponse = await updateHiveStatus({
      hive_id: 1,
      new_status: "Active",
    });
    console.log("Update Hive Status Response:", updateStatusResponse);

    // 5. Get Average Temperature
    const avgTemp = await getAvgTemperature({
      region_id: 5,
      days: 30,
    });
    console.log("Average Temperature:", avgTemp.avg_temperature);

    // 6. Assign Maintenance Plan
    const assignPlanResponse = await assignMaintenancePlan({
      plan_id: 7,
      user_id: 3,
    });
    console.log("Assign Maintenance Plan Response:", assignPlanResponse);
```

```
100
101     // 7. Has Region Access
102     const regionAccess = await hasRegionAccess({
103       user_id: 42,
104       region_id: 5,
105     });
106     console.log("Has Region Access:", regionAccess.has_access);
107
108     // 8. Register Incident
109     const registerIncidentResponse = await registerIncident({
110       hive_id: 1,
111       incident_date: "2023-05-20",
112       description: "Varroa mite infestation detected.",
113       severity: "High",
114     });
115     console.log("Register Incident Response:", registerIncidentResponse);
116
117     // 9. Get Latest Sensor Reading
118     const latestSensor = await getLatestSensorReading({
119       hive_id: 1,
120       sensor_type: "humidity",
121     });
122     console.log("Latest Sensor Reading:", latestSensor);
123
124     // 10. Create Production Report
125     const createReportResponse = await createProductionReport({
126       apiary_id: 1,
127       start_date: "2023-01-01",
128       end_date: "2023-06-30",
129     });
130     console.log("Create Production Report Response:", createReportResponse);
131   } catch (error) {
132     console.error("An error occurred:", error);
133   } finally {
134     client.close();
135   }
136 }
137
138 main();
```

1.6 Провести презентацию проекта.

Смотрите приложение 2