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 №2
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 второй части

Предметная область: гео-распределенная пасека

1.1 Подробное текстовое описание предметной области

1.1.1 Улей

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

1.1.2 Пчелосемья

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

1.1.3 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].

1.1.4 Запись о медосборе

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

1.1.5 Журнал наблюдений

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

1.1.6 Ветеринарный паспорт

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

1.1.7 Система управления

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

1.1.8 План обслуживания

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

1.1.9 Инциденты

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

1.1.10 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].

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

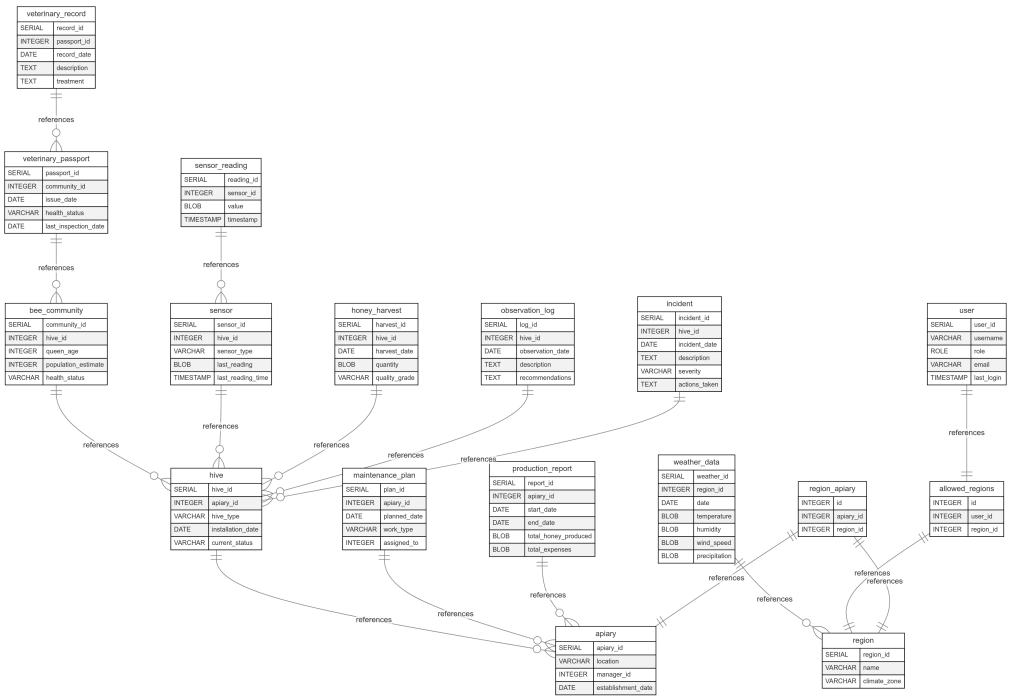

Рис. 1: ER диаграмма

1.3 Реализовать даталогическую модель в реляционной СУБД PostgreSQL

```sql
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');

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,
      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,
      "start_date" DATE,
```

```
163         "end_date" DATE,
164         "total_honey_produced" FLOAT,
165         "total_expenses" FLOAT,
166         PRIMARY KEY("report_id"),
167         UNIQUE("apiary_id", "start_date", "end_date")
168 );
169
170 CREATE TABLE IF NOT EXISTS "weather_data" (
171         "weather_id" SERIAL,
172         "region_id" INTEGER,
173         "date" DATE,
174         "temperature" FLOAT,
175         "humidity" FLOAT,
176         "wind_speed" FLOAT,
177         "precipitation" FLOAT,
178         PRIMARY KEY("weather_id")
179 );
180
181 CREATE TABLE IF NOT EXISTS "user" (
182         "user_id" SERIAL,
183         "username" VARCHAR NOT NULL UNIQUE,
184         "full_name" VARCHAR NOT NULL,
185         "role" ROLE,
186         "email" VARCHAR NOT NULL UNIQUE,
187         "password" VARCHAR NOT NULL,
188         "last_login" TIMESTAMP,
189         PRIMARY KEY("user_id")
190 );
191
192 CREATE TABLE IF NOT EXISTS "allowed_region" (
193         "id" SERIAL NOT NULL,
194         "user_id" INTEGER,
195         "region_id" INTEGER,
196         PRIMARY KEY("id"),
197         UNIQUE ("user_id", "region_id")
198 );
199
200 CREATE TABLE IF NOT EXISTS "region_apiary" (
201         "id" SERIAL NOT NULL UNIQUE,
202         "apiary_id" INTEGER,
203         "region_id" INTEGER,
204         PRIMARY KEY("id")
205 );
206
207 ALTER TABLE
208         "hive"
209 ADD
210         FOREIGN KEY("apiary_id") REFERENCES "apiary"("apiary_id") ON UPDATE NO
            ↪ ACTION ON DELETE CASCADE;
211
212 ALTER TABLE
213         "bee_community"
214 ADD
215         FOREIGN KEY("hive_id") REFERENCES "hive"("hive_id") ON UPDATE NO ACTION ON
            ↪  DELETE CASCADE;
216
217 ALTER TABLE
218         "veterinary_passport"
219 ADD
220         FOREIGN KEY("bee_community_id") REFERENCES "bee_community"("community_id")
            ↪  ON UPDATE NO ACTION ON DELETE CASCADE;
```

```
221
222 ALTER TABLE
223       "veterinary_record"
224 ADD
225       FOREIGN KEY("passport_id") REFERENCES "veterinary_passport"("passport_id")
          ↪  ON UPDATE NO ACTION ON DELETE CASCADE;
226
227 ALTER TABLE
228       "sensor"
229 ADD
230       FOREIGN KEY("hive_id") REFERENCES "hive"("hive_id") ON UPDATE NO ACTION ON
          ↪  DELETE CASCADE;
231
232 ALTER TABLE
233       "sensor_reading"
234 ADD
235       FOREIGN KEY("sensor_id") REFERENCES "sensor"("sensor_id") ON UPDATE NO
          ↪ ACTION ON DELETE CASCADE;
236
237 ALTER TABLE
238       "honey_harvest"
239 ADD
240       FOREIGN KEY("hive_id") REFERENCES "hive"("hive_id") ON UPDATE NO ACTION ON
          ↪  DELETE CASCADE;
241
242 ALTER TABLE
243       "observation_log"
244 ADD
245       FOREIGN KEY("hive_id") REFERENCES "hive"("hive_id") ON UPDATE NO ACTION ON
          ↪  DELETE CASCADE;
246
247 ALTER TABLE
248       "maintenance_plan"
249 ADD
250       FOREIGN KEY("apiary_id") REFERENCES "apiary"("apiary_id") ON UPDATE NO
          ↪ ACTION ON DELETE CASCADE;
251
252 ALTER TABLE
253       "incident"
254 ADD
255       FOREIGN KEY("hive_id") REFERENCES "hive"("hive_id") ON UPDATE NO ACTION ON
          ↪  DELETE CASCADE;
256
257 ALTER TABLE
258       "production_report"
259 ADD
260       FOREIGN KEY("apiary_id") REFERENCES "apiary"("apiary_id") ON UPDATE NO
          ↪ ACTION ON DELETE CASCADE;
261
262 ALTER TABLE
263       "weather_data"
264 ADD
265       FOREIGN KEY("region_id") REFERENCES "region"("region_id") ON UPDATE NO
          ↪ ACTION ON DELETE CASCADE;
266
267 ALTER TABLE
268       "allowed_region"
269 ADD
270       FOREIGN KEY("region_id") REFERENCES "region"("region_id") ON UPDATE NO
          ↪ ACTION ON DELETE NO ACTION;
271
```

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

ALTER TABLE
      "region_apiary"
ADD
      FOREIGN KEY("region_id") REFERENCES "region"("region_id") ON UPDATE NO
         ↪ ACTION ON DELETE NO ACTION;

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

ALTER TABLE
      "sensor"
ALTER COLUMN
      "last_reading_time"
SET
      DEFAULT now();

ALTER TABLE
      "sensor_reading"
ALTER COLUMN
      "timestamp"
SET
      DEFAULT now();
```

1.4 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 $$;
```

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

```
-- Function to populate the database with test data
CREATE OR REPLACE FUNCTION populate_test_data()
RETURNS VOID AS $$
BEGIN
    -- Insert test data for Region
    INSERT INTO Region (name, climate_zone) VALUES
    ('North', 'Temperate'),
    ('South', 'Mediterranean'),
    ('East', 'Continental'),
    ('West', 'Oceanic');

    -- Insert test data for Apiary
    INSERT INTO Apiary (location, establishment_date, region_id) VALUES
    ('Forest Edge', '2020-05-15', 1),
    ('Meadow', '2019-07-20', 2),
    ('Mountain Side', '2021-03-10', 3),
    ('Coastal Area', '2018-09-01', 4);

    -- Insert test data for Hive
    INSERT INTO Hive (apiary_id, installation_date) VALUES
    (1, '2020-06-01'),
    (1, '2020-06-02'),
    (2, '2019-08-01'),
    (3, '2021-04-01'),
```

```
    (4, '2018-10-01');

    -- Insert test data for BeeCommunity
    INSERT INTO BeeCommunity (hive_id, queen_age, population_estimate) VALUES
    (1, 2, 50000),
    (2, 1, 40000),
    (3, 3, 60000),
    (4, 1, 45000),
    (5, 2, 55000);

    -- Insert test data for User
    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;
```

```
86
87 -- Execute the function to populate the database
88 SELECT populate_test_data();
89
90 -- Confirm population
91 SELECT 'Test data inserted successfully!' AS result;
```

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

```
1     -- Create function to create triggers if they don't exist
2  CREATE OR REPLACE FUNCTION create_trigger_if_not_exists(
3     p_trigger_name TEXT,
4     p_table_name TEXT,
5     p_trigger_timing TEXT,
6     p_trigger_event TEXT,
7     p_trigger_function TEXT
8  ) RETURNS VOID AS $$
9  BEGIN
10    -- Check if the trigger already exists
11    IF NOT EXISTS (
12       SELECT 1
13       FROM information_schema.triggers
14       WHERE trigger_name = p_trigger_name
15         AND event_object_table = p_table_name
16    ) THEN
17       -- Create the trigger if it doesn't exist
18       EXECUTE format(
19          'CREATE TRIGGER %I
20           %s %s ON %I
21           FOR EACH ROW
22           EXECUTE FUNCTION %s()',
23          p_trigger_name,
24          p_trigger_timing,
25          p_trigger_event,
26          p_table_name,
27          p_trigger_function
28       );
29       RAISE NOTICE 'Trigger % created on table %', p_trigger_name, p_table_name;
30    ELSE
31       RAISE NOTICE 'Trigger % already exists on table %', p_trigger_name,
             ↪ p_table_name;
32    END IF;
33 END;
34 $$ LANGUAGE plpgsql;
35
36 -- Create triggers
37 CREATE OR REPLACE FUNCTION update_last_login()
38 RETURNS TRIGGER AS $$
39 BEGIN
40    NEW.last_login = CURRENT_TIMESTAMP;
41    RETURN NEW;
42 END;
43 $$ LANGUAGE plpgsql;
44
45 SELECT create_trigger_if_not_exists(
46    'update_user_last_login',
47    '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 $$
BEGIN
    IF NEW.role = 'ADMIN' THEN
        INSERT INTO "allowed_region" (user_id, region_id)
        SELECT NEW.user_id, r.region_id
        FROM "region" r
        ON CONFLICT ("user_id", "region_id") DO NOTHING;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

SELECT create_trigger_if_not_exists(
    'admin_access_trigger',
    'user',
    'AFTER',
    'INSERT OR UPDATE',
    'grant_admin_access'
);

CREATE OR REPLACE FUNCTION update_population_estimate()
RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'INSERT' THEN
        UPDATE "bee_community"
        SET population_estimate = population_estimate + NEW.quantity
        WHERE community_id = (SELECT community_id FROM bee_community WHERE hive_id
            ↪  = NEW.hive_id);
    ELSIF TG_OP = 'DELETE' THEN
        UPDATE "bee_community"
        SET population_estimate = population_estimate - OLD.quantity
        WHERE community_id = (SELECT community_id FROM bee_community WHERE hive_id
            ↪  = OLD.hive_id);
    END IF;
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

SELECT create_trigger_if_not_exists(
    'update_bee_population',
    'honey_harvest',
    'AFTER',
    'INSERT OR DELETE',
    'update_population_estimate'
);

-- Trigger to automatically create a veterinary passport for new bee communities
CREATE OR REPLACE FUNCTION create_veterinary_passport()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO "veterinary_passport" (bee_community_id, issue_date,
        ↪ last_inspection_date)
    VALUES (NEW.community_id, CURRENT_DATE, CURRENT_DATE);
```

```
105     RETURN NEW;
106 END;
107 $$ LANGUAGE plpgsql;
108 
109 SELECT create_trigger_if_not_exists(
110     'new_bee_community_passport',
111     'bee_community',
112     'AFTER',
113     'INSERT',
114     'create_veterinary_passport'
115 );
116 
117 -- Trigger to update production report when new honey harvest is added
118 CREATE OR REPLACE FUNCTION update_production_report()
119 RETURNS TRIGGER AS $$
120 BEGIN
121     INSERT INTO "production_report" (apiary_id, start_date, end_date,
        ↪ total_honey_produced)
122     VALUES (
123         (SELECT apiary_id FROM "hive" WHERE hive_id = NEW.hive_id),
124         DATE_TRUNC('month', NEW.harvest_date),
125         DATE_TRUNC('month', NEW.harvest_date) + INTERVAL '1 month' - INTERVAL '1
            ↪ day',
126         NEW.quantity
127     )
128     ON CONFLICT (apiary_id, start_date, end_date)
129     DO UPDATE SET total_honey_produced = "production_report".total_honey_produced
        ↪  + NEW.quantity;
130     RETURN NEW;
131 END;
132 $$ LANGUAGE plpgsql;
133 
134 SELECT create_trigger_if_not_exists(
135     'update_honey_production',
136     'honey_harvest',
137     'AFTER',
138     'INSERT',
139     'update_production_report'
140 );
141 
142 
143 -- 1. Функциядляполученияобщегоколичествамеда,
        ↪ собранноговконкретномульезаопределенныйпериод
144 CREATE OR REPLACE FUNCTION get_total_honey_harvested(
145     p_hive_id INTEGER,
146     p_start_date DATE,
147     p_end_date DATE
148 ) RETURNS DECIMAL AS $$
149 DECLARE
150     total_honey DECIMAL;
151 BEGIN
152     SELECT COALESCE(SUM(quantity::DECIMAL), 0)
153     INTO total_honey
154     FROM honey_harvest
155     WHERE hive_id = p_hive_id
156     AND harvest_date BETWEEN p_start_date AND p_end_date;
157 
158     RETURN total_honey;
159 END;
160 $$ LANGUAGE plpgsql;
```

```
161
162 -- 2. Процедурадлядобавленияновогонаблюденияв журнал
163 CREATE OR REPLACE PROCEDURE add_observation(
164     p_hive_id INTEGER,
165     p_observation_date DATE,
166     p_description TEXT,
167     p_recommendations TEXT
168 ) AS $$
169 BEGIN
170     INSERT INTO observation_log (hive_id, observation_date, description,
        ↪ recommendations)
171     VALUES (p_hive_id, p_observation_date, p_description, p_recommendations);
172 END;
173 $$ LANGUAGE plpgsql;
174
175 -- 3. Функциядляполученияте кущегосостоянияздоровьяпчелинойобщины
176 CREATE OR REPLACE FUNCTION get_community_health_status(p_community_id INTEGER)
177 RETURNS VARCHAR AS $$
178 DECLARE
179     health_status VARCHAR;
180 BEGIN
181     SELECT vp.health_status
182     INTO health_status
183     FROM veterinary_passport vp
184     WHERE vp.bee_community_id = p_community_id
185     ORDER BY vp.last_inspection_date DESC
186     LIMIT 1;
187
188     RETURN COALESCE(health_status, 'Unknown');
189 END;
190 $$ LANGUAGE plpgsql;
191
192 -- 4. Процедурадляобновлениястатусаулья
193 CREATE OR REPLACE PROCEDURE update_hive_status(
194     p_hive_id INTEGER,
195     p_new_status VARCHAR
196 ) AS $$
197 BEGIN
198     UPDATE hive
199     SET current_status = p_new_status
200     WHERE hive_id = p_hive_id;
201 END;
202 $$ LANGUAGE plpgsql;
203
204 -- 5. Функциядлярасчетасреднейтемпературыврегионезапоследние N дней
205 CREATE OR REPLACE FUNCTION get_avg_temperature(
206     p_region_id INTEGER,
207     p_days INTEGER
208 ) RETURNS DECIMAL AS $$
209 DECLARE
210     avg_temp DECIMAL;
211 BEGIN
212     SELECT AVG(temperature::DECIMAL)
213     INTO avg_temp
214     FROM weather_data
215     WHERE region_id = p_region_id
216     AND date ⩾ CURRENT_DATE - p_days;
217
218     RETURN COALESCE(avg_temp, 0);
219 END;
```

```
220 $$ LANGUAGE plpgsql;
221
222 -- 6. Процедурадляназначенияработниканаплануобслуживания
223 CREATE OR REPLACE PROCEDURE assign_maintenance_plan(
224     p_plan_id INTEGER,
225     p_user_id INTEGER
226 ) AS $$
227 BEGIN
228     UPDATE maintenance_plan
229     SET assigned_to = p_user_id
230     WHERE plan_id = p_plan_id;
231 END;
232 $$ LANGUAGE plpgsql;
233
234 -- 7. Функциядляпроверкидоступапользователякрегиону
235 CREATE OR REPLACE FUNCTION has_region_access(
236     p_user_id INTEGER,
237     p_region_id INTEGER
238 ) RETURNS BOOLEAN AS $$
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 $$ LANGUAGE plpgsql;
251
252 -- 8. Процедурадлярегистрациииинцидента
253 CREATE OR REPLACE PROCEDURE register_incident(
254     p_hive_id INTEGER,
255     p_incident_date DATE,
256     p_description TEXT,
257     p_severity VARCHAR
258 ) AS $$
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;
```

1.7 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
       ↪ );
```