ЗАДАНИЕ

Разработать информационное обеспечение автоматизированной системы управления сетью спортивных клубов.

Функции, выполняемые проектируемыми элементами:

1) Хранение, ввод и изменение следующей информации:
   - данные о спортивных клубах;
   - данные о работниках клуба (тренер/администратор);
   - данные о посетителях;
   - данные об общих тренировках для посетителей;
   - данные о личных тренировках посетителей.
2) Реализация ограничений целостности данных средствами СУБД.
3) Обеспечение безопасного доступа к БД со стороны пользователей и программ.

Требования к проектируемым элементам.

1) Для реализации программного обеспечения должна использоваться СУБД PostgreSQL.
2) Требования к среде реализации интерфейса пользователя не предъявляются.
3) БД должна обеспечивать оперативный доступ к хранящейся в ней информации (время отклика на "простые" запросы не более 10 секунд).
4) БД должна хранить следующую информацию о спортивных клубах:
   - Улица (обязательно);
   - Номер дома (обязательно);
   - Телефон (обязательно).
5) БД должна хранить следующую информацию о работниках:
   - Ф.И.О. (обязательно);
   - Серия и номер паспорта (обязательно);

- Дата рождения (обязательно);
- Пол (обязательно);
- Табельный номер (обязательно);
- Данные о квалификации (обязательно);
- Указание клуба, где он работает (обязательно)
- Роль работника (тренер/администратор) (обязательно).

5) БД должна хранить следующую информацию о посетителях:
   - Ф.И.О. (обязательно);
   - Серия и номер паспорта (обязательно);
   - Дата рождения (обязательно);
   - Пол (обязательно);
   - Дата окончания абонемента (обязательно);
   - Количество оставшихся тренировок (обязательно);
   - Указание на клуб, в котором он занимается (обязательно).
6) БД должна хранить следующую информацию об общих тренировках для посетителей:
   - Название (обязательно);
   - Описание (обязательно);
   - Дата и время проведения (обязательно);
   - Указание на тренера, который ведёт тренировку (обязательно);
   - Указание клуба, где проходит тренировка (обязательно);
   - Количество мест (обязательно).
7) БД должна хранить следующую информацию о личных тренировках для посетителей:
   - Указание на тренера, который проводит тренировку (обязательно);
   - Указание на посетителя, который проходит тренировку;
   - Заметка тренера после тренировки;
   - Дата и время тренировки (обязательно).
8) БД должна удовлетворять следующим ограничениям целостности и правилам обработки:

- Очевидные общие правила целостности (потенциальные ключи и внешние ключи);
- Можно записаться на тренировку только позже текущего времени;
- Нельзя записаться на общую тренировку, если достигнут лимит посетителей;
- Тренировку может везти только тренер из того же клуба;
- Посетитель не может записаться на тренировку к тренеру из другого клуба.

9) БД должна удовлетворять следующим требованиям по безопасности:
   - Для структур, перечисленных в таблице, необходимо выполнять аудит в части регистрации пользователя, выполнявшего последним изменения в БД.

1. Описание предметной области

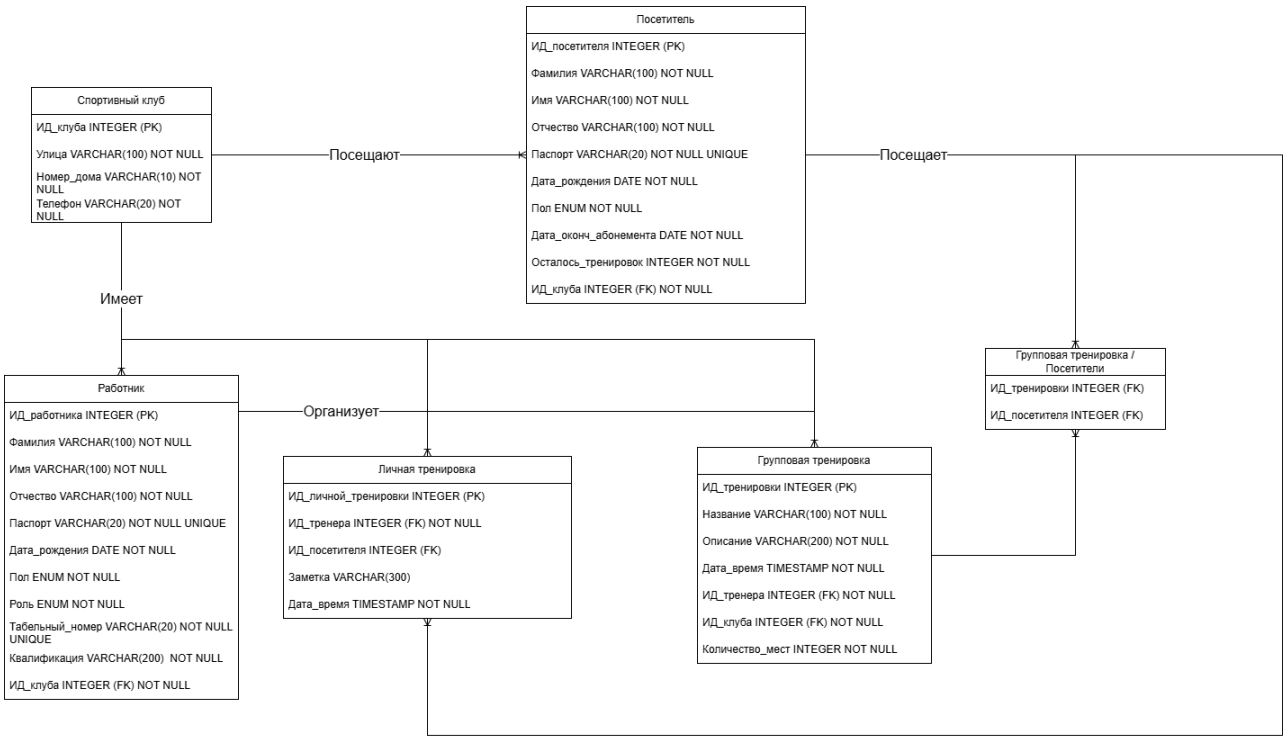
Информационная автоматизированная система для управления сетью спортивных клубов.

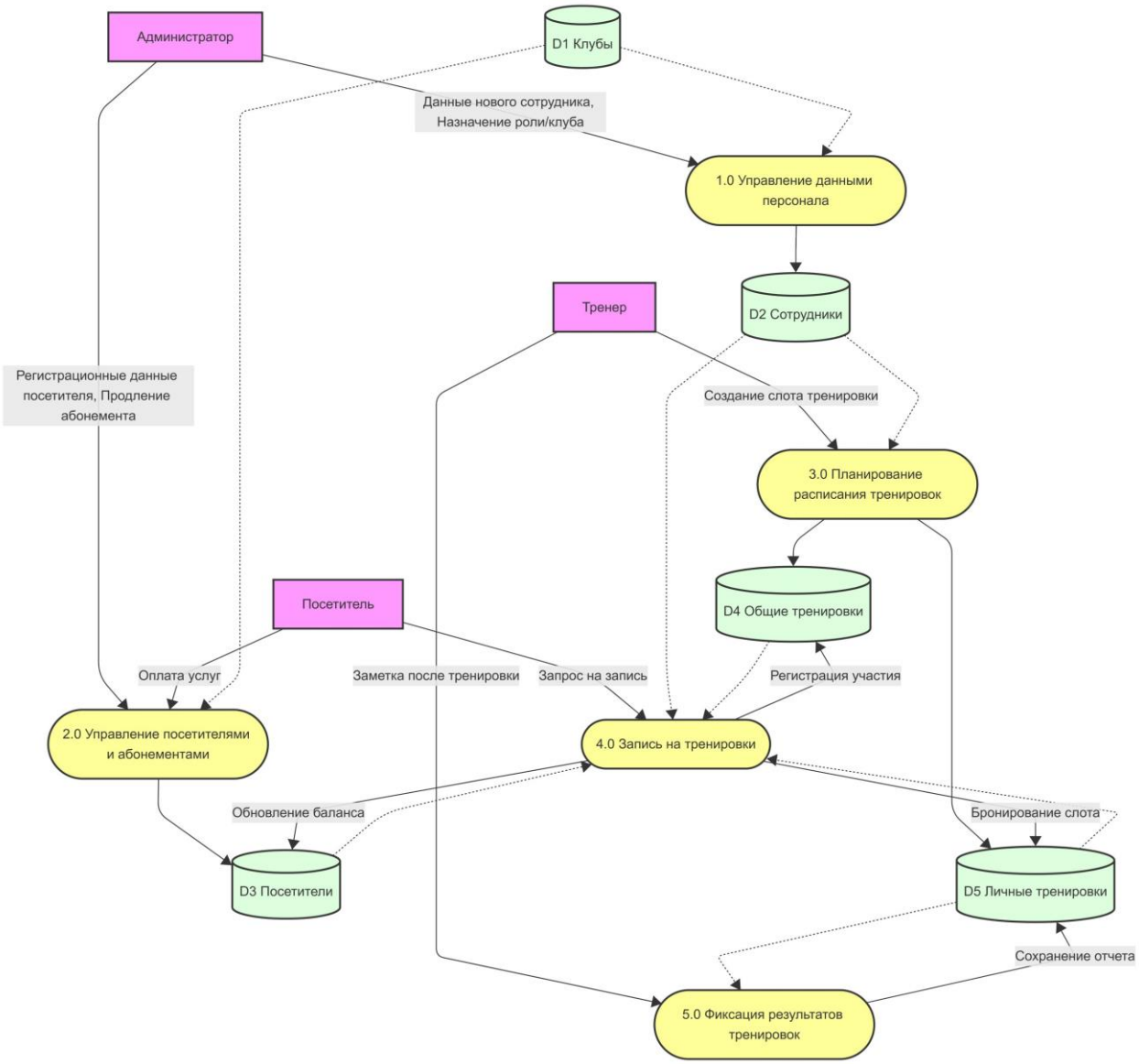
Для регистрации посетителей задействован администратор клуба. Он вносит в БД данные о посетителе (ФИО, паспортные данные и т.д.). Далее, клиент оплачивает свой абонемент для того чтобы пользоваться базовыми услугами клуба. С помощью администратора происходит продление абонемента и покупка дополнительных тренировок.

Также, в клубе есть несколько дополнительных услуг. Личная тренировка – тренера добавляют в БД список тренировок, в которые к ним могут записаться посетители. Для записи посетитель выбирает ещё не занятую личную тренировку и у него должна оставаться хотя бы 1 тренировка. После тренировки, тренер может оставить заметку, чтобы зафиксировать результаты посетителя, указать на что стоит обратить внимание на следующем занятии.

Ещё, в клубе есть общие тренировки. В них могут участвовать более одного посетителя. Принцип записи тот же: тренер создаёт запись об общей тренировке, пользователи записываются на те, в которых ещё есть свободные места.

В свою очередь администратор заносит новую информацию в базу данных при принятии на работу нового рабочего персонала, а также контролирует работу с БД и имеет право на любые операции (выбора, записи, удаления, изменения) с данными.


Посетитель
ИД_посетителя INTEGER (PK)
Фамилия VARCHAR(100) NOT NULL
Имя VARCHAR(100) NOT NULL
Отчество VARCHAR(100) NOT NULL
Паспорт VARCHAR(20) NOT NULL UNIQUE
Дата_рождения DATE NOT NULL
Пол ENUM NOT NULL
Дата_оконч_абонемента DATE NOT NULL
Осталось_тренировок INTEGER NOT NULL
ИД_клуба INTEGER (FK) NOT NULL
Спортивный клуб
ИД_клуба INTEGER (PK)
Улица VARCHAR(100) NOT NULL
Номер_дома VARCHAR(10) NOT NULL
Телефон VARCHAR(20) NOT NULL
Посещают
Посещает
Имеет
Работник
ИД_работника INTEGER (PK)
Фамилия VARCHAR(100) NOT NULL
Имя VARCHAR(100) NOT NULL
Отчество VARCHAR(100) NOT NULL
Паспорт VARCHAR(20) NOT NULL UNIQUE
Дата_рождения DATE NOT NULL
Пол ENUM NOT NULL
Роль ENUM NOT NULL
Табельный_номер VARCHAR(20) NOT NULL UNIQUE
Квалификация VARCHAR(200) NOT NULL
ИД_клуба INTEGER (FK) NOT NULL
Организует
Личная тренировка
ИД_личной_тренировки INTEGER (PK)
ИД_тренера INTEGER (FK) NOT NULL
ИД_посетителя INTEGER (FK)
Заметка VARCHAR(300)
Дата_время TIMESTAMP NOT NULL
Групповая тренировка
ИД_тренировки INTEGER (PK)
Название VARCHAR(100) NOT NULL
Описание VARCHAR(200) NOT NULL
Дата_время TIMESTAMP NOT NULL
ИД_тренера INTEGER (FK) NOT NULL
ИД_клуба INTEGER (FK) NOT NULL
Количество_мест INTEGER NOT NULL
Групповая тренировка / Посетители
ИД_тренировки INTEGER (FK)
ИД_посетителя INTEGER (FK)


Администратор
D1 Клубы
Данные нового сотрудника, Назначение роли/клуба
1.0 Управление данными персонала
Тренер
D2 Сотрудники
Регистрационные данные посетителя, Продление абонемента
Создание слота тренировки
3.0 Планирование расписания тренировок
Посетитель
D4 Общие тренировки
Оплата услуг
Заметка после тренировки
Запрос на запись
Регистрация участия
2.0 Управление посетителями и абонементами
4.0 Запись на тренировки
Обновление баланса
Бронирование слота
D3 Посетители
D5 Личные тренировки
Сохранение отчета
5.0 Фиксация результатов тренировок

2. Разработка логической структуры БД

2.1. SQL-скрипты, генерация объектов БД

После создания логической структуры БД, а именно ER-диаграммы, были выполнены SQL-скрипты для создания таблиц. Далее с помощью инструмента PL/SQL Developer были сгенерированы сначала необходимые последовательности, которые требуются для функций автоинкремента, а затем и сами пакеты. После генерации пакетов из шаблона, в них были добавлены и реализованы, перечисленные в задании ограничения целостности.

2.1.1. Генерация последовательностей

```
CREATE SEQUENCE IF NOT EXISTS seq_club_id
    START WITH 1
    INCREMENT BY 1
    MINVALUE 1
    MAXVALUE 99999999999
    CACHE 20;

CREATE SEQUENCE IF NOT EXISTS seq_employee_id
    START WITH 1
    INCREMENT BY 1
    MINVALUE 1
    MAXVALUE 99999999999
    CACHE 20;

CREATE SEQUENCE IF NOT EXISTS seq_visitor_id
    START WITH 1
    INCREMENT BY 1
    MINVALUE 1
    MAXVALUE 99999999999
    CACHE 20;

CREATE SEQUENCE IF NOT EXISTS seq_group_training_id
    START WITH 1
    INCREMENT BY 1
    MINVALUE 1
```

```
    MAXVALUE 99999999999
    CACHE 20;

CREATE SEQUENCE IF NOT EXISTS seq_personal_training_id
    START WITH 1
    INCREMENT BY 1
    MINVALUE 1
    MAXVALUE 99999999999
    CACHE 20;
```

2.1.2. Генерация таблиц.

```
CREATE TYPE gender_enum AS ENUM ('M', 'F');

CREATE TYPE role_enum AS ENUM ('тренер',
'администратор');

CREATE TABLE IF NOT EXISTS club (
    id_club INTEGER PRIMARY KEY DEFAULT
nextval('seq_club_id'),
    street VARCHAR(100) NOT NULL,
    house_number VARCHAR(10) NOT NULL,
    phone VARCHAR(20) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS employee (
    id_employee INTEGER PRIMARY KEY DEFAULT
nextval('seq_employee_id'),
    second_name VARCHAR(100) NOT NULL,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    passport VARCHAR(20) NOT NULL UNIQUE,
    birth_date DATE NOT NULL,
    gender gender_enum NOT NULL,
    employee_number VARCHAR(20) NOT NULL UNIQUE,
    qualification VARCHAR(200) NOT NULL,
    role role_enum NOT NULL,
    id_club INTEGER NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (id_club) REFERENCES club(id_club) ON
DELETE CASCADE
```

```
);

CREATE TABLE IF NOT EXISTS visitor (
    id_visitor INTEGER PRIMARY KEY DEFAULT
nextval('seq_visitor_id'),
    second_name VARCHAR(100) NOT NULL,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    passport VARCHAR(20) NOT NULL UNIQUE,
    birth_date DATE NOT NULL,
    gender gender_enum NOT NULL,
    subscription_end_date DATE NOT NULL,
    remaining_trainings INTEGER NOT NULL CHECK
(remaining_sessions >= 0),
    id_club INTEGER NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (id_club) REFERENCES club(id_club) ON
DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS group_training (
    id_training INTEGER PRIMARY KEY DEFAULT
nextval('seq_group_training_id'),
    name VARCHAR(100) NOT NULL,
    description VARCHAR(200) NOT NULL,
    training_datetime TIMESTAMP NOT NULL,
    id_trainer INTEGER NOT NULL,
    id_club INTEGER NOT NULL,
    capacity INTEGER NOT NULL CHECK (capacity > 0),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (id_trainer) REFERENCES
employee(id_employee) ON DELETE CASCADE,
    FOREIGN KEY (id_club) REFERENCES club(id_club) ON
DELETE CASCADE,
    CONSTRAINT fk_trainer_role CHECK (
        EXISTS (
            SELECT 1 FROM employee e
            WHERE e.id_employee = id_trainer AND e.role =
'тренер'
        )
    )
);
```

```
CREATE TABLE IF NOT EXISTS personal_training (
    id_personal_training INTEGER PRIMARY KEY DEFAULT
nextval('seq_personal_training_id'),
    id_trainer INTEGER NOT NULL,
    id_visitor INTEGER,
    note VARCHAR(300),
    training_datetime TIMESTAMP NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (id_trainer) REFERENCES
employee(id_employee) ON DELETE CASCADE,
    FOREIGN KEY (id_visitor) REFERENCES
visitor(id_visitor) ON DELETE SET NULL,
    CONSTRAINT fk_personal_trainer_role CHECK (
        EXISTS (
            SELECT 1 FROM employee e
            WHERE e.id_employee = id_trainer AND e.role =
'тренер'
        )
    )
);

CREATE TABLE IF NOT EXISTS visitor_group_training (
    id_visitor INTEGER NOT NULL,
    id_training INTEGER NOT NULL,
    PRIMARY KEY (id_visitor, id_training),
    FOREIGN KEY (id_visitor) REFERENCES
visitor(id_visitor) ON DELETE CASCADE,
    FOREIGN KEY (id_training) REFERENCES
group_training(id_training) ON DELETE CASCADE
);
```

2.1.3. Генерация пакетов

```
CREATE OR REPLACE PACKAGE BODY enum_utils AS
    FUNCTION get_gender_values()
    RETURNS TABLE (value VARCHAR, description VARCHAR) AS $$
    BEGIN
        RETURN QUERY
        SELECT
            enumlabel::VARCHAR as value,
            CASE enumlabel
```

```
                WHEN 'M' THEN 'Мужской'
                WHEN 'F' THEN 'Женский'
            END as description
        FROM pg_enum
        WHERE enumtypid = 'gender_enum'::regtype
        ORDER BY enumsortorder;
    END;

    FUNCTION get_role_values()
    RETURNS TABLE (value VARCHAR, description VARCHAR) AS $$
    BEGIN
        RETURN QUERY
        SELECT
            enumlabel::VARCHAR as value,
            CASE enumlabel
                WHEN 'тренер' THEN 'Тренер'
                WHEN 'администратор' THEN 'Администратор'
            END as description
        FROM pg_enum
        WHERE enumtypid = 'role_enum'::regtype
        ORDER BY enumsortorder;
    END;

    FUNCTION is_valid_gender(p_gender VARCHAR)
    RETURNS BOOLEAN AS $$
    DECLARE
        v_count INTEGER;
    BEGIN
        SELECT COUNT(*) INTO v_count
        FROM pg_enum
        WHERE enumtypid = 'gender_enum'::regtype
          AND enumlabel = p_gender;

        RETURN v_count > 0;
    END;

    FUNCTION is_valid_role(p_role VARCHAR)
    RETURNS BOOLEAN AS $$
    DECLARE
        v_count INTEGER;
    BEGIN
        SELECT COUNT(*) INTO v_count
        FROM pg_enum
        WHERE enumtypid = 'role_enum'::regtype
          AND enumlabel = p_role;

        RETURN v_count > 0;
    END;
END enum_utils;

CREATE OR REPLACE PACKAGE BODY club_management AS
    FUNCTION is_phone_unique(p_phone VARCHAR, p_exclude_id INTEGER DEFAULT NULL)
    RETURN BOOLEAN AS
        v_count INTEGER;
    BEGIN
        SELECT COUNT(*) INTO v_count
        FROM club
        WHERE phone = p_phone
          AND (p_exclude_id IS NULL OR id_club != p_exclude_id);

        RETURN v_count = 0;
    END is_phone_unique;

    PROCEDURE add_club(
        p_street VARCHAR,
        p_house_number VARCHAR,
        p_phone VARCHAR,
        p_result OUT INTEGER,
        p_message OUT VARCHAR
    ) AS
        v_club_id INTEGER;
    BEGIN
        IF p_street IS NULL OR TRIM(p_street) = '' THEN
            p_result := -1;
            p_message := 'Улица не может быть пустой';
            RETURN;
        END IF;

        IF p_house_number IS NULL OR TRIM(p_house_number) = '' THEN
            p_result := -1;
```

```
        p_message := 'Номер дома не может быть пустым';
        RETURN;
    END IF;

    IF p_phone IS NULL OR TRIM(p_phone) = '' THEN
        p_result := -1;
        p_message := 'Телефон не может быть пустым';
        RETURN;
    END IF;

    IF NOT p_phone ~ '^[\+\d\s\-\(\)]+$' THEN
        p_result := -1;
        p_message := 'Некорректный формат телефона';
        RETURN;
    END IF;

    INSERT INTO club (street, house_number, phone)
    VALUES (
        TRIM(p_street),
        TRIM(p_house_number),
        TRIM(p_phone)
    )
    RETURNING id_club INTO v_club_id;

    p_result := v_club_id;
    p_message := 'Клуб успешно добавлен с ID: ' || v_club_id;

    INSERT INTO audit_log (
        table_name, operation, record_id, user_name, details
    ) VALUES (
        'club', 'INSERT', v_club_id, CURRENT_USER,
        'Добавлен клуб: ' || TRIM(p_street) || ', ' || TRIM(p_house_number)
    );

    COMMIT;

EXCEPTION
    WHEN OTHERS THEN
        p_result := -1;
        p_message := 'Ошибка при добавлении клуба: ' || SQLERRM;
        ROLLBACK;
END add_club;

PROCEDURE update_club(
    p_id_club INTEGER,
    p_street VARCHAR DEFAULT NULL,
    p_house_number VARCHAR DEFAULT NULL,
    p_phone VARCHAR DEFAULT NULL,
    p_result OUT INTEGER,
    p_message OUT VARCHAR
) AS
    v_old_street VARCHAR;
    v_old_house VARCHAR;
    v_old_phone VARCHAR;
    v_update_count INTEGER;
BEGIN
    SELECT street, house_number, phone
    INTO v_old_street, v_old_house, v_old_phone
    FROM club
    WHERE id_club = p_id_club;

    IF NOT FOUND THEN
        p_result := -1;
        p_message := 'Клуб с ID ' || p_id_club || ' не найден';
        RETURN;
    END IF;

    IF p_phone IS NOT NULL AND TRIM(p_phone) != v_old_phone THEN
        IF NOT TRIM(p_phone) ~ '^[\+\d\s\-\(\)]+$' THEN
            p_result := -1;
            p_message := 'Некорректный формат телефона';
            RETURN;
        END IF;
    END IF;

    UPDATE club
    SET
        street = COALESCE(TRIM(p_street), street),
        house_number = COALESCE(TRIM(p_house_number), house_number),
        phone = COALESCE(TRIM(p_phone), phone),
```

```
        updated_at = CURRENT_TIMESTAMP
    WHERE id_club = p_id_club
    RETURNING 1 INTO v_update_count;

    IF v_update_count = 1 THEN
        p_result := p_id_club;
        p_message := 'Данные клуба успешно обновлены';

        INSERT INTO audit_log (
            table_name, operation, record_id, user_name, details
        ) VALUES (
            'club', 'UPDATE', p_id_club, CURRENT_USER,
            'Обновлен клуб. Старые данные: ' ||
            v_old_street || ', ' || v_old_house || ', ' || v_old_phone ||
            '. Новые данные: ' ||
            COALESCE(TRIM(p_street), v_old_street) || ', ' ||
            COALESCE(TRIM(p_house_number), v_old_house) || ', ' ||
            COALESCE(TRIM(p_phone), v_old_phone)
        );

        COMMIT;
    ELSE
        p_result := -1;
        p_message := 'Клуб не был обновлен';
    END IF;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        p_result := -1;
        p_message := 'Клуб с ID ' || p_id_club || ' не найден';
    WHEN OTHERS THEN
        p_result := -1;
        p_message := 'Ошибка при обновлении клуба: ' || SQLERRM;
        ROLLBACK;
END update_club;

PROCEDURE delete_club(
    p_id_club INTEGER,
    p_cascade BOOLEAN DEFAULT FALSE,
    p_result OUT INTEGER,
    p_message OUT VARCHAR
) AS
    v_club_info VARCHAR;
    v_has_employees BOOLEAN;
    v_has_visitors BOOLEAN;
    v_has_trainings BOOLEAN;
BEGIN
    SELECT street || ', д.' || house_number || ', тел: ' || phone
    INTO v_club_info
    FROM club
    WHERE id_club = p_id_club;

    IF NOT p_cascade THEN
        SELECT EXISTS(SELECT 1 FROM employee WHERE id_club = p_id_club)
        INTO v_has_employees;

        SELECT EXISTS(SELECT 1 FROM visitor WHERE id_club = p_id_club)
        INTO v_has_visitors;

        SELECT EXISTS(SELECT 1 FROM group_training WHERE id_club = p_id_club)
        INTO v_has_trainings;

        IF v_has_employees OR v_has_visitors OR v_has_trainings THEN
            p_result := -1;
            p_message := 'Нельзя удалить клуб, так как есть связанные записи. ' ||
                         'Используйте p_cascade = TRUE для каскадного удаления.';
            RETURN;
        END IF;
    END IF;

    DELETE FROM club
    WHERE id_club = p_id_club
    RETURNING 1 INTO p_result;

    IF p_result = 1 THEN
        p_message := 'Клуб успешно удален: ' || v_club_info;

        COMMIT;
    ELSE
        p_result := -1;
```

```
            p_message := 'Клуб не найден';
        END IF;

    EXCEPTION
        WHEN OTHERS THEN
            p_result := -1;
            p_message := 'Ошибка при удалении клуба: ' || SQLERRM;
            ROLLBACK;
    END delete_club;
END club_pkg;

CREATE OR REPLACE PACKAGE BODY employee_pkg AS
    FUNCTION validate_employee_data(
        p_passport VARCHAR,
        p_employee_number VARCHAR,
        p_birth_date DATE,
        p_exclude_id INTEGER DEFAULT NULL
    ) RETURN VARCHAR AS
        v_error VARCHAR;
    BEGIN
        IF EXISTS (
            SELECT 1 FROM employee
            WHERE passport = p_passport
              AND (p_exclude_id IS NULL OR id_employee != p_exclude_id)
        ) THEN
            RETURN 'Паспорт уже используется другим работником';
        END IF;

        IF EXISTS (
            SELECT 1 FROM employee
            WHERE employee_number = p_employee_number
              AND (p_exclude_id IS NULL OR id_employee != p_exclude_id)
        ) THEN
            RETURN 'Табельный номер уже используется';
        END IF;

        RETURN NULL;
    END validate_employee_data;

    PROCEDURE add_employee(
        p_second_name VARCHAR,
        p_first_name VARCHAR,
        p_last_name VARCHAR,
        p_passport VARCHAR,
        p_birth_date DATE,
        p_gender VARCHAR,
        p_employee_number VARCHAR,
        p_qualification TEXT,
        p_role VARCHAR,
        p_id_club INTEGER,
        p_result OUT INTEGER,
        p_message OUT VARCHAR
    ) AS
        v_employee_id INTEGER;
        v_validation_error VARCHAR;
        v_club_exists BOOLEAN;
    BEGIN

        IF p_passport IS NULL OR TRIM(p_passport) = '' THEN
            p_result := -1;
            p_message := 'Паспорт не может быть пустым';
            RETURN;
        END IF;

        IF p_employee_number IS NULL OR TRIM(p_employee_number) = '' THEN
            p_result := -1;
            p_message := 'Табельный номер не может быть пустым';
            RETURN;
        END IF;

        IF p_qualification IS NULL OR TRIM(p_qualification) = '' THEN
            p_result := -1;
            p_message := 'Квалификация не может быть пустой';
            RETURN;
        END IF;

        IF NOT enum_utils.is_valid_gender(p_gender) THEN
            p_result := -1;
            p_message := 'Некорректное значение для пола: ' || p_gender;
            RETURN;
```

```
    END IF;

    IF NOT enum_utils.is_valid_role(p_role) THEN
        p_result := -1;
        p_message := 'Некорректное значение для роли: ' || p_role;
        RETURN;
    END IF;

    SELECT EXISTS(SELECT 1 FROM club WHERE id_club = p_id_club)
    INTO v_club_exists;

    IF NOT v_club_exists THEN
        p_result := -1;
        p_message := 'Клуб с ID ' || p_id_club || ' не существует';
        RETURN;
    END IF;

    v_validation_error := validate_employee_data(
        TRIM(p_passport),
        TRIM(p_employee_number),
        p_birth_date
    );

    IF v_validation_error IS NOT NULL THEN
        p_result := -1;
        p_message := v_validation_error;
        RETURN;
    END IF;

    INSERT INTO employee (
        second_name, first_name, last_name, passport, birth_date, gender,
        employee_number, qualification, role, id_club
    ) VALUES (
        TRIM(p_second_name),
        TRIM(p_first_name),
        TRIM(p_last_name),
        TRIM(p_passport),
        p_birth_date,
        p_gender::gender_enum,
        TRIM(p_employee_number),
        TRIM(p_qualification),
        p_role::role_enum,
        p_id_club
    )
    RETURNING id_employee INTO v_employee_id;

    p_result := v_employee_id;
    p_message := 'Работник успешно добавлен с ID: ' || v_employee_id;

    COMMIT;

EXCEPTION
    WHEN OTHERS THEN
        p_result := -1;
        p_message := 'Ошибка при добавлении работника: ' || SQLERRM;
        ROLLBACK;
END add_employee;

PROCEDURE update_employee(
    p_id_employee INTEGER,
    p_second_name VARCHAR DEFAULT NULL,
    p_first_name VARCHAR DEFAULT NULL,
    p_last_name VARCHAR DEFAULT NULL,
    p_qualification TEXT DEFAULT NULL,
    p_role VARCHAR DEFAULT NULL,
    p_id_club INTEGER DEFAULT NULL,
    p_result OUT INTEGER,
    p_message OUT VARCHAR
) AS
    v_old_record employee%ROWTYPE;
    v_validation_error VARCHAR;
    v_club_exists BOOLEAN;
    v_update_count INTEGER;
BEGIN
    SELECT * INTO v_old_record
    FROM employee
    WHERE id_employee = p_id_employee;

    IF NOT FOUND THEN
        p_result := -1;
```

```
        p_message := 'Работник с ID ' || p_id_employee || ' не найден';
        RETURN;
    END IF;

    IF p_role IS NOT NULL AND NOT enum_utils.is_valid_role(p_role) THEN
        p_result := -1;
        p_message := 'Некорректное значение для роли: ' || p_role;
        RETURN;
    END IF;

    IF p_id_club IS NOT NULL AND p_id_club != v_old_record.id_club THEN
        SELECT EXISTS(SELECT 1 FROM club WHERE id_club = p_id_club)
        INTO v_club_exists;

        IF NOT v_club_exists THEN
            p_result := -1;
            p_message := 'Клуб с ID ' || p_id_club || ' не существует';
            RETURN;
        END IF;
    END IF;

    UPDATE employee
    SET
        second_name = COALESCE(TRIM(p_second_name), second_name),
        first_name = COALESCE(TRIM(p_first_name), first_name),
        last_name = COALESCE(TRIM(p_last_name), last_name),
        qualification = COALESCE(TRIM(p_qualification), qualification),
        role = COALESCE(p_role::role_enum, role),
        id_club = COALESCE(p_id_club, id_club),
        updated_at = CURRENT_TIMESTAMP
    WHERE id_employee = p_id_employee
    RETURNING 1 INTO v_update_count;

    IF v_update_count = 1 THEN
        p_result := p_id_employee;
        p_message := 'Данные работника успешно обновлены';

        COMMIT;
    ELSE
        p_result := -1;
        p_message := 'Работник не был обновлен';
    END IF;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        p_result := -1;
        p_message := 'Работник с ID ' || p_id_employee || ' не найден';
    WHEN OTHERS THEN
        p_result := -1;
        p_message := 'Ошибка при обновлении работника: ' || SQLERRM;
        ROLLBACK;
END update_employee;

PROCEDURE dismiss_employee(
    p_id_employee INTEGER,
    p_dismissal_date DATE DEFAULT CURRENT_DATE,
    p_reason TEXT DEFAULT NULL,
    p_result OUT INTEGER,
    p_message OUT VARCHAR
) AS
    v_employee_name VARCHAR;
    v_is_trainer BOOLEAN;
    v_active_trainings INTEGER;
BEGIN
    SELECT second_name, role = 'тренер'::role_enum
    INTO v_employee_name, v_is_trainer
    FROM employee
    WHERE id_employee = p_id_employee;

    IF NOT FOUND THEN
        p_result := -1;
        p_message := 'Работник с ID ' || p_id_employee || ' не найден';
        RETURN;
    END IF;

    IF v_is_trainer THEN
        SELECT COUNT(*) INTO v_active_trainings
        FROM (
            SELECT 1 FROM group_training
            WHERE id_trainer = p_id_employee
```

```
                      AND training_datetime > CURRENT_TIMESTAMP
                    UNION ALL
                    SELECT 1 FROM personal_training
                    WHERE id_trainer = p_id_employee
                      AND training_datetime > CURRENT_TIMESTAMP
                ) t;

                IF v_active_trainings > 0 THEN
                    p_result := -1;
                    p_message := 'Невозможно уволить тренера, у которого есть будущие тренировки
('  ||
                                 v_active_trainings || ' тренировок)';
                    RETURN;
                END IF;
            END IF;

            DELETE FROM employee
            WHERE id_employee = p_id_employee
            RETURNING 1 INTO p_result;

            IF p_result = 1 THEN
                p_message := 'Работник ' || v_employee_name || ' успешно уволен';

                COMMIT;
            END IF;

        EXCEPTION
            WHEN OTHERS THEN
                p_result := -1;
                p_message := 'Ошибка при увольнении работника: ' || SQLERRM;
                ROLLBACK;
        END dismiss_employee;

        PROCEDURE transfer_employee(
            p_id_employee INTEGER,
            p_new_club_id INTEGER,
            p_result OUT INTEGER,
            p_message OUT VARCHAR
        ) AS
            v_old_club_id INTEGER;
            v_employee_name VARCHAR;
            v_club_exists BOOLEAN;
        BEGIN
            SELECT second_name, id_club
            INTO v_employee_name, v_old_club_id
            FROM employee
            WHERE id_employee = p_id_employee;

            IF NOT FOUND THEN
                p_result := -1;
                p_message := 'Работник с ID ' || p_id_employee || ' не найден';
                RETURN;
            END IF;

            IF v_old_club_id = p_new_club_id THEN
                p_result := -1;
                p_message := 'Работник уже работает в этом клубе';
                RETURN;
            END IF;

            SELECT EXISTS(SELECT 1 FROM club WHERE id_club = p_new_club_id)
            INTO v_club_exists;

            IF NOT v_club_exists THEN
                p_result := -1;
                p_message := 'Клуб с ID ' || p_new_club_id || ' не существует';
                RETURN;
            END IF;

            UPDATE employee
            SET id_club = p_new_club_id,
                updated_at = CURRENT_TIMESTAMP
            WHERE id_employee = p_id_employee
            RETURNING 1 INTO p_result;

            IF p_result = 1 THEN
                p_message := 'Работник ' || v_employee_name ||
                             ' переведен из клуба ' || v_old_club_id ||
                             ' в клуб ' || p_new_club_id;
```

```
            COMMIT;
        END IF;

    EXCEPTION
        WHEN OTHERS THEN
            p_result := -1;
            p_message := 'Ошибка при переводе работника: ' || SQLERRM;
            ROLLBACK;
    END transfer_employee;
END employee_pkg;


CREATE OR REPLACE PACKAGE BODY visitor_pkg AS
    FUNCTION validate_visitor_data(
        p_passport VARCHAR,
        p_birth_date DATE,
        p_subscription_months INTEGER,
        p_initial_sessions INTEGER,
        p_exclude_id INTEGER DEFAULT NULL
    ) RETURN VARCHAR AS
    BEGIN
        IF EXISTS (
            SELECT 1 FROM visitor
            WHERE passport = p_passport
              AND (p_exclude_id IS NULL OR id_visitor != p_exclude_id)
        ) THEN
            RETURN 'Паспорт уже используется другим посетителем';
        END IF;

        IF EXTRACT(YEAR FROM AGE(p_birth_date)) < 14 THEN
            RETURN 'Посетитель должен быть старше 14 лет';
        END IF;

        IF p_subscription_months < 1 OR p_subscription_months > 24 THEN
            RETURN 'Срок абонемента должен быть от 1 до 24 месяцев';
        END IF;

        IF p_initial_sessions < 1 OR p_initial_sessions > 100 THEN
            RETURN 'Количество тренировок должно быть от 1 до 100';
        END IF;

        RETURN NULL;
    END validate_visitor_data;

    PROCEDURE register_visitor(
        p_second_name VARCHAR,
        p_passport VARCHAR,
        p_birth_date DATE,
        p_gender VARCHAR,
        p_subscription_months INTEGER,
        p_initial_sessions INTEGER,
        p_id_club INTEGER,
        p_result OUT INTEGER,
        p_message OUT VARCHAR
    ) AS
        v_visitor_id INTEGER;
        v_validation_error VARCHAR;
        v_club_exists BOOLEAN;
        v_end_date DATE;
    BEGIN

        IF p_passport IS NULL OR TRIM(p_passport) = '' THEN
            p_result := -1;
            p_message := 'Паспорт не может быть пустым';
            RETURN;
        END IF;

        IF NOT enum_utils.is_valid_gender(p_gender) THEN
            p_result := -1;
            p_message := 'Некорректное значение для пола: ' || p_gender;
            RETURN;
        END IF;

        SELECT EXISTS(SELECT 1 FROM club WHERE id_club = p_id_club)
        INTO v_club_exists;

        IF NOT v_club_exists THEN
            p_result := -1;
            p_message := 'Клуб с ID ' || p_id_club || ' не существует';
            RETURN;
```

```
    END IF;

    v_validation_error := validate_visitor_data(
        TRIM(p_passport),
        p_birth_date,
        p_subscription_months,
        p_initial_sessions
    );

    IF v_validation_error IS NOT NULL THEN
        p_result := -1;
        p_message := v_validation_error;
        RETURN;
    END IF;

    v_end_date := CURRENT_DATE + (p_subscription_months || ' months')::INTERVAL;

    INSERT INTO visitor (
        second_name, passport, birth_date, gender,
        subscription_end_date, remaining_sessions, id_club
    ) VALUES (
        TRIM(p_ second_name),
        TRIM(p_passport),
        p_birth_date,
        p_gender::gender_enum,
        v_end_date,
        p_initial_sessions,
        p_id_club
    )
    RETURNING id_visitor INTO v_visitor_id;

    p_result := v_visitor_id;
    p_message := 'Посетитель успешно зарегистрирован с ID: ' || v_visitor_id ||
                 '. Абонемент действителен до: ' || v_end_date;

    COMMIT;

EXCEPTION
    WHEN OTHERS THEN
        p_result := -1;
        p_message := 'Ошибка при регистрации посетителя: ' || SQLERRM;
        ROLLBACK;
END register_visitor;

PROCEDURE extend_subscription(
    p_id_visitor INTEGER,
    p_additional_months INTEGER,
    p_additional_sessions INTEGER DEFAULT 0,
    p_result OUT INTEGER,
    p_message OUT VARCHAR
) AS
    v_old_end_date DATE;
    v_new_end_date DATE;
    v_old_sessions INTEGER;
    v_new_sessions INTEGER;
    v_visitor_name VARCHAR;
BEGIN
    SELECT second_name, subscription_end_date, remaining_sessions
    INTO v_visitor_name, v_old_end_date, v_old_sessions
    FROM visitor
    WHERE id_visitor = p_id_visitor;

    IF NOT FOUND THEN
        p_result := -1;
        p_message := 'Посетитель с ID ' || p_id_visitor || ' не найден';
        RETURN;
    END IF;

    IF p_additional_months < 1 OR p_additional_months > 12 THEN
        p_result := -1;
        p_message := 'Количество месяцев должно быть от 1 до 12';
        RETURN;
    END IF;

    IF p_additional_sessions < 0 OR p_additional_sessions > 50 THEN
        p_result := -1;
        p_message := 'Количество дополнительных тренировок должно быть от 0 до 50';
        RETURN;
    END IF;
```

```
    v_new_end_date := GREATEST(v_old_end_date, CURRENT_DATE) +
                      (p_additional_months || ' months')::INTERVAL;
    v_new_sessions := v_old_sessions + p_additional_sessions;

    UPDATE visitor
    SET subscription_end_date = v_new_end_date,
        remaining_sessions = v_new_sessions,
        updated_at = CURRENT_TIMESTAMP
    WHERE id_visitor = p_id_visitor
    RETURNING 1 INTO p_result;

    IF p_result = 1 THEN
        p_message := 'Абонемент продлен для ' || v_visitor_name ||
                     '. Новый срок: ' || v_new_end_date ||
                     ', тренировок: ' || v_new_sessions;

        COMMIT;
    END IF;

EXCEPTION
    WHEN OTHERS THEN
        p_result := -1;
        p_message := 'Ошибка при продлении абонемента: ' || SQLERRM;
        ROLLBACK;
END extend_subscription;

PROCEDURE use_session(
    p_id_visitor INTEGER,
    p_session_type VARCHAR,
    p_training_id INTEGER DEFAULT NULL,
    p_result OUT INTEGER,
    p_message OUT VARCHAR
) AS
    v_remaining_sessions INTEGER;
    v_end_date DATE;
    v_visitor_name VARCHAR;
    v_training_info VARCHAR;
BEGIN
    SELECT second_name, remaining_sessions, subscription_end_date
    INTO v_visitor_name, v_remaining_sessions, v_end_date
    FROM visitor
    WHERE id_visitor = p_id_visitor;

    IF NOT FOUND THEN
        p_result := -1;
        p_message := 'Посетитель с ID ' || p_id_visitor || ' не найден';
        RETURN;
    END IF;

    IF v_end_date < CURRENT_DATE THEN
        p_result := -1;
        p_message := 'Абонемент посетителя истек ' || v_end_date;
        RETURN;
    END IF;

    IF v_remaining_sessions < 1 THEN
        p_result := -1;
        p_message := 'У посетителя не осталось доступных тренировок';
        RETURN;
    END IF;

    IF p_training_id IS NOT NULL THEN
        IF p_session_type = 'group' THEN
            SELECT 'Групповая тренировка: ' || name
            INTO v_training_info
            FROM group_training
            WHERE id_training = p_training_id;
        ELSIF p_session_type = 'personal' THEN
            SELECT 'Личная тренировка с тренером ID: ' || id_trainer
            INTO v_training_info
            FROM personal_training
            WHERE id_personal_training = p_training_id;
        END IF;
    END IF;

    UPDATE visitor
    SET remaining_sessions = remaining_sessions - 1,
        updated_at = CURRENT_TIMESTAMP
    WHERE id_visitor = p_id_visitor
    RETURNING 1 INTO p_result;
```

```
        IF p_result = 1 THEN
            p_message := 'Тренировка списана у ' || v_visitor_name ||
                         '. Осталось тренировок: ' || (v_remaining_sessions - 1);

            COMMIT;
        END IF;

    EXCEPTION
        WHEN OTHERS THEN
            p_result := -1;
            p_message := 'Ошибка при списании тренировки: ' || SQLERRM;
            ROLLBACK;
    END use_session;

    PROCEDURE remove_visitor(
        p_id_visitor INTEGER,
        p_reason VARCHAR DEFAULT 'окончание абонемента',
        p_result OUT INTEGER,
        p_message OUT VARCHAR
    ) AS
        v_visitor_data visitor%ROWTYPE;
        v_active_sessions INTEGER;
    BEGIN
        SELECT * INTO v_visitor_data
        FROM visitor
        WHERE id_visitor = p_id_visitor;

        IF NOT FOUND THEN
            p_result := -1;
            p_message := 'Посетитель с ID ' || p_id_visitor || ' не найден';
            RETURN;
        END IF;

        IF v_visitor_data.remaining_sessions > 0 THEN
            p_result := -1;
            p_message := 'Нельзя удалить посетителя с неиспользованными тренировками (' ||
                         v_visitor_data.remaining_sessions || ' осталось)';
            RETURN;
        END IF;

        SELECT COUNT(*) INTO v_active_sessions
        FROM visitor_group_training vgt
        JOIN group_training gt ON vgt.id_training = gt.id_training
        WHERE vgt.id_visitor = p_id_visitor
          AND gt.training_datetime > CURRENT_TIMESTAMP;

        IF v_active_sessions > 0 THEN
            p_result := -1;
            p_message := 'Посетитель зарегистрирован на ' || v_active_sessions ||
                         ' будущих тренировок';
            RETURN;
        END IF;


        DELETE FROM visitor
        WHERE id_visitor = p_id_visitor
        RETURNING 1 INTO p_result;

        IF p_result = 1 THEN
            p_message := 'Посетитель ' || v_visitor_data. second_name ||
                         ' удален. Причина: ' || p_reason;

            COMMIT;
        END IF;

    EXCEPTION
        WHEN OTHERS THEN
            p_result := -1;
            p_message := 'Ошибка при удалении посетителя: ' || SQLERRM;
            ROLLBACK;
    END remove_visitor;
END visitor_pkg;


CREATE OR REPLACE FUNCTION trg_auto_deduct_session()
RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'INSERT' AND TG_TABLE_NAME = 'visitor_group_training' THEN
       CALL visitor_pkg.use_session(
```

```
                    NEW.id_visitor,
                    'group',
                    NEW.id_training,
                    v_result,
                    v_message
                );

                IF v_result = -1 THEN
                    RAISE EXCEPTION '%', v_message;

    END IF;

    RETURN NEW;
END;

CREATE TRIGGER trg_auto_deduct_on_registration
    AFTER INSERT ON visitor_group_training
    FOR EACH ROW EXECUTE FUNCTION trg_auto_deduct_session();

CREATE OR REPLACE FUNCTION trg_check_training_capacity()
RETURNS TRIGGER AS $$
DECLARE
    v_capacity INTEGER;
    v_registered_count INTEGER;
BEGIN
    IF TG_OP = 'INSERT' AND TG_TABLE_NAME = 'visitor_group_training' THEN
        SELECT capacity INTO v_capacity
        FROM group_training
        WHERE id_training = NEW.id_training;

        SELECT COUNT(*) INTO v_registered_count
        FROM visitor_group_training
        WHERE id_training = NEW.id_training;

        IF v_registered_count > v_capacity THEN
            RAISE EXCEPTION 'Тренировка уже заполнена. Максимальная емкость: %', v_capacity;
        END IF;
    END IF;

    RETURN NEW;
END;

CREATE TRIGGER trg_enforce_training_capacity
    BEFORE INSERT ON visitor_group_training
    FOR EACH ROW EXECUTE FUNCTION trg_check_training_capacity();
```

2.1.4. Генерация view

```
-- 1. Представление для отображения информации о клубах с количеством персонала и посетителей
CREATE OR REPLACE VIEW v_clubs_summary AS
SELECT
    c.id_club,
    c.street,
    c.house_number,
    c.phone,
    COUNT(DISTINCT e.id_employee) as employees_count,
    COUNT(DISTINCT v.id_visitor) as visitors_count,
    COUNT(DISTINCT gt.id_training) as active_trainings_count,
    c.created_at,
    c.updated_at
FROM club c
LEFT JOIN employee e ON c.id_club = e.id_club
LEFT JOIN visitor v ON c.id_club = v.id_club
    AND v.subscription_end_date >= CURRENT_DATE
    AND v.remaining_sessions > 0
LEFT JOIN group_training gt ON c.id_club = gt.id_club
    AND gt.training_datetime > CURRENT_TIMESTAMP
GROUP BY c.id_club, c.street, c.house_number, c.phone, c.created_at, c.updated_at
ORDER BY c.id_club;

COMMENT ON VIEW v_clubs_summary IS 'Сводная информация по клубам с количеством персонала и
активных посетителей';

-- 2. Представление для списка активных посетителей с информацией об абонементе
CREATE OR REPLACE VIEW v_active_visitors AS
```

```
        SELECT
            v.id_visitor,
            v.second_name,
            v.passport,
            EXTRACT(YEAR FROM AGE(v.birth_date)) as age,
            v.gender,
            v.subscription_end_date,
            v.remaining_sessions,
            CASE
                WHEN v.subscription_end_date - CURRENT_DATE <= 7 THEN 'истекает'
                WHEN v.subscription_end_date - CURRENT_DATE <= 30 THEN 'действует'
                ELSE 'долгосрочный'
            END as subscription_status,
            c.street || ', д.' || c.house_number as club_address,
            c.phone as club_phone,
            v.created_at,
            v.updated_at
        FROM visitor v
        JOIN club c ON v.id_club = c.id_club
        WHERE v.subscription_end_date >= CURRENT_DATE
          AND v.remaining_sessions > 0
        ORDER BY v.subscription_end_date, v.full_name;

        COMMENT ON VIEW v_active_visitors IS 'Список активных посетителей с информацией об
абонементе';

        -- 3. Представление для списка работников с детальной информацией
        CREATE OR REPLACE VIEW v_employees_detailed AS
        SELECT
            e.id_employee,
            e.second_name,
            e.employee_number,
            e.role,
            CASE e.gender
                WHEN 'M' THEN 'мужской'
                WHEN 'F' THEN 'женский'
            END as gender,
            EXTRACT(YEAR FROM AGE(e.birth_date)) as age,
            e.qualification,
            c.street || ', д.' || c.house_number as club_address,
            c.phone as club_phone,
            CASE
                WHEN e.role = 'тренер' THEN (
                    SELECT COUNT(*)
                    FROM group_training gt
                    WHERE gt.id_trainer = e.id_employee
                      AND gt.training_datetime > CURRENT_TIMESTAMP
                )
                ELSE 0
            END as upcoming_group_trainings,
            CASE
                WHEN e.role = 'тренер' THEN (
                    SELECT COUNT(*)
                    FROM personal_training pt
                    WHERE pt.id_trainer = e.id_employee
                      AND pt.training_datetime > CURRENT_TIMESTAMP
                )
                ELSE 0
            END as upcoming_personal_trainings,
            e.created_at,
            e.updated_at
        FROM employee e
        JOIN club c ON e.id_club = c.id_club
        ORDER BY e.role, e.full_name;

        COMMENT ON VIEW v_employees_detailed IS 'Детальная информация о работниках со статистикой
тренировок';

        -- 4. Представление для предстоящих групповых тренировок
        CREATE OR REPLACE VIEW v_upcoming_group_trainings AS
        SELECT
            gt.id_training,
            gt.name,
            gt.description,
            gt.training_datetime,
            gt.capacity,
            e.full_name as trainer_name,
            e.qualification as trainer_qualification,
            c.street || ', д.' || c.house_number as club_address,
            (SELECT COUNT(*)
```

```
             FROM visitor_group_training vgt
             WHERE vgt.id_training = gt.id_training) as registered_count,
            gt.capacity - (SELECT COUNT(*)
                           FROM visitor_group_training vgt
                           WHERE vgt.id_training = gt.id_training) as available_spots,
            CASE
                WHEN gt.training_datetime < CURRENT_TIMESTAMP THEN 'завершена'
                WHEN gt.training_datetime - INTERVAL '1 hour' <= CURRENT_TIMESTAMP THEN 'скоро
начнется'
                ELSE 'запланирована'
            END as training_status,
            gt.created_at,
            gt.updated_at
        FROM group_training gt
        JOIN employee e ON gt.id_trainer = e.id_employee
        JOIN club c ON gt.id_club = c.id_club
        WHERE gt.training_datetime > CURRENT_TIMESTAMP - INTERVAL '1 day'
        ORDER BY gt.training_datetime;

        COMMENT ON VIEW v_upcoming_group_trainings IS 'Предстоящие групповые тренировки с информацией
о доступных местах';

        -- 5. Представление для детальной информации о личных тренировках
        CREATE OR REPLACE VIEW v_personal_trainings_detailed AS
        SELECT
            pt.id_personal_training,
            pt.training_datetime,
            e_trainer.full_name as trainer_name,
            e_trainer.qualification as trainer_qualification,
            e_trainer.employee_number as trainer_employee_number,
            v.full_name as visitor_name,
            v.remaining_sessions as visitor_remaining_sessions,
            v.subscription_end_date as visitor_subscription_end,
            pt.note,
            CASE
                WHEN pt.training_datetime < CURRENT_TIMESTAMP THEN
                    CASE
                        WHEN pt.trainer_note IS NOT NULL THEN 'проведена'
                        ELSE 'состоялась (без отзыва)'
                    END
                WHEN pt.training_datetime - INTERVAL '1 hour' <= CURRENT_TIMESTAMP THEN 'скоро
начнется'
                ELSE 'запланирована'
            END as training_status,
            pt.created_at,
            pt.updated_at
        FROM personal_training pt
        JOIN employee e_trainer ON pt.id_trainer = e_trainer.id_employee
        LEFT JOIN visitor v ON pt.id_visitor = v.id_visitor
        ORDER BY pt.training_datetime DESC;

        COMMENT ON VIEW v_personal_trainings_detailed IS 'Детальная информация о личных тренировках';

        -- 6. Представление для расписания тренеров
        CREATE OR REPLACE VIEW v_trainer_schedule AS
        SELECT
            e.id_employee,
            e.full_name as trainer_name,
            'групповая' as training_type,
            gt.name as training_name,
            gt.training_datetime,
            gt.capacity,
            (SELECT COUNT(*)
             FROM visitor_group_training vgt
             WHERE vgt.id_training = gt.id_training) as participants_count,
            c.street || ', д.' || c.house_number as location
        FROM employee e
        JOIN group_training gt ON e.id_employee = gt.id_trainer
        JOIN club c ON gt.id_club = c.id_club
        WHERE gt.training_datetime > CURRENT_TIMESTAMP

        UNION ALL

        SELECT
            e.id_employee,
            e.second_name as trainer_name,
            'личная' as training_type,
            'Личная тренировка' as training_name,
            pt.training_datetime,
            1 as capacity,
```

```
    1 as participants_count,
    (SELECT street || ', д.' || house_number
     FROM club c
     JOIN employee e2 ON c.id_club = e2.id_club
     WHERE e2.id_employee = e.id_employee) as location
FROM employee e
JOIN personal_training pt ON e.id_employee = pt.id_trainer
WHERE pt.training_datetime > CURRENT_TIMESTAMP
  AND pt.id_visitor IS NOT NULL

ORDER BY 1, 5;

COMMENT ON VIEW v_trainer_schedule IS 'Расписание тренировок для всех тренеров';
```