Университет ИТМО
Кафедра вычислительной техники
Базы данных

КУРСОВАЯ РАБОТА

Группа: P3102

Коков Алексей Тимурович
Нестеров Дали Константинович

г. Санкт-Петербург
2018 г.

Предметная область (этап №1)

Система MNU

14 апреля 1982 года человечество вступает в первый контакт с внеземным разумом, когда над Йоханнесбургом зависает гигантский космический корабль. Тем не менее, в течение трёх месяцев из него никто не появляется, и земные учёные сами проникают внутрь, прорезав обшивку корабля — внутри они обнаруживают огромное количество сильно истощенных и больных инопланетян. Пришельцев вывезли с корабля и с тех пор начали оказывать им всяческую помощь.

Предметная область - база данных частной военной компании и одного из крупнейших производителей оружия Multi-National United. По сюжету фильма правительство ЮАР прибегло к услугам данной компании, когда ситуация содержания лагеря в районе №9 стала выходить из-под контроля. Данная база будет лежать в основе этой компании, которая облегчит доступ к различной информации по миру внутри фильма (в особенности о сотрудниках, оборудованию, проектах MNU и пришельцах).

Каждая компания должна иметь в себе сотрудников, а также ответственные за что-либо подразделения, в которые эти сотрудники входят. MNU состоит из 3 частей: научная, охранная и руководящая, причем в каждой из них есть свои главные ученые/командующие/администраторы, ответственные за какие-либо проекты или отдельные действия.

В распоряжении компании также имеются средства передвижения (наземные, а также и воздушные), а на экстренные случаи - различное вооружение. Для разных видов транспорта и вооружения сотрудники должны иметь определенный уровень доступа. К примеру, у рядового охранника имеется доступ к набору нескольких пистолетов-пулеметов, пистолетов и холодного оружия, но отсутствует к транспорту.

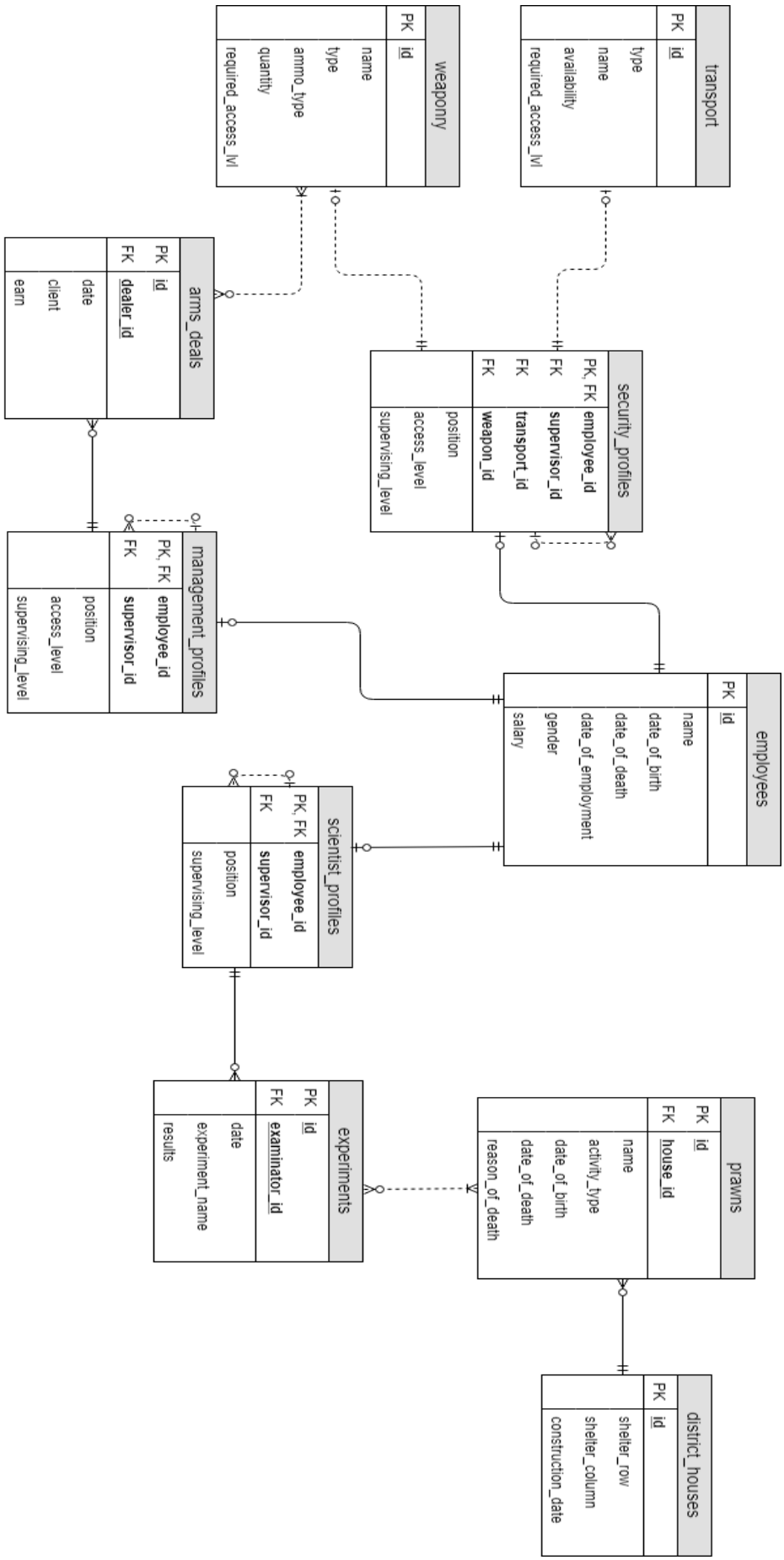
MNU также ответственна за содержание собственно инопланетян, в простонародье называемых “моллюсками”. По этой причине они должны содержаться в базе, в которой хранится информация о них: имя, возраст, род их деятельности (состоят ли они в какой-либо локальной преступной группировке), отсек проживания. Также компания ведет управление районами № 9 и 10, поэтому в базе должна иметься таблица последнего (учитывая, что из района №9 большинство пришельцев было переселено) с данными об отсеках: идентификатор, заселен или нет, проживающие и т.п.

В связи с повышенным интересом к инопланетянам и их технологиям MNU проводит различные медицинские эксперименты над образцами пришельцев: исследование нервной, мышечной системы, их реакции на различные внешние воздействия и т.д. Также компания интересуется инопланетным оружием, которое привезли с собой «моллюски». Но воспользоваться они им не могут, т.к. оно реагирует только на ДНК пришельцев.

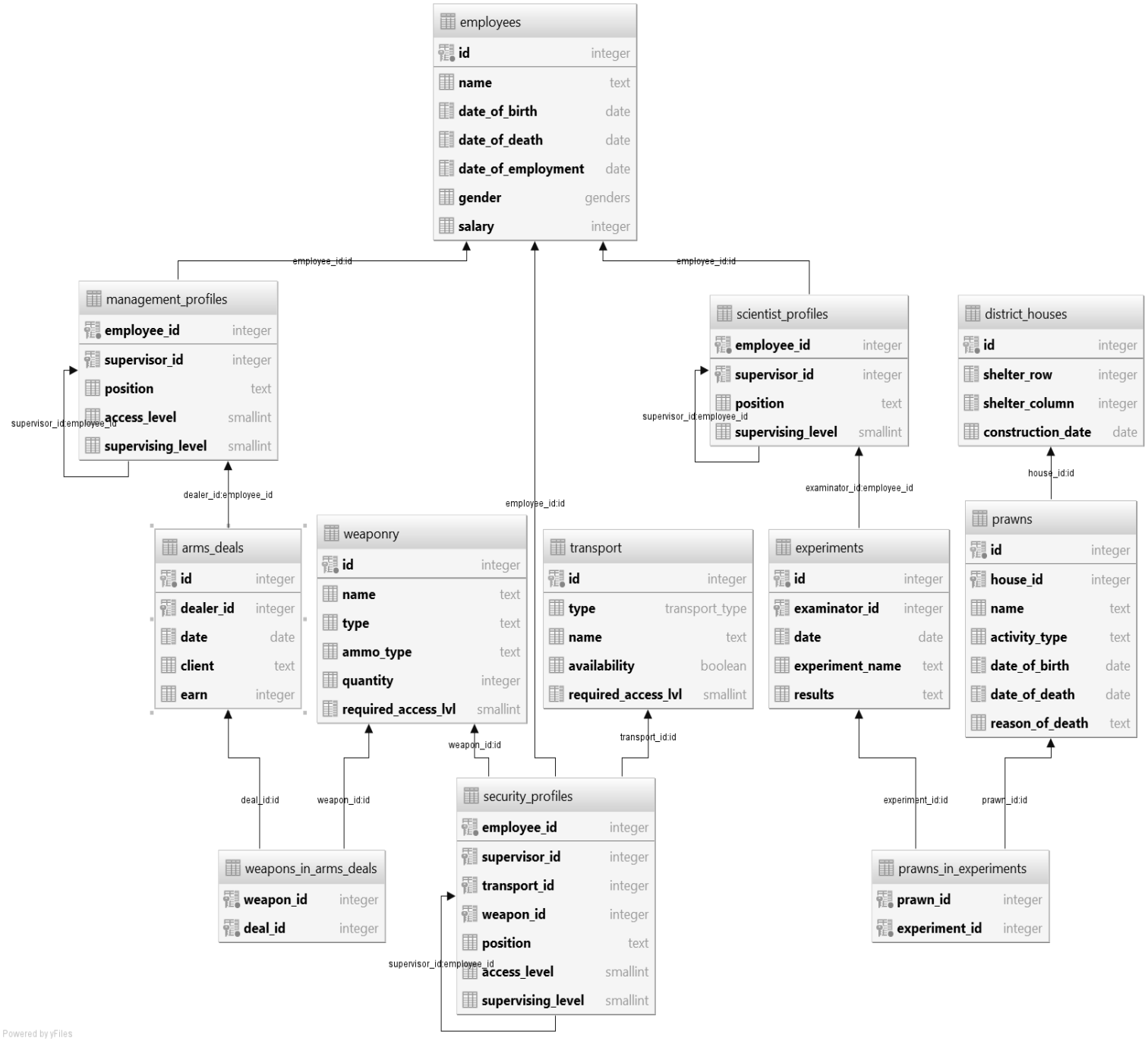
Результаты записываются в некоторый протокол (дата, суть эксперимента, проводящий, испытуемый и т.д.), который затем заносится в таблицу экспериментов.

Чтобы поддерживать стабильную экономическую обстановку в своей компании, MNU занимается торговлей оружием (и не всегда с легальными элементами). MNU сохраняет отчеты о сделках, также ведя учет о постоянных и прибыльных клиентах.

Инфологическая модель (этап №2)



Даталогическая модель (этап №3)



Описание сущностей и атрибутов (этапы №2 и 3)

1. employees – сущность, содержащая всех сотрудников.

   Атрибуты:
   id – уникальный идентификатор сотрудника (тип данных – serial; является первичным ключом);
   name – полное имя сотрудника (тип данных – text);
   date_of_birth – дата рождения сотрудника (тип данных – date);
   date_of_death – дата смерти сотрудника (тип данных – date);
   date_of_employment – дата найма сотрудника (тип данных – date);
   gender – пол сотрудника (тип данных – перечисляемый тип “genders”);
   salary – зарплата сотрудника (тип данных – integer).

   Ограничения целостности:
   invalid_death – дата смерти (если присутствует) не может быть раньше, чем дата рождения/найма;
   invalid_employment – дата найма не может быть раньше, чем дата рождения + 18 лет;
   valid_salary – зарплата должна быть обязательно положительной.

2. security_profiles – сущность, содержащая сотрудников отдела охраны.

   Атрибуты:
   employee_id – идентификатор сотрудника из сущности “employees” (тип данных – integer; является первичным и внешним ключом, ссылающимся на атрибут ‘id’ из сущности ‘employees’);
   supervisor_id – идентификатор начальника сотрудника (тип данных – integer; является внешним ключом, ссылающимся на атрибут ‘employee_id’ из сущности ‘security_profiles’);
   transport_id – идентификатор транспорта, который находится в распоряжения охранника (тип данных – integer; является внешним ключом, ссылающимся на атрибут ‘id’ из сущности ‘transport’);
   weapon_id – идентификатор оружия, которое находится в распоряжении охранника (тип данных – integer; является внешним ключом, ссылающимся на атрибут ‘id’ из сущности ‘weaponry’);
   position – должность сотрудника в отделе (тип данных – text);
   access_level – уровень доступа сотрудника к арсеналу оружия и транспорту (тип данных – smallint);
   supervising_level – уровень расположения сотрудника в иерархии отдела (тип данных – smallint).

   Ограничения целостности:
   valid_levels – уровень доступа должен иметь значения от 0 до 5 включительно, уровень начальства – от 0 до 10.

3. management_profiles – сущность, содержащая сотрудников руководящего отдела.

   Атрибуты:
   employee_id – идентификатор сотрудника из сущности “employees” (тип данных – integer; является первичным и внешним ключом, ссылающимся на атрибут ‘id’ из сущности ‘employees’);
   supervisor_id – идентификатор начальника сотрудника (тип данных – integer; является внешним ключом, ссылающимся на атрибут ‘employee_id’ из сущности ‘management_profiles’);
   position – должность сотрудника в отделе (тип данных – text);
   access_level – уровень доступа сотрудника к арсеналу оружия и транспорту (тип данных – smallint);
   supervising_level – уровень расположения сотрудника в иерархии отдела (тип данных – smallint).

   Ограничения целостности:
   valid_levels – уровень доступа должен иметь значения от 0 до 5 включительно, уровень начальства – от 0 до 10.

4. scientist_profiles – сущность, содержащая сотрудников научного отдела.

   Атрибуты:
   employee_id – идентификатор сотрудника из сущности “employees” (тип данных – integer; является первичным и внешним ключом, ссылающимся на атрибут ‘id’ из сущности ‘employees’);
   supervisor_id – идентификатор начальника сотрудника (тип данных – integer; является внешним ключом, ссылающимся на атрибут ‘employee_id’ из сущности ‘scientist_profiles’);
   position – должность сотрудника в отделе (тип данных – text);
   supervising_level – уровень расположения сотрудника в иерархии отдела (тип данных – smallint).

   Ограничения целостности:
   valid_levels – уровень начальства должен иметь значения от 0 до 10 включительно.

5. transport – сущность, содержащая транспорт, которым располагает компания.

   Атрибуты:
   id – уникальный идентификатор транспорта (тип данных – serial; является первичным ключом);
   type – тип транспорта (тип данных – перечисляемый тип ‘transport_type’);
   name – название транспорта (тип данных – text);
   availability – доступность транспорта (тип данных – boolean);
   required_access_lvl – требуемый уровень доступа для пользования транспортом (тип данных – smallint).

Ограничения целостности:
valid_level – требуемый уровень доступа должен иметь значения от 0 до 5 включительно.

6. weaponry – сущность, содержащая оружие, которым располагает компания.

   Атрибуты:
   id – уникальный идентификатор оружия (тип данных – serial; является первичным ключом);
   type – тип оружия (тип данных – text);
   name – название оружия (тип данных – text);
   ammo_type – тип патронов для оружия (тип данных - text);
   quantity – количество данного оружия (тип данных – integer);
   required_access_lvl – требуемый уровень доступа для пользования оружием (тип данных - smallint).

   Ограничения целостности:
   valid_level – требуемый уровень доступа должен иметь значения от 0 до 5 включительно;
   valid_quantity – количество доступного оружия должно быть обязательно неотрицательным.

7. arms_deals – сущность, содержащая информацию о проведенных оружейных сделках.

   Атрибуты:
   id – уникальный идентификатор сделки (тип данных – serial; является первичным ключом);
   dealer_id – идентификатор сотрудника, который провел сделку (тип данных – integer; является внешним ключом, ссылающимся на атрибут ‘employee_id’ из сущности ‘management_profiles’);
   date – дата проведения сделки (тип данных – date);
   client – клиент, купивший оружие (тип данных – text);
   earn – доход со сделки (тип данных – integer).

   Ограничения целостности:
   valid_earn – доход со сделки обязательно должен быть положительным.

8. prawns – сущность, содержащая информацию о инопланетянах.

   Атрибуты:
   id – уникальный идентификатор инопланетянина (тип данных – serial; является первичным ключом);
   house_id – идентификатор здания, в котором проживает моллюск (тип данных – integer; является внешним ключом, ссылающимся на атрибут ‘id’ из сущности ‘district_houses’);

name – имя инопланетянина (тип данных – text);
activity_type – род деятельности инопланетянина (тип данных – text);
date_of_birth – дата рождения инопланетянина (тип данных – date);
date_of_death – дата смерти инопланетянина (тип данных – date);
reason_of_death – причина смерти инопланетянина (тип данных – text).

Ограничения целостности:
invalid_death - дата смерти (если присутствует) не может быть раньше, чем дата рождения.

9. district_houses – сущность, содержащая информацию о зданиях в районе №10.

   Атрибуты:
   id – уникальный идентификатор здания (тип данных – serial; является первичным ключом);
   shelter_row – расположение/адрес здания по его «строке» (тип данных – integer, уникальный);
   shelter_column – расположение/адрес здания по его «столбцу» (тип данных – integer, уникальный);
   constriction_date – дата сооружения здания (тип данных – date).

10. experiments – сущность, содержащая информацию о проведенных экспериментах над инопланетянами.

    Атрибуты:
    id – уникальный идентификатор эксперимента (тип данных – serial; является первичным ключом);
    examinator_id – идентификатор сотрудника, который провел эксперимент (тип данных – integer; является внешним ключом, ссылающимся на атрибут ‘employee_id’ из сущности ‘scientist_profiles’);
    date – дата проведения эксперимента (тип данных – date);
    experiment_name – название эксперимента (тип данных – text);
    results – результаты проведения эксперимента (тип данных – text).

11. weapons_in_arms_deals – сущность, содержащая информацию об оружии, задействованном/проданном в какой-либо сделке.

    Атрибуты:
    weapon_id – уникальный идентификатор оружия (тип данных – integer; является внешним ключом, ссылающимся на атрибут ‘id’ из сущности ‘weaponry’);
    deal_id – уникальный идентификатор сделки (тип данных – integer; является внешним ключом, ссылающимся на атрибут ‘id’ из сущности ‘arms_deals’).

12. prawns_in_experiments – сущность, содержащая информацию об инопланетянах, над которыми были проведены какие-либо эксперименты.

Атрибуты:
prawn_id – уникальный идентификатор моллюска (тип данных – integer; является внешним ключом, ссылающимся на атрибут ‘id’ из сущности ‘prawns’);
experiment_id - уникальный идентификатор эксперимента (тип данных – integer; является внешним ключом, ссылающимся на атрибут ‘id’ из сущности ‘experiments’).

SQL-код (этапы №3 и 4)

```
-- Типы данных

CREATE TYPE GENDERS AS ENUM ('male', 'female', 'other');
CREATE TYPE TRANSPORT_TYPE AS ENUM ('land', 'air');

-- Таблицы

CREATE TABLE employees (
  id SERIAL PRIMARY KEY,
  name TEXT,
  date_of_birth DATE,
  date_of_death DATE,
  date_of_employment DATE,
  gender GENDERS,
  salary INT,

  CONSTRAINT invalid_death CHECK (date_of_death >= date_of_birth AND date_of_death >= date_of_employment),
  CONSTRAINT invalid_employment CHECK (date_of_employment >= date_of_birth + interval '18 years'),
  CONSTRAINT valid_salary CHECK (salary > 0)
);

CREATE TABLE transport(
  id SERIAL PRIMARY KEY,
  type TRANSPORT_TYPE,
  name TEXT,
  availability BOOLEAN,
  required_access_lvl SMALLINT,

  CONSTRAINT valid_level CHECK (required_access_lvl >= 0 AND required_access_lvl <= 5)
);

CREATE TABLE weaponry(
  id SERIAL PRIMARY KEY,
  name TEXT,
  type TEXT,
  ammo_type TEXT,
  quantity INTEGER,
  required_access_lvl SMALLINT

  CONSTRAINT valid_quantity CHECK (quantity >= 0),
  CONSTRAINT valid_level CHECK (required_access_lvl >= 0 AND required_access_lvl <= 5)
);

CREATE TABLE district_houses (
  id SERIAL PRIMARY KEY,
  shelter_row INTEGER,
  shelter_column INTEGER,
  construction_date DATE,
  UNIQUE (shelter_row, shelter_column)
);

CREATE TABLE prawns (
  id SERIAL PRIMARY KEY,
```

```
  house_id INT REFERENCES district_houses (id),
  name TEXT,
  activity_type TEXT,
  date_of_birth DATE,
  date_of_death DATE,
  reason_of_death TEXT,

  CONSTRAINT invalid_death CHECK (date_of_death >= date_of_birth)
);

CREATE TABLE scientist_profiles (
  employee_id INT PRIMARY KEY REFERENCES employees (id),
  supervisor_id INT REFERENCES scientist_profiles (employee_id),
  position TEXT,
  supervising_level SMALLINT,

  CONSTRAINT valid_levels CHECK (supervising_level >= 0 AND supervising_level <= 10)
);

CREATE TABLE security_profiles (
  employee_id INT PRIMARY KEY REFERENCES employees (id),
  supervisor_id INT REFERENCES security_profiles (employee_id),
  transport_id INT REFERENCES transport (id),
  weapon_id INT REFERENCES weaponry (id),
  position TEXT,
  access_level SMALLINT,
  supervising_level SMALLINT,

  CONSTRAINT valid_levels CHECK (access_level >= 0 AND access_level <= 5
                          AND supervising_level >= 0 AND supervising_level <= 10)
);

CREATE TABLE management_profiles (
  employee_id INT PRIMARY KEY REFERENCES employees (id),
  supervisor_id INT REFERENCES management_profiles (employee_id),
  position TEXT,
  access_level SMALLINT,
  supervising_level SMALLINT,

  CONSTRAINT valid_levels CHECK (access_level >= 0 AND access_level <= 5
                          AND supervising_level >= 0 AND supervising_level <= 10)
);

CREATE TABLE arms_deals(
  id SERIAL PRIMARY KEY,
  dealer_id INT REFERENCES management_profiles (employee_id),
  date DATE,
  client TEXT,
  earn INT,

  CONSTRAINT valid_earn CHECK (earn > 0)
);

CREATE TABLE experiments (
  id SERIAL PRIMARY KEY,
  examinator_id INT REFERENCES scientist_profiles (employee_id),
  date DATE,
  experiment_name TEXT,
  results TEXT
);

CREATE TABLE weapons_in_arms_deals (
  weapon_id INTEGER NOT NULL REFERENCES weaponry (id),
  deal_id INTEGER NOT NULL REFERENCES arms_deals (id)
```

```
);

CREATE TABLE prawns_in_experiments (
  prawn_id INTEGER NOT NULL REFERENCES prawns (id),
  experiment_id INTEGER NOT NULL REFERENCES experiments (id)
);


-- Функции и триггеры

CREATE FUNCTION check_weapons_before_deal()
  RETURNS TRIGGER AS $$
  DECLARE
    required_access_level SMALLINT;
    employee_access_level SMALLINT;
    deal_id INTEGER;
    weapon_count INTEGER;
    BEGIN
    weapon_count := (SELECT quantity FROM weaponry where id = NEW.weapon_id);
    deal_id := (SELECT id FROM arms_deals WHERE id = NEW.deal_id);
    required_access_level := (SELECT required_access_lvl FROM weaponry WHERE id = NEW.weapon_id);
    employee_access_level := (SELECT access_level FROM management_profiles, arms_deals WHERE
                              (deal_id = arms_deals.id AND
                             arms_deals.dealer_id = management_profiles.employee_id));

    IF employee_access_level < required_access_level THEN
      RAISE EXCEPTION 'Employee''s admission level must be higher to access this weapon.';
    ELSE IF weapon_count = 0 THEN
      RAISE EXCEPTION 'Insufficient quantity of weapons. Try again later.';
      ELSE
        UPDATE weaponry SET quantity = quantity-1 WHERE id = NEW.weapon_id;
      END IF;
    END IF;
  RETURN NEW;
  END;
   $$
LANGUAGE plpgsql;

CREATE FUNCTION check_weapons_before_using()
  RETURNS TRIGGER AS $$
  DECLARE
    required_access_level SMALLINT;
    weapon_count INTEGER;
    BEGIN
    weapon_count := (SELECT quantity FROM weaponry where id = NEW.weapon_id);
    required_access_level := (SELECT required_access_lvl FROM weaponry WHERE id = NEW.weapon_id);
    IF NEW.access_level < required_access_level THEN
      RAISE EXCEPTION 'Employee''s admission level must be higher to access this weapon.';
    ELSE IF weapon_count = 0 THEN
      RAISE EXCEPTION 'Insufficient quantity of weapons. Try again later.';
      ELSE
        UPDATE weaponry SET quantity = quantity-1 WHERE id = NEW.weapon_id;
      END IF;
    END IF;
    RETURN NEW;
  END;
  $$
LANGUAGE plpgsql;

CREATE FUNCTION check_transport_before_using()
  RETURNS TRIGGER AS $$
  DECLARE
    required_access_level SMALLINT;
    transport_availability BOOLEAN;
```

```
    BEGIN
    transport_availability := (SELECT availability FROM transport WHERE id = NEW.transport_id);
    required_access_level := (SELECT required_access_lvl FROM transport WHERE id = NEW.transport_id);
    IF NEW.access_level < required_access_level THEN
      RAISE EXCEPTION 'Employee''s admission level must be higher to access this transport.';
    ELSE IF (transport_availability = FALSE) THEN
      RAISE EXCEPTION 'This transport is not available right now. Try again later.';
      ELSE
        UPDATE transport SET availability = FALSE WHERE id = NEW.transport_id;
      END IF;
    END IF;
    RETURN NEW;
  END;
  $$
LANGUAGE plpgsql;

CREATE FUNCTION check_examinator_death_and_enlistment()
  RETURNS TRIGGER AS $$
  DECLARE
    examinator_enlistment date;
    examinator_death date;
    BEGIN
    examinator_enlistment := (SELECT date_of_employment FROM employees WHERE id = NEW.examinator_id);
    examinator_death := (SELECT date_of_death FROM employees WHERE id = NEW.examinator_id);
    IF NEW.date > examinator_death THEN
      RAISE EXCEPTION 'Scientist is already dead by the time of the experiment.';
      ELSE IF NEW.date < examinator_enlistment THEN
        RAISE EXCEPTION 'Scientist has not been enlisted yet by the time of the experiment.';
      END IF;
    END IF;
    RETURN NEW;
  END;
  $$
LANGUAGE plpgsql;

CREATE FUNCTION check_dealer_death_and_enlistment()
  RETURNS TRIGGER AS $$
  DECLARE
    dealer_enlistment date;
    dealer_death date;
    BEGIN
    dealer_enlistment := (SELECT date_of_employment FROM employees WHERE id = NEW.dealer_id);
    dealer_death := (SELECT date_of_death FROM employees WHERE id = NEW.dealer_id);
    IF NEW.date > dealer_death THEN
      RAISE EXCEPTION 'Employee is already dead by the time of the deal.';
      ELSE IF NEW.date < dealer_enlistment THEN
        RAISE EXCEPTION 'Employee has not been enlisted yet by the time of the deal.';
      END IF;
    END IF;
    RETURN NEW;
  END;
  $$
LANGUAGE plpgsql;

CREATE FUNCTION check_for_supervising_validity()
  RETURNS TRIGGER AS $$
  DECLARE
    supervisor_id_old INTEGER;
    supervisor_level SMALLINT;
    table_name TEXT;
    BEGIN
    table_name = quote_ident(tg_table_name);
    IF table_name = 'security_profiles' THEN
      supervisor_id_old := (SELECT supervisor_id FROM security_profiles WHERE employee_id =
```

```
NEW.supervisor_id);
     supervisor_level := (SELECT supervising_level FROM security_profiles where employee_id =
NEW.supervisor_id);

    ELSIF table_name = 'scientist_profiles' THEN
     supervisor_id_old := (SELECT supervisor_id FROM scientist_profiles WHERE employee_id =
NEW.supervisor_id);
     supervisor_level := (SELECT supervising_level FROM scientist_profiles where employee_id =
NEW.supervisor_id);

    ELSIF table_name = 'management_profiles' THEN
     supervisor_id_old := (SELECT supervisor_id FROM management_profiles WHERE employee_id =
NEW.supervisor_id);
     supervisor_level := (SELECT supervising_level FROM management_profiles where employee_id =
NEW.supervisor_id);

    END IF;

    IF supervisor_id_old = NEW.employee_id THEN
      RAISE EXCEPTION 'Employees cannot supervise each other.';
    ELSE IF supervisor_level <= NEW.supervising_level THEN
       RAISE EXCEPTION 'Supposed supervisor''s level is not sufficient.';
      END IF;
    END IF;
    RETURN NEW;
  END;
  $$
LANGUAGE plpgsql;


CREATE FUNCTION return_weapon_after_using()
  RETURNS TRIGGER AS $$
    BEGIN
    IF tg_op = 'UPDATE' THEN
      IF (NEW.weapon_id IS NULL AND OLD.weapon_id IS NOT NULL) OR (NEW.weapon_id <> OLD.weapon_id)
THEN
        UPDATE weaponry SET quantity = quantity+1 WHERE id = OLD.weapon_id;
      END IF;
      RETURN NEW;
    ELSIF tg_op = 'DELETE' THEN
      IF (OLD.weapon_id IS NOT NULL) THEN
        UPDATE weaponry SET quantity = quantity+1 WHERE id = OLD.weapon_id;
      END IF;
      RETURN old;
    END IF;
  END;
  $$
LANGUAGE plpgsql;

CREATE FUNCTION return_transport_after_using()
  RETURNS TRIGGER AS $$
    BEGIN
    IF tg_op = 'UPDATE' THEN
     IF (NEW.transport_id IS NULL AND OLD.transport_id IS NOT NULL) OR (NEW.transport_id <> OLD.transport_id)
THEN
        UPDATE transport SET availability = TRUE WHERE id = OLD.transport_id;
      END IF;
      RETURN NEW;
    ELSIF tg_op = 'DELETE' THEN
      IF (OLD.transport_id IS NOT NULL) THEN
        UPDATE transport SET availability = TRUE WHERE id = OLD.transport_id;
      END IF;
      RETURN old;
    END IF;
```

```
  END;
  $$
LANGUAGE plpgsql;


CREATE TRIGGER ability_to_deal_weapons
  BEFORE INSERT
  ON weapons_in_arms_deals
  FOR EACH ROW EXECUTE PROCEDURE check_weapons_before_deal();

CREATE TRIGGER ability_to_use_weapons
  BEFORE INSERT OR UPDATE
  ON security_profiles
  FOR EACH ROW EXECUTE PROCEDURE check_weapons_before_using();

CREATE TRIGGER ability_to_use_transport
  BEFORE INSERT OR UPDATE
  ON security_profiles
  FOR EACH ROW EXECUTE PROCEDURE check_transport_before_using();

CREATE TRIGGER validity_of_experiment
  BEFORE INSERT
  ON experiments
  FOR EACH ROW EXECUTE PROCEDURE check_examinator_death_and_enlistment();

CREATE TRIGGER validity_of_deal
  BEFORE INSERT
  ON arms_deals
  FOR EACH ROW EXECUTE PROCEDURE check_dealer_death_and_enlistment();

CREATE TRIGGER security_supervising
  BEFORE INSERT OR UPDATE
  ON security_profiles
  FOR EACH ROW EXECUTE PROCEDURE check_for_supervising_validity();

CREATE TRIGGER scientist_supervising
  BEFORE INSERT OR UPDATE
  ON scientist_profiles
  FOR EACH ROW EXECUTE PROCEDURE check_for_supervising_validity();

CREATE TRIGGER management_supervising
  BEFORE INSERT OR UPDATE
  ON management_profiles
  FOR EACH ROW EXECUTE PROCEDURE check_for_supervising_validity();

CREATE TRIGGER weapon_return
  BEFORE UPDATE OR DELETE
  ON security_profiles
  FOR EACH ROW EXECUTE PROCEDURE return_weapon_after_using();

CREATE TRIGGER transport_return
  BEFORE UPDATE OR DELETE
  ON security_profiles
  FOR EACH ROW EXECUTE PROCEDURE return_transport_after_using();

-- Индексы

CREATE INDEX employee_birth ON employees USING BTREE (date_of_birth);
CREATE INDEX employee_death ON employees USING BTREE (date_of_death);
CREATE INDEX employee_enlist ON employees USING BTREE (date_of_employment);

CREATE INDEX fk_house_id ON prawns USING BTREE (house_id);
CREATE INDEX prawn_birth ON prawns USING BTREE (date_of_birth);
```

```
CREATE INDEX prawn_death ON prawns USING BTREE (date_of_death);

CREATE INDEX fk_prawn_in_experiment_id ON prawns_in_experiments USING BTREE (prawn_id);
CREATE INDEX fk_prawn_experiment_id ON prawns_in_experiments USING BTREE (experiment_id);

CREATE INDEX fk_weapon_in_deal_id ON weapons_in_arms_deals USING BTREE (weapon_id);
CREATE INDEX fk_weapon_deal_id ON weapons_in_arms_deals USING BTREE (deal_id);

CREATE INDEX fk_examinator_id ON experiments USING BTREE (examinator_id);
CREATE INDEX experiment_date ON experiments USING BTREE (date);

CREATE INDEX fk_dealer_id ON arms_deals USING BTREE (dealer_id);
CREATE INDEX deal_date ON arms_deals USING BTREE (date);

CREATE INDEX fk_sci_employee_id ON scientist_profiles USING BTREE (employee_id);
CREATE INDEX fk_sec_employee_id ON security_profiles USING BTREE (employee_id);
CREATE INDEX fk_man_employee_id ON management_profiles USING BTREE (employee_id);

CREATE INDEX fk_sci_supervisor_id ON scientist_profiles USING BTREE (supervisor_id);
CREATE INDEX fk_sec_supervisor_id ON security_profiles USING BTREE (supervisor_id);
CREATE INDEX fk_man_supervisor_id ON management_profiles USING BTREE (supervisor_id);

CREATE INDEX fk_transport_id ON security_profiles USING BTREE (transport_id);
CREATE INDEX fk_weapon_id ON security_profiles USING BTREE (weapon_id);

CREATE INDEX tra_access_lvl ON transport USING BTREE (required_access_lvl);
CREATE INDEX wea_access_lvl ON weaponry USING BTREE (required_access_lvl);
```