

Факультет программной инженерии и компьютерной техники

Информационные системы и базы данных

Курсовая работа

Преподаватель: Гаврилов Антон Валерьевич

Выполнили: Кульбако Артемий Юрьевич Р33113

# Предметная область

**База данных для частной военной компании**

Частная военная компания имеет штаб СОТРУДНИКОВ (у работников необходимо узнать ИМЯ, ФАМИЛИЮ, ДАТУ\_РОЖДЕНИЯ, ОБРАЗОВАНИЕ и текущий СЕМЕЙНЫЙ\_СТАТУС, а также хранить его ДАТУ\_ЗАЧИСЛЕНИЯ на службу), которые могут занимать различные военные и невоенные ДОЛЖНОСТИ (НАЗВАНИЕ\_ДОЛЖНОСТИ, ЗАРПЛАТУ ВОИНСКОЕ\_ЗВАНИЕ если есть, номер комплекта ЭКИПИРОВКИ, и тип ВООРУЖЕННЫХ\_СИЛ (также работники могут занимать гражданские должности)).

Типы ВС: СВ, ВМФ, ВКС.

Каждый сотрудник имеет МЕДКАРТУ (с информацией о РОСТЕ в см, ВЕСЕ в кг, ГРУППЕ\_КРОВИ (по системе AB0), ПЕРЕНЕСЁННЫХ\_ТРАВМАХ/ЗАБОЛЕВАНИЯХ, БИОЛОГИЧЕСКОМ\_ПОЛЕ) и закрепленную за ним БАЗУ, являющуюся его основным местом прибывания (с информацией о МЕСТОПОЛОЖЕНИИ базы и её СТАТУСЕ).

Сотрудники могут отправляться на МИССИИ (нужно хранить НАЗВАНИЕ, ДАТУ\_И\_ВРЕМЯ\_СТАРТА и ЗАВЕРШЕНИЯ, ЮРИДИЧЕСКИЙ\_СТАТУС, МЕСТО\_ОТПРАВЛЕНИЯ и ПРИБЫТИЯ, ВРАГОВ), а также историю миссий сотрудников.

Миссия является большой частью военной КАМПАНИИ (должно содержать НАЗВАНИЕ, ЗАКАЗЧИКА, ПРИБЫЛЬ, ЗАТРАТЫ и СТАТУС\_ВЫПОЛЕНИЯ) на ТРАНСПОРТЕ (НАЗВАНИЕ, ТИП, необходимо также знать, когда СОСТОЯНИЕ находится), принадлежащем чвк (транспорт, естественно не утилизируется после миссии и может быть использован повторно).

Для безопасности, стоит хранить историю всех ТЕХОСМОТРОВ (с номерами ТРАНСПОРТА, ОБСЛУЖИВАЮЩЕГО, ДАТЫ\_ОСМОТРА).

Людям с военной должностью должен выдаваться комплект ЭКИПИРОВКИ (где может быть (но необязательно) КАМУФЛЯЖ, СРЕДСТВА\_КОММУНИКАЦИИ, РАЗВЕДКИ, МЕДИКАМЕНТЫ и ПРОЧЕЕ).

Экипировка в обязательном порядке должна включать один из ИРП (с описание о БЕЛКАХ, ЖИРАХ, УГЛЕВОДАХ и КАЛОРИЙНОСТИ, блюдах ЗАВТРАКА, ОБЕДА, УЖИНА, ПИЩЕВЫХ\_ДОБАВКАХ).

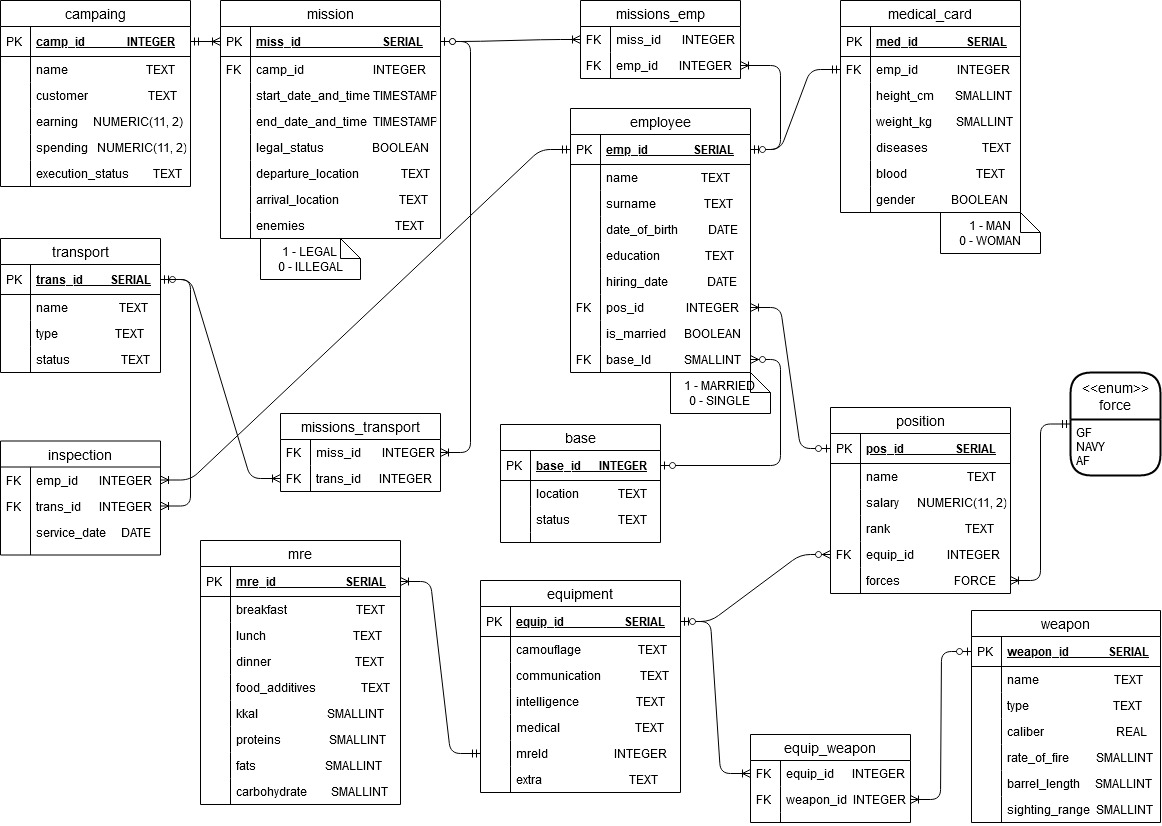
Экипировка может содержать одно или несколько ОРУЖИЙ (с подробными техническими характеристиками, будь то НАЗВАНИЕ, ТИП, КАЛИБР, СКОРОСТРЕЛЬНОСТЬ, ДЛИНА\_СТВОЛА, ПРИЦЕЛЬНАЯ\_ДАЛЬНОСТЬ).

**Процессы**

1. Тех, кто не имеет воинских званий, нельзя отправлять на боевые миссии.
2. Информационная система должна учитывая какие сотрудники отправились на миссии (один и тот же сотрудник не может находиться на двух миссиях одновременно).
3. Работников неподходящих по физическим данным запрещено устраивать как военных сотрудников.
4. Необходимо хранить историю инспекций транспорта, а транспорт со статусами «сломан» или «в ремонте» нельзя использовать в операциях.
5. Если за базой не закреплён ни один сотрудник, стоит закрыть её.
6. Стараться отправлять на боевые операции при прочих равных в первую очередь неженатых военных, давно не участвовавших в миссиях, имеющих большой опыт работы.

# Инфологическая модель

# Даталогическая модель



# Схема

*-- https://stackoverflow.com/questions/7296846/how-to-implement-one-to-one-one-to-many-and-many-to-many-relationships-while-de  
CREATE TABLE* base  
(  
 base\_id *SERIAL PRIMARY KEY*,  
 *location TEXT NOT NULL*,  
 status *TEXT NOT NULL*);  
  
*CREATE TABLE* mre  
(  
 mre\_id *SERIAL PRIMARY KEY*,  
 breakfast *TEXT NOT NULL*,  
 lunch *TEXT NOT NULL*,  
 dinner *TEXT NOT NULL*,  
 food\_additives *TEXT*,  
 kkal *SMALLINT NOT NULL CHECK* (kkal >= 1000),  
 proteins *SMALLINT NOT NULL CHECK* (proteins > 0),  
 fats *SMALLINT NOT NULL CHECK* (fats > 0),  
 carbohydrate *SMALLINT NOT NULL CHECK* (carbohydrate > 0)  
);  
  
*CREATE TABLE* equipment  
(  
 equip\_id *SERIAL PRIMARY KEY*,  
 camouflage *TEXT*,  
 communication *TEXT*,  
 intelligence *TEXT*,  
 medical *TEXT*,  
 mre\_id *INTEGER NOT NULL REFERENCES* mre,  
 extra *TEXT*);  
  
*CREATE TYPE force AS ENUM* ('GF', 'NAVY', 'AF');  
  
*CREATE TABLE position*(  
 pos\_id *SERIAL PRIMARY KEY*,  
 *name TEXT NOT NULL*,  
 salary *NUMERIC*(11, 2) *NOT NULL CHECK* (salary >= 12130),  
 rank *TEXT*,  
 equip\_id *INTEGER REFERENCES* equipment *ON DELETE SET NULL*,  
 forces *FORCE*);  
  
*CREATE TABLE* employee  
(  
 emp\_id *SERIAL PRIMARY KEY*,  
 *name TEXT NOT NULL*,  
 surname *TEXT NOT NULL*,  
 date\_of\_birth *DATE NOT NULL CHECK* (DATE\_PART('year', AGE(date\_of\_birth)) >= 18),  
 education *TEXT*,  
 hiring\_date *DATE NOT NULL DEFAULT CURRENT\_DATE*,  
 pos\_id *INTEGER NOT NULL REFERENCES position ON DELETE RESTRICT*,  
 is\_married *BOOLEAN NOT NULL*,  
 base\_id *INTEGER REFERENCES* base *ON DELETE SET NULL*);  
  
*CREATE TABLE* medical\_card  
(  
 med\_id *SERIAL PRIMARY KEY*,  
 emp\_id *INTEGER NOT NULL REFERENCES* employee *ON DELETE CASCADE*,  
 height\_cm *SMALLINT NOT NULL*,  
 weight\_kg *SMALLINT NOT NULL*,  
 diseases *TEXT*,  
 blood *TEXT NOT NULL*,  
 gender *BOOLEAN NOT NULL*);  
  
*CREATE TABLE* weapon  
(  
 weapon\_id *SERIAL PRIMARY KEY*,  
 *name TEXT NOT NULL*,  
 *type TEXT NOT NULL*,  
 caliber *REAL CHECK* (caliber > 0),  
 rate\_of\_fire *SMALLINT CHECK* (rate\_of\_fire > 0),  
 sighting\_range\_m *SMALLINT CHECK* (sighting\_range\_m > 0)  
);  
  
*CREATE TABLE* campaign  
(  
 camp\_id *SERIAL PRIMARY KEY*,  
 *name TEXT NOT NULL*,  
 customer *TEXT NOT NULL*,  
 earning *NUMERIC*(11, 2) *NOT NULL CHECK* (earning >= 0),  
 spending *NUMERIC*(11, 2) *NOT NULL CHECK* (spending >= 0),  
 execution\_status *TEXT*);  
  
*CREATE TABLE* mission  
(  
 miss\_id *SERIAL PRIMARY KEY*,  
 camp\_id *INTEGER NOT NULL REFERENCES* campaign *ON DELETE CASCADE*,  
 start\_date\_and\_time *TIMESTAMP*,  
 end\_date\_and\_time *TIMESTAMP*,  
 legal\_status *BOOLEAN*,  
 departure\_location *TEXT*,  
 arrival\_location *TEXT*,  
 enemies *TEXT*);  
  
*CREATE TABLE* transport  
(  
 trans\_id *SERIAL PRIMARY KEY*,  
 *name TEXT NOT NULL*,  
 *type TEXT NOT NULL*,  
 status *TEXT NOT NULL*);  
  
*CREATE TABLE* equip\_weapon  
(  
 equip\_id *INTEGER NOT NULL REFERENCES* equipment,  
 weapon\_id *INTEGER NOT NULL REFERENCES* weapon  
);  
  
*CREATE TABLE* missions\_transport  
(  
 miss\_id *INTEGER NOT NULL REFERENCES* mission,  
 trans\_id *INTEGER NOT NULL REFERENCES* transport  
);  
  
*CREATE TABLE* inspection  
(  
 emp\_id *INTEGER NOT NULL REFERENCES* employee,  
 trans\_id *INTEGER NOT NULL REFERENCES* transport,  
 service\_date *DATE NOT NULL DEFAULT CURRENT\_DATE*);  
  
*CREATE TABLE* missions\_emp  
(  
 miss\_id *INTEGER NOT NULL REFERENCES* mission,  
 emp\_id *INTEGER NOT NULL REFERENCES* employee  
);

# Триггеры и функции

*--1: Тех, кто не имеет воинских званий, нельзя отправлять на боевые миссии.  
CREATE FUNCTION* is\_military\_on\_mission() *RETURNS trigger AS* $$  
 *DECLARE* enemy *TEXT*;  
 *DECLARE* rank *TEXT*;  
 *BEGIN* enemy = (*SELECT* enemies *FROM* mission *WHERE* miss\_id = *new*.miss\_id);  
 rank = (*SELECT* rank *FROM position JOIN* employee *USING* (pos\_id) *WHERE* emp\_id = *new*.emp\_id);  
 *IF* (enemy *IS NOT NULL OR* !~~ '') *AND* (rank *IS NULL OR* ~~ '') *THEN  
 RAISE EXCEPTION* 'Cannot set not military employee to a combat mission';  
 *ELSE RETURN new*;  
 *END IF*;  
 *END*;  
$$ *LANGUAGE* plpgsql;  
  
*CREATE TRIGGER* is\_military\_on\_mission *BEFORE INSERT OR UPDATE ON* missions\_emp  
 *FOR EACH ROW EXECUTE PROCEDURE* is\_military\_on\_mission();  
  
*/\*  
2: Информационная система должна учитывая какие сотрудники отправились на миссии (один и тот же сотрудник не  
 может находиться на двух миссиях одновременно).  
\*/  
CREATE FUNCTION* check\_periods\_of\_emp\_missions() *RETURNS trigger AS* $$  
 *DECLARE* inserted\_miss mission;  
 *BEGIN* inserted\_miss = (*SELECT* \* *FROM* mission *WHERE* miss\_id = *new*.miss\_id);  
 *IF* (*TRUE*) *IN* (  
 *SELECT* (inserted\_miss.start\_date\_and\_time, inserted\_miss.end\_date\_and\_time) *OVERLAPS* (start\_date\_and\_time, end\_date\_and\_time) *FROM* mission  
 *WHERE* miss\_id *IN* (*SELECT* miss\_id *FROM* missions\_emp *WHERE* emp\_id = *new*.emp\_id)) *THEN  
 RAISE EXCEPTION* 'This worker cannot be assigned to a mission as he was on another mission at the time';  
 *ELSE RETURN new*;  
 *END IF*;  
 *END*;  
$$ *LANGUAGE* plpgsql;  
  
*CREATE TRIGGER* check\_emp\_mission\_period *BEFORE INSERT OR UPDATE ON* missions\_emp  
 *FOR EACH ROW EXECUTE PROCEDURE* check\_periods\_of\_emp\_missions();  
  
*CREATE INDEX* mission\_period *ON* mission *USING* btree(start\_date\_and\_time, end\_date\_and\_time);  
  
*--3: Работников неподходящих по физическим данным запрещено устраивать как военных сотрудников (рост < 150 см или вес < 45 кг).  
CREATE FUNCTION* check\_physical\_condition() *RETURNS trigger AS* $$  
 *DECLARE h SMALLINT*;  
 *DECLARE w SMALLINT*;  
 *BEGIN  
 SELECT* height\_cm, weight\_kg *INTO h*, *w FROM* medical\_card *JOIN* employee *USING* (emp\_id) *WHERE* emp\_id = *new*.emp\_id;  
 *IF h* < 150 *OR w* < 45 *THEN  
 RAISE EXCEPTION* 'Cannot hire this employee to military position because his physical data does not require the minimum';  
 *ELSE RETURN new*;  
 *END IF*;  
 *END*;  
$$ *LANGUAGE* plpgsql;  
  
*CREATE TRIGGER* check\_physical\_condition *BEFORE INSERT OR UPDATE ON* employee  
 *FOR EACH ROW EXECUTE PROCEDURE* check\_physical\_condition();  
  
*/\*  
 4: Необходимо хранить историю инспекций транспорта (реализована отдельной таблицей),  
 а транспорт со статусами «сломан» или «в ремонте» нельзя использовать в операциях.  
 \*/  
CREATE FUNCTION* check\_transport\_condition() *RETURNS trigger AS* $$  
 *BEGIN  
 IF* (*SELECT* status *FROM* transport *WHERE* trans\_id = *new*.trans\_id *AND* status = 'available') *IS NULL THEN  
 RAISE EXCEPTION* 'Cannot set not available transport to mission';  
 *ELSE RETURN new*;  
 *END IF*;  
 *END*;  
$$ *LANGUAGE* plpgsql;  
  
*CREATE TRIGGER* check\_transport\_condition *BEFORE INSERT OR UPDATE ON* missions\_transport  
 *FOR EACH ROW EXECUTE PROCEDURE* check\_transport\_condition();  
  
*-- 5: Если за базой не закреплён ни один сотрудник, стоит закрыть её.  
CREATE FUNCTION* close\_empty\_bases() *RETURNS SETOF void AS* $$  
 *BEGIN  
 DELETE FROM* base *WHERE* base\_id *IN* (*SELECT* \* *FROM* base\_count\_emp);  
 *END*;  
$$ *LANGUAGE* plpgsql;  
  
*CREATE MATERIALIZED VIEW* base\_count\_emp *AS* (*SELECT* base\_id *FROM* base *JOIN* employee *USING* (base\_id) *GROUP BY* base\_id *HAVING* COUNT(emp\_id) = 0);  
  
*CREATE FUNCTION* update\_base\_count\_emp() *RETURNS trigger AS* $$  
 *BEGIN  
 REFRESH MATERIALIZED VIEW* base\_count\_emp;  
 *RETURN new*;  
 *END*;  
$$ *LANGUAGE* plpgsql;  
  
*CREATE TRIGGER* update\_base\_count\_emp *AFTER INSERT OR UPDATE OR DELETE ON* employee  
 *FOR EACH ROW EXECUTE PROCEDURE* update\_base\_count\_emp();  
  
*/\*  
6: Стараться отправлять на боевые операции при прочих равных в первую очередь неженатых военных, давно не  
 участвовавших в миссиях, имеющих большой опыт работы.  
 \*/  
CREATE FUNCTION* get\_combat\_candidates(n *int DEFAULT* 1) *RETURNS* employee *AS* $$  
 *BEGIN  
 SELECT* emp\_id *FROM* employee  
 *JOIN position USING* (pos\_id)  
 *JOIN* missions\_emp *USING* (emp\_id)  
 *JOIN* mission *USING* (miss\_id)  
 *WHERE* rank *IS NOT NULL OR* !~~ ''  
 *ORDER BY* is\_married *DESC*, end\_date\_and\_time *DESC*, hiring\_date *DESC  
 LIMIT* n;  
 *END*;  
$$ *LANGUAGE* plpgsql;  
  
*CREATE INDEX* pos\_rank *ON position USING* btree(rank);

# Индексы

Естественно, большая часть обращений будут происходит к таблицам, связанным с бизнес-процессами информационной системы, поэтому при оптимизации нужно делать ставку на них.

1-ую функцию оптимизировать смысла нет, т.к. при обращении к 2 из 3 трёх таблиц поиск и так происходит через индексы, ибо условие отбора работает с первичными ключами таблиц.

Во 2-ой функции, для избежание пересечения временных промежутков добавляемой миссии и существующих, нам необходимо получить этот самый интервал для каждой записи, и индекс здесь действительно будет к месту.

*CREATE INDEX* mission\_period *ON* mission *USING* btree(start\_date\_and\_time, end\_date\_and\_time);

|  |  |  |
| --- | --- | --- |
|  | -> |  |

**Выигрыш в треть.**

В 3-ем необходимо сопоставлять работников и их некоторые параметры из их медицинской карты. Т.к. выборка на «нижнем уровне» дерева запроса осуществляет по ключу таблицы, индекс не нужен.

4-ый: аналогично 3-ему.

В 5-ой функции стоит сделать материализованное представление, которое будет содержать базы и количество их сотрудников, а логику обновления этой таблицы запускать по запросу на изменение данных Employee.

*CREATE MATERIALIZED VIEW* base\_count\_emp *AS* (*SELECT* base\_id *FROM* base *JOIN* employee *USING* (base\_id) *GROUP BY* base\_id *HAVING* COUNT(emp\_id) = 0);  
  
*CREATE FUNCTION* update\_base\_count\_emp() *RETURNS trigger AS* $$  
 *BEGIN  
 REFRESH MATERIALIZED VIEW* base\_count\_emp;  
 *END*;  
$$ *LANGUAGE* plpgsql;  
  
*CREATE TRIGGER* update\_base\_count\_emp *BEFORE INSERT OR UPDATE OR DELETE ON* employee  
 *FOR EACH ROW EXECUTE FUNCTION* update\_base\_count\_emp();

В последней функции присутствует условие отбора записей, не по ключу, и в таблице (правда записей в таблице не так уж и много). Создадим индекс:

*CREATE INDEX* pos\_rank *ON position USING* btree(rank);

|  |  |  |
| --- | --- | --- |
|  | -> |  |

**Время незначительно сократилось.**