

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).

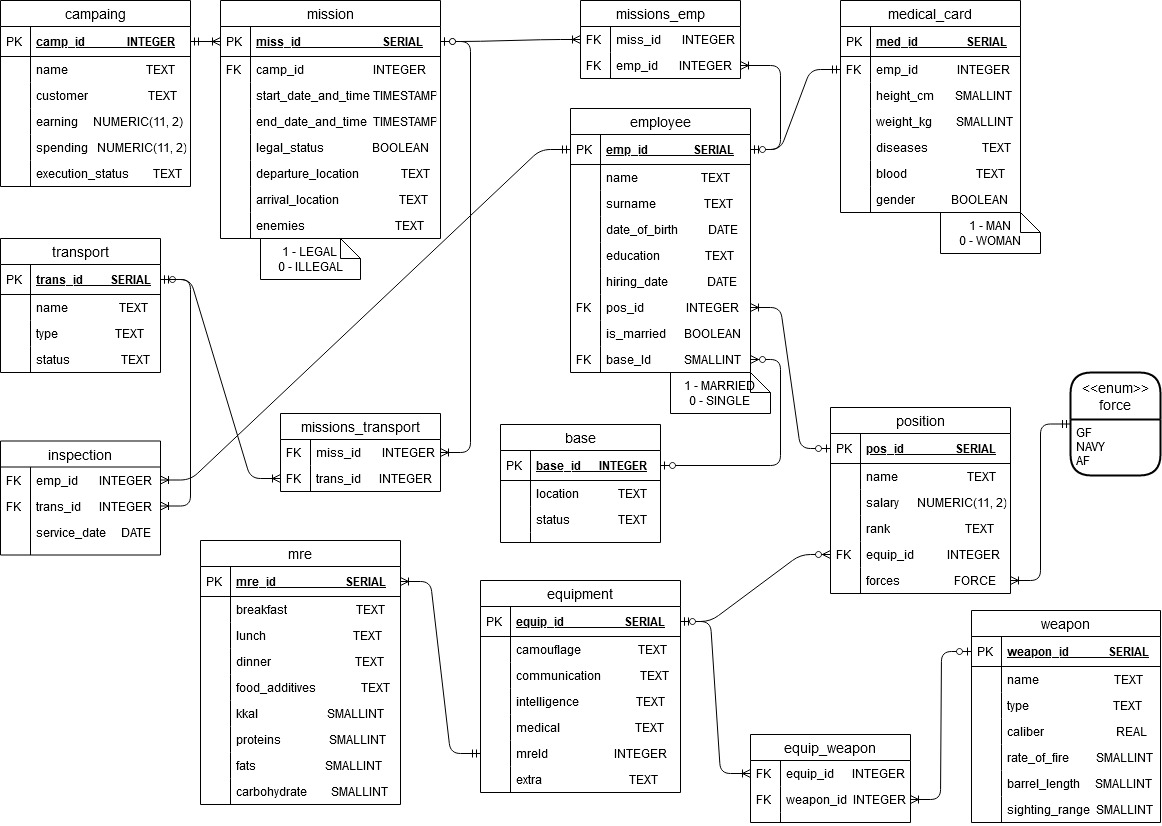
Экипировка может содержать одно или несколько ОРУЖИЙ (с подробными техническими характеристиками, будь то НАЗВАНИЕ, ТИП, КАЛИБР, СКОРОСТРЕЛЬНОСТЬ, ДЛИНА\_СТВОЛА, ПРИЦЕЛЬНАЯ\_ДАЛЬНОСТЬ).

**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*,  
 status *TEXT*);  
  
*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 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 TYPE* blood *AS ENUM* ('0+', '0-', 'A+', 'A-', 'B+', 'B-', 'AB+', 'AB-');  
  
*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*);  
  
*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* $$  
 *DECLARE* card medical\_card;  
 *BEGIN* card = (*SELECT* height\_cm, weight\_kg *FROM* medical\_card *JOIN* employee *USING* (emp\_id) *WHERE* emp\_id = *new*.emp\_id);  
 *IF* card.height\_cm < 150 *OR* card.weight\_kg < 45 *THEN  
 RAISE EXCEPTION* 'Cannot hair this employee to military position because his physical data does not require the minimum';  
 *ELSE RETURN new*;  
 *END IF*;  
 *END*;  
$$ *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* $$  
 *BEGIN  
 REFRESH MATERIALIZED VIEW* base\_count\_emp;  
 *END*;  
$$ *LANGUAGE* plpgsql;  
  
*CREATE TRIGGER* update\_base\_count\_emp *BEFORE 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.**

# Генерация данных

*package* com.testpassword  
  
*import* io.ktor.util.\*  
*import* org.jetbrains.exposed.sql.Table  
*import* org.jetbrains.exposed.sql.`java-time`.CurrentDateTime  
*import* org.jetbrains.exposed.sql.`java-time`.date  
*import* org.jetbrains.exposed.sql.`java-time`.datetime  
*import* org.jetbrains.exposed.sql.`java-time`.year  
*import* org.jetbrains.exposed.sql.insert  
*import* org.jetbrains.exposed.sql.select  
*import* org.jetbrains.exposed.sql.selectAll  
*import* org.json.JSONObject  
*import* org.postgresql.util.PGobject  
*import* java.io.File  
*import* java.time.LocalDate  
*import* java.time.ZoneId  
*import* java.util.\*  
*import* kotlin.random.Random  
  
*/\*  
https://stackoverflow.com/questions/45723803/how-to-use-postgresql-enum-type-via-kotlin-exposed-orm  
https://blog.jdriven.com/2019/07/kotlin-exposed-a-lightweight-sql-library/  
 \*/  
  
class* PGEnum<T : Enum<T>>(enumTypeName: String, enumValue: T?) : PGobject() {  
 *init* {  
 value = enumValue?.name  
 type = enumTypeName  
 }  
}  
  
*inline fun* <*reified* T : Enum<T>> Table.postgresEnumeration(  
 columnName: String,  
 postgresEnumName: String  
) = customEnumeration(columnName, postgresEnumName,  
 **{** value **->** enumValueOf<T>(value *as* String) **}**, **{** PGEnum(postgresEnumName, **it**) **}**)  
  
*interface Generable* { *fun* generateAndInsert(n: Int = 0) = Unit }  
  
*object* Base: Table("base"), *Generable* {  
  
 *val* baseId = integer("base\_id").autoIncrement().primaryKey()  
 *val* location = text("location")  
 *val* status = text("status")  
  
 *override fun* generateAndInsert(n: Int) {  
 *val* statuses = setOf("working", "closed", "destroyed", "abandoned", "unknown", "captured", "for\_sale")  
 (1..n).forEach **{** Base.insert **{  
 it**[location] = *F*.address().city()!!  
 **it**[status] = statuses.random()  
 **}  
 }** }  
}  
  
*object* MRE: Table("mre"), *Generable* {  
  
 *val* mreId = integer("mre\_id").autoIncrement().primaryKey()  
 *val* breakfast = text("breakfast").nullable()  
 *val* lunch = text("lunch").nullable()  
 *val* dinner = text("dinner").nullable()  
 *val* foodAdditives = text("food\_additives")  
 *val* kkal = integer("kkal").nullable().check **{ it** greaterEq 1000 **}** *val* proteins = integer("proteins").nullable().check **{ it** greater 0 **}** *val* fats = integer("fats").nullable().check **{ it** greater 0 **}** *val* carbohydrate = integer("carbohydrate").nullable().check **{ it** greater 0 **}** *override fun* generateAndInsert(n: Int) {  
 (1..n).forEach **{** MRE.insert **{  
 it**[breakfast] = *F*.food().dish()  
 **it**[lunch] = *F*.food().dish()  
 **it**[dinner] = *F*.food().dish()  
 **it**[foodAdditives] = generateSequence **{** *F*.food().ingredient() **}**.take(3).joinToString(separator = "\_")  
 **it**[kkal] = *F*.number().numberBetween(1000, 4000)  
 **it**[proteins] = *F*.number().numberBetween(1, 400)  
 **it**[fats] = *F*.number().numberBetween(1, 400)  
 **it**[carbohydrate] = *F*.number().numberBetween(1, 400)  
 **}  
 }** }  
}  
  
*object* Equipment: Table("equipment"), *Generable* {  
  
 *val* equipId = integer("equip\_id").autoIncrement().primaryKey()  
 *val* camouflage = text("camouflage")  
 *val* communication = text("communication")  
 *val* intelligence = text("intelligence")  
 *val* medical = text("medical")  
 *val* mreId = reference("mre\_id", MRE.mreId)  
 *val* extra = text("extra")  
  
 *override fun* generateAndInsert(n: Int) {  
 *val* camouflages = setOf("black", "khaki", "olive", "fleckerteppich", "strichtarn", "cce", "vegetata",  
 "flora", "forest", "pixel", "woodland", "accupat", "desert", "city")  
 *val* communications = setOf("radio\_set", "map", "mobile\_satellite", "signal\_flares")  
 *val* intelligences = setOf("binoculars", "drone", "radar", "thermal\_visor")  
 *val* medicals = setOf("bandage", "harness", "antibiotics", "alcohol", "scissors", "tweezers", "antiseptic",  
 "ammonia", "nitroglycerine", "adrenalin")  
 *val* mreIds = MRE.selectAll().limit(n).map **{ it**[MRE.mreId] **}** (1..n).forEach **{** Equipment.insert **{  
 it**[camouflage] = camouflages.random()  
 **it**[communication] = communications.random()  
 **it**[intelligence] = intelligences.random()  
 **it**[medical] = medicals.random()  
 **it**[mreId] = mreIds.random()  
 **}  
 }** }  
}  
  
*enum class* FORCES { *GF*, *NAVY*, *AF* }  
  
*object* Position: Table("Position"), *Generable* {  
  
 *val* posId = integer("pos\_id").autoIncrement().primaryKey()  
 *val* name = text("name").nullable()  
 *val* salary = decimal("salary", 2, 11).nullable().check **{ it** greater 12130.0 **}** *val* rank = text("rank")  
 *val* equipId = reference("equip\_id", Equipment.equipId)  
 *val* forces = postgresEnumeration<FORCES>("forces", "force")  
  
 *override fun* generateAndInsert(n: Int) {  
 *val* ranks = setOf("student\_officer", "private\_second\_class", "private\_first\_class", "junior\_sergeant",  
 "sergeant", "senior\_sergeant", "petty\_officer", "ensign", "senior\_ensign", "junior\_lieutenant", "lieutenant",  
 "senior\_lieutenant", "captain", "major", "lieutenant\_colonel", "colonel") *//, "major\_general", "lieutenant\_general",  
 //"colonel\_general", "army\_general", "marshal") исключим высшие звания из генерации  
 val* armyPositions = setOf("medic", "miner", "scout", "security", "driver", "torpedo\_operator", "pilot",  
 "mechanic", "engineer", "navigator", "orderly", "duty", "coach", "artilleryman", "gunner", "sniper", "spy")  
 *val* equipIds = Equipment.selectAll().limit(n).map **{ it**[Equipment.equipId] **}** (1..n).forEach **{** Position.insert **{** *val* empPos = *if* (Random.nextDouble(1.0, 100.0) >= 65) armyPositions.random() *else F*.job().position()  
 **it**[name] = empPos  
 **it**[salary] = *F*.number().randomDouble(2, 12130, 1000000).toBigDecimal()  
 *if* (empPos *in* armyPositions) **it**[rank] = ranks.random()  
 **it**[equipId] = equipIds.random()  
 **it**[forces] = FORCES.values().random()  
 **}  
 }** }  
}  
  
*object* Employee: Table("employee"), *Generable* {  
  
 *val* empId = integer("emp\_id").autoIncrement().primaryKey()  
 *val* name = text("name").nullable()  
 *val* surname = text("surname").nullable()  
 *val* dateOfBirth = date("date\_of\_birth").nullable().check **{ it** less (CurrentDateTime().year() - 18) **}** *val* education = text("education")  
 *val* hiringDate = date("hiring\_date").defaultExpression(CurrentDateTime().date())  
 *val* posId = reference("pos\_id", Position.posId)  
 *val* isMarried = bool("is\_married").nullable()  
 *val* baseId = reference("base\_id", Base.baseId)  
  
 @InternalAPI *override fun* generateAndInsert(n: Int) {  
 *val* posIds = Position.selectAll().limit(n).map **{ it**[Position.posId] **}** *val* baseIds = Base.selectAll().map **{ it**[Base.baseId] **}** (1..n).forEach **{** *val* newbieId = Employee.insert **{  
 it**[name] = *F*.name().firstName()  
 **it**[surname] = *F*.name().lastName()  
 **it**[dateOfBirth] = *F*.date().birthday(18, 70).toLocalDateTime().toLocalDate()  
 **it**[education] = *F*.educator().university()  
 **it**[hiringDate] = *F*.date().between(  
 *// здесь и далее - условной день основания компании.* Date.from(LocalDate.of(2014, 3, 18).atStartOfDay(ZoneId.systemDefault()).toInstant()),  
 Date()  
 ).toLocalDateTime().toLocalDate()  
 **it**[posId] = posIds.random()  
 **it**[isMarried] = Random.nextBoolean()  
 **it**[baseId] = baseIds.random()  
 **}**[empId]  
 MedicalCard.insert **{  
 it**[empId] = newbieId  
 **it**[height\_cm] = Random.nextInt(150, 200)  
 **it**[weight\_kg] = Random.nextInt(40, 120)  
 **it**[diseases] = generateSequence **{** *F*.medical().diseaseName() **}** .take(Random.nextInt(0, 5))  
 .joinToString(separator = "\_")  
 **it**[blood] = *F*.name().bloodGroup()  
 **it**[gender] = Random.nextBoolean()  
 **}  
 }** }  
}  
  
*object* MedicalCard: Table("medical\_card"), *Generable* {  
  
 *val* medId = integer("med\_id").autoIncrement().primaryKey()  
 *val* empId = reference("emp\_id", Employee.empId)  
 *val* height\_cm = integer("height\_cm").nullable()  
 *val* weight\_kg = integer("weight\_kg").nullable()  
 *val* diseases = text("diseases")  
 *val* blood = text("blood").nullable()  
 *val* gender = bool("gender").nullable()  
}  
  
*object* Weapon: Table("weapon"), *Generable* {  
  
 *val* weaponId = integer("weapon\_id").autoIncrement().primaryKey()  
 *val* name = text("name").nullable()  
 *val* type = text("type").nullable()  
 *val* caliber = float("caliber").check **{ it** greater 0.0 **}** *val* rateOfFire = integer("rate\_of\_fire").check **{ it** greater 0 **}** *val* sightingRange\_m = integer("sighting\_range\_m").check **{ it** greater 0 **}** *override fun* generateAndInsert(n: Int) {  
 *val* rawData = File("resources/static/guns.json").readLines().joinToString(separator = "")  
 *val* jsonBody = JSONObject(rawData).getJSONArray("weapons")  
 jsonBody.forEachIndexed **{** i, \_ **->** *val* w = jsonBody.getJSONObject(i)  
 Weapon.insert **{  
 it**[name] = w.getString("name")  
 **it**[type] = w.getString("type")  
 **it**[caliber] = w.getFloat("caliber")  
 **it**[rateOfFire] = w.getInt("rate\_of\_fire")  
 **it**[sightingRange\_m] = w.getInt("sighting\_range\_m")  
 **}  
 }** }  
}  
  
*object* Campaign: Table("campaign"), *Generable* {  
  
 *val* campId = integer("camp\_id").autoIncrement().primaryKey()  
 *val* name = text("name").nullable()  
 *val* customer = text("customer").nullable()  
 *val* earning = decimal("earning", 2, 11).nullable().check **{ it** greaterEq 0.0 **}** *val* spending = decimal("spending", 2, 11).nullable().check **{ it** greaterEq 0.0 **}** *val* executionStatus = text("execution\_status")  
  
 *override fun* generateAndInsert(n: Int) {  
 *val* statuses = setOf("completed", "in the process", "failed", "canceled")  
 (1..n).forEach **{** Campaign.insert **{  
 it**[name] = *F*.ancient().titan()  
 **it**[customer] = *F*.name().fullName()  
 **it**[earning] = *F*.number().randomDouble(2, 500000, 100000000).toBigDecimal()  
 **it**[spending] = *F*.number().randomDouble(2, 0, 10000000).toBigDecimal()  
 **it**[executionStatus] = *if* (Random.nextInt(1, 100) >= 70) statuses.random() *else* statuses.first()  
 **}  
 }** }  
}  
  
*object* Mission: Table("mission"), *Generable* {  
  
 *val* missId = integer("miss\_id").autoIncrement().primaryKey()  
 *val* campId = reference("camp\_id", Campaign.campId)  
 *val* startDateAndTime = datetime("start\_date\_and\_time")  
 *val* endDateAndTime = datetime("end\_date\_and\_time")  
 *val* legalStatus = bool("legal\_status")  
 *val* departureLocation = text("departure\_location")  
 *val* arrivalLocation = text("arrival\_location")  
 *val* enemies = text("enemies")  
  
 @InternalAPI *override fun* generateAndInsert(n: Int) {  
 *val* campIds = Campaign.selectAll().map **{ it**[Campaign.campId] **}** (1..n).forEach **{** *val* st = *F*.date().between(  
 Date.from(LocalDate.of(2014, 3, 18).atStartOfDay(ZoneId.systemDefault()).toInstant()),  
 Date()  
 )  
 *val* et = *F*.date().between(st, Date())  
 Mission.insert **{  
 it**[campId] = campIds.random()  
 **it**[startDateAndTime] = st.toLocalDateTime()  
 **it**[endDateAndTime] = et.toLocalDateTime()  
 **it**[legalStatus] = Random.nextBoolean()  
 **it**[departureLocation] = "${*F*.address().latitude()} ${*F*.address().longitude()}"  
 **it**[arrivalLocation] = "${*F*.address().latitude()} ${*F*.address().longitude()}"  
 **it**[enemies] = arrayOf(*F*.nation().nationality(), *F*.name().fullName()).random()  
 **}  
 }** }  
}  
  
*object* Transport: Table("transport"), *Generable* {  
  
 *val* transId = integer("trans\_id").autoIncrement().primaryKey()  
 *val* name = text("name").nullable()  
 *val* type = text("type").nullable()  
 *val* status = text("status")  
  
 *override fun* generateAndInsert(n: Int) {  
 *val* statuses = setOf("available", "under\_repair", "destroyed", "broken")  
 *val* rawData = File("resources/static/transports.json").readLines().joinToString(separator = "")  
 *val* jsonBody = JSONObject(rawData).getJSONArray("transport")  
 jsonBody.forEachIndexed **{** i, el **->** *val* w = jsonBody.getJSONObject(i)  
 Transport.insert **{  
 it**[name] = w.getString("name")  
 **it**[type] = w.getString("type")  
 **it**[status] = statuses.random()  
 **}  
 }** }  
}  
  
*object* WeaponsInEquipment: Table("equip\_weapon"), *Generable* {  
  
 *val* equipId = reference("equip\_id", Equipment.equipId).nullable()  
 *val* weaponId = reference("weapon\_id", Weapon.weaponId).nullable()  
  
 *override fun* generateAndInsert(n: Int) {  
 *val* weaponIds = Weapon.selectAll().map **{ it**[Weapon.weaponId] **}** Equipment.selectAll().map **{ it**[Equipment.equipId] **}**.forEach **{** e **->** WeaponsInEquipment.insert **{  
 it**[equipId] = e  
 **it**[weaponId] = weaponIds.random()  
 **}  
 }** }  
}  
  
*object* TransportOnMissions: Table("missions\_transport"), *Generable* {  
  
 *val* missId = reference("miss\_id", Mission.missId).nullable()  
 *val* transId = reference("trans\_id", Transport.transId).nullable()  
  
 *override fun* generateAndInsert(n: Int) {  
 *val* transIds = Transport.selectAll().map **{ it**[Transport.transId] **}** Mission.selectAll().map **{ it**[Mission.missId] **}**.forEach **{** m **->** *if* (Random.nextBoolean())  
 TransportOnMissions.insert **{  
 it**[missId] = m  
 **it**[transId] = transIds.random()  
 **}  
 }** }  
}  
  
*object* Inspection: Table("inspection"), *Generable* {  
  
 *val* empId = reference("emp\_id", Employee.empId)  
 *val* transId = reference("trans\_id", Transport.transId)  
 *val* serviceDate = date("service\_date").defaultExpression(CurrentDateTime().date()).nullable()  
  
 @InternalAPI *override fun* generateAndInsert(n: Int) {  
 *val* transIds = Transport.selectAll().map **{ it**[Transport.transId] **}** Employee.leftJoin(Position)  
 .slice(Employee.empId, Position.name)  
 .select **{** Position.name inList listOf("mechanic", "engineer") **}** .map **{ it**[Employee.empId] **}** .forEach **{** e **->** *if* (Random.nextBoolean())  
 Inspection.insert **{  
 it**[empId] = e  
 **it**[transId] = transIds.random()  
 **it**[serviceDate] = *F*.date().between(  
 Date.from(LocalDate.of(2014, 3, 18).atStartOfDay(ZoneId.systemDefault()).toInstant()),  
 Date()  
 ).toLocalDateTime().toLocalDate()  
 **}  
 }** }  
}  
  
*object* EmployeeOnMission: Table("missions\_emp"), *Generable* {  
  
 *val* missId = reference("miss\_id", Mission.missId).nullable()  
 *val* empId = reference("emp\_id", Employee.empId).nullable()  
  
 *override fun* generateAndInsert(n: Int) {  
 *val* missIds = Mission.selectAll().map **{ it**[Mission.missId] **}** Employee.leftJoin(Position).select **{** Position.rank neq "" **}**.map **{ it**[Employee.empId] **}**.forEach **{** e **->** EmployeeOnMission.insert **{  
 it**[missId] = missIds.random()  
 **it**[empId] = e  
 **}  
 }** }  
}

*package* com.testpassword  
  
*import* com.github.javafaker.Faker  
*import* io.ktor.util.\*  
*import* org.jetbrains.exposed.sql.\*  
*import* org.json.JSONObject  
*import* java.io.File  
*import* java.time.LocalDate  
*import* java.time.ZoneId  
*import* java.util.\*  
*import* kotlin.random.Random  
  
*val F* = Faker()  
  
*class* ReferenceEntityError(e: Table): Error() {  
 *override val* message = "Reference entity table ${e.tableName} does not contains such records"  
}  
  
*object* RecordsGenerator {  
  
 *fun* fillDb(sizes: *Map*<*Generable*, Int>) = sizes.forEach **{ it**.key.generateAndInsert(**it**.value) **}**}