База резюме

Имеются следующие тестовые данные в виде резюме:

1. ФИО: Петров Петр Петрович

Дата рождения: 12.12.1986

Контакты: +375(29)123-45-67, http://github.com/petya, petrovich@gmail.com

Пол: мужчина

Технологии: Git, Spring Boot, HTML

2. ФИО: Иванов Иван Иванович

Дата рождения: 4.4.1997

Контакты: +375(29)87-65-43, http://github.com/vanya, skype:ivanko

Пол: мужчина

Технологии: Git, JavaEE, Java Core

3. ФИО: Морская Мария Васильевна

Дата рождения: 07.11.1999

Контакты: +375(29)999-99-99, https://www.linkedin.com/in/mariya/

Пол: женщина

Технологии: Maven, REST, Spring

Первая часть.

Необходимо спроектировать базу данных и заполнить её тестовыми данными указанными выше. Тип базы данных - Oracle или PostgreSQL. Результатом этой части задания должен быть текстовый файл с SQL-инструкциями для создания и заполнения базы данных (CREATE TABLE… INSERT… вот это всё). Выполняя это задание важно помнить про нормализацию баз данных.

Вторая часть.

Необходимо создать Maven-проект (jar), в которой описать JPA-сущности (@Entity) для таблиц, созданных в первом задании. А также набор классов, который помогал бы строить простые SQL-запросы. Фактически,

механизм должен давать нам возможность получать данные из БД без знания SQL. Что-то вроде собственной реализации Criteria API.

Механизм должен предоставлять возможность указать, что мы хотим получить (например объекты класса «Резюме») а так же критерии отбора (например «имя» равно «Петр» и «фамилия» равна «Петров»).

Минимальные требования к реализации:

· ваш класс должен уметь отдельно возвращать строку запроса и отдельно значения параметров запроса.

· для построения параметров запроса должен использоваться паттерн «Строитель».

Например: exp.equal(**"name"**, **"Петр"**).and().equal(**"surname"**, **"Петров"**).and().equal(**"patronymic"**, **"Петрович"**);

· генерироваться должен SQL-запрос, но нужно предусмотреть возможность, чтобы когда-нибудь в будущем кто-то другой смог бы добавить реализацию, которая будет генерировать JPQL или HQL и при этом не пришлось бы переписывать ваши классы.

Также в составе проекта должно быть два теста:

1. Составление (при помощи механизма описанного выше) и выполнение запроса, который бы выбрал все резюме, в которых фамилия равна «Морская», имя = «Мария», отчество = «Васильевна» и как верный результат - получение третьего резюме из БД

2. Составление (при помощи механизма описанного выше) и выполнение запроса, который бы выбрал все резюме, в которых фамилия заканчивается на «ов» или пол – женский и получение всех трёх резюме из бд.

Глобальная задача – сделать механизм как можно более удобным для использования. С одной стороны, в условиях ограниченного времени вам важно реализовать только тот функционал, который будет проверен тестами. С другой стороны вам нужно построить механизм так, чтобы его было удобно расширить новым функционалом (например в будущем добавить in, between, больше и меньше и так далее).

Исходя из предыдущего абзаца важно отметить, что оцениваться будет не объем функционала, а его качество и удобство использования.

Важно постараться сделать механизм максимально независимым. Например, сделать так, чтобы если имя таблицы поменятся, нам не пришлось бы вносить изменения в тест. Только в Entity.

Так же важно постараться сделать механизм гибким. Статические методы и классы-утилиты сильно снижают гибкость (например, мы не можем их переопределять при наследовании), а связи объектов только при помощи интерфейсов, наоборот, повышают и гибкость и независимость.

Все эти моменты (за исключением минимальных требований) не являются обязательными, но могут помочь вам получить преимущество.

Результатом выполнения этой части задания должен быть архив с исходными кодами проекта (каталог src и pom.xml).