#### Задача 1

Напишите скрипт с вручную заданными параметрами либо хранимую процедуру, которая получает на вход дату и идентификатор пользователя, а возвращает состояние этого пользователя на указанную дату на основе таблиц аудита.

Не забудьте о связи с таблицей статусов пользователя и механизме soft delete.

#### Задача 2

Переделайте скрипт (процедуру) из задачи 1 так, чтобы он работал и в случае, если ключ сущности составной.

#### Задача 3

Доработайте скрипт (процедуру) так, чтобы на вход можно было подать имя таблицы, дату, первичный ключ (простой или составной) и получить состояние сущности на эту дату.

#### Задача 4

Напишите скрипт, который выгружает объединённые данные по торгам из таблиц cislink.\* и tander.\* со следующими полями:

* наименование товара;
* потребность сети;
* место поставки (РЦ);
* дата требуемой поставки;
* дата окончания торгов;
* статус торгов (активные, архивные);
* цена, установленная сетью;
* минимальная предложенная цена;
* название торговой сети.

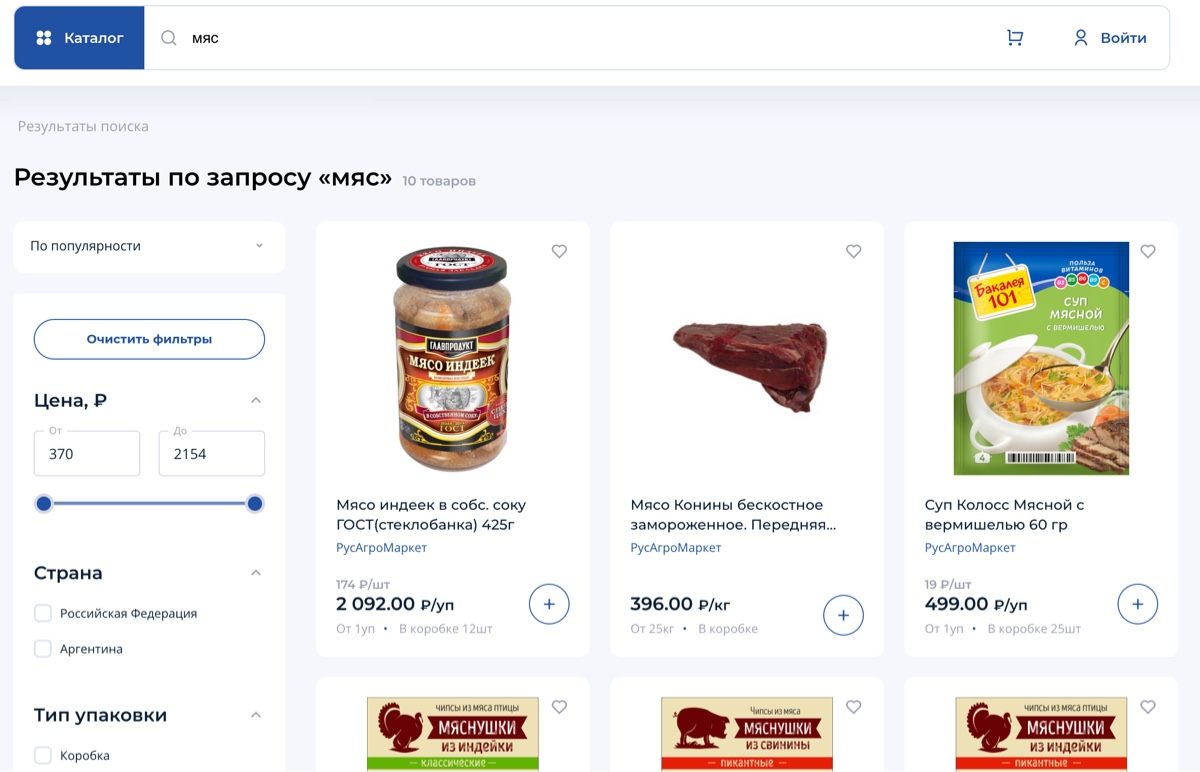
Итоговые данные должны быть типизированы (числовые колонки должны иметь определённый тип данных — в отличие от исходных данных, которые хранятся в строковом виде). Обратите также внимание, что часть данных из источников может отсутствовать или не вписываться в схему. Предложите варианты, как такие данные можно трансформировать.

#### Задача 5

Напишите процедуру получения списка доступных фильтров для поиска продуктов. На вход подаётся текстовая строка поиска, а также набор фильтров по следующим полям:

* цена;
* страна происхождения;
* тип упаковки.

Необходимо вернуть значения этих фильтров (цена, страна и тип упаковки), для которых при указанной текстовой строке будут получены какие-либо результаты. То есть, если пользователь захочет уточнить запрос (пример ниже), нужно показать ему, при каких значениях фильтров по этому запросу будет выдан хотя бы один продукт.



#### Советы и рекомендации

* Некоторые сервисы (например, Businessprofile и Userprofile) имеют таблицы для хранения аудита: данные, которые описывают для каждой строки и ячейки, кто, когда и какое значение установил. Такие таблицы позволяют восстановить значение каждой строки на конкретное время.
* Некоторые сущности (например, Company, User, Preorder и другие) имеют отдельную таблицу для хранения истории изменения их статусов с соответствующими названиями: {EntityName}Status: UserStatus, CompanyStatus и так далее. Связь между таблицей статусов и основной таблицей осуществляется по внешнему ключу OwnerId = ID.
* Часть данных хранится в ненормализованном виде в формате JSON: например, все дополнительные параметры уведомлений вынесены в словарь Payload, в котором хранятся пары «ключ — значения», необходимые для уточнения адресатов уведомлений.
* Во всех сущностях реализован принцип soft delete: данные не удаляются из таблиц, а лишь помечаются удалёнными через поле SoftDeletedLevel. Это поле равно нулю для актуальных сущностей и больше нуля — для удалённых.
* В логике soft delete необходимо учесть возможность каскадного удаления зависимых сущностей. Недостаточно просто пометить строку флагом «удалена», нужно решить, как быть с зависимыми сущностями: их тоже нужно помечать удалёнными и откатывать обратно так, чтобы восстановилось исходное состояние.

Этот механизм детально описан в [серии статей на The Reformed Programmer](https://www.thereformedprogrammer.net/ef-core-in-depth-soft-deleting-data-with-global-query-filters/#building-solution-3-cascade-softdeleteservice).