Тестовое задание для PHP back-end разработчика

1. Скидка на продукты

Есть продукты A, B, C, D, E, F, G, H, I, J, K, L, M. Каждый продукт стоит определенную сумму.

Есть набор правил расчета итоговой суммы:

1. Если одновременно выбраны A и B, то их суммарная стоимость уменьшается на 10% (для каждой пары A и B)
2. Если одновременно выбраны D и E, то их суммарная стоимость уменьшается на 5% (для каждой пары D и E)
3. Если одновременно выбраны E,F,G, то их суммарная стоимость уменьшается на 5% (для каждой тройки E,F,G)
4. Если одновременно выбраны A и один из [K,L,M], то стоимость выбранного продукта уменьшается на 5%
5. Если пользователь выбрал одновременно 3 продукта, он получает скидку 5% от суммы заказа
6. Если пользователь выбрал одновременно 4 продукта, он получает скидку 10% от суммы заказа
7. Если пользователь выбрал одновременно 5 продуктов, он получает скидку 20% от суммы заказа
8. Описанные скидки 5,6,7 не суммируются, применяется только одна из них
9. Продукты A и C не участвуют в скидках 5,6,7
10. Каждый товар может участвовать только в одной скидке. Скидки применяются последовательно в порядке описанном выше.

Необходимо написать программу на PHP, которая, имея на входе набор продуктов (один продукт может встречаться несколько раз) рассчитывала суммарную их стоимость.

Программу необходимо написать максимально просто и максимально гибко. Учесть, что список продуктов практически не будет меняться, также как и типы скидок. В то время как правила скидок (какие типы скидок к каим продуктам) будут меняться регулярно.

Все параметры задаются в программе статически (пользовательский ввод обрабатывать не нужно). Оценивается подход к решению задачи.
Тщательное тестирование решения проводить не требуется.

В задаче оценивается:

- Выполнение задания
- Подход к решению задачи
- Время выполнения

2. Перестройка предложения

Пожалуйста, разработайте функцию\класс для "перемешивания" предложения.

Символ | является разделителем слов-вариантов. Например:

"{Пожалуйста|Просто} сделайте так, чтобы это {удивительное|крутое|простое} тестовое предложение {изменялось {быстро|мгновенно} случайным образом|менялось каждый раз}."

На выходе должно получаться:

"Пожалуйста сделайте так, чтобы это крутое тестовое предложение изменялось каждый раз." или "Просто сделайте так, чтобы это удивительное тестовое предложение изменялось мгновенно случайным образом".

В задаче оценивается:

- Выполнение задания
- Подход к решению задачи
- Время выполнения

3. Работа с БД

Создать один sql-запрос, который возвратит данные, представленные в следующей таблице:

OrderName	TestersName

Заказ 1	AA A, BB B (Auto-dealer.ru), DD D (autoPotrebitel.ru), CC C (autoRATING.ru)
Заказ 2	BB B (Auto-dealer.ru), CC C (autoRATING.ru)
Заказ 3	BB B, AA A (Auto-dealer.ru)
Заказ 4	DD D (autoPotrebitel.ru)

Данная выборка представляет собой список заказов с привязанными к ним тестерами. Причем, тестеры перечисляются через запятую, и в скобках указывается организация, к которой они принадлежат. Если в колонке TestersName получается несколько тестеров, которые принадлежат одной организации, то наименование организации не дублируется, а указывается только один раз в конце.

Вторая часть задачи (вопрос на пять) - как написать наиболее оптимальный запрос для двух крайних абстрактных случаев:

а) очень много данных (десятки тысяч записей)

б) очень мало данных (десятки, сотни записей)

Данные к задаче:

```
CREATE TABLE Order_Tester (
 OrderID int(11) NOT NULL default '0',
 TesterID int(11) NOT NULL default '0',
 PRIMARY KEY (OrderID,TesterID),
 KEY OrderID (OrderID),
 KEY TesterID (TesterID)
```

```
) ENGINE=InnoDB DEFAULT CHARSET=cp1251;
INSERT INTO Order_Tester VALUES (1, 1);
INSERT INTO Order_Tester VALUES (1, 2);
INSERT INTO Order_Tester VALUES (1, 3);
INSERT INTO Order_Tester VALUES (1, 4);
INSERT INTO Order_Tester VALUES (2, 2);
INSERT INTO Order_Tester VALUES (2, 3);
INSERT INTO Order_Tester VALUES (3, 1);
INSERT INTO Order_Tester VALUES (3, 2);
INSERT INTO Order_Tester VALUES (4, 4);
CREATE TABLE Orders (
 ID int(11) NOT NULL auto_increment,
 Name varchar(50) default NULL,
 PRIMARY KEY (ID)
) ENGINE=InnoDB DEFAULT CHARSET=cp1251;
INSERT INTO Orders VALUES (1, 'Order 1');
INSERT INTO Orders VALUES (2, 'Order 2');
INSERT INTO Orders VALUES (3, 'Order 3');
INSERT INTO Orders VALUES (4, 'Order 4');
CREATE TABLE Organizations (
 ID int(11) NOT NULL default '0',
 Name varchar(50) default NULL,
 PRIMARY KEY (ID)
) ENGINE=InnoDB DEFAULT CHARSET=cp1251;
INSERT INTO Organizations VALUES (1, 'Auto-dealer.ru');
INSERT INTO Organizations VALUES (2, 'autoRATING.ru');
INSERT INTO Organizations VALUES (3,
'autoPotrebitel.ru'); CREATE TABLE Testers (
```

```
 ID int(11) NOT NULL auto_increment,
 FirstName varchar(50) default NULL,
 LastName varchar(50) default NULL,
 OrganizationID int(11) NOT NULL default '0',
 PRIMARY KEY (ID),
 KEY OrganizationID (OrganizationID)
) ENGINE=InnoDB DEFAULT CHARSET=cp1251;
INSERT INTO Testers VALUES (1, 'A', 'AA', 1);
INSERT INTO Testers VALUES (2, 'B', 'BB', 1);
INSERT INTO Testers VALUES (3, 'C', 'CC', 2);
INSERT INTO Testers VALUES (4, 'D', 'DD', 3);
ALTER TABLE `Order_Tester`
 ADD CONSTRAINT order_tester_ibfk_2 FOREIGN KEY (TesterID) REFERENCES
testers (ID),
 ADD CONSTRAINT order_tester_ibfk_1 FOREIGN KEY (OrderID) REFERENCES
orders (ID);
```