Дополнительные задачи

TASK #1

1. Что будет, если один игрок вызовет параллельно 1000 раз один REST метод?

Произойдет нарушение консистентности данных, пользователь сможет списать больше монет чем у него есть на балансе. Это произойдет из-за того, что валидация данных будет происходить с уже "устаревшим" балансом, с которого уже списали монеты в параллельном запросе.

2. Как избежать возможных последствий?

Работать с балансом игрока в транзакции с уровнем изоляции `Serializable` или блокировать строку через `FOR UPDATE` .

3. Какая последовательность действий в операции? Расширьте логику

Достаточно работать с балансом внутри транзакции с уровнем изоляции `Serializable` , с учетом что такой подход снижает производительность СУБД:

1. Валидируем токен игрока, если не валиден выдаем ошибку - errors.validate.token
2. Открываем транзакцию
3. Запрос в таблицу **REWARDS**: id = rewardId & campaign_id = campaignId. Если записи нет, выдаем ошибку - errors.reward.notFound и закрываем транзакцию
4. Запрос в таблицу **PLAYERS**: id = player_id & campaign_id = campaignId. Если запись не найдена, выдаем ошибку - errors.player.notFound и закрываем транзакцию
5. **ВАЛИДАЦИЯ ДАННЫХ**
    - Если **PLAYERS**.balance_coins < **REWARDS**.cost_coins, выдаем ошибку - errors.validate.cost и закрываем транзакцию
6. **ОБВНОВЛЯЕМ PLAYERS**
    - balance_coins = current - **REWARDS**.cost_coins
    - balance_crystals = current + **REWARDS**.award_crystals
7. Закрываем транзакцию
8. Возвращаем ответ на фронт

TASK #2

1. Какую БД предлагаете использовать для хранения и обработки записей и почему?

Clickhouse из-за его высоких показателей RPS и возможности безболезненно оптимизировать хранение данных под указанную задачу.

2. Как предлагаете оптимизировать нагрузку на сервер и БД?

Как вариант можно дополнительно создать материализованное представление в Clickhouse с engine `SummingMergeTree` с группировкой по `player_id` и дате `toDate(event_time)` . При начислении баллов сумма за день будут автоматически обновляться. Это даст уменьшение количества хранимых данных. Указав ключами сортировки следующие столбцы:

1. `toYear(event_time) as year`
2. `toMonth(event_time) as month`
3. `toDate(event_time) as day`

, можно добиться ускорения выполнения запроса.

TASK #3

1. Почему описанная выше схема не сработает? Опишите какой будет сценарий

Схема не сработает т.к. в public REST запросе внутри транзакции происходит блокировка строки с балансом пользователя в таблице **PLAYERS**, но к этой же строке обратиться уже **INTERNAL REST = consume** на изменение баланса в таблице **PLAYERS**. Внутренний метод не сможет успешно выполнить запрос на изменение баланса и выдаст ошибку errors.balance.notEnough.

2. Как защитить себя от паралельных одинаковых запросов из игры? по сути от фрода

Можем ограничить количество запросов в минуту от пользователя или кэшировать запрос с хранением в минуту.

3. Какие могут быть последствия от параллельных запросов из игры? Например, один пользователь отправит одновременно 1000 запросов

Нарушением конститнености данных и излишней нагрузкой на HTTP сервер, СУБД.

TASK #4

Ниже описаны решения пунктов 1-3 и 6, пункты 4-5 решены в пунктах 1-3. По идее таблицу **PLAYERS** стоит декомпозировать на несколько таблиц, чтобы не смешивать справочную информацию, пароль и баланс игрока. Но не стал этого делать, чтобы не отходить от поставленной задачи.

Обозначения: * Синий цвет - primary key ** - индексируемые параметры * C - создание * U - обновление * R - обязательное поле или нет

1. Описать таблицу профиля игрока DAILYBOX_PROGRESS

Поле `first_accr_reward_time` необходимо для проверки периода, когда можно получить награды. Поле `last_accr_reward_time` нужно для проверки факта отсутствия начислений за текущий день. И поле `next_reward_id` хранит идентификатор следующей награды для обеспечения последовательного начисления наград, по умолчанию стоит ID первой награды.

Атрибут	Тип	Описание	Значение	C	U	R
id	int	идентификатор прогресса	1	serial		+
player_id*	int	идентификатор игрока	1	+		+
first_accr_reward_time*	timestamp	дата и время начисления первой награды	now	+		+
last_accr_reward_time	timestamp	дата и время начисления последней награды	yesterday	+	+	+
next_reward_id	int	идентификатор следующей награды	1	+	+	+

2. Описать таблицу профиля игрока PLAYERS

Атрибут	Тип	Описание	Значение	C	U	R
id *	int	идентификатор игрока	1	serial		+
name	string	имя игрока	alex	+	+	+
email	string	почта игрока		+	+	
balance_coins	int	баланс монет	100	+	+	+
balance_crystals	int	баланс кристаллов	0	+	+	+
password	string	пароль игрока	qwer123	+	+	+

3. Описать таблицу настроек ежедневных наград DAILYBOX_REWARDS

Поле `next_reward_id` указывает на идентификатор следующей награды после текущей, изменяя это поле можно регулировать последовательность выдачи наград. В поле `expiration_period` хранится количество дней отведенных на сбор всего набора. По аналогии можно контролировать период начисления для каждой награды, а не глобально как описано сейчас.

Атрибут	Тип	Описание	Значение	C	U	R
id *	int	идентификатор награды	1	serial		+
award_crystals	int	сколько выдает кристаллов	1	+	+	+
cost_coins	int	стоимость в монетах	1	+	+	+
next_reward_id	int	идентификатор следующей награды	1	+	+	
expiration_period	int	кол-во дней на сбор всего набора	1	+	+	+

6. Описать логику REST запроса из игры для авторизованного игрока

REST API game/v1/playerId:int/get-dailybox-reward

request

```
TYPE: Post
HEADER: autorization=accessToken
URL: playerIdId=int
Payload: {}
```

response

```
{
    "rewardId": int, // идентификатор награды
    "balanceCoins": int, // текущий баланс монет
    "balanceCrystals": int // текущий баланс кристаллов
}
```

1. Валидируем токен игрока, если не валиден выдаем ошибку - errors.validate.token
2. Открываем транзакцию
3. SELECT FOR UPDATE **PLAYERS**: id = playerId. Если запись не найдена, выдаем ошибку - errors.player.notFound и закрываем транзакцию
4. SELECT FOR UPDATE **DAILYBOX_PROGRESS**: player_id = playerId. Если запись не найдена, создаём новую запись с дефолтными значениями, но player_id присваиваем значение **PLAYERS**.id
5. Запрос в таблицу **DAILYBOX_REWARDS**: id = **DAILYBOX_PROGRESS**.next_reward_id. Если записи нет, выдаем ошибку - errors.reward.notFound и закрываем транзакцию
6. **ВАЛИДАЦИЯ ДАННЫХ**
    - Если **PLAYERS**.balance_coins < **DAILYBOX_REWARDS**.cost_coins, выдаем ошибку - errors.validate.cost и закрываем транзакцию
    - Если toDate(**DAILYBOX_PROGRESS**.last_accr_reward_time) = toDate(now), выдаем ошибку - errors.validate.doubleAccr и закрываем транзакцию
    - Если now - **DAILYBOX_PROGRESS**.first_accr_reward_time > **DAILYBOX_REWARDS**.expiration_period, выдаем ошибку - errors.validate.expirationPeriod и закрываем транзакцию
7. **ОБВНОВЛЯЕМ PLAYERS**
    - balance_coins = current - **DAILYBOX_REWARDS**.cost_coins
    - balance_crystals = current + **DAILYBOX_REWARDS**.award_crystals
8. **ОБВНОВЛЯЕМ DAILYBOX_PROGRESS**
    - last_accr_reward_time = now
    - next_reward_id = **DAILYBOX_REWARDS**.next_reward_id

9. Закрываем транзакцию
10. Возвращаем ответ на фронт