# Задача 11

## Постановка задачи

1) Имеется некоторый пул транзакций для загрузки в OeBS из внешней системы (для простоты пусть это будет таблица apps.t\_import , а сама транзакция содержит только номер, дату и сумму ).

2) Загрузка транзакций выполняется загрузчиком, который вызовет стандартный public API, в виде 10 параллельных сессий, вместе которые обеспечивают загрузку данных со скоростью не менее 200 транзакций в секунду.  
3) Исходный пул постоянно пополняется новыми данными, это могут быть дубликаты или обновления уже имеющихся записей.

## Допущения

1) Транзакции не зависимы друг от друга. Например транзакция с ID 125 никак не зависит от транзакции с ID 126.

2) Пункт 3 указывает, что пул пополняется данными, включая обновления имеющихся записей. Полагаю, что это означает появление транзакций в пуле с существующим в системе ID.

Например, в системе есть транзакция

{id: 123, date: 01.01.2019, amount:100, creation\_date:01.01.2019, last\_update\_date:01.0.2019}.

В пул поступает сообщение

{id: 123, date: 01.01.2019, amount:200}

Никто не мешает добавить в пул сообщение для той же транзакции вида

{id: 123, date: 01.01.2019, amount:300}

Какое из сообщений считать последним и какой в итоге должна быть сумма - не ясно. Если последовательность применения сообщений измерять по моменту добавления в пул, то необходимо маркировать данные при доставке в пул. Однако надежнее маркировать в самом сообщении, например, время возникновения события в системе-источнике. Тогда корректность данных не будет зависеть от времени их загрузки в пул (ведь пулов тоже может быть несколько, как и загрузчиков).

В любом случае, в сообщении необходмо иметь атрибут, который характеризует последовательность применения сообщений. Мы не станем рассматривать реализацию этого требования, как часть загрузчика.

Пример

{id: 123, date: 01.01.2019, amount:200, update\_date: 03.02.2019}

{id: 123, date: 01.01.2019, amount:300,update\_date: 02.02.2019}

Тогда реальная сумма на число 05.02.2019 должна быть 200, хотя сообщения пришли в другом порядке.

Для верной обработки подобных ситуаций необходимо сравнивать последнее время изменения транзакции в системе (или другой маркер) и время в сообщении и, например, игнорировать сообщение, если оно возникло раньше, чем последнее изменение транзакции (то есть в системе более актуальные данные, чем в сообщении). Такую проверку логично возложить на само АПИ- сделать обертку стандартной АПИ и до вызова стандартного АПИ сверять сначала время.

Таким образом мы отделяем логику решения зависимостей от загрузчика, упрощая его разработку.

Отсюда следует, что Для загрузчика не важна последовательность обработки сообщений.

## **Реализация**

### Постоянное чтение пула

Этот метод состоит в том, что все сессии конкурентно читают таблицу доступом full scan и питаются залочить строки. Если блокировка накладывается успешно, то происходит обработка – вызов стандартного АПИ, если не успешно, то ищется следующая свободная строка.

В oracle, начиная с версии 11g, существует конструкция select for update skip locked, с помощью которой просто реализовать этот метод. Я выбрал для реализации именно его, уделив большое внимание на тестирование, тем более что он позвояет удовлетворить требования. Детали по инсталяции и запуску тестов в отдельном документе HowTo.doc

Плюсы

- не требуются индексы на таблице пула

- простая реализация

Минусы

- теоретически, обработка должна быть медленнее, чем при разделении пула на пачки.

### Возможные варианты реализации

Существует множество вариантов реализации обработки пула сообщений . Наиболее приемлемые решения в рамках поставленных требований, на мой взгляд, я привожу ниже.

Система состоит из трех существенных частей: загрузчиков, пула данных и обработчиков. В постановке задачи есть требования по скорости обработки данных и количестве потоков – 10 сессий и 200 вызовов стандартного АПИ в секунду, это около 20 вызовов для одного потока, в среднем 0,05 сек на вызов. Вероятно, основную часть времени будет все-таки выполняться АПИ, требования к нему уже достаточно жесткие.

По условию задачи пул пополняется новыми данными, которые также нужно обработать. В идеале, для беспрерывной работы интерфейса, скорость поступления данных должна быть не меньше скорости обработки. Ограничение снизу это 200 операций в секунду, что не составляет проблем. Это дает возможность дополнительных проверок на входе, до попадения в пул, например проверка дубликатов (см отдельный параграф)

Основной задачей является все-таки обработка сообщений из пула, где критически важно не потерять сообщения, важно не повторять обработку одних и тех же сообщений. Требование о дубликатах рассматривается в отдельном параграфе.

Работа обработчика состоит из нескольких частей – выборка данных из пула, вызов АПИ и удаление данных из пула. Этап вызова АПИ от нас не зависит, основная сложность в выборке.

Так как выборка будет паралельной, то важно не допустить блокировки сессий друг другом и не допустить обработки одного сообщения разными сессиями. Для этого необходимо перед обработкой наложить блокировку на сообщение, используя различные конструкции for update nowait, for update skip locked.

В конкурентной среде не желательно применение множества индексов на таблице пула, так как на их поддержку также уходит время. Записи в таблице постоянно удаляются и появляются снова, индекс становится разреженным и более глубоким, чем необходимо. Поэтому по возможности стоит воздержаться от индексирования пула.

Для уменьшения конкуренции за строки и блоки между разными сессиями возможно применить разделение пула на группы\пачки строк.

### Разделение на группы по rowid

Вариант предполагает предварительно разделить таблицу с данными на N пачек (chunks), каждая из которых состоит из нижней и верхней границы rowid. Для ускорения создания списка частей не обязательно читать весь пул, достаточно запросить словарь данных, таблица dba\_extends. Этот подход реализован в стандартном пакете oracle dbms\_parallel\_execute

1) запускается процедура определения пачек, при обращении к словарю

2) запускаются обработчики, которые забирают пачки и обрабатывают их. Доступ к таблице пула происходит по rowid, никаких индексов не требуется.

3) по окончанию обработки своей пачки обработчик берет следующую пачку и так пока все пачки не будут обработаны.

Однако за это время в таблице появились новые данные, их нужно снова обработать. Тогда обработчик (или отдельная сессия- менеджер) запускает снова нарезку пачек (пункт 1) и снова идет обработка.

Сложности подхода следующие

1) когда кончаются все пачки и когда нарезаются новые, то возможна пауза в обработке данных. Хотя нарезка занимает мало времени, все же пауза не желательна. Для этого можно сделать отдельную сессию-менеджера, которая следит за наличием пачек и добавляет новые.

2) вставка в таблицу происходит неравномерно, данные могут добавляться в одни и те же extents, а другие после обработки и очистки, могут оставаться пустыми. Для более равномерного распределения данных по extents возможно применить секционирование таблицы пула (partitioning) по хэш-функции, например на основе ID транзакции.

Плюсы

- не нужно создавать индексы в пуле

- прямой доступ по rowid к таблице

Минусы

- средняя сложность реализации

- возможны паузы в обработке

- неравномерное распределение строк между обработчиками

Этот метод более подходит для обработки больших статических таблиц без добавления данных в пул.

### Разделение на группы по ID

Этот метод похож на предыдущий, только в качестве ключа разделения берется ID сущности в пуле, тоже вариант реализации dbms\_parallel\_execute

Плюсы

- более равномерное разделение по пачкам

Минусы

- необходим индекс по ID

- средняя сложность реализации

- возможны паузы в обработке

### Advanced Queue

В этом методе в качестве пула выбирается очередь oracle, которая позволяет одновременно и добавлять данные и считывать с помощью стандартного АПИ oracle.

Не все загрузчики позволяют добавлять данные в очередь, однако это легко можно обойти с помощью INSTEAD OF TRIGGER на вьюшке, который будет класть в очередь. Тогда загручики будут вставлять во вьюшку также, как в обычную таблицу.

Плюсы

- простота реализации

- забота о блокировках лежит на oracle API, который многократно протестирован

Минусы

-затруднен поиск дубликатов в непосредственно в пуле

Если бы не требование с дубликатами, я бы предпочел этот проверенный метод.

## **Борьба с дубликатами**

Допущение: Дубликатами являются сообщения с одинаковыми значениями бизнес-полей и служебных полей типа даты изменения\события.

Представим себе, что пул данных еще пустой. Избежать обработку дубликатов можно либо на стадии записи данных в пул, либо на стадии их обработки потребителем.

Оптимальнее предотвращать появление дубликатов на стадии добавлении в пул, тогда дубликаты не появляются вообще и не требуют доп. места и обработки.

Другой способ – проверка на дубликаты обработчиком. Таким образом, мы признаем, что в пуле уже есть дубликаты и не производим обработку, если сообщение является дубликатом.

### Полное избавление от дубликатов

Единственной реализацией в Oracle, гарантирующей уникальность данных при наличии конкуренции за ресурсы, являются уникальные ключи. Уникальные индексы способны сами взять на себя заботу об отсутствии дубликатов при большом количестве параллельных вставок.

Есть и минусы - такая реализация требует места в памяти\жестком диске, замедляет саму вставку. При добавлении в таблицу с ключем похожих данных одновременно возникают горячие блоки, когда затрягиваются те же листовые блоки индекса, например с последовательными ключами типа sequence. В добавок если 2 сессии вставляют записи с одним значением ключа и на таблице есть уникальный индекс, то одна сессия блокируется, пока другая не закончит транзакцию. Нужно понимать, что полное избавление от дубликатов обходится совсем не бесплатно.

Выбраный и реализованный мной подход отсеивает дубликаты при вставке в пул с помощью instead of trigger for view. Загрузчик может использовать View как обычную таблицу и делать вставку в нее. Тригер ищет такие же записи в пуле перед вставкой и если они найдены, то это считается дубликатом и его добавление в пул не происходит.

Этот подход хорошо работает, когда нет нескольких сессий, одновременно вставляющих данные с тем же ID транзакции, что очень маловероятно. К тому же цена дубликата достаточно мала, всего 0,05 сек, что вместе с малым риском события делает этот способ достаточно хорошим.

Плюсы

- простая реализация

- удовлетворяет требованиям в скорости вставки, более 200 записей в секунду.

Минусы

- требуется создание индекса на таблице пула

- возможны дубликаты, если несколько загрузчиков одновременно вставляют данные с теь же ID

### Дополнительные вопросы

*1) Может ли одна и та же транзакция одновременно или со сдвигом по времени грузиться всеми 10 параллельными потоками?*

Возможно при определенной реализации. Например, если не разделять пул на части и не запрашивать блокировку перед чтением очередного сообщения. Просто читать курсорами из пула.

*2) Как организовать пул таким образом, чтобы заранее не назначать (не распределять) задания между параллельными сессиями?*

Реализации прямым чтением и advanced queue подробно расписаны выше.

3) *Какие нюансы могут возникнуть, если внешних систем, использующих общий пул, будет несколько?*

Как обсуждалось выше, важно иметь метку, которая определит последовательность возникновения события, даже если оно пришло из нескольких систем.

При выбраной мною реализации подавления дубликатов, одновременная вставка транзакций с одним ID разными системами, могут все-таки возникнуть дубликаты в пуле. Я думаю это маловероятно и не критично, подробнее в соотв. разделе

*4)Если пулов с параллельным доступом будет несколько?*

Явно сложнее будет бороться с дубликатами.