Тюнинг планов обмена 1С
(решение проблемы ожидания на блокировках СУБД)

Предлагаемое решение основано на предыдущих публикациях автора на эту тему, а также на его практическом опыте:

Планы обмена 1С

https://infostart.ru/1c/articles/899200/

Анализ блокировок СУБД: таблица изменений плана обмена 1С

https://infostart.ru/1c/articles/1346977/

Использование таблиц СУБД в качестве очередей сообщений

https://infostart.ru/1c/articles/1214312/

Описание проблемы.

Планы обмена 1С широко используются в качестве **механизма регистрации изменений** данных прикладных объектов 1С с тем, чтобы впоследствии экспортировать их во внешние, заинтересованные в этих данных, информационные системы.

Для этих целей платформа создаёт для каждого объекта метаданных 1С, участвующего в обмене, таблицу регистрации изменений. При этом выполняется, так называемая, регистрация без данных, то есть в этой таблице регистрируется только первичный ключ объекта. Это аналогично механизму регистрации изменений SQL Server, который называется **change tracking** (не путать с *change data capture*, который выполняет регистрацию с данными).

Особенностью реализации change tracking от 1С является то, что на каждое изменение, сколько бы их ни было, в таблице регистрации создаётся **всегда только одна запись**.

Параллельно над этой записью может выполняться несколько отличных друг от друга или идентичных по функциональности операций, а именно:
1. Регистрация изменения (создание или обновление).
2. Выгрузка изменений (изменение статуса записи).
3. Квитирование доставки получателем (удаление записи).
Таким образом **запись** изменения становится объектом конфликта (**конкурентного доступа**) со стороны различных транзакций.

Практика показывает, что наибольшие проблемы создаёт операция выгрузки изменений, которая порождается вызовом метода менеджера планов обмена "ВыбратьИзменения". Однако, любая из вышеуказанных операций может быть причиной длительного ожидания на блокировках и создавать проблемы. Особенно ярко это проявляется в высоко нагруженных системах.

Решение проблемы.

К сожалению, сама архитектура механизма 1С такова, что **простого решения этой проблемы нет.** Кроме всего прочего потому, что на таблицах регистрации изменений основана ещё и логика разрешения возможных коллизий данных.

Коллизии данных - это отдельная большая тема, которая здесь не будет освещаться. Для этого есть другие публикации:

Обнаружение и разрешение коллизий данных: альтернативная реализация типовой стратегии РИБ 1С
https://infostart.ru/1c/articles/1377988/

Распределённые алгоритмы РИБ 1С
https://infostart.ru/1c/articles/1469979/

Тем не менее, для решения подобных проблем с ожиданием на блокировках существуют типовые приёмы программирования СУБД. Одним из таких инструментов являются триггеры.

Суть решения заключается в том, чтобы "расшить" конкурентный доступ таким образом, чтобы минимизировать его влияние или полностью устранить. В данном случае эта цель достигается тем, что одна запись превращается в несколько.

Другими словами каждое изменение должно регистрироваться как отдельная запись или событие. Постольку поскольку изменение схемы данных типовой таблицы 1С невозможно (платформа просто поломается от этого), необходимо создать для этих целей отдельную таблицу — очередь событий. При помощи триггеров операция записи в типовую таблицу регистрации изменений 1С будет перенаправляться в нашу таблицу, каждый раз используя только команду **INSERT.**

При этом сама платформа 1С ничего об этом "знать" не будет. Для неё этот механизм абсолютно прозрачен. Единственным нюансом, однако, будет являться только то, что типовая таблица 1С не будет содержать соответствующих записей, наличие которых в некоторых случаях используется алгоритмом разрешения коллизий данных РИБ 1С.

Стоит отметить, что удаление триггера СУБД будет автоматически возвращать систему в исходное состояние использования типовых алгоритмов 1С.

Реализация нашего "коварного плана" для различных СУБД может быть разной, в зависимости от тех или иных особенностей конкретной СУБД. Кроме этого, нужно учесть **одновременное использование типового и альтернативного механизмов** регистрации изменений, например, для различных планов обмена.

Важный нюанс: *таблица регистрации изменений объекта метаданных 1С всегда одна и используется всеми планами обмена в равной степени!*

Мы рассмотрим два варианта: SQL Server и PostgreSQL.

Для обоих вариантов СУБД необходимо предварительно создать таблицу-очередь. Кроме этого, если ничего создавать не хочется, то можно использовать для этих целей соответствующий регистр сведений 1С.

Сделаем допущение, что мы выполняем тюнинг таблицы регистрации изменений для справочника "Номенклатура" и она называется, например, **_ReferenceChngR123**. Решение для регистров будет аналогичным за исключением того факта, что первичный ключ в таком случае может быть составным.

Альтернативная таблица-очередь (код SQL Server).

```
CREATE TABLE ОчередьИзменений
(
   НомерСообщения bigint IDENTITY(1,1) PRIMARY KEY,
   ПланОбмена binary(4)  NOT NULL,
   УзелОбмена binary(16) NOT NULL,
   Ссылка     binary(16) NOT NULL -- первичный ключ объекта 1С
);
```

Решение для SQL Server.

Программирование решения при помощи **INSTEAD OF** триггеров для SQL Server не слишком удобное по причине того, что в коде триггеров придётся повторять логику SQL кода от 1С для тех операций, которые должны использовать типовую регистрацию изменений, а не альтернативную, что имеет два недостатка:
1. Этот код от 1С может поменяться (маловероятно).
2. Этот код будет сложно поддерживать (очень вероятно).

В конечном итоге был выбран вариант **AFTER** триггера.

```
-- Команды UPDATE и DELETE от 1С игнорируются
-- (Что за команды такие и почему — см. в статьях выше)
-- Исключение: есть какие-то удивительные особенности
--             конкретной информационной базы 1С ...

CREATE TRIGGER tr_its_a_magic ON _ReferenceChngR123
AFTER INSERT NOT FOR REPLICATION
AS

-- Настройка альтернативного механизма для конкретного узла обмена
-- Если фильтр по плану обмена или узлу не нужен, то следует
-- отредактировать предложение WHERE соответствующим образом

DECLARE @ПланОбмена binary(4)  = 0x00000024;
DECLARE @УзелОбмена binary(16) = 0x9CD5408D5C93CC8E11ED1E6FC225E9C8;

-- 1. Следует иметь ввиду, что в триггер может "прилететь" набор
из нескольких записей, например, по нескольким узлам обмена.

-- 2. Использование DELETE - "костыль" из-за отказа от INSTEAD OF
триггера. Мы удаляем только что вставленную в типовую таблицу
регистрации изменений 1С запись и тут же перенаправляем её в нашу.

DELETE _ReferenceChngR123
OUTPUT deleted._NodeTRef, deleted._NodeRRef, deleted._IDRRef
  INTO ОчередьИзменений(ПланОбмена, УзелОбмена, Ссылка)
  FROM _ReferenceChngR123 INNER JOIN inserted
    ON _ReferenceChngR123._NodeTRef = inserted._NodeTRef
   AND _ReferenceChngR123._NodeRRef = inserted._NodeRRef
   AND _ReferenceChngR123._IDRRef   = inserted._IDRRef
 WHERE _ReferenceChngR123._NodeTRef = @ПланОбмена
   AND _ReferenceChngR123._NodeRRef = @УзелОбмена
```

Код триггера в принципе очень простой.

Проще придумать не удалось =)

Решение для PostgreSQL.

Реализация триггера для PostgreSQL требует наличия триггерной функции. Кроме этого, в данной СУБД удобнее всего использовать **BEFORE** триггер. Код функции и триггера аналогичен коду SQL Server, поэтому оставляю его без особых комментариев.

```
CREATE OR REPLACE FUNCTION fn_change_tracking_magic()
RETURNS trigger AS $$ BEGIN

 IF NEW._nodetref = '\\x00000024'
AND NEW._noderref = '\\x9cd7408d5c93cc8e11ed313e285cd6a6' THEN

   INSERT INTO ОчередьИзменений(ПланОбмена, УзелОбмена, Ссылка)
   VALUES (NEW._nodetref, NEW._noderref, NEW._idrref);

   RETURN NULL; -- Отменяем выполнение команды 1С

ELSE

   RETURN NEW; -- Продолжаем выполнение команды 1С

END IF;

END $$ LANGUAGE 'plpgsql';
```

Код триггера, использующего функцию выше:

```
CREATE TRIGGER tr_change_tracking_magic
BEFORE INSERT ON _referencechngr264
FOR EACH ROW
EXECUTE PROCEDURE fn_change_tracking_magic();
```

Практическая реализация.

Следует напомнить, что триггер создаётся для каждого объекта метаданных 1С в отдельности. Создание и управление всеми необходимыми триггерами вручную может создавать практическую сложность применения альтернативного механизма регистрации изменений.

В связи с этим предполагается реализация соответствующего административного интерфейса пользователя в **DaJet Studio**, а также использование **автоматической генерации** необходимого **кода SQL** для обеих вышеуказанных СУБД. Для экспорта данных из регистра сведений можно использовать средства самой 1С, а из таблицы-очереди предполагается использовать скрипты (запросы) **1QL.**