
PLARIUM

/ UDEV MEETUP /

Работа с MS SQL Server в высоконагруженном проекте

Павел
Матлашов

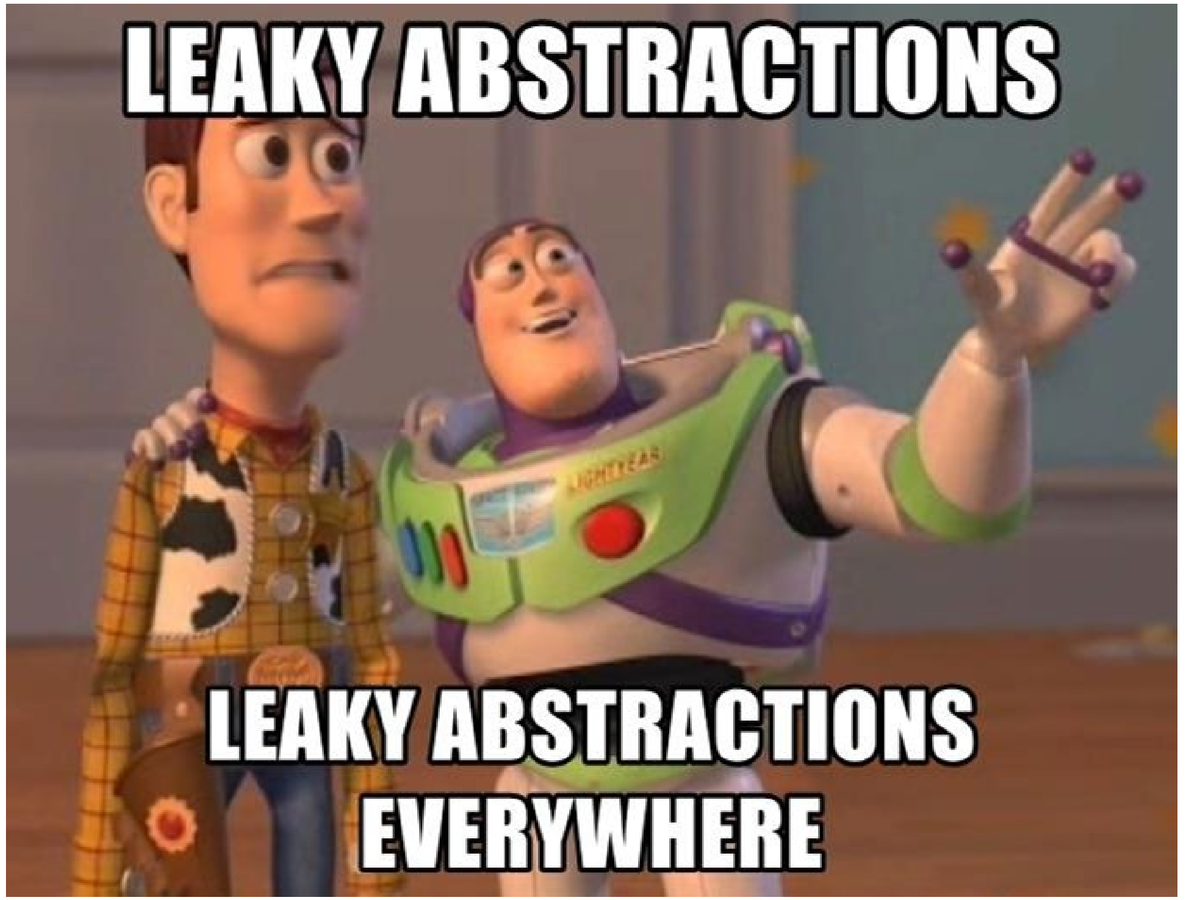
Head of Game Technology Division

PLARIUM

CONTENTS

ПОЧЕМУ МЫ ЗДЕСЬ СОБРАЛИСЬ?

Как и многие другие прекрасные концепции и реализованные на их базе инструменты, SQL и RDBMS во многом задумывались как нечто универсальное и идеальное. Вам всего лишь нужно составить модель, запросы на декларативном языке, и инструмент всё сделает за вас оптимальным способом и без багов.
На деле всё, как обычно, оказалось иначе :)



ПОЧЕМУ МЫ ЗДЕСЬ СОБРАЛИСЬ?

Скорее всего, для эффективного решения сложных задач вам придется узнать о физической структуре БД, страницах, блокировках, взаимоблокировках, логах, кешах, планах выполнения запросов, подсказок для запросов, статистике, соединениях, индексах, транзакциях, уровнях изоляции, протоколах, пулах соединений и многом другом...

/ 2 /

Базовые принципы использования MS SQL в highload-проектах

ПОЧЕМУ КОГДА-ТО ВЫБРАЛИ ИМЕННО MS SQL SERVER?

- Был хороший опыт
- Проверенный продукт
- ACID-гарантии
- Хорошо поддерживает гибридную модель
- NoSql-решения того времени были недостаточно хороши

ПОЧЕМУ ДО СИХ ПОР ИСПОЛЬЗУЕМ MS SQL?

- Опыта стало значительно больше
- Стоимость миграции на другие решения значительна
- Риск при миграции на другие решения высок
- NoSql-решения нашего времени не до конца нас удовлетворяют

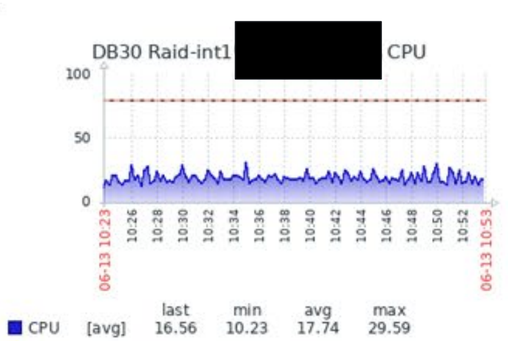
КЛЮЧЕВЫЕ ОСОБЕННОСТИ:

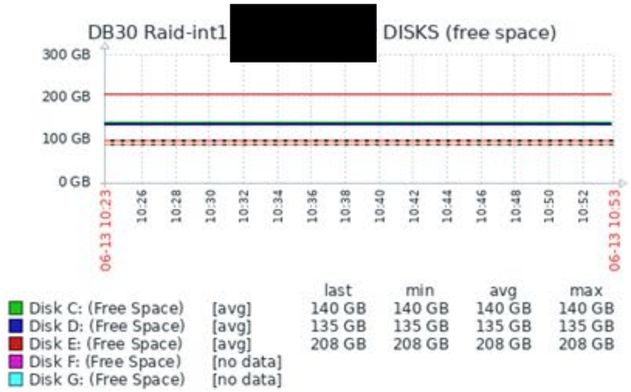
- ADO.NET + легковесная обертка
- Простая схема данных, мало таблиц
- Простые миграции только вверх + расширение для поддержки независимости модулей
- Очень много денормализации, вплоть до NoSQL-подхода в хранении;

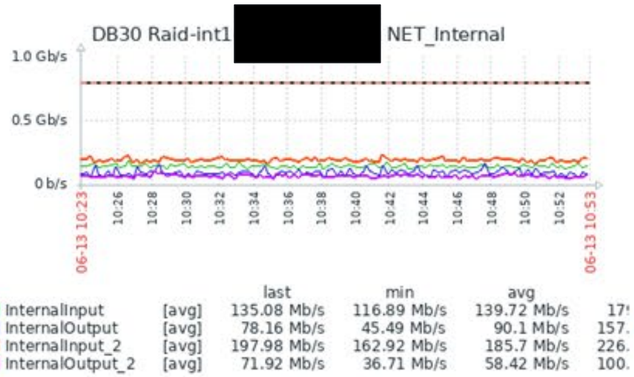
  при этом необходимы и ACID-свойства
- Внимательное отношение к индексам

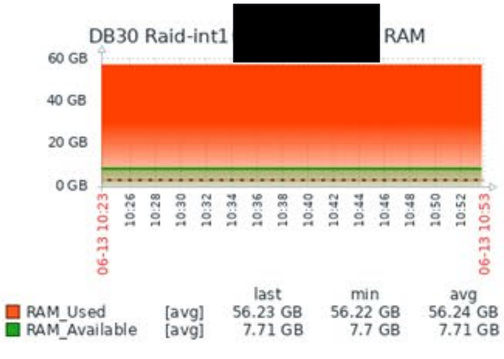
ИНСТРУМЕНТЫ ДЛЯ МОНИТОРИНГА И ДИАГНОСТИКИ

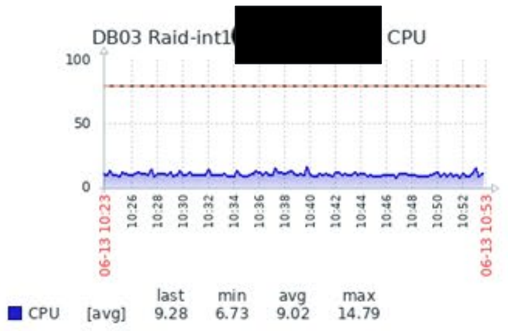
Zabbix

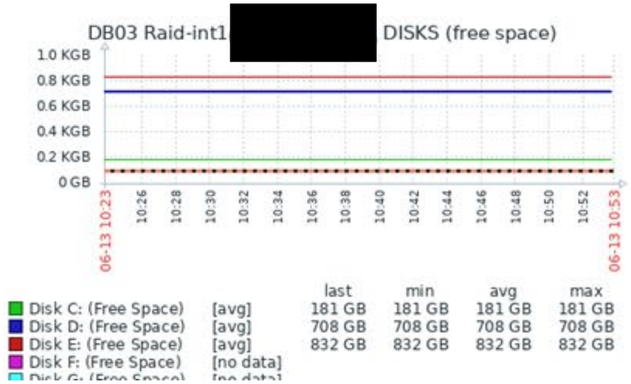


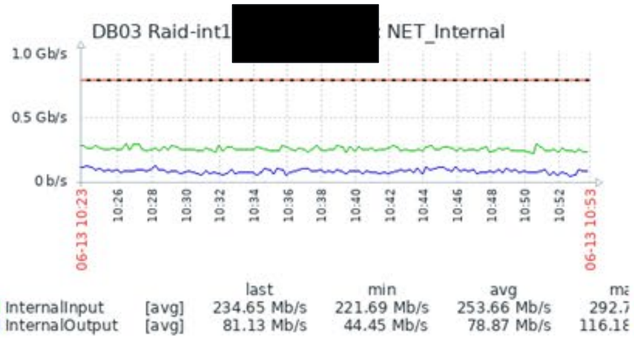


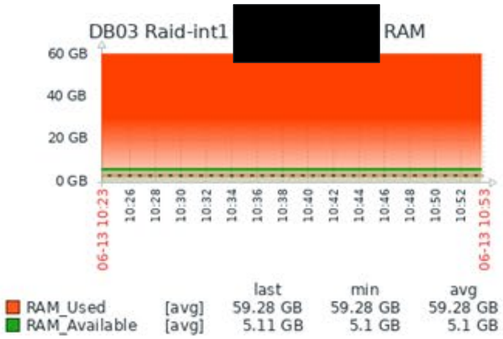












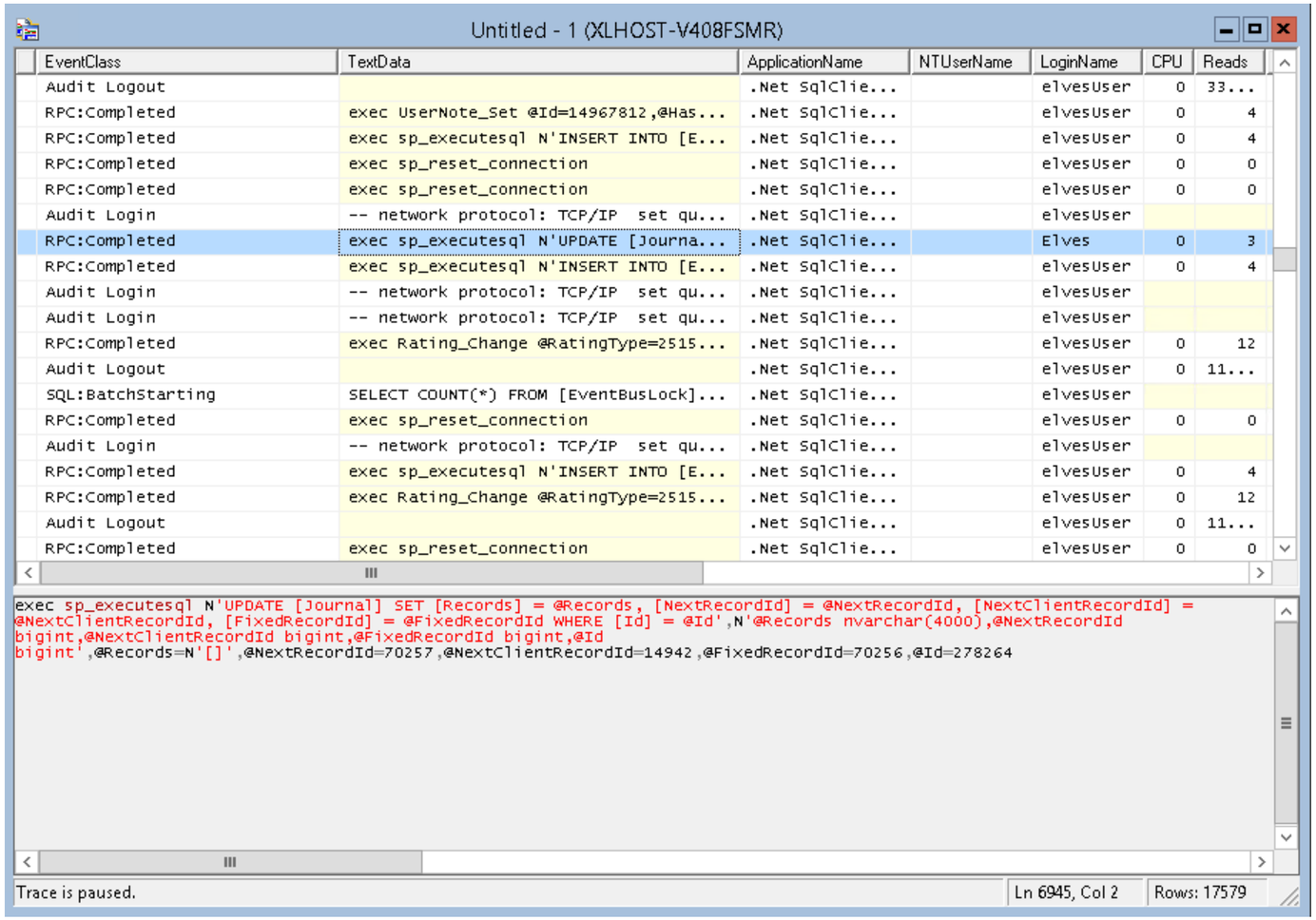
ИНСТРУМЕНТЫ ДЛЯ МОНИТОРИНГА И ДИАГНОСТИКИ

Profiler

db_UpdateUser	111.22	0.6% 142295194	29.7% (6.8%)	26.9% 77.51% (3%)	33.05 / 7.51 ms	25.61 / 2.81 ms	7.43 / 4.69 ms	0 (0) ... 23940 (1250) ms

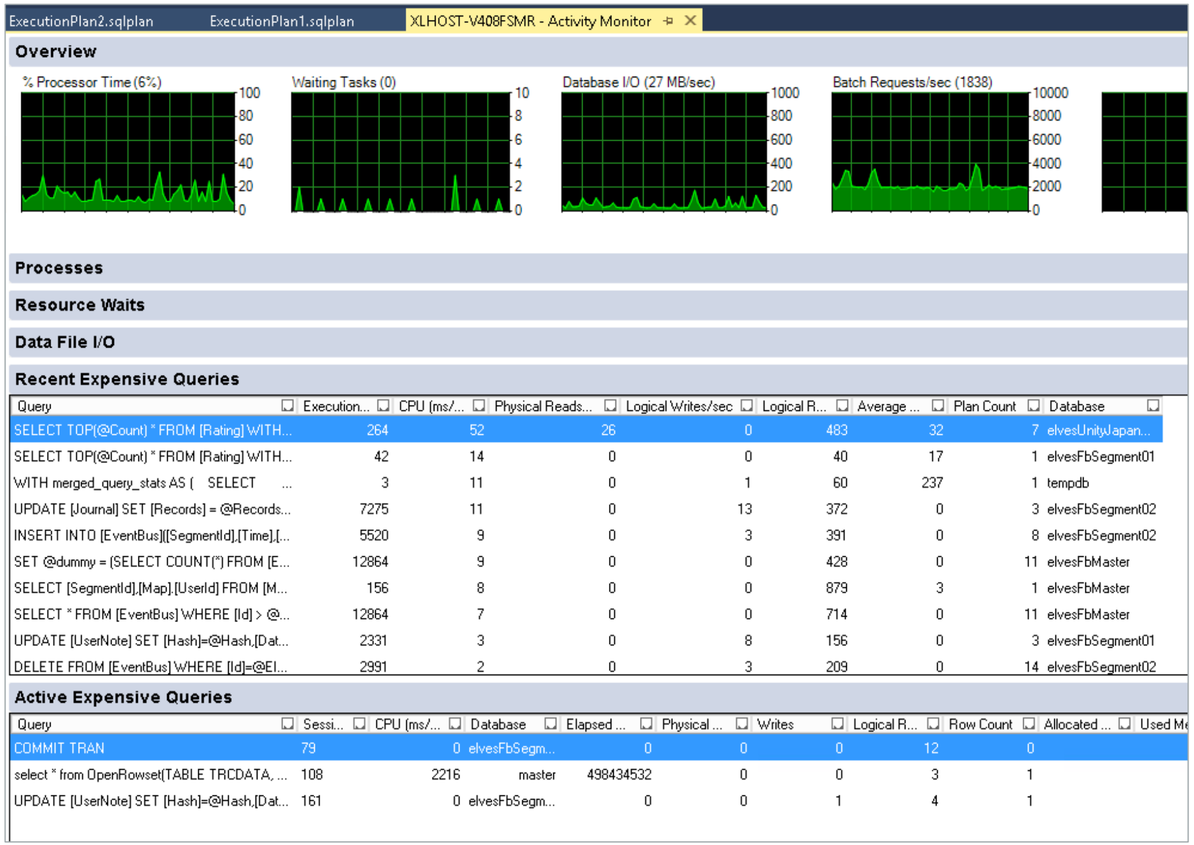
ИНСТРУМЕНТЫ ДЛЯ МОНИТОРИНГА И ДИАГНОСТИКИ

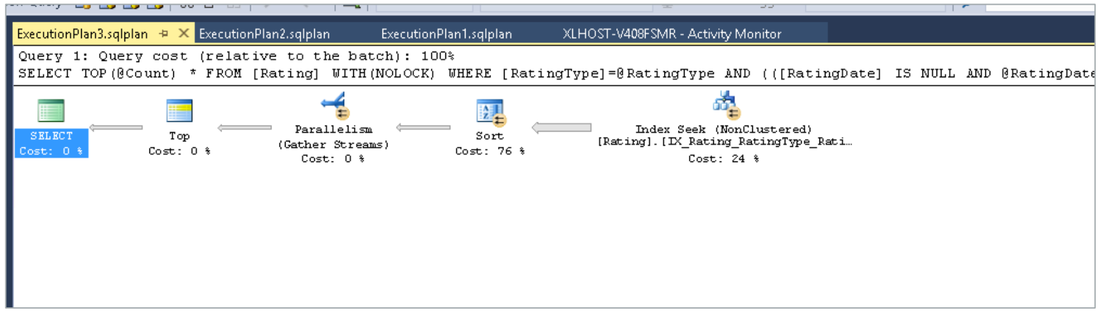
SQL Server Profiler

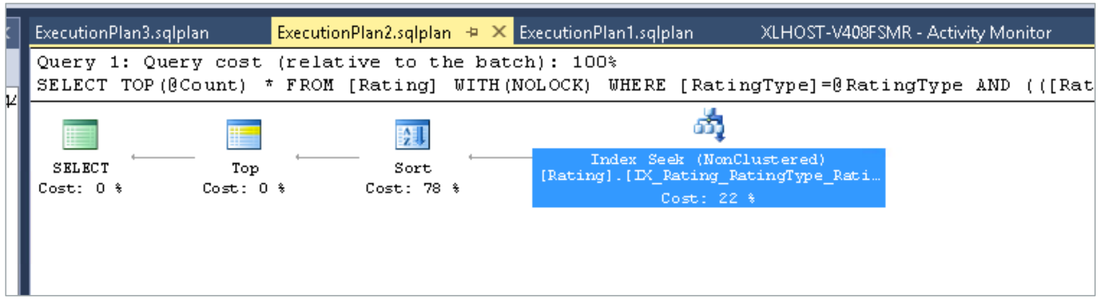


ИНСТРУМЕНТЫ ДЛЯ МОНИТОРИНГА И ДИАГНОСТИКИ

SQL Server Management Studio







/ 3 /

Всего понемногу

ПЕРВИЧНЫЕ КЛЮЧИ, ОГРАНИЧЕНИЯ, ИНДЕКСЫ

- В основном суррогатные ключи
- Балансируем создание индексов под сценарии использования
- Индексы не только для select, но и для update
- Не боимся включать данные в индекс
- Стараемся проверять эффективность и применимость индексов на тестовых данных
- Мониторим продакшн
- Меньше ограничений на уровне базы, исключения

ПОДСКАЗКИ ПЛАНИРОВЩИКУ

NOLOCK

- Сначала использовали массово, полагаясь на синхронизацию на уровне приложения
- Со временем поймали несколько багов с большими полями и жизненным циклом соединений и транзакций
- Избавились во многих местах

UPDLOCK/TABLOCK/HOLDLOCK и т. д.

- Применяли во многих местах в процессе познания MS SQL :)
- Почти от всего избавились, кроме хитрых случаев (о них дальше)

FORCESEEK/INDEX

- В некоторых случаях планировщик запросов строит странные планы
- После анализа пытаемся подсказать ему, на который из индексов обратить внимание
- Используем аккуратно

XACT_ABORT

```
USE AdventureWorks2012;
GO
IF OBJECT_ID(N't2', N'U') IS NOT NULL
    DROP TABLE t2;
GO
IF OBJECT_ID(N't1', N'U') IS NOT NULL
    DROP TABLE t1;
GO
CREATE TABLE t1
    (a INT NOT NULL PRIMARY KEY);
CREATE TABLE t2
    (a INT NOT NULL REFERENCES t1(a));
GO
INSERT INTO t1 VALUES (1);
INSERT INTO t1 VALUES (3);
INSERT INTO t1 VALUES (4);
INSERT INTO t1 VALUES (6);
GO
```

```
SET XACT_ABORT OFF;
GO
BEGIN TRANSACTION;
INSERT INTO t2 VALUES (1);
INSERT INTO t2 VALUES (2); -- Foreign key error.
INSERT INTO t2 VALUES (3);
COMMIT TRANSACTION;
GO
SET XACT_ABORT ON;
GO
BEGIN TRANSACTION;
INSERT INTO t2 VALUES (4);
INSERT INTO t2 VALUES (5); -- Foreign key error.
INSERT INTO t2 VALUES (6);
COMMIT TRANSACTION;
GO
-- SELECT shows only keys 1 and 3 added.
-- Key 2 insert failed and was rolled back, but
-- XACT_ABORT was OFF and rest of transaction
-- succeeded.
-- Key 5 insert error with XACT_ABORT ON caused
-- all of the second transaction to roll back.
SELECT *
    FROM t2;
GO
```

XACT_ABORT. ДОПОЛНИТЕЛЬНЫЕ ОСОБЕННОСТИ.

Что *еще* будет, если этот флаг выключить?

- Клиентские тайм-ауты
- Жизненный цикл транзакций по задумке Microsoft
- Как же быть? :)

/ 4 /

Копаем глубже

ПОСЛЕДОВАТЕЛЬНАЯ ЗАПИСЬ/ЧТЕНИЕ

- Влияние денормализации на память
- Придумали TextBuffer, аналогично StringBuilder
- Заменили максимальное количество мест
- Снизили нагрузку на LOH
- Когда добрались до базы

ПОСЛЕДОВАТЕЛЬНАЯ ЗАПИСЬ/ЧТЕНИЕ — ПРИМЕРЫ

2 usages PLARIUM\k.shamushkina +1 *

```
protected RefCountedDisposable<TextBuffer> ReadColumnChars(IDataReader reader, int index)
{
  var result = new RefCountedDisposable<TextBuffer>(new TextBuffer());
  var buffer = new char[4096];
  long read, pos = 0;
  while ((read = reader.GetChars( i: index, pos, buffer,  bufferoffset: 0, buffer.Length)) > 0)
  {
    result.Value.Append(buffer,  index: 0,  count: (int) read);
    pos += read;
  }

  result.TryAddRef();
  return result;
}
```

2 usages PLARIUM\k.shamushkina +1 *

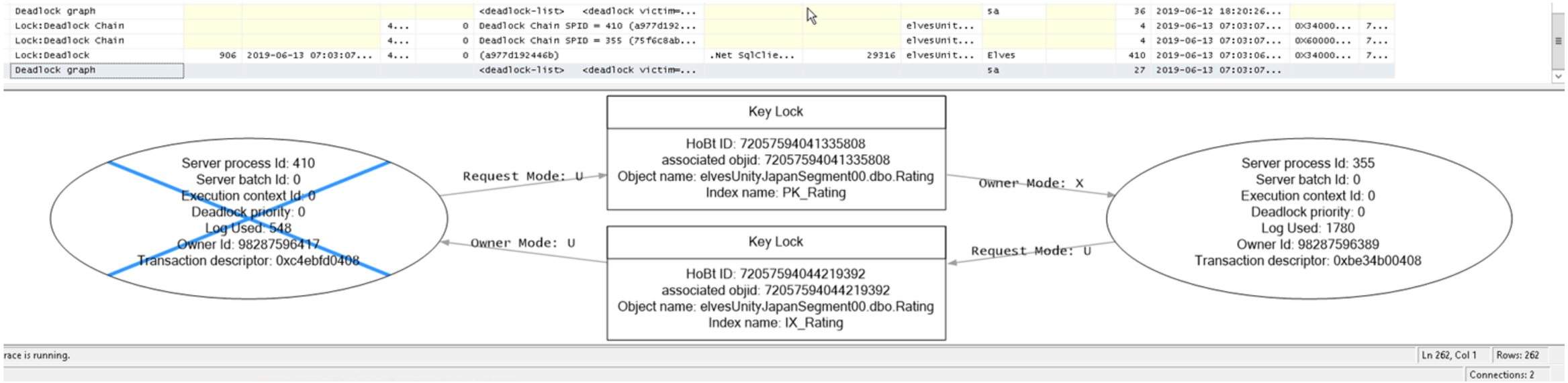
```
protected RefCountedDisposable<ChunkMemoryStream> ReadColumnBytes(IDataReader reader, int index)
{
  var result = new RefCountedDisposable<ChunkMemoryStream>(new ChunkMemoryStream());
  var buffer = new byte[4096];
  long read, pos = 0;

  while ((read = reader.GetBytes( i: index, pos, buffer,  bufferoffset: 0, buffer.Length)) > 0)
  {
    var count = (int) read;
    result.Value.Write(buffer,  offset: 0, count);
    pos += read;
  }

  result.Value.Seek( offset: 0, SeekOrigin.Begin);
  result.TryAddRef();
  return result;
}
```

НА ЧТО ОБРАЩАТЬ ВНИМАНИЕ

- IN, влияние на производительность
- Загрязнение кеша планов выполнения
- Альтернативы IN?
- Почему еще загрязняется кеш?
- Как может подвести преобразование типов?
- Bulk Insert/Upsert
- Пулы соединений

ВЗАИМОБЛОКИРОВКИ

- Причины возникновения
- Их вообще можно избежать?
- А в реальности?



ВЗАИМОБЛОКИРОВКИ

- Обнаруживаем по логам и статистике ошибок
- Используем SQL Server Profiler для отлова
- Стараемся соблюдать в основных транзакциях порядок обращения к таблицам
- Пробуем ***LOCK подсказки, чаще не получается
- Снижаем количество обращений к ресурсам
- Регулируем параллелизм на уровне приложения (concurrency codes etc.)

/ 5 /

Когда всё нужно брать в свои руки

ЗАДАЧА

- EventBus, краткое введение
- Гарантии – Eventual Consistency, Ordering
- Поэтому нужно обеспечить вычитку записей из таблицы в порядке их добавления.

  Просто “SELECT * FROM [EventBus] WHERE [Id] > @Id ORDER BY [Id]”?

  Этого достаточно?

ЗАДАЧА

- Не совсем :)
- Записи с меньшими Id появляются в таблице позже записей с большими
- Начали думать о неком барьере, попытке дождаться завершения всех транзакций
- Решения?

ПОПЫТКИ РЕШИТЬ ПРОБЛЕМУ

Попытка № 1

1. Окружаем запрос на вычитку TABLOCK - SELECT * FROM [EventBus] WITH(TABLOCK) WHERE [Id] > @Id ORDER BY [Id]
2. Мотивация?
3. Провал :)
4. Возможно, мы что-то не так понимаем?

ЛИРИЧЕСКОЕ ОТСТУПЛЕНИЕ

IDENTITY поле, что оно гарантирует? Посмотрим в MSDN

«Столбцы идентификаторов можно использовать для формирования значений ключей. Свойство идентификаторов столбца гарантирует следующее.

- **Каждое новое значение будет сформировано на основе текущих аргументов seed и increment.**
- **Каждое новое значение для определенной транзакции будет отлично от других параллельных транзакций для таблицы.**

Свойство идентификаторов столбца не гарантирует следующее.

- **Уникальность значения** – уникальность значения следует обеспечить с помощью ограничения PRIMARY KEY или UNIQUE либо индекса UNIQUE.
- **Последовательные значения в пределах транзакции** – при вставке транзакцией нескольких строк не гарантируется, что для них будут назначены последовательные значения. Это связано с тем, что в таблице могут выполняться другие параллельные операции вставки. Если значения должны быть последовательными, то транзакция должна использовать монопольную блокировку для таблицы или уровень изоляции SERIALIZABLE.
- **Последовательные значения после перезапуска сервера или других ошибок** – SQL Server может сохранять значения идентификаторов в кеше для обеспечения высокой производительности, и некоторые из присвоенных значений могут быть потеряны при сбое базы данных или перезагрузке сервера. Это может вызвать пропуски в значениях идентификатора при вставке. Если пропуски недопустимы, приложение должно использовать собственный механизм для создания значений ключей. Использование генератора последовательностей с параметром NOCACHE может привести к ограничению пропусков в незафиксированных транзакциях.
- **Повторное использование значений** – свойства идентификаторов, созданные конкретным свойством идентификатора с заданными аргументами seed и increment, не используются повторно подсистемой. Если определенная инструкция вставки завершается с ошибкой или производится ее откат, использованные значения идентификаторов не будут созданы повторно. Это может привести к появлению пропусков при создании последующих значений идентификаторов».

ПОПЫТКИ РЕШИТЬ ПРОБЛЕМУ

Попытка № 2

1. Моноблокировка на уровне приложения
2. Мотивация?
3. Провал :)

ПОПЫТКИ РЕШИТЬ ПРОБЛЕМУ

Попытка № 3

1. RW-блокировка на уровне приложения
2. Ну дальше понятно :)

ПОПЫТКИ РЕШИТЬ ПРОБЛЕМУ

Попытка № 4 **Теоретическая**

1. Можно сделать таблицу со следующим значением идентификатора события, и в транзакции в первую очередь увеличивать его и брать для нового события.
2. Почему не будем даже пробовать?

ПОПЫТКИ РЕШИТЬ ПРОБЛЕМУ

Попытка № 5 **Теоретическая, усложненная**

1. Делаем несколько next id записей, выбираем нужную посредством, например, делением id источника по модулю на количество и повторяем всё то же самое, что и в прошлой попытке.
2. Почему отказались?

ПОПЫТКИ РЕШИТЬ ПРОБЛЕМУ

Попытка № 6

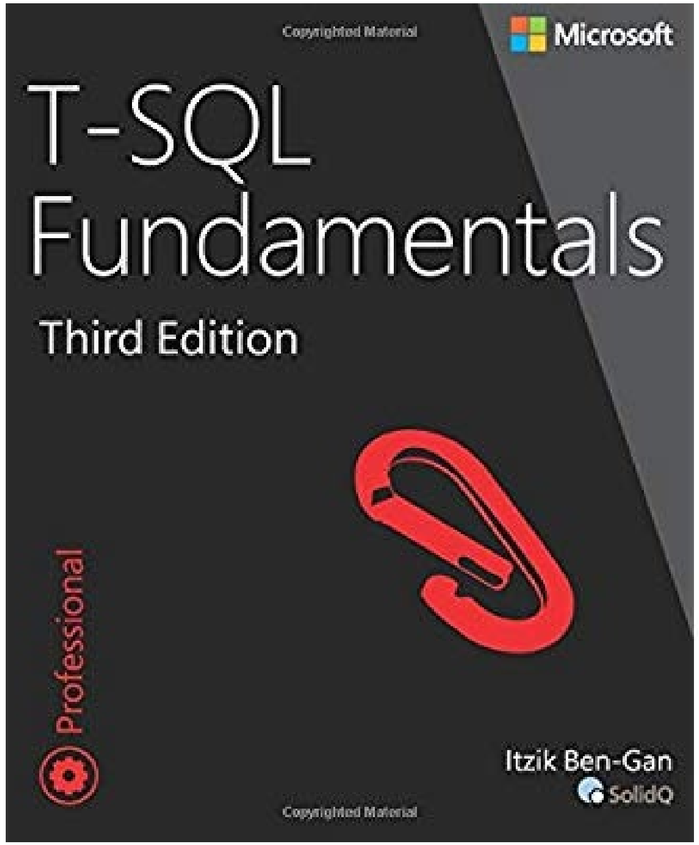
```
using (new PerfWatch( name: "eventbus_GetInsertLock"))
  cx.Sql("SELECT COUNT(*) FROM [EventBusLock] WITH (TABLOCK,HOLDLOCK)").Execute();
```

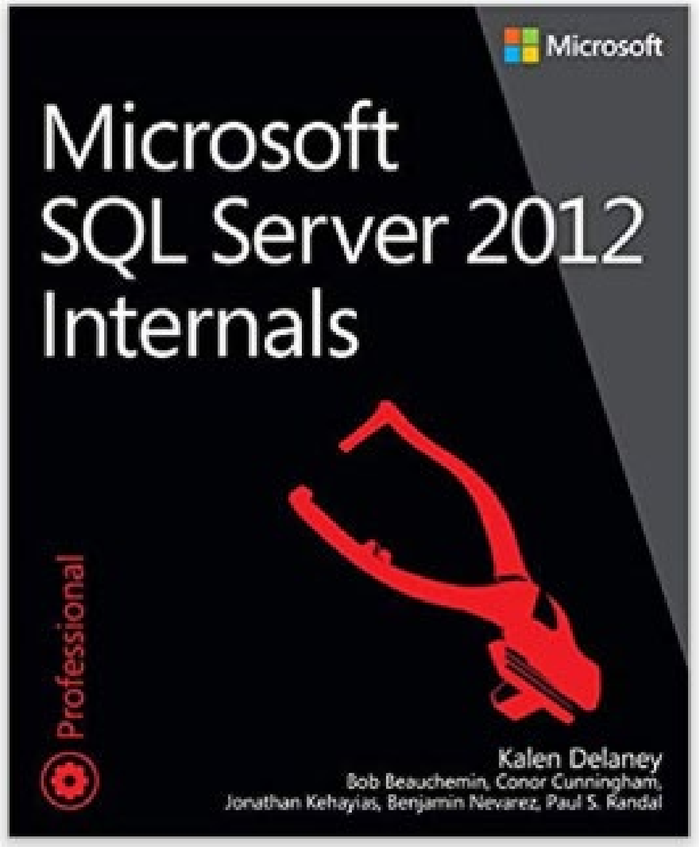
1. А может, сделаем RW-блокировку на уровне базы?
2. Добавляем таблицу предварительного блокирования, к которой каждая транзакция должна обратиться перед доступом в основную.
3. Вставляющие события транзакции берут в ней shared-блокировку и держат ее до конца.
4. Читающая транзакция берет в ней эксклюзивную блокировку.
5. Получаем нормальную очень локальную и короткую обратную RW-блокировку.
   Таким образом, генерация следующего значения производится с учетом порядка владения блокировкой.
6. PROFIT :)

```
? usages  ? overrides  Pavel Matlashov +1
private static List<EventBusEntity> GetLockedIfNeeded(DataContext cx)
{
  if (!EventBusSettings.Value.GuardEventInsertion) return null;

  using (new PerfWatch( name: "eventbus_GetLockedEvents"))
  {
    return cx.Sql("BEGIN TRANSACTION;" +
                  "DECLARE @dummy int;" +
                  "SET @dummy = (SELECT COUNT(*) FROM [EventBusLock] WITH (TABLOCKX));" +
                  "SELECT * FROM [EventBus];" +
                  "COMMIT TRANSACTION;")
      .ReadAll(ReadEvent);
  }
}
```

/ 6 /

ВЫВОДЫ?





THANK YOU

TAKE THE WORLD