

РЕАЛИЗАЦИЯ КОНСИСТЕНТНОСТИ ДАННЫХ В РАСПРЕДЕЛЕННЫХ СИСТЕМАХ

Павлов В.А.

*Московский государственный технологический университет «СТАНКИН»,
Москва, Россия*

Ключевые слова: ссылочная целостность данных, распределенные системы, консистентность, отслеживание удаления и изменения данных, микросервисная архитектура.

Аннотация. Одним из подходов при реализации высоконагруженных вычислительных систем является построение распределенных систем, где выполнение функций разделено между множеством узлов, которые, взаимодействуя друг с другом, решают поставленные задачи. Для обеспечения своей работы узлы могут использовать базы данных, которые, как и сами узлы, могут быть физически размещены на различных серверах. В таком случае остро встает вопрос поддержания ссылочной целостности (консистентности) данных, которыми оперируют узлы системы. В настоящее время разработаны методы поддержания консистентности в рамках одной БД, для решения данной задачи в условиях распределенных систем требуется разработать алгоритмы работы и архитектуру системы отслеживания изменения и удаления данных.

DATA CONSISTENCY IMPLEMENTATION IN DISTRIBUTED SYSTEMS

Pavlov V.A.

Moscow state university of technology "STANKIN", Moscow, Russia

Keywords: data referential integrity, distributed systems, data consistency, data changes and deletions tracking, microservice architecture.

Abstract. Distributed systems are an option to build high volume system, which consist of set separated nodes that interact with each other to provide required information. To ensure their work nodes may use databases, which, like the nodes themselves, may be physically located on different servers. In that case the task is to maintain data referential integrity (consistency). For now, there are methods for maintaining consistency within a single database and to solve this issue for distributed systems it is required to develop algorithms and architecture of the system to track changes and deletions of data.

Задача поддержания консистентности данных в условиях реализации распределенной вычислительной системы гарантирует работу с корректной (с точки зрения логики) и актуальной (с технической стороны) информацией каждому узлу системы. Консистентность или целостность данных – соответствие хранимых в базе данных информации логике и структуре, определенной для базы данных (БД). Современные СУБД отлично справляются с задачей поддержания целостности данных в рамках одной БД. Однако, в случае реализации распределенной системы чаще всего происходит разделение всех данных на разные БД, нередко физически размещенных на различных серверах и работающих под управлением различных СУБД. В таком случае требуется реализация отдельного узла, который отслеживает операции над данными и обеспечивает целостность данных. В случае

использования технологии EF Core для доступа к информации в БД данная задача сохраняет актуальность в виду того, что EF Core опирается на функции конкретной СУБД, к которой обращается за предоставлением данных. Соответственно при построении системы, реализующей в своей архитектуре одновременную работу с несколькими БД, приходится самостоятельно решать задачу консистентности данных, в виду того, что широко распространенных систем для решения поставленной задачи не существует. Решение данной задачи осложняется тем, что она опирается на конкретные архитектурные особенности, которые применяются в каждом отдельном проекте [1].

Согласно теореме CAP (Consistency, availability, partition tolerance) консистентность данных является одним из свойств в реализации систем распределенных вычислений:

- согласованность/целостность данных (Consistency) – гарантия непротиворечивости данных в каждый момент времени для всех узлов распределенной вычислительной системы;

- доступность (Availability) – гарантия корректного отклика на запрос к системе, при этом нет гарантий что ответы всех узлов системы будут совпадать;

- устойчивость к разделению (Partition tolerance) – разделение системы на изолированные узлы не приведет к изменению итогового отклика.

Сама же теорема CAP гласит, что в реализации распределенных систем возможно обеспечить одновременно исполнение не более двух из описанных выше свойств [2].

Стоит отметить, что целостность данных не гарантирует их истинность, а лишь отвергает заведомо невозможные значения. Выделяется несколько типов правил – ограничения целостности, которые накладывают ограничения на возможное состояние БД, среди них:

- ограничения значений – задают логические ограничения на данные, которые могут содержаться в таблицах БД (дата рождения пользователя не может быть больше текущей даты, значение концентрации рабочих растворов может изменяться в пределах от 0% до 100% и т.д.);

- ограничение отношений – задают ограничения на внешние ключи, которые являются указателями на записи в других таблицах;

- ограничение типов – задают тип данных, который может храниться в том или ином столбце таблицы (целочисленный, строкой, байтовый массив и т.д.).

Рассмотрим более подробно ограничение отношений. Главная задача данного ограничения состоит в управлении данными, которые могут быть сохранены в таблице в виде внешнего ключа, хранящего ссылку на запись в другой таблице, а также – отслеживание изменений данных в таблице, на которую ссылается внешний ключ. Например, при удалении записи из таблицы, куда ссылался внешний ключ, нарушается ссылочная целостность связи таблиц, и сам внешний ключ будет ссылаться на удаленную запись, что может вызвать разные последствия от простой недоступности данных с

недостовверными внешними ключами, до критических ошибок в обработке данных записей. В современных СУБД есть базовые механизмы, которые могут обеспечить выполнение данного ограничения, например, каскадная ссылочная целостность, которая может определять, какие действия будут выполняться ядром СУБД при удалении или изменении данных, на которые ссылаются внешние ключи. Существуют следующие каскадные действия:

- BLOCK (NO ACTION) – блокировка изменения\удаления данных в таблице, на которую ссылается внешний ключ, с формированием сообщения об ошибке;

- CASCADE – изменение или удаление строк, ссылающихся на родительскую таблицу, в случае изменения или удаления строк из нее;

- SET NULL – удаление значения внешнего ключа (установка значения NULL) при изменении или удалении строки из родительской таблицы, на которую он ссылается

- SET DEFAULT – установка значения по умолчанию, которое определяется при проектировании БД.

Данный механизм прекрасно справляется с реализацией ссылочной целостности данных, однако, только в пределах одной БД. При реализации распределенной системы, построенной на микросервисной архитектуре, самым подходящим решением является передача функций по контролю целостности под управление отдельному сервису и выделение общих функций в отдельную библиотеку, которая может быть использована на каждом отдельном узле системы [3].

В основе построения системы поддержания целостности данных лежит идея о необходимости централизованного хранения информации обо всех созданных сущностях и о взаимосвязях между ними для идентификации того, в какой БД и в какой таблице находится та или иная строка, на которую ссылается внешний ключ.

Данная задача может быть решена путем использования трех таблиц, структура которых представлена на рисунке 1:

- ExistingEntities – список всех сущностей, которые были созданы в рамках всей системы с указанием ключа и таблицы (полный путь, включающий: БД, схему БД и конкретную таблицу), которая содержит данную сущность;

- HardLockedEntities – список сущностей блокирующих удаление конкретных сущностей, на которые они ссылаются;

- SoftLockedEntities – список сущностей, требующих удаление внешнего ключа в случае если происходит удаление записи, на которую они ссылаются.

В таблицу ExistingEntities должны попадать все новые записи, которые создаются в БД разными узлами распределенной системы. Для этого реализуется собственный метод сохранения данных в БД, который оборачивает в себя стандартный метод EF Core и кроме сохранения самой сущности в БД, добавляет запись в таблицу ExistingEntities тем самым пополняя централизованный список всех сущностей. Дополнительно с

сохранением информации о самой сущности проходит проверка – не помечено ли какое-либо свойство сохраняемой сущности специальным атрибутом, который информирует систему о том, что данное свойство является внешним ключом на другую сущность.

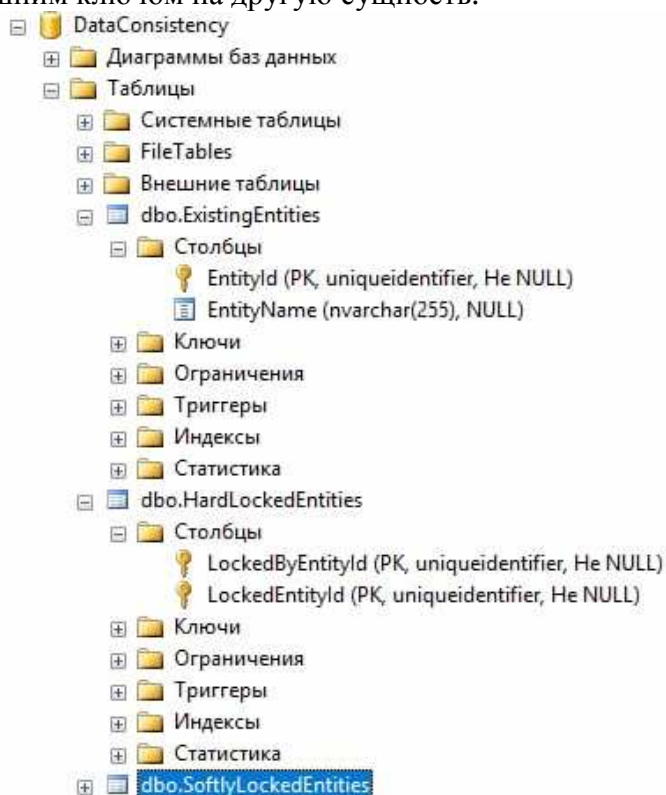


Рис. 1. Структура базы данных

Всего применяется 3 различных атрибута, которые позволяют реализовывать каскадные функции поддержания целостности данных:

- HadrLock – свойство помеченное данным атрибутом запрещает удалять сущность, на которую ссылается свойство, являясь внешним ключом.

- CascadeDelete – при удалении сущности, на которую ссылается свойство помеченное данным атрибутом, будет удалена и сама запись содержащая внешний ключ.

- ReplaceWithNull/SoftLock – при удалении сущности, на которую ссылается свойство помеченное данным атрибутом, само свойство примет значение NULL – внешний ключ удаляется.

Сами по себе атрибуты представляют специальный механизм языка программирования, позволяющий встроить дополнительные метаданные, как классу в целом, так и его методам и свойствам по отдельности (рис. 2).

```
[Column("gTraceabilityStartFillingInspectionCharacteristicId")]
[ExternalEntity(Constraints.HardLock)]
2 references | 0 changes | 0 authors, 0 changes
public Guid? TraceabilityStartFillingInspectionCharacteristicId { get; set; }
```

Рис. 2. Применение атрибута определения каскадной функции

Если в сохраняемой сущности есть свойства, которые помечены атрибутами `HardLock` или `ReplaceWithNull\SoftLock`, то происходит дополнительно запись информации в соответствующие таблицы для последующего отслеживания (рис. 3).

```
1 reference | 0 authors, 0 changes
private async Task SaveChangedEntityToDataConsistencyDb(SavedEntityWrapper entity)
{
    var whoHaveAlreadyBeenHardLocked = _dataConsistencyManager.GetHardLockedEntities(entity.Id);
    var whoHaveAlreadyBeenSoftlyLocked = _dataConsistencyManager.GetSoftLockedEntities(entity.Id);

    // 2.1.
    var entitiesForHardlock = entity.FieldsWithExternalEntity.Where(o => o.Constraint == Constraints.HardLock
                                                                    && o.Value.HasValue)
                                                                    .Select(o => o.Value.Value)
                                                                    .ToList();

    // 2.2.
    var entitiesForSoftlock = entity.FieldsWithExternalEntity
                                .Where(o => (o.Constraint == Constraints.CascadeDelete || o.Constraint == Constraints.ReplaceWithNull)
                                && o.Value.HasValue)
                                .Select(o => o.Value.Value)
                                .ToList();

    // 2.3.
    if (_dataConsistencyManager.CheckHardLock(entity.Id)) // если сохраняемая сущность жестко залочена;
    {
        // выбрать те поля у сохраняемой сущности, которые имеют атрибут каскадного блока;
        var fieldswithCascadeLock = entity.FieldsWithExternalEntity.Where(o => o.Constraint == Constraints.CascadeDelete
                                                                              && o.Value.HasValue);

        // исключить id-шники каскадно заблокированные из мягкого блока и переместить их в жесткий блок
        entitiesForHardlock.AddRange(fieldswithCascadeLock.Select(o => o.Value.Value));
        foreach (var removedId in fieldswithCascadeLock.Select(o => o.Value.Value))
            entitiesForSoftlock.Remove(removedId);
    }

    // 2.4.
    var entitiesWaitingForUnhardlock = whoHaveAlreadyBeenHardLocked.Except(entitiesForHardlock);
    var entitiesWaitingForUnsoftlock = whoHaveAlreadyBeenSoftlyLocked.Except(entitiesForSoftlock);

    var entitiesWaitingForHardlock = entitiesForHardlock.Except(whoHaveAlreadyBeenHardLocked);
    var entitiesWaitingForSoftlock = entitiesForSoftlock.Except(whoHaveAlreadyBeenSoftlyLocked);

    // 2.5.
    // unlock external entities
    foreach (var entityForUnlock in entitiesWaitingForUnhardlock)
        await _dataConsistencyManager.UnlockEntity(entity.Id, entityForUnlock, LockType.HardLock);

    foreach (var entityForUnlock in entitiesWaitingForUnsoftlock)
        await _dataConsistencyManager.UnlockEntity(entity.Id, entityForUnlock, LockType.SoftLock);

    // set lock to external entities
    foreach (var entityForLock in entitiesWaitingForSoftlock)
        await _dataConsistencyManager.SoftLockEntity(entity.Id, entityForLock);

    foreach (var entityForLock in entitiesWaitingForHardlock)
        await _dataConsistencyManager.HardLockEntity(entity.Id, entityForLock);
}
```

Рис. 3. Метод сохранения сущностей в БД и записи информации о них в БД
консистентности

Алгоритм сохранения сущностей со стороны поддержания целостности данных выглядит следующим образом.

- Получение всех сущностей для `HardLock`:
 - выбор всех внешних ключей из сохраняемой сущности;
 - выбор внешних ключей, которые подлежат `HardLock`.
- Получение всех сущностей для `SoftLock`:
 - выбор всех внешних ключей из сохраняемой сущности;
 - выбор внешних ключей, которые подлежат `SoftLock`.
- Если сохраняемая сущность имеет атрибут `HardLock`, то это значит, что она сама не может быть каскадно удалена, а значит должна изменить свой статус с `SoftLock` (проставляется по умолчанию для всех сущностей) на `HardLock`.

- Удаление из списка сущностей, ожидающих Lock, те, что раньше уже были помещены в таблицы HardLockedEntities\SoftLockedEntities.

- Сохранение сущностей в соответствующих таблицах.

Общий вид метода, реализующего описанный выше алгоритм, представлен на рисунке 3.

При удалении сущности каждый узел распределенной системы отправляет ряд сообщений в брокер AMQP сообщений RabbitMQ, который прослушивается сервисом консистентности.

– Сообщение об удалении сущности с идентификатором удаляемой сущности – при получении этого сообщения сервис консистентности проверяет возможность удаления по соответствующим таблицам и, если находит блокировку другой сущностью, оперативно информирует об этом сервис-источник сообщения прерывая процедуру удаления. Если удаляемая сущность имеет атрибут SoftLock, то сервис консистентности также через брокера сообщений уведомляет все остальные узлы распределенной системы об удалении сущности, на которую они ссылались. При получении данного сигнала сервисы удаляют значение внешнего ключа и проставляют значение NULL.

– Сообщение об удалении внешних ключей с определением каскадной операции. Если для внешнего ключа определены операции HardLock\SoftLock, то данный внешний ключ просто удаляется из соответствующей таблицы разрешая выполнение дальнейших операций с сущностью, на которую указывал внешний ключ. Если для внешнего ключа определена операция CascadeDelete, то сервис консистентности отправляет сигнал о необходимости удаления записи в соответствующий узел распределенной системы, которая при удалении сущности также отправляет сообщение об удалении для проверки возможности удаления конкретной записи. Такой подход позволяет проверить всю цепочку ссылок сущностей друг на друга и предотвратить каскадное удаление в случае, если какая-то запись будет жестко заблокирована. Тем самым реализуется транзакционность выполнения операции удаления записей из БД, когда операция либо выполняется полностью (удаляются все сущности по цепочке), либо не выполняется совсем, сохраняя тем самым целостность ссылок.

Применение описанного подхода позволяет сохранить сквозную ссылочную целостность данных в рамках распределенной вычислительной системы, где каждый узел может иметь свою собственную БД под управлением разных СУБД, и не допустить ошибок обработки данных при изменении или удалении записей. А реализация сервиса на основе данного подхода позволяет автоматизировать выполнение каскадных функций, сняв с разработчиков задачи реализации их вручную и минимизирует риск ошибки.

Список литературы

1. Артамонов Ю.С. Разработка распределенных приложений сбора и анализа данных на базе микросервисной архитектуры / Ю.С. Артамонов, С.В. Востокин //

- Известия Самарского научного центра Российской академии наук. – 2016. – Т.18. – № 4-4. – С. 688-693.
2. Зыкин В.С. Ссылочная целостность данных в корпоративных информационных системах // Информатика и ее применения. – 2015. – Т. 9. – № 3. – С. 97-105. – DOI: 10.14357/19922264150310.
 3. Лучинин З.С. Формирование ссылочной целостности в документо-ориентированных базах данных / З.С. Лучинин, И.Г. Сидоркина // Информационные технологии в электротехнике и электроэнергетике: материалы IX Всероссийской научно-технической конференции, Чебоксары, 05–07 июня 2014 года. – Чебоксары: Чувашский государственный университет им. И.Н. Ульянова, 2014. – С. 339-341.

References

1. Artamonov Yu.S. Development of distributed applications for data collection and analysis based on microservice architecture / Yu. S. Artamonov, S.V. Vostokin // News of Samara Scientific Center of the Russian Academy of Sciences. 2016. Vol. 18. No. 4-4. P. 688-693.
2. Zykin V.S. Data Reference Integrity in Corporate Information Systems // Informatics and its Applications. 2015. Vol. 9. No. 3. P. 97-105. DOI: 10.14357/19922264150310.
3. Luchinin Z.S. Document-oriented databases referential integrity formation / Z.S. Luchinin, I.G. Sidorkina // Information technologies in electrical engineering and power engineering: materials of IX All-Russian scientific and technical conference, Cheboksary, 05-07 June 2014. – Cheboksary: Chuvash State University n.a. I.N. Ulyanov, 2014. – P. 339-341.

Павлов Василий Анатольевич – старший преподаватель	Pavlov Vasily Anatolevich – senior lecturer
basilpavlov730@gmail.com	

Received 06.06.2022