



# OASIS DECENTRALIZED DATABASE

## Система управления базами данных

### Техническое описание

Anton Filatov<sup>1</sup>, Dmitry Kochin<sup>2</sup>

August, 2017 - March, 2019

---

<sup>1</sup> <https://www.linkedin.com/in/antonfilatov>



# Содержание

|                                      |    |
|--------------------------------------|----|
| <b>Введение</b>                      | 4  |
| <b>Сущности системы</b>              | 5  |
| Субъекты системы                     | 5  |
| Объекты системы                      | 6  |
| <b>Узел</b>                          | 7  |
| <b>Координатор</b>                   | 7  |
| <b>Пользователь</b>                  | 8  |
| <b>Клиент</b>                        | 8  |
| <b>Пространство таблиц</b>           | 8  |
| <b>Таблица</b>                       | 8  |
| Уровень репликации иパーティционирования | 9  |
| <b>Поле</b>                          | 9  |
| Формат поля                          | 10 |
| <b>Метка времени</b>                 | 10 |
| <b>Запись</b>                        | 10 |
| Счетчик версии                       | 11 |
| Подпись записи                       | 11 |
| <b>Индекс</b>                        | 11 |
| Кластерный индекс                    | 11 |
| Некластерный индекс                  | 12 |
| Локальный индекс                     | 12 |
| <b>Ограничение</b>                   | 13 |
| Статическое ограничение              | 13 |
| Динамическое ограничение             | 13 |
| Умное ограничение                    | 14 |
| <b>Запрос</b>                        | 14 |
| Запрос доступа к данным              | 14 |
| Запрос модификации данных            | 15 |
| Уровень консистентности запроса      | 16 |

---

<sup>2</sup> <https://www.linkedin.com/in/kochin>



|   |    |
|---|----|
| <b>Чек</b>                                  | 18 |
| Чек доступа к данным                        | 18 |
| Чек модификации данных                      | 19 |
| Чек хранения                                | 19 |
| Чек с накопительным итогом                  | 20 |
| Диапазон нумерации чеков                    | 20 |
| Выпуск чеков                                | 20 |
| Погашение чеков                             | 21 |
| <b>Блэйминг</b>                             | 22 |
| <b>Хилинг</b>                               | 22 |
| Гонка конкурентных записей                  | 23 |
| <b>Типы данных</b>                          | 24 |
| <b>Структура объектов</b>                   | 26 |
| Структура пространства таблиц               | 28 |
| Структура таблицы                           | 28 |
| Структура поля                              | 30 |
| Структура подписи                           | 30 |
| Структура записи                            | 30 |
| Структура запроса                           | 32 |
| Структура подтверждения получения данных    | 33 |
| Структура ответа успешного поиска           | 34 |
| Структура ошибки запроса                    | 34 |
| Структура чека доступа и модификации данных | 35 |
| Структура чека хранения данных              | 35 |
| Структура диапазона чеков                   | 36 |
| Структура чека с накопительным итогом       | 36 |
| <b>Протокол</b>                             | 37 |
| Управление пространствами таблиц            | 40 |
| Управление таблицами                        | 41 |
| Управление полями                           | 42 |
| Управление индексами                        | 44 |
| Управление ограничениями                    | 47 |
| Диапазон распределения записей              | 49 |
| Распределение записей                       | 53 |
| Управление записями                         | 54 |
| Условия поиска                              | 56 |
| Формирование запросов                       | 57 |



|                            |           |
|----------------------------|-----------|
| Ответ узла на запрос       | 57        |
| Выпуск чеков               | 59        |
| Гашение чеков              | 60        |
| Получение вознаграждения   | 64        |
| <b>Особенности системы</b> | <b>64</b> |
| Ограничение чтения         | 65        |

## Введение

Oasis.DDB - это публичная, децентрализованная, основанная на публичном блокчейне, распределенная нереляционная система управления базами данных с открытым исходным кодом.

Oasis.DDB является полностью децентрализованной базой данных. Это означает, что в ней отсутствует не только единая точка отказа, но и единая точка контроля. То есть, все узлы Oasis.DDB равноправны, и введение в сеть нового узла Oasis.DDB доступно всем. В результате Oasis.DDB представляет собой облачную базу данных, работающую на предоставленных участниками сообщества серверах. Участники при этом разнородны и не принадлежат единой организации, именно поэтому в Oasis.DDB отсутствует единая точка контроля.

Полная децентрализация налагает на Oasis.DDB требования к устойчивости к злонамеренному поведению отдельных её участников. То есть, участник, предоставляя узел сети Oasis.DDB, не должен быть способен нарушить её работу, даже если он модифицирует программное обеспечение узла для нанесения вреда.

Ниже приведены принципы работы узлов Oasis.DDB. При этом основное внимание уделено описанию взаимодействия узлов. Низкоуровневое хранение данных на отдельных узлах не рассматривается, потому что предполагается, что для этого будет использоваться одна из существующих реализаций баз данных.



Тот факт что Oasis.DDB является нереляционной базой данных, говорит о некоторых особенностях её работы. А в частности невозможность использования принципов ACID (**A**tomicity, **C**onsistency, **I**solation, **D**urability). Вместо них используется семантика BASE (**B**asically **A**vailable, **S**oft state, **E**ventual **c**onsistency), позволяющая выбирать в процессе работы компромисс между согласованностью данных, доступностью и устойчивостью к разделению распределенной системы. Подробнее о решении этих проблем и особенностях работы описано в разделе [особенности системы](#).

## Сущности системы

### Субъекты системы

Основными субъектами системы являются:

- Узел - участник системы, хранящий и передающий данные, предоставленные пользователями за вознаграждение.  
Непосредственный способ хранения данных не имеет значения, главное факт возможности работать в соответствии с протоколом. Узел получает вознаграждение за выполненную работу от пользователя.
- Координатор - участник системы, принимающий запросы клиентов, проверяющий формат входных данных, транслирующий их на узлы, и объединяющий результаты для ответа пользователю. Может быть представлен как узлом так и клиентом. Координатор получает вознаграждение за выполненную работу от узлов. Если в качестве координатора выступает клиент пользователя, то вознаграждение будет возвращено пользователю после расчетов с узлами.
- Пользователь - участник системы, предоставляющий для хранения и запрашивающий для обработки данные. Вознаграждает узлы за выполненную работу.



- Клиент - программный компонент или приложение, работающее от имени пользователя, которое форматирует предоставленные данные (для хранения и обработки) и производит расчеты с узлами.

## Объекты системы

- Пространство таблиц - совокупность описаний таблиц и правил доступа к ним. Создание и управление пространством таблиц производится с помощью смарт-контрактов.
- Таблица - совокупность полей, описывающих формат хранения данных, ограничений, индексов для поиска и записей, содержащих данные пользователей. Управление доступом к таблице производится с помощью смарт-контрактов относительно табличного пространства.
- Поле - описание способа хранения данных, идентификатор доступа к ним. Управление полями производится с помощью смарт-контрактов относительно таблицы в пространстве.
- Запись - совокупность данных, передаваемых пользователем для хранения и обработки. Структура данных должна соответствовать полям в таблице, куда они будут записаны, и отвечать требованиям ограничений таблицы.
- Индекс - подразделяется на первичный (кластерный), служащий для определения узла, на котором должна храниться запись, и некластерный индекс, используемый для поиска данных и работы ограничений. Управление некластерными индексами производится с помощью смарт-контрактов относительно таблицы в пространстве. Первичный индекс может быть назначен только один на таблицу и после создания не может быть изменен.
- Ограничение - какое-либо условие проверки данных. Данные, не удовлетворяющие одному или нескольким ограничениям, не могут быть добавлены в таблицу.
- Запрос - средство управления данными. Подразделяется на запрос доступа и запрос изменения.



## OASIS DECENTRALIZED DATABASE

- Запрос доступа - позволяет осуществлять поиск и получение данных.
- Запрос изменения - позволяет производить добавление, изменение и удаление данных.
- Чек - основное средство расчетов между субъектами системы.  
Существует [несколько видов чеков](#), в зависимости от оплачиваемых ими операций. Чеки в конечном итоге [подлежат погашению](#) через блокчейн.

Подробнее о каждом из субъектов и объектов системы далее. Определения даны для сравнения со стандартными объектами реляционных и нереляционных систем управления баз данных и установления сходств и различий.

## Узел

Основным элементом построения системы считается Узел. На узлах хранятся данные, они выполняют их поиск и проверку на соответствие условиям ограничений, и они также могут выступать в роли [координаторов](#).

## Координатор

Координатор - это роль узла или клиента, в которой он выступает при передаче запросов. Координаторы являются транспортной сущностью системы. Они отвечают за координацию запросов клиентов и ответов узлов и объединение поисковых результатов. Также они выполняют сверку данных и улаживание конфликтов при обнаружении неконсистентности. При этом координаторы получают вознаграждение от узлов за успешно устраненную неконсистентность (подробнее в разделе [хилинг](#)) и успешную передачу данных от узлов клиенту и от клиента узлам. Также координаторы предагрегируют ответы от узлов и объединяют их в общий результат.



## Пользователь

Лицо, от имени которого работает Клиент. Пользователь предоставляет данные и управляет функциями клиента для осуществления запросов к Oasis.DDB, и предоставляет средства для расчетов с узлами. Пользователь идентифицируется своим публичным ключом, а приватный ключ служит для подписи чеков, запросов и данных, передаваемых на узлы.

## Клиент

Средство работы с системой. Может представлять из себя программу для какой-либо операционной системы, библиотеку для использования в языках программирования или любое другое решение, позволяющее общаться с системой Oasis.DDB в соответствии с [протоколом](#). Как и узел, может выступать в роли [координатора](#).

## Пространство таблиц

Пространство таблиц представляет собой логическую группировку таблиц и прав управления. Обычно одно пространство таблиц выделяется на приложение. В нем группируются таблицы, хранящие сущности приложения. Пространство позволяет контролировать выполняемые с таблицами операции для разных ролей пользователей приложения. Создание и управление табличными пространствами производится через блокчейн с использованием смарт-контракта. Вся информация о правах управления структурой базы данных (создание и удаление таблиц, изменения полей) также хранится в пространстве таблиц в блокчейне.

## Таблица



Таблица хранит непосредственно данные пользователей, организованные в записи. Сам механизм хранения записей может отличаться на разных узлах, но он должен обеспечивать возможность выполнения запросов согласно протокола Oasis.DDB. У каждой таблицы должен быть назначен первичный (кластерный) [индекс](#), по которому [данные будут распределяться по узлам сети](#). Права доступа к таблице, так же как и для табличных пространств, хранятся в блокчейне. Но в отличие от прав табличного пространства они разграничивают права на создание, изменение данных. Доступ на получение данных в системе Oasis.DDB имеют все участники, поэтому для [ограничения доступа чтения](#) рекомендуется использовать шифрование, но поиск в этом случае будет ограничен.

## Уровень репликации иパーティцирования

Также у таблицы есть параметры репликации и разбиения (партицирования). Они указывают, какое количество реплик будет существовать для данной таблицы, и на сколько частей она будет разделена соответственно. Максимальное количество узлов, обслуживающих данную таблицу, может быть рассчитано перемножением этих параметров. А минимальное количество узлов - это наибольшее значение из уровня репликации и уровня партицирования, чтобы каждый узел хранил разные диапазоны разных реплик. Например, для обслуживания таблицы с уровнем репликации 9 и уровнем партицирования 100 понадобится не менее 100 и не более 900 узлов.

## Поле

Поле - это определение хранимых данных. Из полей и [индексов](#) складывается структура таблицы. Каждое поле имеет уникальное имя, позволяющее идентифицировать его внутри таблицы, и формат, описывающий, какого типа данные могут храниться в этом поле. Информация



о полях, как и прочая информация о структуре, хранится в блокчейне и управляетя с помощью смарт-контрактов.

## Формат поля

Формат поля включает в себя тип поля и размер поля. Тип поля характеризует информацию, хранимую в поле, и возможные способы обработки. От типа зависит, какие операции и фильтры будут применимы в запросах доступа. Размер поля, относительно типа, накладывает базовые ограничения на размер хранимых данных. Запись не может содержать информации в поле больше, чем установлено его размером.

## Метка времени

Все временные метки измеряются количеством секунд, прошедших с 1 января 1970 года в UTC. Поскольку в децентрализованной системе синхронизация времени нетривиальна, во временных метках в Oasis.DDB указывается время, синхронизированное по блокчейну (метке времени последнего блока). Особенностью таких меток является невысокая разрешающая способность (для Ethereum - около 15 секунд). Но для целей Oasis.DDB достаточно такой точности.

## Запись

Непосредственно данные, подлежащие хранению в Oasis.DDB. Данные записи разделены по полям, соответствуют их формату и относятся к конкретной таблице в табличном пространстве. Запись формируется клиентом из информации, предоставленной и подписанной пользователем. Запись не может содержать поля, не принадлежащие указанной в записи таблице, и должна ссылаться на существующие поля, таблицу и табличное пространство.



Каждая запись имеет [метку времени](#), синхронизированную с блокчейном (время последнего выпущенного блока).

## Счетчик версии

Каждая запись имеет последовательный счетчик версии, недоступный для изменения напрямую. Он контролирует конфликты модификации записи и последовательность изменений. Каждый запрос на добавление записи устанавливает счетчик записи равным единице (если запись добавляется на место удаленной, то счетчик версии продолжается с версии, следующей за версией удаленной записи), а последующие изменения инкрементируют его значение на единицу. Таким образом, данный счетчик позволяет легко определять и разрешать конфликты конкурентных изменений записи в процессе работы.

## Подпись записи

Каждая запись обязательно имеет заголовок, подписанный пользователем, создавшим эту запись. В подпись заголовка входит хэш всех полей записи, [метка времени](#) и [счетчик версии](#).

## Индекс

По сути, индексы - это реестры записей, позволяющие находить их (записи) в системе. Они могут быть построены по одному или нескольким полям и бывают трех видов: кластерный, некластерный (поисковой) и локальный индекс.

## Кластерный индекс

Кластерный (первичный) индекс является глобальным уникальным идентификатором записи. Он используется для определения узлов, на которых будет сохранена запись, и с которых будет производиться чтение. Для каждой таблицы может существовать только один такой индекс. В Oasis.DDB



кластерный индекс реализуется с помощью хэшей, при этом [распределяется](#) сразу весь диапазон возможных значений хэш-функции по узлам, способным его обслуживать. Кластерный индекс может быть композитным и может включать несколько полей [примитивных и составных типов](#), исключая типы коллекций.

## **Некластерный индекс**

Некластерные индексы существуют для ускорения доступа к данным и организации некоторых видов ограничений. Некластерные индексы, как и первичные, являются глобальными и распределяются по узлам независимо от записей, по которым они построены. У таких индексов, так же как и у таблиц, есть параметр уровня репликации иパーティционирования, указывающий на количество узлов, на которые будет реплицирован данный индекс, и на сколько частей будет разделен. Ускорение доступа происходит за счет уменьшения количества запросов к узлам в большинстве случаев. При наличии индексов, условия запроса на поиск сначала проверяются по ним на узлах, хранящих индекс, затем запрос уходит только на те узлы, которые были найдены в индексе. Также нужно заметить, что при наличии таких индексов может уменьшиться скорость добавления и изменения данных, т.к. необходимо будет обслуживать (обновлять) все индексы, в которых участвуют изменяемые поля.

## **Локальный индекс**

Локальные индексы используются на узлах, хранящих данные. Локальный индекс является видом некластерного индекса, но распределяется локально (вместе с данными). В итоге на каждом узле хранится свой локальный индекс, по которому производится поиск записей в пределах узла. Локальные индексы могут быть удобны для сортировки результатов запроса поиска или использованы как глобальные некластерные индексы без дополнительного распределения на мелких таблицах с цельным кластерным индексом (весь



диапазон значений кластерного индекса, а соответственно и все записи, хранятся в пределах одного узла).

## Ограничение

Под ограничением в Oasis.DDB понимаются некоторые условия, которым должно удовлетворять значение поля, чтобы запись была принята узлами.

Ограничения могут устанавливаться на конкретное поле либо всю таблицу. К записи может быть применено несколько различных ограничений, установленных для полей и таблицы, в итоге результаты проверок будут конъюнктивно объединены (логическое умножение) в один итоговый.

Ограничения проверяются узлами и контролируются координатором. Они подразделяются на статические, динамические и "умные".

### Статическое ограничение

Статические ограничения, или ограничения поля - это условия, которым должно отвечать значение в записи для конкретного поля. В статических ограничениях можно использовать только константы и ссылаться лишь на новое значение поля, для которого они установлены, присутствующее в передаваемой записи. При этом записи, не удовлетворяющие условиям статических ограничений, отвергаются узлами.

### Динамическое ограничение

Динамические ограничения - это условия, которым должна отвечать запись. В условиях динамических ограничений допускаются ссылки на другие поля как передаваемой, так и имеющейся (в случае изменения данных) записи, и встроенные функции, работающие со всеми записями таблицы, такие как проверка уникальности значения. Динамические ограничения позволяют добавить некоторые удобства контроля консистентности, присущие реляционным системам управления базами данных, но отсутствующие в



большинстве нереляционных. Динамические ограничения проверяются как на узлах, так и координатором, если есть возможность использовать индексы для проверки условий.

## Умное ограничение

Умные ограничения - это проверка записи, основанная на выполнении функций в смарт-контракте. Как и в динамических ограничениях, в качестве аргументов функций смарт-контракта допускаются ссылки на другие поля как передаваемой, так и имеющейся (в случае изменения данных) записи, и встроенные функции. Вызов допускается только для константных (не требующих транзакций) функций чтения, возвращающих лишь один булевый результат.

## Запрос

Запрос - это пакет информации, предоставленной и подписанной пользователем, содержащий структурированные соответствующим образом запись (непосредственно информацию) и, при необходимости, чеки для оплаты. Запрос передается клиентом с помощью координатора на узлы для получения, добавления и изменения данных. В запросах указываются операции только по работе с данными, изменение структуры базы данных производится непосредственно в блокчейне. Каждый запрос, как и запись, подписывается пользователем. Каждый запрос выполняется согласно указанному в нем [уровню консистентности](#), который выбирается логикой работы клиента или непосредственно пользователем.

## Запрос доступа к данным

Запросы доступа к данным используются для поиска и получения данных, хранящихся на узлах. Запрос доступа к данным содержит условия поиска и [чеки оплаты поиска данных](#). При выполнении этого запроса координатор разбирает условия поиска, проверяя, можно ли использовать индексы для



уменьшения количества опрашиваемых узлов. Затем запрос транслируется на узлы, на которых хранятся данные, попавшие в условия выборки. Узлы после выполнения необходимых операций с данными (поиск, фильтрация, сортировка) возвращают их клиенту через координатор в обмен на [чеки оплаты чтения данных](#).

### **Запрос версии данных**

Это **легковесный** запрос, позволяющий получить текущую версию записи, которую необходимо добавить, изменить или удалить. Содержит набор значений первичных ключей записей и чеки на оплату выполнения запроса. Ответом на данный запрос является версия записи для каждого первичного ключа в запросе.

### **Запрос модификации данных**

Данный вид запроса производит добавление или изменение уже имеющихся данных в базе Oasis.DDB. Каждый запрос изменения данных имеет [запись](#), включающую новую [версию](#) записи (если запись создается впервые, то версия записи будет равна единице, а если запись вставляется на место удаленной, то версия записи будет следующей за версией удаленной записи), метку времени и, при необходимости, чеки для оплаты хранения записи (для новой записи чеки оплаты хранения обязательны). Запросы модификации данных координатор транслирует на целевые узлы (отвечающие за диапазон кластерного индекса, в котором находится запись) и получает от них подтверждение получения записи (необходимое для разрешение проблемы [гонки конкурентных записей](#)) и вознаграждение.

### **Запрос добавления данных**

В запросе добавления данных содержится запись, которую необходимо добавить. Запрос на добавление данных будет принят, только если запись с указанными значениями [первичного ключа](#) и [версии](#) отсутствует либо была удалена, в ином случае запрос добавления данных будет отвергнут узлом.



## Запрос изменения данных

В запросе изменения данных содержится запись, состоящая из полей с новыми значениями (которые заменят хранящиеся), [первичного ключа](#), [метки времени](#) и счетчика версии целевой записи, увеличенного на единицу. Запись в запросе подписывается, используя хэши значений всех полей (возвращаются в запросе доступа вместе с текущей версией записи), но содержит только измененные данные.

## Запрос удаления данных

В запросе удаления данных содержится запись со счетчиком версии, следующей за версией записи, подлежащей удалению. Данные в такой записи не передаются, а подпись формируется только по полю версии и [метке времени](#). Удаленная запись не возвращается в ответах на запросы доступа к данным.

## Уровень консистентности запроса

Уровни консистентности позволяют управлять параметрами CAP (**C**onsistency, **A**vailability, **P**artition tolerance) динамически в процессе работы системы. Каждый клиент сам выбирает, какому из параметров отдать предпочтение в зависимости от выполняемой функции. Существует несколько уровней консистентности для запросов доступа и модификации данных:

- **ALL** - запрос должны принять все относящиеся к нему узлы. Самый медленный и самый консistentный запрос. При возвращении результата запроса с данным уровнем консистентности можно гарантировать, что запись будет прочитана при любом уровне консистентности запроса доступа.
- **QUORUM** - запрос должен принять кворум узлов, относящийся к нему. Кворумом является минимальное количество узлов, необходимое для достижения большинства (т.е. больше половины узлов для каждой записи). Например, для таблицы с уровнем партицирования 2 и



## OASIS DECENTRALIZED DATABASE

уровнем репликации 3 в худшем случае (когда заранее не известен первичный ключ искомой записи) будут опрошены 4 узла (по 2 узла на диапазон). Если же первичный ключ известен, то опрошены будут только узлы, обслуживающие диапазон, в который входит этот ключ, то есть будет опрошено лишь по одному узлу на большую половину реплик - в данном случае только 2 узла. Это гарантирует, что информация хранится на большинстве узлов. При обнаружении неконсистентности в данных до возвращения ответа будет произведено автоматическое устранение неконсистентности ([хилинг](#)). Это гарантирует, что возвращенная запись распространится на остальные узлы и не будет затерта другой версией. Таким образом, это второй по степени консистентности уровень запроса. При возвращении результата запроса с данным уровнем консистентности можно гарантировать, что запись будет прочитана при уровне консистентности [запроса доступа](#) выше или равном уровню quorum. Жертвуя доступностью, пользователь повышает скорость ответа и устойчивость к разделению.

- **ONE** - запрос должен принять хотя бы один узел. Если это запрос модификации данных, то его должен принять хотя бы один узел, обслуживающий диапазон первичного ключа, и по одному узлу на каждый индекс. Если это запрос доступа к данным, то его должны принять по одному узлу на каждую часть диапазона первичных ключей либо узел индекса и узел данных, найденный в индексе. Другими словами, ответ доступа к данным возвращается после опроса только одной реплики данных. Является самым ненадежным, но быстрым и дешевым уровнем консистентности. Может быть использован для доступа и модификации некритичной информации либо при очень редких и не срочных изменениях.
- **10P ... 90P** - указывает процентное соотношение узлов, которые должны ответить на запрос. Позволяет использовать тонкую настройку запросов. Общее число узлов вычисляется из уровня партицирования и



## OASIS DECENTRALIZED DATABASE

уровня репликации таблицы, но для запроса модификации и доступа к данным они различаются. В таблице с уровнем репликации 21 и уровнемパーティционирования 100 общее количество узлов, на которых хранятся записи таблицы, составляет максимум 2100. При выполнении запроса модификации данных с уровнем доступа 10Р координатор вернет ответ, если запрос принял хотя бы 2 узла (10% всех реплик отвечающих за диапазон, которому принадлежит первичный ключ модифицируемой записи). При запросе доступа к данным с теми же параметрами запрос распределится на все части двух реплик (10% от всех реплик), т.е. на 200 узлов.

Следует заметить, что координатор после возвращения ответа на запрос модификации данных клиенту может продолжить распространять запись на все относящиеся к ней узлы независимо от выбранного уровня консистентности для получения вознаграждения от этих узлов. Но гарантировать, что запись попала на все узлы после получения успешного ответа клиентом, может только уровень консистентности all.

## Чек

Чеки являются основным средством расчетов с узлами в Oasis.DDB. Чеки позволяют проводить большую часть взаиморасчетов вне блокчейна, то есть, обеспечивают высокую скорость и низкую стоимость транзакций. Пользователь выписывает чеки узлам в награду за совершающую ими работу. Все чеки имеют [номер из заданного диапазона](#), идентификатор узла, которому он был выписан, и подписываются приватным ключом пользователя. Подробнее о структуре чеков описано в разделе [структура чека](#). Существует несколько видов чеков:

### Чек доступа к данным



## OASIS DECENTRALIZED DATABASE

Подтверждает факт совершения оффчайн оплаты запросов поиска и получения данных. Передаются узлам в обмен на результат выполнения запроса.

### **Чек поиска**

Чеком поиска оплачивается поиск записей на узлах (пока это фиксированная сумма, возможно будет вычисляться в зависимости от сложности условий запроса).

### **Чек чтения**

Чеком чтения оплачиваются полученные от узлов данные (сумма зависит от объема переданных данных).

## **Чек модификации данных**

Подтверждает факт совершения оффчайн оплаты запросов создания и изменения данных. Прикрепляются к сформированным запросам, передаваемым на узлы.

### **Чек создания и изменения записи**

Чеками создания и изменения записи оплачивается создание новых или обновление существующих записей на узлах (сумма зависит от объема записываемых данных).

### **Чек удаления**

Чеками удаления оплачиваются удаления записей на узлах.

## **Чек хранения**

Чеками хранения оплачивается факт хранения данных. Чек привязан к заголовку записи, включающему первичный ключ записи, версию записи и время создания записи, синхронизированному по последнему блоку в блокчейне. Таким образом, данный вид чека не может быть погашен без наличия версии записи (или как минимум её заголовка), на обеспечение хранения которой он выписан. Чек хранения выписывается на весь период



хранения записи, но оплачивается либо по истечении срока хранения записи, либо при обновлении записи новой версией.

## Чек с накопительным итогом

Является итоговым чеком. В него объединяются все виды чеков, выписываемых пользователем с помощью [диапазонов нумерации](#). Именно этот вид чека подлежит прямому обналичиванию через смарт-контракт.

## Диапазон нумерации чеков

Каждый чек должен быть явно идентифицирован при включении в [чек с накопительным итогом](#) во избежание повторного включения в итоговую сумму. Для этого используются диапазоны нумерации чеков. Каждый диапазон создается для каждого клиента на узле и имеет уникальный идентификатор и версию. Диапазон хранит массив последовательностей погашенных чеков. При [выписывании чека](#) в него включается идентификатор узла, идентификатор диапазона и номер из этого диапазона (см. [структура чека](#)).

## Выпуск чеков

В соответствии со [структурой чека](#), чеки выпускаются с номерами из [диапазона](#), соответствующего клиенту пользователя (приложению работающему с Oasis.DDB). Диапазон выпускается на конкретный экземпляр клиента; если пользователь работает из нескольких клиентов, то на каждый клиент будет создан свой диапазон. Созданный диапазон можно считать подобием долговременной сессии. Чеки создаются и подписываются автоматически самим клиентом. Чтобы получить диапазон, клиент генерирует UUID, который становится уникальным идентификатором диапазона, либо использует имеющийся. Таким образом, клиентам не требуется синхронизация для работы, каждый выписывает чеки внутри своего диапазона, и они включаются узлами в чек с накопительным итогом отдельно. Также нет необходимости знать состояние узла, что позволяет начать работу немедленно, без дополнительных затрат на синхронизацию.



## Погашение чеков

Чеки создания, модификации и хранения данных не могут быть обналичены напрямую, для этого они должны быть объединены в текущий чек с накопительным итогом. Происходит это по инициативе узла. Обработка чеков происходит на клиенте в фоне с низким приоритетом.

### Погашение чеков доступа и модификации данных

Узел передает на клиент чек с накопительным итогом и чеки данных, которые он хочет погасить, последние версии диапазонов, которым принадлежат чеки данных, и недостающие для расчета хэша корня хэши ветвей и листьев дерева Меркла, корень которого хранится в чеке с накопительным итогом. Клиент проверяет подписи чеков и диапазонов и вычисляет корень дерева Меркла из переданных хэшей, затем проверяет, чтобы он совпадал с корнем, переданным в чеке с накопительным итогом. После валидации входных данных клиент отмечает чеки в диапазонах как погашенные, увеличивая итог накопительного чека на сумму погашенных чеков (подписывая новые версии диапазонов), и перестраивает дерево Меркла с учетом новых значений. После расчетов он заменяет значение корня дерева Меркла на новое и подписывает получившийся чек с накопительным итогом. Затем клиент возвращает новые диапазоны и чек с накопительным итогом узлу.

### Погашение чеков хранения данных

Погашение чеков хранения происходит так же, как и гашение чеков модификации и доступа, но дополнительно с чеком присыпается заголовок записи, хранение которой он обеспечивает. При погашении чека хранения в случае истечения срока хранения записи для проверки валидности погашения может быть выполнен [запрос последней версии записи](#). В ответ может быть возвращен новый чек хранения, если срок хранения требуется продлить. В случае, когда погашение чека хранения происходит при появлении новой версии записи, к гасимому чеку прикладывается чек хранения и заголовок



новой версии записи, подписанные пользователем, создавшим новую запись. Таким образом, погашению подлежит не срок хранения, на который выписан чек, а время фактического хранения, от времени создания текущего чека до времени создания чека хранения новой версии записи.

Подробнее гашение чеков описано в разделе [протокол](#).

## Блейминг

Блейминг - это механизм обвинения узлов в некорректной работе и последующее установление виновности и привлечение к ответственности. Все участники системы могут использовать этот механизм, чтобы указать на недобросовестно исполняющего свои обязанности участника системы. Для этого в блокчейне реализован смарт-контракт с методами, соответствующими каждой возникшей ситуации, принимающими на вход идентификатор другого участника системы и другие параметры. При накоплении участником определенного количества обвинений (блэймов) смарт-контрактом производится привлечение участника к ответственности путем предусмотренных контрактом штрафов.

## Хилинг

Хилинг - это механизм устранения неконсистентности записей в системе. В процессе работы распределенной системы могут возникать ситуации, когда доступ к некоторым участникам сети ограничен или отсутствует (так называемое разделение). При доступе к данным это не критично, т.к. непосредственного изменения информации не происходит и, если запись была распространена на все узлы, которыми она обслуживается, то достаточно хотя бы одного узла, на котором она хранится, чтобы её получить. Иначе дело обстоит при модификации данных, при недоступности некоторых узлов запись



на них не попадает. Для устранения такой неконсистентности используется хилинг. Сама процедура восстановления происходит при запросах доступа к данным с уровнем консистентности выше того, в котором находится система в данный момент. Если координатор, запрашивая записи, находит несоответствие (например, некоторые реплики возвращают более старые записи либо записи на них отсутствуют), он может начать процедуру распространения данных на неконсистентные узлы. Для устранения он запрашивает запись с одной из консистентных реплик (всю запись либо те части, которые изменились с предыдущей версии) и отправляет на неконсистентные реплики за вознаграждение. Данная процедура происходит в фоне, после ответа клиенту и по инициативе координатора, при условии, что можно явно установить требуемый уровень консистентности. Например, при доступе с уровнем консистентности запроса one и консистентности условий поиска all при опросе всех реплик для нахождения самой быстрой может быть обнаружен неконсистентный узел, но т.к. это не влияет на возможность ответа, сначала будет возвращен ответ клиенту с самой новой версией записи, а затем будет предпринята попытка обновления записи на устаревшем узле. Но при невозможности установить текущее состояние (консистентных узлов недостаточно для требуемого уровня консистентности ответа, например, quorum), попытка пересылки новейшей версии записи будет предпринята перед ответом клиенту до достижения требуемой в запросе консистентности. Данная механика может замедлить ответы на запросы с наиболее консистентными уровнями, но позволяет автоматически исправлять ошибки рассинхронизации данных и гарантировать требуемый уровень консистентности в конечном счете, даже при периодических проблемах доступа к узлам и разделении системы.

## Гонка конкурентных записей

В данном механизме может возникнуть ситуация, что будут существовать две различные версии записи с одним номером. Например, два клиента изменили одну и ту же запись с уровнем репликации 9 с версии 1 до версии 2



различным образом, с самым низким уровнем консистентности - one (т.е. ответ клиенты получили при попадании записи на как минимум одну из реплик).

После возвращения ответов клиентам, координаторы, с которыми они работали, могут продолжить распространять записи, которые были переданы через них на остальные узлы. Это распространение будет происходить одновременно и конкурентно, при этом при попытке записать одну версию записи на узел, который уже содержит другую версию записи, возникнет ошибка. Координатор, получивший ошибки от большинства реплик, останавливает распространение своей версии записи, а другой координатор, распространивший свою версию записи на большинство узлов, при возникновении ошибки при распространении передает узлу [подтверждения](#) от большинства реплик, что они получили эту запись. Узел при получении подтверждений от большинства реплик проверяет их валидность и при успешной проверке принимает другую версию записи. Таким образом, координатор, успевший распространить запись на большинство узлов, в итоге получает вознаграждение от всех узлов. Такая же ситуация может складываться и при хилинге, но при этом сначала вычисляется версия записи, хранящаяся на большинстве реплик, и гонки конкурентных записей не возникает.

## Типы данных

Все числовые типы в Oasis.DDB - знаковые, кодированные дополнительным кодом (two's complement coded), числовые типы с плавающей точкой отвечают стандарту IEEE 754, а форматы временных типов - стандарту ISO 8601.

Примитивные типы:

- Boolean - логический тип. Может принимать 2 возможных значения: true или false.



- Integer - числовой тип, размером 32 бита. От -2147483648, до 2147483647.
- Long - числовой тип, размером 64 бита. От -9223372036854775808 до 9223372036854775807.
- Float - числовой тип с плавающей точкой, одинарной точности размером 32 бита.
- Double - числовой тип с плавающей точкой, двойной точности размером 64 бита.
- Decimal - числовой тип произвольной точности. Размер зависит от выбранного масштаба и точности.
- String - строковый тип. Состоит из указанной кодировки символов и массива октетов закодированной текстовой строки.
- Binary - бинарный тип. Массив октетов.
- Time - временной тип, включающий дату. Размер зависит от выбранной точности.
- Duration - временной тип отражающий промежуток времени. Размер зависит от выбранной точности.

Коллекции:

- List - список элементов, с сохранением последовательности.
- Bag - множество элементов. В зависимости от настроек может быть множеством уникальных элементов или мульти множеством.
- Map - ассоциативный массив элементов с примитивным типом в качестве ключа. В будущем будет расширен до применения составных типов в качестве ключа.

Составные:

- Structure - объект с фиксированным набором элементов. Доступ к элементам может производиться как в ассоциативном массиве по номеру или имени, если массив именованный.



## Структура объектов

В данном разделе приводится общее описание структуры объектов без уточнения типов данных, использованных для каждого элемента. Это необходимо для упрощения описания операций над объектами, используя ссылки к их элементам.

Описания приведены в следующем формате:

$A [B, ?C, D_{0..n}]$ , где

$A$  - родительский элемент для структурных элементов  $B$ ,  $C$  и  $D$ . То есть  $A$  содержит  $B$ ,  $C$  и  $D$ .

$B$ - обязательный элемент родительского элемента  $A$ .

$C$ - необязательный элемент родительского элемента  $A$ .

$D_{0..n}$ - множество элементов родительского элемента  $A$ , от 0 до  $n$ .

Также возможны ссылки на структурные элементы:

$B^A$  - ссылка на структурный элемент  $B$  родительского элемента  $A$ . То есть  $B$  содержится в  $A$ .

$C^{A.X.Z}$  - ссылка на структурный элемент  $C$  родительского элемента  $A$ . При этом элемент  $X$  является родительским для  $A$ , а  $Z$  является родительским для  $X$ . То есть  $Z$  содержит  $X$ ,  $X$  содержит  $A$ , а  $A$  содержит  $C$ .

$D_3^A$ - ссылка на третий элемент множества  $D$  родительского элемента  $A$ .

Если элемент является функцией, вычисляемой из других элементов, к нему применяется следующий формат:

$A [S(B^A), B, C, \#E(B^A, C^A)]$ , где

$A$ - родительский элемент.

$B$  и  $C$ - структурные элементы родительского элемента  $A$ .



$S(B^A)$ - хранимый структурный элемент родительского элемента  $A$ , вычисленный из структурного элемента  $B$  родительского элемента  $A$ . То есть  $S(B^A)$  содержится в  $A$ , являясь при этом результатом функции от  $B^A$ .

$\#E(B^A, C^A)$ - структурный элемент родительского элемента  $A$ , не хранящийся в родительском элементе, но вычисляемый из структурных элементов  $B$  и  $C$  родительского элемента  $A$ .

Если структурный состав элемента не может быть определен или зависит от факторов, которые не могут быть описаны в данном формате, то он описывается следующим образом:

$X [?]$ , где

$X$  - родительский элемент, структурные элементы которого не могут быть определены.

Если структурный состав совпадает с другим элементом либо не важен, он может быть опущен; в этом случае опущенные элементы заменяются многоточием:

$X [Y, \dots]$ , где

$X$  - родительский элемент, не все структурные элементы которого важны.

$Y$  - структурный элемент родительского элемента  $X$ , который важен в текущем контексте.

Если родительский элемент может иметь только один структурный элемент из набора в зависимости от ситуации, это может быть записано следующим образом:

$X [Y || Z, S(Y^x || Z^x)]$ , где

$X$  - родительский элемент.

$Y$  и  $Z$  - структурные элементы, один из которых (но не оба) должны присутствовать в родительском элементе  $X$ .



$S$  - структурный элемент родительского элемента  $X$ , который вычисляется из одного элемента  $Y$  или  $Z$  в зависимости от их наличия.

## Структура пространства таблиц

Пространство таблиц включает в себя имя, ограничения доступа и набор таблиц, принадлежащих этому пространству.

$Tbs [Tsn, Rst, Tbl_{0..n}]$ , где

$Tsn$ - имя табличного пространства.

$Rst$ - ограничения доступа для всех элементов в рамках этого пространства.

$Tbl_{0..n}$ - набор таблиц в этом пространстве.

## Структура таблицы

Таблица состоит из имени, набора полей, кластерного индекса и наборов внутренних индексов, внешних индексов, динамических ограничений и умных ограничений. В каждом из индексов присутствует набор полей. В кластерном и [внешнем индексе](#) также присутствуют параметры репликации, данных и ключей соответственно. У внешнего индекса есть уникальное имя, позволяющее использовать его в описании ограничений и запросов доступа к данным.

## Структура параметров репликации и партицирования

Параметры репликации и партицирования состоят из значения уровня репликации и значения уровня партицирования. Они вынесены в отдельную структуру, т.к. всегда указываются парой.

$Rpc [Rpl, Prl, Rpk_{0..r,0..p}]$ , где

$Rpl$ - уровень репликации (количество реплик).

$Prl$ - уровень партицирования (количество частей на реплику).

$Rpk_{0..r,0..p}$ - пространство реплик диапазонов распределения.



## Структура таблицы

$Tbl [Tbn, Fld_{0..n}, Cdx, Idx_{0..m}, Edx_{0..l}, Trs_{0..k}],$  и

$Tbl [Cdx[Rpc, Fln_{0..a}^{Fld.Tbl}], Idx[Idn, Fln_{0..b}^{Fld.Tbl}]_{0..m}, Edx[Edn, Rpc, Fln_{0..c}^{Fld.Tbl}]_{0..l}, \dots],$  и

$Tbl [Trs[Trn, Trt, Trc]_{0..k}],$  где

$Tbn$ - имя таблицы.

$Fld_{0..n}$ - набор полей таблицы.

$Cdx$ - [кластерный индекс](#), использующийся для распределения записей по узлам.

$Rpc^{Cdx}$ - параметры репликации иパーティционирования таблицы.

$Fln_{0..a}^{Cdx}$ - имена полей, участвующих в индексации первичного ключа.

$Idx_{0..m}$ - набор [внутренних индексов](#).

$Idn^{Idx}$ - имя внутреннего индекса.

$Fln_{0..b}^{Idx}$ - имена полей таблицы, участвующих во внутреннем индексе.

$Edx_{0..l}$ - набор [внешних не кластерных индексов](#).

$Edn^{Edx}$ - имя внешнего индекса.

$Rpc^{Edx}$ - параметры репликации внешнего индекса.

$Fln_{0..c}^{Edx}$ - имена полей таблицы, участвующих во внешнем индексе.

$Trs_{0..k}$ - набор [динамических](#) и ["умных"](#) ограничений для текущей таблицы.

$Trn$ - имя ограничения, уникальное для таблицы.

$Trt$ - тип ограничения (динамическое или умное) указывающее формат строки условия ограничения.

$Trc$ - условия ограничения, соответствующие указанному формату.



## Структура поля

Поле включает в себя имя поля, тип хранимых данных, параметры, уточняющие характеристики указанного типа данных (например, размер и точность) и набор [статических ограничений](#).

*Fld [Fln, T, ?Tp, ?Frc]*, где

*Fln*- имя поля в таблице.

*T*- [тип данных](#).

*Tp*- параметры указанного типа данных.

*Frc*- условия [статического ограничения](#) поля, соответствующие формату поля.

## Структура подписи

Подпись состоит из идентификатора восстановления ключа (public key recovery ID) и двух частей самой подписи, сформированной алгоритмом ECDSA.

*Sig [Sv, Sr, Ss]*, где

*Sv*- заголовок подписи, идентификатор восстановления ключа.

*Sr* и *Ss*- части цифровой подписи.

## Структура записи

Каждая запись состоит из подписи хэша заголовка записи, самого заголовка, списка измененных или созданных полей со значениями и чеков оплаты создания и хранения записи. В свою очередь хэш записи строится из хешей от пар имен и значений всех полей (не только измененных, а всех полей, которые будут содержаться в записи после распространения на узлы) и хешей чеков оплаты этой записи. Заголовок содержит имя табличного пространства, которому принадлежит изменяемая таблица, имя таблицы, значения полей первичного ключа в порядке, указанном в кластерном



индексе, версию записи (следующей за версией той записи, которую предполагается изменить), отпечаток времени, синхронизированного со временем последнего блока в блокчейне и хэш самой записи. Поля в записи представляют собой пары имен и значений.

### **Структура заголовка**

$Hdr$  [ $\#Hash(\dots), Tsn, Tbn, Pk[?], V, Et, Tim, Hen = Hash^{Ent}$ ], и

$Hdr$  [ $\#Hash(Tsn^{Hdr}, Tbn^{Hdr}, Pk^{Hdr}, V^{Hdr}, Et^{Hdr}, Tim^{Hdr}, Hen^{Hdr}), \dots$ ], где

$Tsn$ - имя табличного пространства, которому принадлежит таблица.

$Tbn$ - имя таблицы в табличном пространстве.

$Pk$ - набор значений полей первичного ключа в порядке, указанном в индексе первичного ключа в таблице.

$V$ - версия записи.

$Et$ - тип записи: новая запись, модификация старой, удаленная запись.

$Tim$ - временной отпечаток создания записи.

$Hen$ - хэш записи, которой принадлежит заголовок.

$Hash$ - хэш заголовка записи, построенный по именам табличного пространства, таблицы, значениям первичного ключа, версии записи, временного отпечатка и хэша записи, которой принадлежит заголовок.

### **Структура поля**

$Fld$  [ $\#Hash(Fln^{Fld}, Hash(Flv^{Fld}))$ ,  $Fln, Flv$ ], где

$Fln$ - имя поля.

$Flv$ - значение поля.

$Hash$ - хэш от имени и хэша значения поля.



## Структура записи

$Ent$  [ $\#Hash(Hash^{Fld.Ent})$ ,  $Sig(Hash^{Hdr.Ent})$ ,  $Hdr$ ,  $Fld_{0..n}$ ,  $Chq_{0..m}$ ], где

$Sig$ - [подпись](#), построенная по хэшу заголовка.

$Hdr$ - заголовок записи.

$Fld_{0..n}$ - поля записи.

$Chq_{0..m}$ - чеки оплаты хранения и модификации записи.

$Hash$ - хэш записи, вычисляемый от хэшей всех полей.

## Структура запроса

Каждый запрос в общем случае включает в себя подпись, тип запроса и уровень консистентности. Запросы модификации включают в себя запись, хэш которой включается в формирование подписи. Запросы поиска включают в себя условия поиска, хэш которых также включается в подпись запроса. Условия поиска включают в себя тип поискового запроса, строку запроса, чеки оплаты поиска и подпись. Хэш-функция условий поиска строится по типу запроса, строке запроса и хэшам чеков оплаты запроса. Подписывается только хэш условий поиска.

## Структура запроса модификации данных

$Rm$  [?  $Sig(Cl^{Rm}, Hash^{Hdr.Ent.Rm})$ ,  $Cl$ ,  $Ent$ ], где

$Sig$ - [подпись](#), построенная по хэшу заголовка записи, уровню консистентности и типу запроса.

$Cl$ - [уровень консистентности](#) запроса.

$Ent$ - запись, которую необходимо добавить или изменить.

## Структура условий поиска

$Qry$  [ $\#Hash(Tsn^{Qry}, Tbn^{Qry}, Qt^{Qry}, S^{Qry})$ ,  $Sig(Hash^{Qry})$ ,  $Tsn$ ,  $Tbn$ ,  $Qt$ ,  $S$ ,  $Chq$ ], где



$Tsn$ - имя табличного пространства, которому принадлежит таблица.

$Tbn$ - имя таблицы в табличном пространстве.

$Qt$ - тип условий поиска, указывающий формат, использованный в строке поиска.

$S$ - строка поиска в соответствии с форматом типа условий.

$Chq$ - чек оплаты поиска узлам.

$Hash$ - хэш поискового запроса, вычисляемый от его типа, строки запроса, имени табличного пространства и таблицы в этом пространстве.

### **Структура запроса доступа к данным**

$Rs$  [ $?Sig(Cl^{Rs}, Hash^{Qry.Rs}), Cl, Qry$ ], где

$Sig$ - [подпись](#), построенная по хэшу записи, уровню консистентности и типу запроса.

$Cl$ - [уровень консистентности](#) запроса.

$Qry$ - условия поиска и оплата запроса.

### **Структура подтверждения получения данных**

Узлом после успешного получения и записи данных возвращается структура, состоящая из хэша заголовка принятой записи, его подписи этого хэша и чека оплаты работы координатора.

$Ace$  [ $Sig(Hen^{Ace}), Hen = Hash^{Hdr.Ent}, Chq$ ], где

$Sig$ - [подпись](#) хэша заголовка записи узлом, принявшим данные.

$Hen$ - хэш заголовка записи, которая была принята.

$Chq$ - чек, выписанный узлом координатору, передавшему принятую запись.



## Структура ответа успешного поиска

В случае успешного поиска данных узел возвращает поисковые записи, состоящие из заголовка реальной записи, подписи реальной записи, полей реальной записи, которые были выбраны к возвращению в поисковом запросе, и полей-дополнений, содержащих хэши значений вместо значений полей.

$Srs [Enr [#Ent, Sig^{Ent.Enr}, Hdr^{Ent.Enr}, Fld_{0..n}^{Ent.Enr}, Fls [Fln, Flh]_{0..m}]_{0..p}],$  где

$Enr$ - поисковая запись.

$Ent$ - реальная запись, соответствующая найденной.

$Sig$ - подпись реальной записи.

$Hdr$ - заголовок реальной записи.

$Fld_{0..n}$ - поля реальной записи, которые требуется вернуть.

$Fls_{0..m}$ - поля-дополнения, соответствующие полям записи, которые существуют в реальной записи, но не требуются в ответе поиска.

$Flh$ - хэш значения записи.

## Структура ошибки запроса

Ошибка выполнения запроса включает подпись хэша ошибки, хэш заголовка записи при обработке запроса модификации либо хэш условий поиска при обработке запроса доступа, код ошибки и текстовое описание ошибки, если это необходимо.

$Err [\#Hash(H^{Err}, Ec^{Err}, ?Msg^{Err}), Sig(Hash^{Err}), H = (Hash^{Hdr.Ent} || Hash^{Qry}), Ec, ?Msg],$  где

$Sig$ - [подпись](#) хэша ошибки узлом, на котором она произошла.

$H$ - хэш заголовка записи, которая не была принята, либо запроса поиска, который не может быть выполнен.



*Ecs*- код ошибки выполнения запроса.

? *Msg*- текстовое описание ошибки, если необходимо.

## Структура чека доступа и модификации данных

Чек состоит из идентификатора диапазона, в котором выписан чек, номера в этом диапазоне, идентификатора узла, которому он выписан, непосредственно суммы вознаграждения, которое получит узел при его погашении, и подписи. Также чек имеет подпись хэша, сформированного из полей чека.

*Chq* [<#Hash, *Sig*(*Hash*<sup>Chq</sup>), *Tim*, *Rid*, *Num*, *Nid*<sub>0..n</sub>, *Amt*], и

*Chq* [<#Hash(*Tim*<sup>Chq</sup>, *Rid*<sup>Chq</sup>, *Num*<sup>Chq</sup>, *Nid*<sub>0..n</sub><sup>Chq</sup>, *Amt*<sup>Chq</sup>), ...], где

*Sig*- [подпись](#), построенная по хэшу чека.

*Rid*- идентификатор диапазона, в котором выписан данный чек.

*Num*- номер чека в диапазоне.

*Nid*<sub>0..n</sub>- идентификаторы узлов, которым выписан данный чек.

*Amt*- сумма вознаграждения.

*Hash*- хэш чека, построенный по идентификатору диапазона, номеру чека, идентификатору узла и сумме чека.

## Структура чека хранения данных

Чеки хранения отличаются от чеков доступа и модификации тем, что в подпись чека хранения включается заголовок записи, на хранение которой он был выписан.

*Chq* [*Sig*(*Hash*<sup>Hdr.Ent</sup>, *Hash*<sup>Chq</sup>), ...], где

*Sig*- [подпись](#), построенная по хэшу заголовка записи, на хранение которой выписан данный чек, и хэшу самого чека.



## Структура диапазона чеков

Диапазон нумерации чеков состоит из идентификатора, набора непрерывных последовательностей погашенных чеков и подписи, построенной по хэшу диапазона. Хэш диапазона в свою очередь строится по идентификатору диапазона и парам номеров чеков (наименьшего и наибольшего) непрерывных последовательностей погашенных чеков. При этом полный диапазон хранится на узле, а клиент генерирует и хранит лишь его идентификатор и номер последнего выданного чека.

$Rng [\#Hash(Id^{Rng}, Pay[Lcq, Hcq]_{0..n}^{Rng}), Sig(Hash^{Rng}), Id, Pay[Lcq, Hcq]_{0..n}]$ , где

*Sig*- [подпись](#), построенная по хэшу диапазона.

*Id*- уникальный идентификатор диапазона, созданный клиентом.

*Pay<sub>0..n</sub>*- непрерывные последовательности погашенных чеков.

*Lcq*- наименьший номер чека в непрерывной последовательности.

*Hcq*- наибольший номер чека в непрерывной последовательности.

*Hash*- хэш диапазона, построенный по идентификатору диапазона и интервалам погашенных чеков.

## Структура чека с накопительным итогом

Чек с накопительным итогом состоит из подписи, построенной по его хэшу, идентификатора узла, временного промежутка, корня дерева Меркля и итоговой суммы чека. Временные отпечатки итогового чека необходимы для ограничения включения в итог чеков, временной отпечаток которых раньше времени начала итогового чека. Все чеки, выписанные до времени начала итогового чека, принимаются узлом без их оплаты. При погашении итогового чека в блокчейне сохраняется итоговая сумма и временной отпечаток самого позднего выписанного чека, включенного в этот итоговый чек, который становится временем начала при выпуске нового чека с накопительным



итогом. Хэш итогового чека строится по идентификатору узла, корню дерева Меркла и итоговой сумме.

$Tcq$  [ $\#Hash(Nid^{Tcq}, Mtr^{Tcq}, Amt^{Tcq}), Sig(Hash^{Tcq}), Nid, Tms, Tml, Mtr, Amt]$ , где

$Sig$ - [ПОДПИСЬ](#), построенная по хэшу итогового чека.

$Nid$ - идентификатор узла.

$Tms$ - временной отпечаток начала итогового чека.

$Tml$ - временной отпечаток самого позднего выписанного чека, включенного в итоговый чек.

$Mtr$ - корневой хэш дерева Меркла, построенного по хэшам чековых диапазонов.

$Amt$ - итоговая сумма всех включенных чеков.

## Протокол

Для описания осуществляемых в рамках протокола операций и отношений элементов необходимо расширить формат описания следующими обозначениями.

Под группой понимается объединение элементов, удовлетворяющих условию, присущему данной группе. Например, группа таблиц, каждая из которых имеет хоть один внешний индекс:  $\langle Tbl \mid Edx_{0..n}^{Tbl} \neq \emptyset \rangle$ .

Применительно к группам структурных элементов полезно рассматривать дополнительные ограничения. А именно: группа элементов может быть дополнительно ограничена условием принадлежности определенному родительскому элементу. Если в группе не указан элемент, к которому она относится, такая группа называется абсолютной, и в нее включаются все указанные в группе элементы. Относительные же группы указывают,



относительно какого элемента необходимо выбрать подмножество элементов, указанных в группе. Обозначаются ссылки на группу следующим образом:

$\langle A \rangle$ - ссылка на группу элементов  $A$  относительно всей системы.

$\langle A_{0..n} \rangle := A_{0..n}$ - ссылка на группу множества элементов  $A$  без ограничений, равна этому множеству по определению.

$\langle B^A \rangle$ - ссылка на группу структурных элементов  $B$  родительского элемента  $A$  относительно всей системы. Также  $\langle B^A \rangle = \langle B \rangle^{(A)}$ .

$\langle C^B \rangle^A$ - ссылка на группу структурных элементов  $C$  родительского элемента  $B$  относительно элемента  $A$ .

Также у групп могут быть ограничения включения элементов, которые указываются после вертикальной черты:

$\langle a \mid a \in \langle A \rangle \rangle := \langle A \rangle$ - ссылка на группу элементов  $A$  относительно всей системы. Общий случай.

$\langle A \mid A \neq B \rangle := \langle A \mid \neg B \rangle$ - ссылка на группу элементов  $A$  кроме элемента  $B$  относительно всей системы.

$\langle M \mid n = N^M \rangle$ - ссылка на группу элементов  $M$  относительно всей системы, структурный элемент  $N$  которых равен  $n$ .

$\langle A[S, X_{0..n}] \mid x \notin \langle X \rangle^A \rangle$ - ссылка на группу элементов  $A$  относительно всей системы, в группу дочерних элементов  $X$  которой не входит элемент  $x$ .

$\langle S^A \mid x \notin \langle X \rangle^A \rangle$ - ссылка на группу структурных элементов  $S$  родительского элемента  $A$  относительно всей системы, в группу дочерних элементов  $X$  родительского элемента  $A$  которой не входит элемент  $x$ .

Операции и функции обозначаются как и вычисляемые элементы объектов круглыми скобками с аргументами, необходимыми для вычисления. Они могут быть как именованными, так и неименованными.



Функции, не влияющие на состояние системы, обозначаются обычным равенством. Отсутствие побочных эффектов позволяет использовать их в качестве условий:

$(A, B) = Y$ - функция, позволяющая получить имеющийся в системе элемент  $Y$  из элементов  $A$  и  $B$ , и условие, что для элементов  $A$  и  $B$  возможно получить имеющийся в системе элемент  $Y$ .

$(A, B) \neq Y$ - функция, возвращающая ошибку при попытке получить  $Y$  из элементов  $A$  и  $B$ . Условие, что для элементов  $A$  и  $B$  нет возможности получить имеющийся в системе элемент  $Y$ .

$check(A, B)$ - именованная функция без определения. Определение именованной функции может быть дано отдельно по имени этой функции, либо контекстом подразумевается, что определение известно.

Неименованная функция без определения служит простой группировкой:

$$(A) := A \Rightarrow (A \parallel B) := A \parallel B \Rightarrow (A \parallel B)_{0..n}^X := A_{0..n}^X \parallel B_{0..n}^X.$$

Операции, изменяющие состояние системы, обозначаются через вызов функции и обозначение следствия. Слева от стрелки (обозначающей следствие) указывается вызов операции и, возможно, текущее состояние системы, справа - состояние, в которую система перешла в результате вызова:

$(A) \rightarrow B \in \langle B \rangle$ - операция, порождающая новый элемент  $B$  (входящий в группу элементов  $B$ ) из элемента  $A$ .

$(X) \rightarrow B \notin \langle B \rangle$ - операция, исключающая существующий элемент  $B$  (входящий во множество элементов  $B$ ) из этого множества, используя элемент  $X$ . При этом подразумевается существование функции  $(X) = B$ , которая отображает элемент  $X$  множества  $\langle X \rangle$  на элемент  $B$  множества  $\langle B \rangle$ .

Для различия одноименных элементов используется унарная система штрихов. Например:  $A \neq A' \neq A'' \neq A'''$ .



## Управление пространствами таблиц

Пространства таблиц хранятся в блокчейне, и управление ими производится с помощью вызова функций смарт-контрактов.

### Создание пространства таблиц

Табличное пространство создается выполнением функции смарт-контракта от имени пространства (если в системе еще не существует пространства с таким именем) и набора ограничений доступа при условии, что в ограничениях доступа присутствует разрешение создавать пространство с таким именем для пользователя, вызвавшего функцию создания. При успешном завершении в системе создается пространство с указанным именем и ограничениями доступа.

$(Tsn, Rst) \rightarrow Tbs [Tsn, Rst, \dots] \in \langle Tbs \rangle \Rightarrow Tsn \in \langle Tsn^{Tbs} \rangle$ , если

$Tsn \notin \langle Tsn^{Tbs} \rangle$ , и

$createTablespace^{Rst}(Tsn)$ , где

$Tsn$ - имя нового пространства таблиц.

$Rst$ - ограничения доступа для всех элементов в рамках этого пространства таблиц, допускающее создание пространства таблиц с таким именем.

$createTablespace^{Rst}$ - функция проверки наличия права создания пространства таблиц с указанным именем.

### Удаление пространства таблиц

Табличное пространство удаляется вызовом функции смарт-контракта от имени существующего табличного пространства при условии, что в ограничениях доступа присутствует разрешение удалять табличное пространство для пользователя, вызвавшего функцию удаления.

$(Tsn) \rightarrow Tbs \notin \langle Tbs \rangle \Rightarrow Tsn \notin \langle Tsn^{Tbs} \rangle$ , если



$\exists Tbs \in \langle Tbs \rangle, Tsn = Tsn^{Tbs}$ , и

$deleteTablespace^{Rst.Tbs}(Tsn)$ , где

$Tsn$ - имя существующего пространства таблиц.

$deleteTablespace^{Rst.Tbs}$ - функция проверки наличия права удаления пространства таблиц с указанным именем.

## Управление таблицами

Таблицы, как и пространства таблиц, хранятся в блокчейне в рамках этих пространств, и управление ими производится с помощью вызова функций смарт-контрактов.

### Создание таблицы

Таблица создается в табличном пространстве выполнением функции смарт-контракта от имени пространства таблиц и имени таблицы (если в пространстве таблиц еще не существует таблицы с таким именем) при условии, что в ограничениях доступа пространства таблиц присутствует разрешение создавать таблицу с таким именем для пользователя, вызвавшего функцию создания. При успешном завершении в табличном пространстве создается таблица с указанным именем.

$(Tsn, Tbn) \rightarrow Tbl [Tbn, \dots] \in \langle Tbl \rangle^{Tbs} \Rightarrow Tbn \in \langle Tbn^{Tbl} \rangle^{Tbs}$ , если

$\exists Tbs \in \langle Tbs \rangle, Tsn = Tsn^{Tbs}$ , и

$Tbn \notin \langle Tbn^{Tbl} \rangle^{Tbs}$ , и

$createTable^{Rst.Tbs}(Tsn, Tbn)$ , где

$Tbn$ - имя новой таблицы, уникальное для пространства таблиц.

$Tsn$ - имя существующего пространства таблиц, в котором будет создана таблица.



$\text{createTable}^{Rst.Tbs}$ - функция проверки наличия права создания таблицы с указанным именем в табличном пространстве.

## Удаление таблицы

Таблица удаляется из табличного пространства вызовом функции смарт-контракта от имени существующего табличного пространства и имени таблицы в этом пространстве при условии, что в ограничениях доступа присутствует разрешение удалять таблицу в пространстве таблиц для пользователя, вызвавшего функцию удаления.

$(Tsn, Tbn) \rightarrow Tbl \notin \langle Tbl \rangle^{Tbs} \Rightarrow Tbn \notin \langle Tbn^{Tbl} \rangle^{Tbs}$ , если

$\exists Tbs \in \langle Tbs \rangle, Tsn = Tsn^{Tbs}$ , и

$\exists Tbl \in \langle Tbl \rangle^{Tbs}, Tbn = Tbn^{Tbl.Tbs}$ , и

$\text{deleteTable}^{Rst.Tbs}(Tsn, Tbn)$ , где

$Tbn$ - имя удаляемой таблицы.

$Tsn$ - имя существующего пространства таблиц, из которого удаляется таблица.

$\text{deleteTable}^{Rst.Tbs}$ - функция проверки наличия права удаления таблицы с указанным именем из табличного пространства в ограничениях доступа этого пространства.

## Управление полями

Поля таблицы хранятся в блокчейне в рамках самой таблицы; управление ими, как и таблицами, производится с помощью вызова функций смарт-контрактов.

### Создание поля

Поле создается в таблице выполнением функции смарт-контракта от имени пространства таблиц, имени таблицы, имени нового поля (если в



таблице еще не существует поля с таким именем), типа данных для этого поля и параметров выбранного типа данных при условии, что в ограничениях доступа пространства таблиц присутствует разрешение создавать поле с таким именем для пользователя, вызвавшего функцию создания. При успешном завершении в таблице создается поле с указанным именем, типом данных и настройками типа данных.

$(Tsn, Tbn, Fln, T, ?Tp) \rightarrow Fld [Fln, T, ?Tp, \dots] \in \langle Fld \rangle^{Tbl.Tbs} \Rightarrow Fln \in \langle Fln^{Fld} \rangle^{Tbl.Tbs}$ ,

если

$\exists Tbs \in \langle Tbs \rangle, Tsn = Tsn^{Tbs}$ , и

$\exists Tbl \in \langle Tbl \rangle^{Tbs}, Tbn = Tbn^{Tbl.Tbs}$ , и

$Fln \notin \langle Fln^{Fld} \rangle^{Tbl.Tbs}$ , и

$createField^{Rst.Tbs}(Tsn, Tbn, Fln)$ , где

$Fln$ - имя нового поля, уникального для таблицы.

$Tbn$ - имя существующей таблицы в пространстве таблиц.

$Tsn$ - имя существующего пространства таблиц.

$T$ - [тип данных](#) нового поля.

$?Tp$ - параметры указанного типа данных.

$createField^{Rst.Tbs}$ - функция проверки наличия права создания поля с указанным именем в таблице табличного пространства, в ограничениях доступа этого пространства.

## Удаление поля

Поле удаляется из таблицы табличного пространства вызовом функции смарт-контракта от имени существующего табличного пространства, имени таблицы и имени существующего поля в этой таблице при условии, что в



ограничениях доступа присутствует разрешение удалять поле в таблице пространства таблиц для пользователя, вызвавшего функцию удаления.

$(Tsn, Tbn, Fln) \rightarrow Fld \notin \langle Fld \rangle^{Tbl.Tbs} \Rightarrow Fln \notin \langle Fln^{Fld} \rangle^{Tbl.Tbs}$ , если

$\exists Tbs \in \langle Tbs \rangle, Tsn = Tsn^{Tbs}$ , и

$\exists Tbl \in \langle Tbl \rangle^{Tbs}, Tbn = Tbn^{Tbl.Tbs}$ , и

$\exists Fld \in \langle Fld \rangle^{Tbl.Tbs}, Fln = Fln^{Fld.Tbl.Tbs}$ , и

$deleteField^{Rst.Tbs}(Tsn, Tbn, Fln)$ , где

$Fln$ - имя поля, удаляемого из таблицы.

$Tbn$ - имя существующей таблицы, в пространстве таблиц.

$Tsn$ - имя существующего пространства таблиц.

$deleteField^{Rst.Tbs}$ - функция проверки наличия права удаления поля с указанным именем из таблицы табличного пространства в ограничениях доступа этого пространства.

## Управление индексами

Индексы создаются функциями смарт-контрактов в блокчейне и используются узлами при выполнении запросов. Индексы позволяют определить ответственные за обслуживания записи узлы и уменьшить количество опрашиваемых узлов при поиске.

### Создание кластерного индекса

$(Tsn, Tbn, Fln_{0..n}, Rpl, Prl) \rightarrow Cdx^{Tbl.Tbs} = Cdx [Rpc[Rpl, Prl, \dots], Fln_{0..n}]$ , если

$\exists Tbs \in \langle Tbs \rangle, Tsn = Tsn^{Tbs}$ , и

$\exists Tbl \in \langle Tbl \rangle^{Tbs}, Tbn = Tbn^{Tbl.Tbs}$ , и

$Fln_{0..n} \in \langle Fln^{Fld} \rangle^{Tbl.Tbs}, Fln = Fln^{Fld.Tbl.Tbs}$ , и



OASIS DECENTRALIZED DATABASE

$publishTable^{Rst.Tbs}(Tsn, Tbn)$ , где

$Rpc$ - параметры репликации и партицирования записей таблицы.

$Rpl$ - параметры репликации записей таблицы.

$Prl$ - параметры партицирования записей таблицы.

$Fln_{0..n}$ - имена полей таблицы, по которым будет строиться кластерный индекс.

$Tbn$ - имя существующей таблицы в пространстве таблиц.

$Tsn$ - имя существующего пространства таблиц.

$publishTable^{Rst.Tbs}$ - функция проверки наличия права создания кластерного индекса таблицы в табличном пространстве в ограничениях доступа этого пространства.

### **Создание внешнего индекса**

$(Tsn, Tbn, Fln_{0..n}, Edn, Rpl, Prl) \rightarrow Edx [Edn, Rpc[Rpl, Prl, \dots], Fln_{0..n}] \in \langle Edx \rangle^{Tbl.Tbs} \Rightarrow$

$\Rightarrow Edn \in \langle Edn^{Edx} \rangle^{Tbl.Tbs}$ , если

$\exists Tbs \in \langle Tbs \rangle, Tsn = Tsn^{Tbs}$ , и

$\exists Tbl \in \langle Tbl \rangle^{Tbs}, Tbn = Tbn^{Tbl.Tbs}$ , и

$Fln_{0..n} \in \langle Fln^{Fld} \rangle^{Tbl.Tbs}, Fln = Fln^{Fld.Tbl.Tbs}$ , и

$Edn \notin \langle Edn^{Edx} \rangle^{Tbl.Tbs}$ , и

$createExtIndex^{Rst.Tbs}(Tsn, Tbn)$ , где

$Edn$ - имя нового внешнего индекса, уникальное для таблицы.

$Rpc$ - параметры репликации и партицирования внешнего индекса.

$Fln_{0..n}$ - имена полей таблицы, по которым будет строиться внешний индекс.



$Tbn$ - имя существующей таблицы в пространстве таблиц.

$Tsn$ - имя существующего пространства таблиц.

$createExtIndex^{Rst.Tbs}$ - функция проверки наличия права создавать внешний индекс таблицы в табличном пространстве в ограничениях доступа этого пространства.

### **Удаление внешнего индекса**

$(Tsn, Tbn, Edn) \rightarrow Edx \notin \langle Edx \rangle^{Tbl.Tbs} \Rightarrow Edn \notin \langle Edn^{Edx} \rangle^{Tbl.Tbs}$ , если

$\exists Tbs \in \langle Tbs \rangle, Tsn = Tsn^{Tbs}$ , и

$\exists Tbl \in \langle Tbl \rangle^{Tbs}, Tbn = Tbn^{Tbl.Tbs}$ , и

$\exists Edx \in \langle Edx \rangle^{Tbl.Tbs}, Edn = Edn^{Edx.Tbl.Tbs}$ , и

$deleteExtIndex^{Rst.Tbs}(Tsn, Tbn, Edn)$ , где

$Edn$ - имя существующего внешнего индекса в таблице.

$Tbn$ - имя существующей таблицы в пространстве таблиц.

$Tsn$ - имя существующего пространства таблиц.

$deleteExtIndex^{Rst.Tbs}$ - функция проверки наличия права удалять внешний индекс таблицы в табличном пространстве в ограничениях доступа этого пространства.

### **Создание внутреннего индекса**

$(Tsn, Tbn, Fln_{0..n}, Idn) \rightarrow Idx [Idn, Fln_{0..n}] \in \langle Idx \rangle^{Tbl.Tbs} \Rightarrow Idn \in \langle Idn^{Idx} \rangle^{Tbl.Tbs}$ ,

если

$\exists Tbs \in \langle Tbs \rangle, Tsn = Tsn^{Tbs}$ , и

$\exists Tbl \in \langle Tbl \rangle^{Tbs}, Tbn = Tbn^{Tbl.Tbs}$ , и

$Fln_{0..n} \in \langle Fln^{Fld} \rangle^{Tbl.Tbs}, Fln = Fln^{Fld.Tbl.Tbs}$ , и



$Idn \notin \langle Idn^{Idx} \rangle^{Tbl.Tbs}$ , и

$createIntIndex^{Rst.Tbs}(Tsn, Tbn, Idn)$ , где

$Idn$ - имя нового внутреннего индекса, уникальное для таблицы.

$Tln_{0..n}$ - имена полей таблицы, по которым будет строиться внешний индекс.

$Tbn$ - имя существующей таблицы в пространстве таблиц.

$Tsn$ - имя существующего пространства таблиц.

$createIntIndex^{Rst.Tbs}$ - функция проверки наличия права создавать внутренний индекс таблицы в табличном пространстве в ограничениях доступа этого пространства.

### **Удаление внутреннего индекса**

$(Tsn, Tbn, Idn) \rightarrow Idx \notin \langle Idx \rangle^{Tbl.Tbs} \Rightarrow Idn \notin \langle Idn^{Idx} \rangle^{Tbl.Tbs}$ , если

$\exists Tbs \in \langle Tbs \rangle, Tsn = Tsn^{Tbs}$ , и

$\exists Tbl \in \langle Tbl \rangle^{Tbs}, Tbn = Tbn^{Tbl.Tbs}$ , и

$\exists Idx \in \langle Idx \rangle^{Tbl.Tbs}, Idn = Idn^{Idx.Tbl.Tbs}$ , и

$deleteIntIndex^{Rst.Tbs}(Tsn, Tbn, Idn)$ , где

$Idn$ - имя существующего внутреннего индекса в таблице.

$Tbn$ - имя существующей таблицы в пространстве таблиц.

$Tsn$ - имя существующего пространства таблиц.

$deleteIntIndex^{Rst.Tbs}$ - функция проверки наличия права удалять внутренний индекс таблицы в табличном пространстве в ограничениях доступа этого пространства.

### **Управление ограничениями**



Ограничения создаются функциями смарт-контрактов в блокчейне и проверяются узлами при выполнении запросов.

### Создание статических ограничений

$(Tsn, Tbn, Fln, Frc) \rightarrow Frc^{Fld.Tbl.Tbs} = Frc$ , если

$\exists Tbs \in \langle Tbs \rangle, Tsn = Tsn^{Tbs}$ , и

$\exists Tbl \in \langle Tbl \rangle^{Tbs}, Tbn = Tbn^{Tbl.Tbs}$ , и

$\exists Fld \in \langle Fld \rangle^{Tbl.Tbs}, Fln = Fln^{Fld.Tbl.Tbs}$ , и

$createFieldRestriction^{Rst.Tbs}(Tsn, Tbn, Fln)$ , где

$Fln$ - имя поля, для которого создается ограничение.

$Frc$ - условия ограничения, соответствующие формату поля.

$Tbn$ - имя существующей таблицы в пространстве таблиц.

$Tsn$ - имя существующего пространства таблиц.

### Создание динамических и умных ограничений

$(Tsn, Tbn, Trn, Trt, Trc) \rightarrow Trs [Trn, Trt, Trc] \in \langle Trs \rangle^{Tbl.Tbs} \Rightarrow$

$\Rightarrow Trn \in \langle Trn^{Trs} \rangle^{Tbl.Tbs}$ , если

$\exists Tbs \in \langle Tbs \rangle, Tsn = Tsn^{Tbs}$ , и

$\exists Tbl \in \langle Tbl \rangle^{Tbs}, Tbn = Tbn^{Tbl.Tbs}$ , и

$Trn \notin \langle Trn^{Trs} \rangle^{Tbl.Tbs}$ , и

$createTableRestriction^{Rst.Tbs}(Tsn, Tbn, Trn)$ , где

$Trn$ - имя нового ограничения записей таблицы, уникальное для таблицы.

$Trt$ - тип ограничения ([динамическое](#) или [умное](#)), указывающий формат строки условия ограничения.



$T_{rc}$ - условия ограничения, соответствующие указанному формату.

$T_{tn}$ - имя существующей таблицы в пространстве таблиц.

$T_{sn}$ - имя существующего пространства таблиц.

## **Диапазон распределения записей**

При создании кластерного индекса таблицы генерируется набор диапазонов распределения записей по значениям первичных ключей в соответствии с настройками репликации иパーティционирования индекса. Диапазоном распределения записей называется подпространство в одномерном пространстве всех возможных значений хэш-функции от значений полей, выбранных в качестве первичного ключа. Параметры репликации влияют на количество диапазонов таким образом, что значение параметраパーティционирования задает количество непересекающихся подпространств, покрывающих все пространство значений хэш-функции, а параметр репликации задает количество реплик (копий) каждого диапазона в системе. Таким образом, множество диапазонов распределения является двухмерным пространством подпространств возможных значений хэш-функции. Это пространство определяется уровнем репликации и уровнемパーティционирования таблицы. Либо его можно представить трехмерным пространством значений хэш-функции, эквивалентным своей проекции на одномерное пространство всех возможных значений хэш-функции. То есть все значения хэш-функции в этом трехмерном пространстве принадлежат одномерному пространству всех возможных ее значений и ограничены этим одномерным пространством.

### **Диапазон распределения при единичном уровне репликации**

Относительно одной реплики существует набор диапазонов распределения, количество которых равно уровнюパーティционирования кластерного индекса таблицы. Для каждого диапазона распределения существует включенный в него набор значений хэш-функции (равный



количеству возможных значений хэш-функции от значений полей кластерного индекса таблицы, деленному на уровень партицирования этого индекса) из пространства всех возможных значений хэш-функции от полей кластерного индекса таблицы. Таким образом, любой диапазон включает в себя хотя бы одно значение, и каждый диапазон одной реплики включает значения, не включенные в другие диапазоны этой реплики.

То есть:

$$P[Epi, k_{0..n}]_{0..p}, \text{ и}$$

$$P \neq \emptyset, P \subseteq K \Leftrightarrow \langle k \rangle^P \subseteq K, \text{ при этом}$$

$$P_a \cap P_b = \emptyset \wedge |P_a| \approx |P_b|, \text{ где}$$

$$p = Prl^{Rpc.Cdx.Tbl},$$

$$n = |K| \div p,$$

$$0 \leq a \leq p, 0 \leq b \leq p, a \neq b.$$

$K$ - пространство возможных значений хэш-функции от полей первичного ключа.

$Epi$ - идентификатор диапазона распределения.

$P$ - диапазон распределения в пространстве возможных значений.

$k_{0..n}$ - значения хэш-функций, которые включает диапазон.

$Prl$ - уровень партицирования кластерного индекса таблицы.

### **Реплики диапазонов распределения**

Количество реплик всех диапазонов распределения для одного пространства возможных значений хэш-функции от значений полей кластерного индекса таблицы равно уровню репликации этого индекса. Таким образом, каждая реплика одного пространства эквивалентна остальным репликам этого пространства, если в нее включены диапазоны, включающие



те же значения хэш-функции, что и диапазоны других реплик этого пространства, и множество значений всех диапазонов одной реплики является множеством всех возможных значений хэш-функции от значений полей кластерного индекса таблицы.

$R[Eri, P[Epi, k_{0..n}]_{0..p}]_{0..r} := Rpk[Eri, Epi, k_{0..n}]_{0..r, 0..p} \Leftrightarrow k_z^{Rpk_{x,y}} := k_z^{P_y R_x}$ , при этом

$R_a \simeq R_b \Leftrightarrow \langle P \rangle^{R_a} \simeq \langle P \rangle^{R_b} := \langle k^P \rangle^{R_a} \equiv \langle k^P \rangle^{R_b} \equiv K$ , где

$r = Rpl^{Rpc.Cdx.Tbl}$ ,

$p = Prl^{Rpc.Cdx.Tbl}$ ,

$n = |K| \div p$ ,

$0 \leq a \leq r, 0 \leq b \leq r, a \neq b$ .

$R$ - реплика диапазонов распределения.

$P$ - диапазон распределения в пространстве возможных значений.

$k_{0..n}$ - значения хэш-функций, которые включает диапазон.

$K$ - пространство возможных значений хэш-функции.

$Rpk_{0..r, 0..p}$ - пространство реплик диапазонов значений хэш-функции.

$Eri$ - идентификатор реплики диапазона распределения.

$Rpl$ - уровень репликации кластерного индекса таблицы.

$Prl$ - уровень партицирования кластерного индекса таблицы.

## Создание диапазона распределения

Диапазон распределения создается при [создании кластерного индекса](#).

$(Tsn, Tbn, Rpl, Prl, \dots) \rightarrow Cdx [Rpc[Rpk[k_{0..n}, \dots]_{0..r, 0..p}, \dots], \dots]$ , если

$\exists Tbs \in \langle Tbs \rangle, Tsn = Tsn^{Tbs}$ , и



$\exists Tbl \in \langle Tbl \rangle^{Tbs}, \quad Tbn = Tbn^{Tbl.Tbs}$ , где

$$p = Prl^{Rpc},$$

$$r = Rpl^{Rpc},$$

$$n = |K| \div p.$$

После создания пространства диапазонов распределения они назначаются узлам для обслуживания. При этом один узел не может обслуживать разные диапазоны одной реплики и один и тот же диапазон разных реплик. Также несколько узлов не могут обслуживать один и тот же диапазон одной реплики.

$\langle Rpk^{Tbl.Tbs} \rangle^{\langle N \rangle} \subseteq \langle Rpk^{Rpc.Tbl.Tbs} \rangle$ , при этом

$\langle Rpk^{Tbl.Tbs} \rangle^{N_x} \not\subset \langle Rpk^{Tbl.Tbs} \rangle^{\langle N \mid \neg N_x \rangle}$ , и

$\forall Rpk_{i,j}^{Tbl.Tbs} \in \langle Rpk^{Tbl.Tbs} \rangle^{N_x}, \quad \langle Rpk_{0..r,j} \mid \neg Rpk_{i,j} \rangle^{Tbl.Tbs} \not\subset \langle Rpk^{Tbl.Tbs} \rangle^{N_x} \wedge$

$\wedge \langle Rpk_{i,0..p} \mid \neg Rpk_{i,j} \rangle^{Tbl.Tbs} \not\subset \langle Rpk^{Tbl.Tbs} \rangle^{N_x}$ , где

$$N_{0..n} := \langle N \rangle,$$

$$0 \leq x \leq n,$$

$$0 \leq i \leq r, \quad 0 \leq j \leq r.$$

$N$ - зарегистрированный в системе узел.

Если представить пространство диапазонов  $Rpk_{0..r,0..p}$  в виде матрицы  $A$  размерностью  $r \times p$ , то после назначения узлу диапазона  $a_{ij} := Rpk_{i,j}$  тому же узлу остается доступной матрица для выбора диапазона назначения, полученная путем вычеркивания  $i$ -строки и  $j$ -столбца из матрицы  $A$  и элементов, назначенных другим узлам.

Например (для упрощения возьмем хэш-функцию с  $|K| = 100$ ):

1. Создается кластерный индекс таблицы с уровнем репликации 3 и уровнем партицирования 5. Следовательно, количество диапазонов



распределения будет равно  $5 \cdot 3 = 15$ , а количество возможных значений в одном диапазоне  $100 \div 5 = 20$ .

2. После создания индекса диапазонам назначаются обслуживающие их узлы. Таким образом, понадобится не менее 5 узлов для назначения всех диапазонов. В таблице показано распределение диапазонов по 5 узлам ( $A, B, C, D$  и  $E$ ) с соблюдением правил распределения для узла. Распределение диапазонов по узлам будет следующим:
- узел  $A$  получает диапазоны  $\{Rpk_{r1,p1}, Rpk_{r2,p2}, Rpk_{r3,p3}\}$ ,
- узел  $B$  получает диапазоны

$$\{Rpk_{r1,p5}, Rpk_{r2,p1}, Rpk_{r3,p2}\},$$

узел  $C$  получает диапазоны  $\{Rpk_{r1,p4}, Rpk_{r2,p5}, Rpk_{r3,p1}\}$ ,

узел  $D$  получает диапазоны  $\{Rpk_{r1,p2}, Rpk_{r2,p3}, Rpk_{r3,p4}\}$ ,

узел  $E$  получает диапазоны  $\{Rpk_{r1,p3}, Rpk_{r2,p4}, Rpk_{r3,p5}\}$ .

Таким образом, каждый узел обслуживает не все пространство возможных значений хэш-функции от первичного ключа, а лишь 3/5 всех возможных значений, в данном случае  $20 \cdot 3 = 60$ .

## Распределение записей

После распределения диапазонов в процессе работы с системой пользователи могут создавать записи в таблицах. Создание записи предполагает обязательное наличие значений полей первичного ключа. При передаче записи координатору на обработку от значений полей первичного ключа вычисляется хэш-функция (которая использовалась для построения диапазонов), по которой определяются узлы, отвечающие за обработку этой записи, которым и передается запись на обработку в соответствии с выбранным уровнем консистентности запроса.

- Определение узлов, обслуживающих запись, по первичному ключу этой записи производится следующим образом:

$$Ns_{0..m} = \{Ns \mid Ns \in \langle N \rangle \wedge h(Pk^{Hdr.Ent}) \in \langle k^{Rpk.Tbl.Tbs} \rangle^{Ns} \}, \text{ если}$$



## OASIS DECENTRALIZED DATABASE

$\exists Tbs \in \langle Tbs \rangle, Tsn^{Hdr.Ent} = Tsn^{Tbs}$ , и

$\exists Tbl \in \langle Tbl \rangle^{Tbs}, Tbn^{Hdr.Ent} = Tbn^{Tbl.Tbs}$ , где

$N$ - зарегистрированный в системе узел.

$Pk^{Hdr.Ent}$ - первичный ключ записи, хранящийся в ее заголовке.

$h$ - хэш-функция, использованная при создании диапазонов таблицы.

Таким образом, запись обслуживают те узлы, на которые назначены диапазоны распределения, к которой относится запись, и в диапазоны которых попадает значение хэш-функции от первичного ключа в заголовке этой записи.

- Определение узлов без ограничения по первичному ключу, например, для поиска записей, производится по идентификатору реплики:

$Ns_{0..m} = \{Ns \mid Ns \in \langle N \rangle \wedge Rpk_{\langle I \rangle, 0..p}^{Tbl.Tbs} \subseteq \langle Rpk^{Tbl.Tbs} \rangle^{Ns}\}$ , где

$\langle I \rangle := i_{0..n} \Rightarrow |\langle I \rangle| = n$ , и

$0 \leq i \leq Rpl^{Tbl.Tbs}$ , и

$0 \leq n \leq Cl$ .

$Rpl^{Tbl.Tbs}$ - уровень репликации таблицы.

$I$ - набор идентификаторов реплик, зависящий от уровня консистентности запроса.

$Cl$ - уровень консистентности запроса.

Также

$N$ - узлы, зарегистрированные в системе.

$Ns_{0..m}$ - узлы, обслуживающие записи.

$Rpk$ - диапазон распределения записей таблицы.

## Управление записями

Для сохранения или изменения данных в системе пользователь формирует записи и отправляет их координатору. Координатор в свою очередь направляет записи узлам, отвечающим за их обслуживание, получая от узлов вознаграждение, и возвращает пользователю ответ.



1. Клиент формирует неподписанный заголовок для определенной структуры таблицы:

$(Tsn, Tbn, V, Et) \rightarrow Hdr [Tsn, Tbn, V, Et, Tim, \dots]$ , если

$\exists Tbs \in \langle Tbs \rangle, Tsn^{Hdr.Ent} = Tsn^{Tbs}$ , и

$\exists Tbl \in \langle Tbl \rangle^{Tbs}, Tbn^{Hdr.Ent} = Tbn^{Tbl.Tbs}$ , и

$createEntry^{Rst.Tbs}(Tsn, Tbn)$ , где

$Tbn$ - имя существующей таблицы, в которую должна быть включена новая запись.

$Tsn$ - имя существующего пространства таблиц, в котором находится таблица.

$V$ - версия записи.

$Et$ - тип записи: новая запись, модификация старой, удаленная запись.

$Tim$ - временной отпечаток создания записи.

$createEntry^{Rst.Tbs}$ - функция проверки наличия права создания записей в таблице с указанным именем в табличном пространстве в ограничениях доступа этого пространства.

2. Пользователь предоставляет набор значений полей таблицы клиенту, который формирует из них поля новой записи:

$(Rv_{0..n}) \rightarrow Fld [Fln(Rv), Flv(Rv)]_{0..n}$ , где

$Rv_{0..n}$ - исходные значения, из которых определяются имена полей, и формируются значения полей записи.

$Fld$ - поля записи, построенные из исходных данных.

3. Клиент формирует первичный ключ из сформированных полей записи:

$\dots \rightarrow Pk^{Hdr} = Pk[Fln_{0..k} = \langle Flv^{Fld} \mid Fln^{Fld} \in \langle Fln \rangle^{Cdx.Tbl.Tbs} \rangle]$ .

4. Клиент определяет набор узлов, на которых будет храниться запись, и формирует чеки оплаты записи и хранения данных для этих узлов:

$\dots \rightarrow Chq^{Ent.Tbl.Tbs} = Chq [Sig(Hash^{Chq}), Rid, Num, Nid = Nid_{0..x}^{Ns}, Amt, \dots]_{0..m}$ ,  
где

$Ns_{0..x} = \langle Ns \mid Ns \in \langle N \rangle \wedge h(Pk^{Hdr.Ent}) \in \langle k^{Rpk.Tbl.Tbs} \rangle^{Ns} \rangle$ .

$N$ - узлы, зарегистрированные в системе.

$Ns_{0..x}$ - узлы, обслуживающие диапазон распределения записей, в



который попадает хэш первичного ключа записи.

*Rid*- идентификатор диапазона чеков, в котором выписан данный чек.

*Num*- номер чека относительно диапазона чеков.

*h*- хэш-функция, использованная при создании диапазонов распределения записей.

*Rpk<sup>Tbl.Tbs</sup>*- диапазон распределения записей таблицы.

5. Клиент формирует подписанную запись:

$\dots \rightarrow Ent [Sig(Hash^{Hdr.Ent}), Hdr[Hen = Hash^{Ent}, \dots], Fld_{0..n}, Chq_{0..m}]$ , где

*Ent*- новая запись, подписанная пользователем.

## Условия поиска

Условия поиска формируются из строки поиска в определенном формате, чека на оплату поиска и подписи. Проверка соответствия строки формату поиска производится непосредственно на узлах. В случае ошибки в условиях поиска узел получает оплату, как если бы он выполнил поисковой запрос.

1. Клиент формирует набор чеков для оплаты поиска на узлах (случай без использования некластерного индекса):

$(Tsn, Tbn, Cl, Qt, S) \rightarrow Chq [Sig(Hash^{Chq}), Rid, Num, Nid_{0..m} = Nid_{0..m}^{Ns}, Amt]$ ,

если

$\exists Tbs \in \langle Tbs \rangle, Tsn^{Hdr.Ent} = Tsn^{Tbs}$ , и

$\exists Tbl \in \langle Tbl \rangle^{Tbs}, Tbn^{Hdr.Ent} = Tbn^{Tbl.Tbs}$ , где

$Ns_{0..m} = \langle Ns | Ns \in \langle N \rangle \wedge Rpk_{\langle I \rangle, 0..p}^{Tbl.Tbs} \subseteq \langle Rpk^{Tbl.Tbs} \rangle^{Ns} \rangle$ ,

$\langle I \rangle := i_{0..n} \Rightarrow |\langle I \rangle| = n, 0 \leq i \leq Rpl^{Tbl.Tbs}, 0 \leq n \leq Cl$ ,

*Rpl<sup>Tbl.Tbs</sup>*- уровень репликации таблицы.

*I*- набор идентификаторов реплик, зависящий от уровня консистентности запроса.

*Cl*- уровень консистентности поиска.

*Tsn*- имя табличного пространства, которому принадлежит таблица.

*Tbn*- имя таблицы в табличном пространстве.

2. Клиент формирует подписанные условия поиска:

$\dots \rightarrow Qry [Sig(Hash^{Qry}), Tsn, Tbn, Qt, S, Chq]$ , где



$Qt$ - тип условий поиска, указывающий формат, использованный в строке поиска.

$S$ - строка поиска в соответствии с форматом типа условий.

## Формирование запросов

Запрос формируется клиентом из записи или условий поиска и уровня консистентности. При этом уровень консистентности запроса может быть меньше уровня консистентности условий поиска. Если консистентность запроса меньше консистентности условий поиска, запрос распределяется по всем узлам, которые указаны в чеке оплаты поиска, а ответ возвращается от количества узлов, вычисленного по уровню консистентности запроса, которые быстрее остальных ответили на запрос. Это позволяет поднять скорость доступа к записям за счет увеличения цены запросов и увеличить вероятность [хилинга](#). Если уровень консистентности запроса больше уровня консистентности условий поиска, то для ответа пользователю будет использован уровень с меньшей консистентностью. При этом, если клиент сам выступает координатором, то формирование подписи в запросах необязательно. Рассмотрим более общий вариант, когда подпись требуется:

$$(Cl, Qry \parallel Ent) \rightarrow$$

$$\rightarrow Rm[Sig(Cl^{Rm}, Hash^{Hdr.Ent.Rm}), Cl, Ent] \parallel Rs[Sig(Cl^{Rs}, Hash^{Qry.Rs}), Cl, Qry], \text{ где}$$

$Rm$ - запрос модификации данных.

$Rs$ - запрос доступа к данным.

$Cl$ - уровень консистентности запроса.

## Ответ узла на запрос

В зависимости от типа запроса узел формирует ответ, подтверждающий обработку запроса этим узлом.

### Ответ узла на запрос модификации данных



Ответ на запрос модификации включает в себя подписанный хэш заголовка записи, которую узел успешно сохранил. Ответ служит как подтверждением пользователю об успешном сохранении записи, так и основанием для хилинга в случае [гонки конкурентных записей](#).

$(Rm[Ent, \dots]) \rightarrow Ace [Sig(H^{Ace}), Hen = Hash^{Hdr.Ent}, Chq]$ , где

*Rm*- запрос модификации данных.

*Sig*- [подпись](#) хэша заголовка записи узлом, принял данным.

*Hen*- хэш заголовка записи, которая была принята.

*Chq*- чек, выписанный узлом координатору, передавшему принятую запись.

*Ent*- сохраняемая запись.

### Ответ узла на запрос поиска

Ответ на запрос поиска состоит из множества найденных на узле записей, отвечающих условиям поиска. В каждой записи присутствует подписанный хэш заголовка (как и в случае ответа на запрос модификации данных), сама запись с указанными в условиях поиска полями и хэши недостающих полей для обеспечения выполнения модификации записи без дополнительных запросов.

$(Rs[Qry, \dots]) \rightarrow Ent [Fld_{0..n}, \dots]_{0..p} \rightarrow$   
 $\rightarrow Enr [\#Ent, Sig^{Ent.Enr}, Hdr = Hdr^{Ent.Enr}, Fld_{0..m} = Fld_{0..m}^{Ent.Enr}, Fls_{0..l}]_{0..p} \rightarrow$   
*Srs* [*Enr*]<sub>0..p</sub>], где

$Fls_{0..l}^{Enr} = \langle Fls(F) \mid F \in Fld_{0..n}^{Ent} \wedge F \notin Fld_{0..m} \rangle^{Enr}$ ,

$Fls(Fls) \rightarrow Fls [Fln^{Fls}, Flh = Hash^{Flv.Fls}]$ .

*Rs*- запрос поиска.

*Srs*- ответ узла на запрос поиска.



*Qry*- условия поиска.

*Ent<sub>0..p</sub>*- найденные записи, отвечающие условиям поиска.

*Fld<sub>0..n</sub>*- все поля найденной записи, отвечающие условиям поиска.

*Fld<sub>0..m</sub>*- поля, которые необходимо вернуть в соответствии с условиями поиска.

*Fls<sub>0..l</sub>*- поля-дополнения, соответствующие полям записи, которые существуют в реальной записи, но не требуются в ответе.

При получении ответов от нескольких узлов координатор группирует их по первичному ключу и сравнивает версии записей. Если обнаруживается несколько версий записи (неконсистентность записи), предпринимается попытка [хилинга](#). Ответ пользователю возвращается только при обнаружении (либо при достижении) указанного в запросе уровня консистентности. При полностью консистентном состоянии системы количество и версии записей на всех узлах одной реплики одинаковы.

## Выпуск чеков

Общий принцип выпуска чеков в рамках операций создания записи и поискового запроса был рассмотрен выше. И, как упоминалось, чеки выпускаются для оплаты обработки, хранения и доступа к записям пользователей. Чек может быть выписан как одному, так и нескольким узлам одновременно. Каждый выпущенный чек имеет номер, в рамках уникального для клиента диапазона номеров, и идентификатор этого диапазона. Если узел встретит в запросе чек с тем же номером, что уже когда-то был погашен либо ожидает погашения, он откажется обрабатывать такой запрос, вернув ошибку. В этом случае диапазон номеров считается ненадежным, и клиент создает новый диапазон для продолжения работы.

$(Rng, Ns_{0..x}) \rightarrow Chq [Sig(Hash^{Chq}), Tim, Rid = Id^{Rng}, Num(Rng), Nid_{0..x}^{Ns}, Amt]_{0..m}$ ,

где


$$\text{Num}(\text{Rng}) \rightarrow \text{Num} = \text{Lcn}^{\text{Rng}}, \text{Rng}'[\text{Lcn} = \text{Lcn}^{\text{Rng}} + 1, \dots].$$

$Ns_{0..x}$ - узлы, на имя которых необходимо выписать чек.

$\text{Rng}$ - диапазон чеков, в котором выписывается чек.

$\text{Num}$ - номер чека, относительно диапазона чеков.

$\text{Lcn}$ - номер последнего чека, выписанный в диапазоне чеков.

$\text{Tim}$ - время, синхронизированное по последнему блоку в блокчейне.

Генерация диапазона номеров достаточно тривиальна. Для этого клиент использует функцию генерации UUID, которая становится идентификатором диапазона и выпускает чеки, нумеруя их с единицы.

$$(\text{Uuid}) \rightarrow \text{Rng} [\text{Id} = \text{Uuid}, \text{Lcn} = 1, \dots].$$

При отсутствии чека с накопительным итогом для пользователя у узла пользователь может его создать. Для этого он обращается в смарт-контракт, получая оттуда сумму последнего оплаченного им чека с накопительным итогом и отпечаток времени самого позднего включенного туда чека.

## Гашение чеков

Гашение чеков подразумевает включение их в сумму итогового чека для последующего обналичивания. Чтобы избежать включения одного и того же чека в итоговый чек несколько раз, используется квази-полное двоичное дерево Меркля, построенное по диапазонам чеков. В основу проверки включения чека в итоговый чек положено доказательство Меркля. Из структуры видно, что для проверки включения чека необходимо хранить лишь диапазоны чеков и корень дерева, остальные же элементы могут быть вычислены из них. При обнаружении коллизии чеков (узел обнаружил чек, который уже был включен в диапазон, либо узел уже получил чек с тем же номером) узел возвращает ошибку, получив которую клиент отказывается от использования этого диапазона чеков и генерирует новый.

**Структура дерева Меркла чека с накопительным итогом**

В каждой ветви присутствует информация о минимальном и максимальном идентификаторе диапазона, который может быть в нее включен, при этом корень всегда включает весь диапазон значений идентификаторов. Левый (нулевой) и правый (единичный) элементы дерева всегда упорядочены по возрастанию включаемых идентификаторов от минимального к максимальному. Диапазоны идентификаторов элементов дерева на одном уровне никогда не пересекаются.

$Mct [Mcr(E(Mcb^{Mct})_{\{0,1\}}), \#Mcb(E(Mcb^{Mct} || Rng)_{\{0,1\}})_{0..n}],$  и

$Mcr [Mhs, \#E_{\{0,1\}}],$  и

$Mcb [Lid, Hid, Mhs, \#E_{\{0,1\}}],$  и

$E [Lid, Hid, Mhs],$  где

$E(Mcb) = E[Lid = Lid^{McB}, Hid = Hid^{McB}, Mhs = Mhs^{McB}],$  и

$E(Rng) = E[Lid = Id^{Rng}, Hid = Id^{Rng}, Mhs = Hash^{Rng}],$  и

$Mcr(E_{\{0,1\}}) = Mcr[Mhs = h(Mhs^{E_0}, Mhs^{E_1}), \#E_{\{0,1\}}],$  и

$Mcb(E_{\{0,1\}}) = Mcb [Lid = Lid^{E_0}, Hid = Hid^{E_1}, Mhs = h(Mhs^{E_0}, Mhs^{E_1}), \#E_{\{0,1\}}],$  и

$\forall E, Lid^E < Hid^E,$  и

$\forall E_{\{0,1\}}, Hid^{E_0} < Lid^{E_1},$  и

$\forall Mcb, \forall Mcr, (\langle x \mid x \in U \wedge Lid^{E_0} \leq x \leq Hid^{E_0} \rangle \not\subseteq \langle y \mid y \in U \wedge Lid^{E_1} \leq y \leq Hid^{E_1} \rangle)^{Mcb || Mcr}.$

$U$ - множество всех возможных значений идентификаторов диапазонов чеков.

$E$ - элемент дерева Меркла.

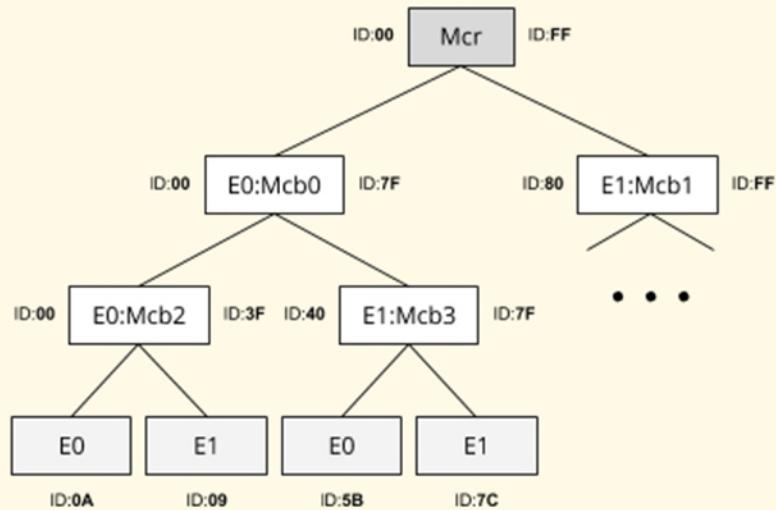
$Mcr$ - корень дерева Меркла, построенный



*Mcb*- ветви дерева.

*Mcl*- листья дерева Меркла, являющиеся хэшами непосредственно данных.

### Графическое представление дерева Меркла



### Построение дерева

Как уже было упомянуто, двоичное дерево Меркла является квази-полным, т.е. все его ветви уже разбиты по диапазонам так, что каждая дочерняя ветвь от корня включает в себя половину диапазона родителя. При этом хэш ветви, которая иерархически включает в себя лишь один лист, равен хэшу этого листа. При включении в дерево или изменении уже включенного листа пересчитывается не все дерево, а лишь хэши ветвей, в иерархии которых находится этот лист (от листа к корню). В данном случае лист дерева строится из диапазона чеков.

### Проверка включения диапазона

Общий принцип проверки заключается в том, что узел направляет клиенту пользователя чек с накопительным итогом, чеки хранения, доступа и модификации данных, соответствующие им диапазоны (если эти диапазоны уже были ранее включены в итоговый чек) и недостающие для расчета корня дерева Меркла ветви (как можно ближе к корню). Клиент проверяет



соответствие чеков (хранения, доступа и модификации данных) относящимся к ним диапазонам, помечает их в диапазонах как погашенные (при этом пересчитывается хэш диапазона), включая их суммы в итоговую сумму чека с накопительным итогом и обновляя в нем временной отпечаток самого позднего оплаченного чека при необходимости. Затем пересчитывает часть дерева Меркла в соответствии с новыми значениями хэшей диапазонов, используя переданные недостающие хэши ветвей, заменяя корень . И заново подписывает изменившиеся диапазоны и итоговый чек. После этого клиент возвращает узлу новый чек с накопительным итогом и изменившиеся диапазоны, чьи хэши были включены в дерево. При отсутствии чека с накопительным итогом либо при погашении чеков, чьи диапазоны еще не были включены в дерево, клиент формирует, подписывает и включает их в дерево.

$(Mct, \#Mcb_{0..x}^{Mct}, Rng_{0..y}, Chq_{0..z}) \rightarrow Mct'[Mcr, \#Mcb_{0..n}],$  при

$$Mcr^{Mct'} = Mcr(E(Mcb^{Mct} \parallel Mcb^{Mct'} \parallel Rng'(Rng, Chq_{...}))_{\{0,1\}}),$$

$$Mcb_{0..n}^{Mct'} = Mcb(E(Mcb^{Mct} \parallel Mcb^{Mct'} \parallel Rng'(Rng, Chq_{...}))_{\{0,1\}})_{0..n},$$

$$Mcb_{0..x}^{Mct} = \langle Mcb \mid \langle i \mid i \in U \wedge Lid^{E_0} \leq i \leq Hid^{E_0} \rangle^{Mcb} \not\subseteq \langle Id \rangle^{Rng_{0..y}} \rangle^{Mct},$$

$$Mcb_j^{Mct'} = Mcb_k^{Mct} \Leftrightarrow Lid^{Mcb_j^{Mct'}} = Lid^{Mcb_k^{Mct}} \wedge Hid^{Mcb_j^{Mct'}} = Hid^{Mcb_k^{Mct}},$$

$Rng'(Rng, Chq_{0..s}) = Rng'[\#Hash, Sig(Hash), Pay_{...}, \dots],$  где

$$Chq_{0..s} = \langle c \mid c \in Chq_{0..z} \wedge Rid^c = Id^{Rng} \rangle,$$

$$\langle p \mid p \in \langle Num^{Chq} \rangle \wedge Lcq^{Pay} \leq p \leq Hcq^{Pay} \rangle^{Rng} \cup \langle Num \rangle^{Chq_{0..s}} \subseteq$$

$$\subseteq \langle p' \mid p' \in \langle Num^{Chq} \rangle \wedge Lcq^{Pay} \leq p' \leq Hcq^{Pay} \rangle^{Rng'},$$
 где

$Mct$ - исходное дерево Меркла, хранящееся в итоговом чеке.

$Mcb_{0..x}^{Mct}$ - недостающие ветви исходного дерева Меркла для построения нового дерева.



$Rng_{0..y}$ - диапазоны, в которые необходимо включить чеки.

$Chq_{0..z}$ - чеки, которые необходимо включить в диапазоны.

$Mct'$ - новое дерево меркла, построенное из новых диапазонов и недостающих ветвей.

$Rng'$ - новые диапазоны, в которые были включены относящиеся к ним чеки.

## Получение вознаграждения

Для получения вознаграждения используется функция смарт-контракта. Функция принимает на вход чек с накопительным итогом и перечисляет на счет узла со счета пользователя, который подписал итоговый чек, средства в размере суммы, указанной в итоговом чеке, за вычетом суммы всех ранее выданных средств по этому чеку. Также в блокчейне запоминается указанный в итоговом чеке максимальный временной отпечаток включенного в него чека поиска, хранения и модификации данных. Это необходимо для восстановления итогового чека.

## Особенности системы

Как упоминалось выше, Oasis.DDB является нереляционной, распределенной, децентрализованной системой управления базами данных, использующей семантику BASE (**B**asically **A**vailable, **S**oft state, **E**ventual consistency). Здесь необходимо разделять понятия распределенности и децентрализации. Распределенные системы могут иметь централизованное управление, контролирующее работу системы. Oasis.DDB же - распределенная система, не имеющая единого контролирующего центра, и все управление и настройки производятся автоматически и согласованы с участниками системы.

Также в Oasis.DDB нет фиксированных параметров CAP (**C**onsistency, **A**vailability, **P**artition tolerance). Каждый клиент выбирает приоритет одного из



свойств по необходимости в запросах доступа и изменения данных. При этом степень репликации таблицы влияет на параметр доступности (availability), а балансом указываемого уровня консистентности в запросах доступа к данным и изменения данных контролируется консистентность (consistency) и устойчивость к разделению (partition tolerance).

## **Ограничение чтения**

Все записи в системе Oasis.DDB доступны для чтения. Это продиктовано необходимостью доступа к ней узлов для работы и автоматического распределения данных в системе. Чтобы ограничить чтение критически важной информации, рекомендуется хранить ее в зашифрованном виде. К сожалению, поиск по ней будет ограничен. Для решения этой проблемы возможно определить ключевые слова, которые будут использованы для поиска, и хранить их рядом (в той же записи) с шифрованными данными. Важно понимать, что это может частично или полностью скомпрометировать зашифрованную информацию, и поэтому следует уделять особое внимание способу выбора ключевых слов. Ключевые слова, хранящиеся в записи, можно заменить на хэши от ключевых слов, улучшив тем самым скрытие информации. Однако поиск по хэшам ключевых слов будет чувствителен к регистру и словоформам. Кроме того, со временем можно составить словарь соответствия хэшей оригиналным ключевым словам, тем самым компрометируя их.