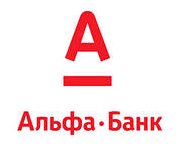
**Обучение оптимизации запросов**



**Введение.**

Данный документ является приложением к презентации, предназначен для более подробного описания обучения, в то время как презентация содержит лишь самые основные моменты.

Обучение предназначено для аналитиков и начинающих разработчиков. В документе содержатся только основы. Не содержит углубленных знаний, предназначенных для более продвинутых пользователей, чтобы легче было усваивать материал для людей, не подготовленных к этому. Также аналитикам обычно не требуется знать все аспекты оптимизации.

Слушатели после прохождения данного обучающего курса должны:

* понимать структуру выполнения запроса
* уметь просматривать и понимать план выполнения запроса
* владеть основами оптимизации запросов
* избегать типичных ошибок в написании запросов
* уметь пользоваться вспомогательными «фишками» оракла

Основные цели получения данных навыков:

* более быстрое получение из БД необходимой информации
* избегание повышенной нагрузки на базу

**Таблица.**

Таблица – структура БД, состоящая из блоков, в которой хранятся строки.

Необходимо отметить следующие моменты:

* поскольку в блоке хранятся строки целиком, а не отдельно атрибуты, то при чтении нескольких столбцов все равно сканируется вся таблица, поэтому скорость чтения не повышается.
* таблица может содержать пустоты, которые занимают место, повышая время чтения таблицы. Вплоть до того, что данных может не содержаться, но при этом таблица будет весить гигабайты данных, на чтение которых потребуется достаточно много времени.
* чтение таблиц происходит с дисков. Это трудоемкая операция. При этом данные таблицы могут после первого чтения сохраняться в оперативной памяти и при последующем чтении считывание данных идет уже не с диска, а из оперативной памяти. Что в разы повышает скорость чтения таблицы.

*Пример:*

select \* from dwh.CARDTRANSACTION\_HTRAN where value\_day between '01.10.2012' and '30.10.2012' and card\_uk = -1;

Время первого выполнения – 28 секунд.

Время повторного выполнения – 9 секунд.

**Секционирование.**

Таблицы можно разбивать на секции, которые в свою очередь можно разбивать на подсекции. Логически секция функционирует как отдельная таблица, то есть если при фильтрации в запросе наложить необходимое условие, то чтение будет идти уже не всей таблицы, а определенной партиции (нескольких партиций, сабпартиций).

*Пример:*

select \* from dwh.account\_hdim where validto = '31.12.5999';

Таблица счетов партиционирована по атрибуту validto, накладывая такое ограничение мы прочитаем 120 млн строк вместо 380 млн, то есть только одну партицию PMAX.

Какие партиции будут читаться можно посмотреть в плане выполнения запроса, данный материал рассматривается ниже.

**Если есть возможность читать таблицу не целиком, а отдельные секции, то это необходимо делать в обязательном порядке!**

**Индексы.**

Индекс – структура БД, позволяющая быстро находить конкретные значения. Является альтернативным способом чтения данных из таблицы полному сканированию.

*Пример:*

select \* from dwh.account\_hdim where validto = '31.12.5999' and uk = 9411526815;

Запрос по индексу отрабатывает за доли секунды. В то время как при полном сканировании он работал бы до нескольких минут.

**Индексы предпочтительнее полного сканирования при необходимости чтения не более 3-5%, так как при большем чтении затраты индексного доступа превышают полное сканирование.**

Если же по индексу сканировать всю таблицу, то время ее чтения больше чтения полным сканированием в 20-30 раз! Поэтому **в случае сомнений, каким образом прочитать таблицу лучше это делать полным сканированием**.

В следующих случаях оптимизатор НЕ использует индексный доступ:

* Стоимость полного сканирования по оценке оптимизатора оптимальнее индексного доступа.

*Пример:*select \* from dwh.account\_hdim where validto = '31.12.5999' and uk between 0 and 9411526815;

В данном случае по оценке оптимизатора будет считана примерно треть таблицы, поэтому он предпочел полное сканирование

* На индекс накладывается какая-либо функция (как строковая так и численная).

*Пример:*   
select \* from dwh.account\_hdim where validto = '31.12.5999' and uk **+ 1** = 9411526815 + 1;

Если же индекс создан по какой-либо функции, то для его использования в запросе нужно использовать аналогичную функцию.

**План.**

План – выбранный оптимизатором порядок выполнения запроса, то есть доступ к данным, метод соединения, порядок выполнения, различные трансформации – сортировки, фильтрации и тд.

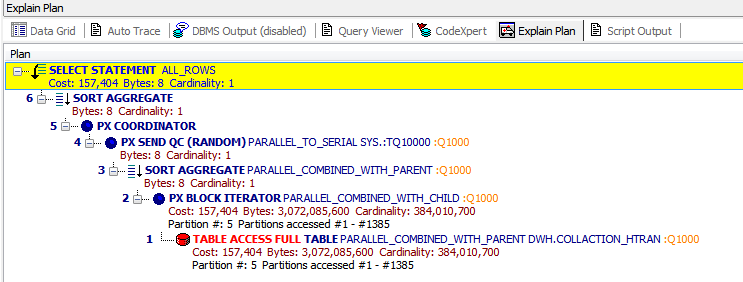
На план влияют:

* То как написан запрос, – какие таблицы, какие фильтрации, какие условия соединений и тд
* Наличие актуальной статистики на запрашиваемых объектах. Отсутствие статистики даже на одной таблице может испортить план выполнения запроса, что приведет к тому, что запрос будет работать очень долго.

Существует несколько основных способов посмотреть план запроса, причем как до его запуска, так и во время его непосредственного выполнения:

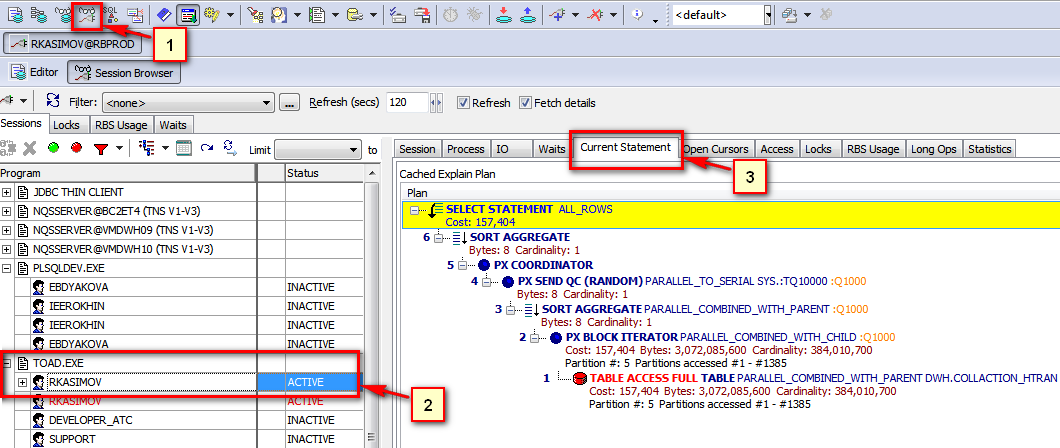
1. Ctrl + E
2. Нажав на панели инструментов следующую иконку - 

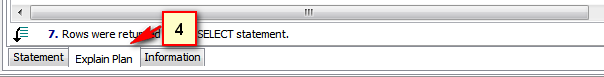
Эти два варианта приводят к отображению плана в Desktop Panels:



При необходимости просмотра плана выполняющегося запроса (как своего так и любого на базе):

1. Зайти в Session Browser, далее находим нужный запрос, затем перейти в Current Statement, Explain Plan





При просмотре плана необходимо обращать внимание на:

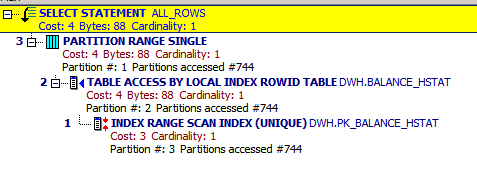
* Кардинальность (cardinality) – показатель сколько строк по оценке оптимизатора будет либо после чтения таблицы либо после соединения.
* Объем (bytes) – объем данных в байтах.

*Пример:*

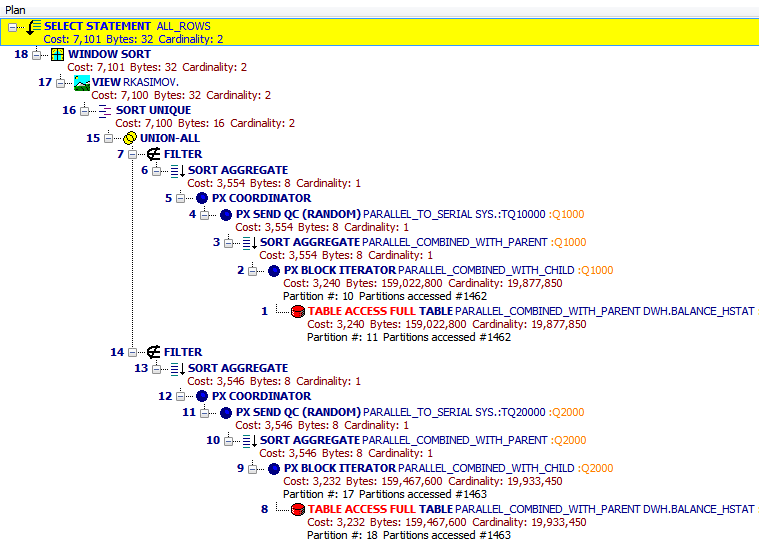


Видно, что по оценке кардинальность 15.9 млрд строк, объем – 1.4 Тб

* Метод доступа – полное чтение (как в предыдущем примере) или по индексу:



* Метод соединения (рассматривается ниже)
* Различные трансформации – объединение (union all), сортировка (sort или sort group by, windows sort говорит о том, что наложена аналитическая функция), фильтрация данных (filter):



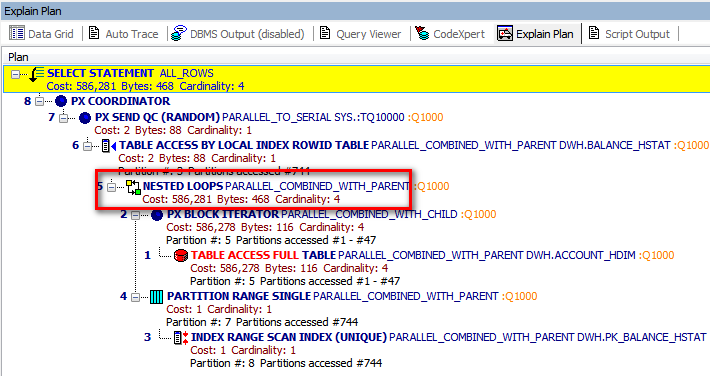
Приведен пример плана, в котором есть различные сортировки, фильтрации, UNION ALL.

**Методы соединения.**

Существует несколько основных методов соединения наборов данных:

1. **Nested loops** – вложенный цикл. Каждая строчка из первого набора посылается поочередно на соединение на второй набор.

*Пример:*select a.account\_number, b.\*  
from dwh.account\_hdim a, dwh.balance\_hstat b  
where a.uk = b.account\_uk and b.value\_day = '01.01.2013' and a.account\_number like '9998100098779%'



Из таблицы счетов выбрано 4 строки, далее каждая по вложенному циклу посылается на соединение с таблицей остатков, которая сканируется по индексу.

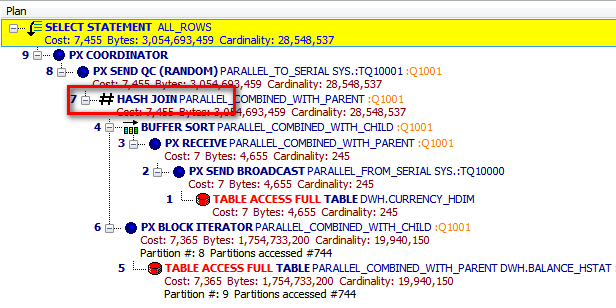
Оптимально при выборе из второго набора небольшого объема данных по индексу.

1. **Hash-join** - хэш-соединение. По исходной таблице строится хэш-функция, затем идет сканирование второй таблицы. Хэш-функция строится в оперативной памяти, если ее не хватает, то начинает использоваться дисковое пространство (TEMP), что **значительно снижает скорость выполнения запроса**.

*Пример:*

select a.iso\_ccode, b.\*

from dwh.currency\_hdim a, dwh.balance\_hstat b   
where a.uk = b.currency\_uk and b.value\_day = '01.01.2013' and a.validto = '31.12.5999'

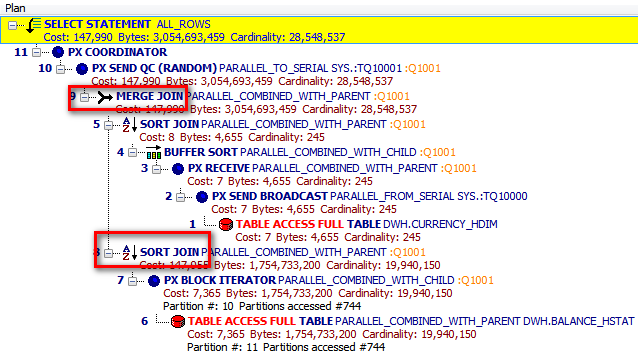


Оптимально при соединении маленькой и большой таблицы без наложения сильной фильтрации данных.

**При хэш-соединении первой обязательна должна идти меньшая таблица!** Если оптимизатор по каким-то причинам (некорректная статистика, неоптимальные хинты) выбрал обратный порядок соединения, то запрос может из быстро работающего превратиться в долго работающий и даже вообще не отработать и упасть с ошибкой нехватки пространства TEMP.

1. **Merge join** - соединение слиянием. Оба входных набора сортируются по атрибутам соединения, далее происходит соединение по отсортированным данным.

*Пример:*



Обычно не является оптимальным методом доступа, от него необходимо избавляться.

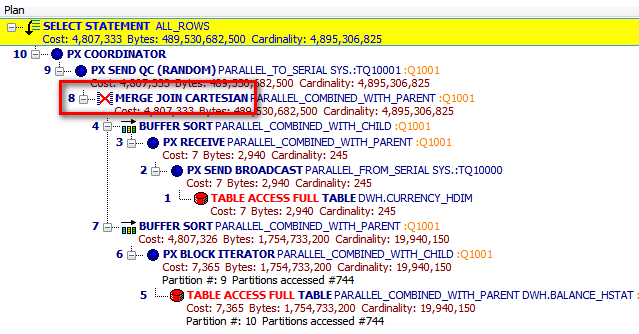
1. **Merge join Cartesian** - декартово произведение. Означает перемножение обоих наборов. Обычно является следствие ошибки написания запроса или отсутствия статистики. От этого **необходимо сразу избавляться**. Допустимо только в случае соединения маленьких таблиц, когда их наборы действительно нужно перемножить.

*Пример:*

select a.iso\_ccode, b.\*

from dwh.currency\_hdim a, dwh.balance\_hstat b

where b.value\_day = '01.01.2013' and a.validto = '31.12.5999'



**Сортировка.**

Для сортировки, как и для хэш или мердж соединений выделяется область в оперативной памяти.

Если выделенной области не хватает, то начинает использоваться дисковое пространство (TEMP), **что резко снижает производительность запроса**. Зачастую одна и та же сортировка в зависимости от нагрузки на базе может выполняться либо полностью в оперативной памяти либо с использованием пространства TEMP, что значительно увеличивает время выполнения запроса.

При нехватке пространства TEMP **запрос упадет** с соответствующей ошибкой.

Отличие трансформации UNION от UNION ALL заключается в том, что первая приводит к сортировке данных для обеспечения уникальности. Поэтому если сортировка не нужна, то лучше использовать трансформацию UNION ALL.

К сортировкам приводят следующие операции:

* ORDER BY – непосредственно сама сортировка
* GROUP BY - группировка данных также требует сортировки данных
* UNION – сортировка с обеспечением уникальности
* Любые аналитические функции

Сортировка трудоемкая операция, поэтому необходимо где возможно ее либо избегать либо сокращать объем сортируемых данных.

**Основные хинты.**

Хинт – подсказка оптимизатору, что в определенном месте необходимо скорректировать или стабилизировать план выполнения запроса.

Синтаксис – пишется как /\*+ hint\*/ или --+ hint.

Ставится сразу после основного оператора – select, update, delete, merge, insert.

В скобках, где это необходимо подставляется синоним таблицы, если он есть, иначе само название таблицы. Так как если в запросе используется синоним таблицы, а в хинт прописать само название таблицы, то этот хинт проигнорируется.

В разработке употребление хинтов должно быть оправданным, желательно получать оптимальные и стабильные планы без них, например настроив сбор статистики. Так как они лишают оптимизатор гибкости выбора плана в зависимости от различных показателей.

Для разового использования использование хинтов нет никаких ограничений. Главное понимать смысл использования хинта, будучи уверенным, что с хинтом план запроса получается оптимальнее, чем без него.

**Использовать хинты нужно аккуратно, так как один** **некорректный хинт может превратить хорошо работающий запрос в неработоспособный**!

Основные хинты:

* **use\_hash** – использовать для указанных таблиц хэш-соединение. Пример – /\*+ use\_hash (a, b, c)\*/ означает, что для таблиц a, b, c соединение нужно производить по хэшу.
* **use\_nl** – использовать соединение по вложенному циклу. Пример - /\*+ use\_nl(a)\*/ означает, что с таблицей а необходимо соединяться по вложенному циклу.
* **full** – производить полное сканирование таблицы. Пример - /\*+ full(a)\*/ означает, что таблицу а нужно сканировать полностью, а не по индексу.
* **index** – производить сканирование таблицы по индексу. Пример - /\*+ index(a b)\*/ - производить сканирование таблицы a по индексу b
* **no\_merge** – не раскрывать подзапрос. Ставится в подзапросе, показывает оптимизатору, что данный подзапрос необходимо выполнить полностью, иначе оптимизатор может решить, что оптимальнее будет его раскрыть и производить соединения по-другому.
* **cardinality** – кардинальность (количество строк) при выборке из таблицы, необходимо для помощи оптимизатору, если он по каким-то причинам ошибается, а мы можем ему подсказать более точное значение. Пример - /\*+ cardinality(a 15)\*/ - кардинальность выборки из таблицы а равна 15
* **parallel** – параллельность, то есть чтение таблицы в несколько параллелей. Необходимо использовать только для крупных таблиц, на которых не установлена параллельность на уровне настроек таблицы. Пример – */\*+ parallel(pl 4)\*/* - использовать на таблице pl параллельность 4.

*Пример:*  
select */\*+ use\_hash(pl) parallel(pl 4)\*/* pl.\* from dmfr.pltransaction\_tran pl

inner join

(select */\*+ no\_merge cardinality(a 15)\*/* uk from dmfr.accountclause\_sdim a

where accountclause\_number like '5587%') acc on

acc.uk = pl.accountclause\_mark\_uk  
where pl.value\_day = '01.01.2013'

**Дополнительные полезные возможности ORACLE.**

1. **ROWNUM** – ограничитель количества строк.

Синтаксис – ставится в условии фильтрации как … and rownum <= x. В данном случае запрос вернет не более x строк.

Используется в следующих случаях:

* Достаточно посмотреть первые x строк
* Необходимо проверить наличие определенных записей в таблице или партиции, при этом неважно их количество. Тогда запрос выглядит примерно следующим образом - Select \* from table\_name where value\_day = date and rownum <=1

1. **WITH** – вынесенный подзапрос. Используется в случае, если один и тот же подзапрос используется несколько раз внутри запроса. Используется как для удобства (чтобы не описать один и тот же блок несколько раз), так и для оптимизации (чтобы один блок выполнялся один раз).

Синтаксис:

With a as (select */\*+ materialize\*/* …)

Select …

В данном примере вынесен из запроса один подзапрос. При этом он заматериализован (добавлен хинт /\*+ materialize\*/), то есть он выполнится один раз, а его результат будет храниться во временной области TEMP.

При необходимости вынесения нескольких подзапросов синтаксис будет следующим:

With a as (select…),

B as (select…)

Select …

То есть запросы перечисляются через запятую.

1. **EXISTS** – проверка наличия записей. Является более предпочтительной по сравнению с функцией IN, так как в некоторых случаях функция IN приводит к неоптимальному плану выполнения запроса.

*Пример:*

select a.\* from dmfr.account\_sdim where exists

(select 1 from dmfr.gl2account\_sdim gl where gl.uk = a.gl2account\_uk and gl.balance\_flag = '?')

**Полезные вьюшки.**

В базе очень много системных вьюшек из которых можно достать всю необходимую информацию по любому объекту – его нахождение, объем, его структура, количество строк (приблизительное по статистике) и др.

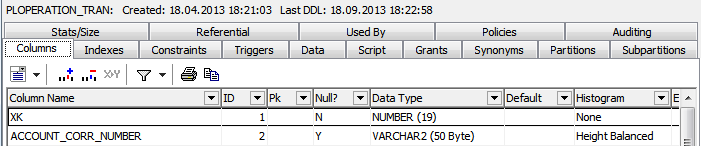
Ниже приведены самые основные вьюшки и описание того, что в них можно найти.

|  |  |
| --- | --- |
| Dba\_segments | Объем данных таблицы, индекса, партиции и тд |
| All\_tables | Информация по таблице – схема, количество строк (примерное по статистике), настройки таблицы и др. |
| All\_tab\_columns | Информация по атрибутам таблицы – наименование, тип, ограничение |
| All\_tab\_partitions | Информация по партициям таблицы – название, значение, настройки |
| All\_part\_key\_columns, all\_subpart\_key\_columns | Можно посмотреть по каким атрибутам (саб)партиционирована таблица |

*Пример:* если нужно знать примерное количество строк в таблице, то достаточно выполнить запрос:

select num\_rows from all\_tables where table\_name = :a and owner = :b

Также в TOAD есть способ быстро получить нужную информацию – для этого необходимо либо выбрать таблицу в Schema Browser либо написав ее в редакторе нажать Ctrl + щелкнуть по ней левой клавишей мыши. Появится примерно следующее окно:



Можно здесь получить следующую полезную информацию:

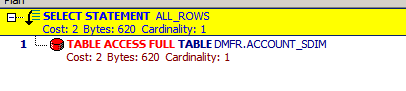
* Когда была создана таблица или дата последнего DDL
* Набор и типы атрибутов – на вкладке Columns
* Какие индексы есть у таблицы – вкладка Indexes
* Просмотреть сами данные – вкладка Data
* Партиции и количество строк в партициях (по статистике) – вкладка Partitions
* Размер таблицы, количество строк, настройки – вкладка Stats/Size

**Тренировочные запросы.**

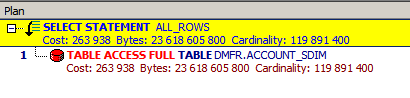
Ниже приведен ряд запросов с планами и комментарии к ним, некоторые из них правильные, некоторые имеют типичные ошибки. Необходимо разобраться с каждым из них, понять в чем суть проблемы, как его можно исправить.

1. Выборка из счетов:

Select \* from dmfr.account\_sdim

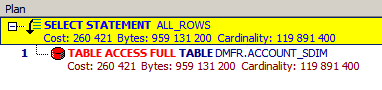
1) 

В данном случае приведен пример отсутствия статистики на таблице. По оценкам оптимизатора кардинальность выборки 1 (0 он никогда не пишет, поэтому 1). В данном случае необходим сбор статистики по таблице.

2) 

Тот же самый запрос, но кардинальность корректная, что говорит о том, что статистика на таблице актуальная.

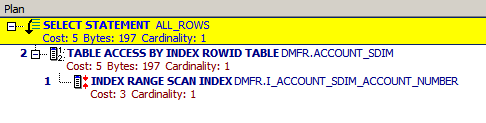
3) select uk from dmfr.account\_sdim



Если сравнить с предыдущим запросом, то видно, что считывается то же количество строк, но к себе мы считаем уже вместо 23,6 Гб около 1 Гб данных. Также необходимо отметить, что хоть мы и считаем себе в десятки раз меньше данных, но прочитать все равно придется всю таблицу, поэтому на чтение мы потратим такое же время как и в предыдущем запросе.

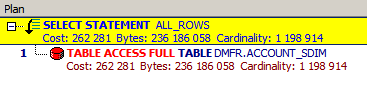
1. Выборка из счетов по индексу.

1) select \* from dmfr.account\_sdim where account\_number = '61304810509570000510';



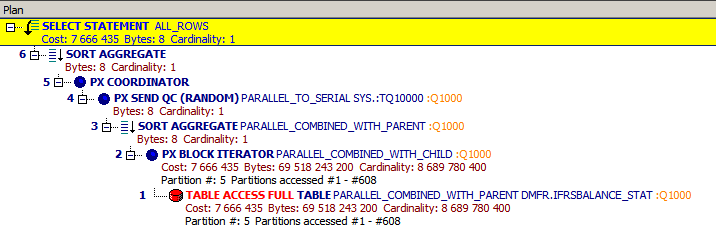
Чтение счетов идет по индексу, выбирается одна запись за доли секунды.

2) select \* from dmfr.account\_sdim a where substr(account\_number, 1, 19) || '0' = '61304810509570000510';



Происходит подавление индекса из-за наложения на индексируемый атрибут функции. В результате оптимизатор не может использовать индекс, хотя запрос в принципе идентичен предыдущему. Как итог запрос вместо долей секунды будет работать несколько секунд. Также сбивается кардинальность выборки.

1. Поиск максимального дня в остатках:

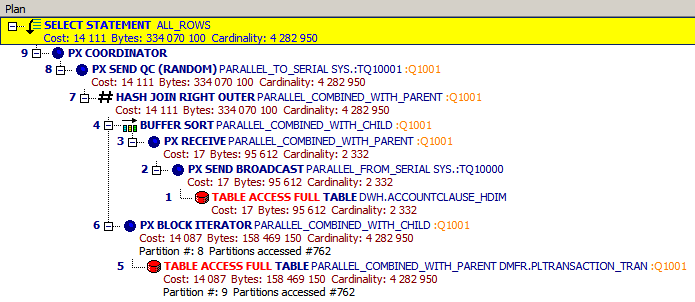
select max(value\_day) from dmfr.ifrsbalance\_stat;

Крайне тяжелый для базы запрос. При этом запросы такого плана распространены. Читаются полностью все остатки, почти 9 млрд строк, 1.5 Тб данных! Хотя понятно, что можно не проверять старые года и месяца и речь о проверке идет только о новых датах, которые можно проверить каждую в отдельности, используя конструкцию ROWNUM.

1. Соединение проводок со статьями:
2. select a.trn\_source\_no, b.name\_eng

from dmfr.pltransaction\_tran a

left join dmfr.accountclause\_sdim b on a.accountclause\_mark\_uk = b.uk

where a.value\_day = '31.01.2013' and a.deleted\_flag = 'N'

Корректный план – соединение маленькой и большой таблицы с помощью хэш-джойна. На проводки по статьям фильтрации не накладывается, поэтому соединение вложенным циклом было бы крайне неоптимальным.

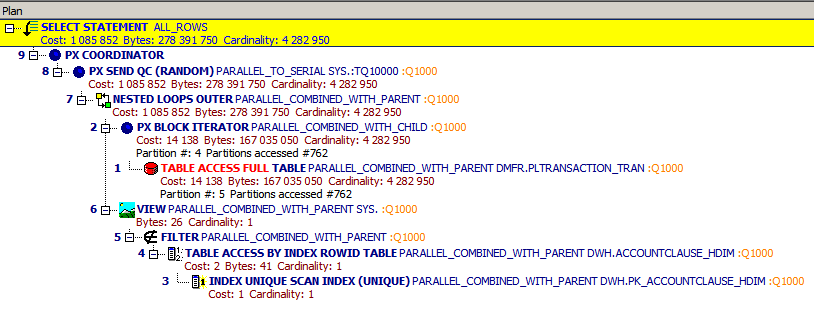
1. select a.trn\_source\_no, b.name\_eng

from dmfr.pltransaction\_tran a

left join dmfr.accountclause\_sdim b on

a.accountclause\_mark\_uk = b.uk **and a.deleted\_flag = 'N'**

where a.value\_day = '31.01.2013';



Распространенная ошибка, заключается в переносе условия фильтрации основной таблицы в условие соединения, при этом условие соединение – внешнее (left join). Данная ошибка приводит к крайне плохому плану – 4 млн проводок соединяются со статьями через вложенный цикл. Такой запрос будет выполняться очень долго и создаст высокую нагрузку на базе.

Исправление зависит от того что необходимо – если проверка по deleted\_flag должна быть в фильтрации, то она туда и переносится (см. предыдущий пример), если же такая проверка все же необходима именно в условии соединения, то необходимо воспользоваться функцией decode(). Тогда запрос выглядит так:

select a.trn\_source\_no, b.name\_eng

from dmfr.pltransaction\_tran a

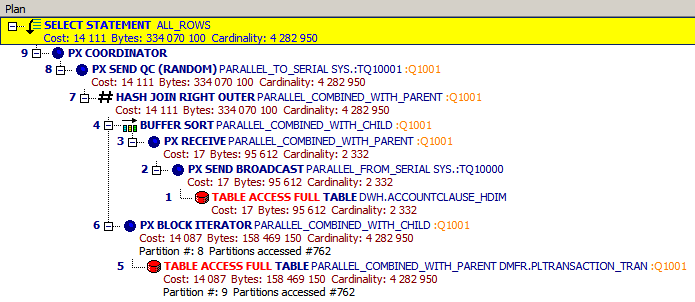
left join dmfr.accountclause\_sdim b on

a.accountclause\_mark\_uk = b.uk and *--a.deleted\_flag = 'N'*

**decode(a.deleted\_flag, 'N', 1, b.accountclause\_number, 0, 0) = 1**

where a.value\_day = '31.01.2013';

То есть сначала должно быть сравнение с нужным значением, потом с каким-нибудь атрибутом из второй таблицы, так чтобы равенство никогда не произошло. Тогда получаем снова следующий корректный план:



Стоит отметить что точно такой же план будет получен, если поменять left join на inner join:

select a.trn\_source\_no, b.name\_eng

from dmfr.pltransaction\_tran a

**inner** join dmfr.accountclause\_sdim b on

a.accountclause\_mark\_uk = b.uk **and a.deleted\_flag = 'N'**

where a.value\_day = '31.01.2013';

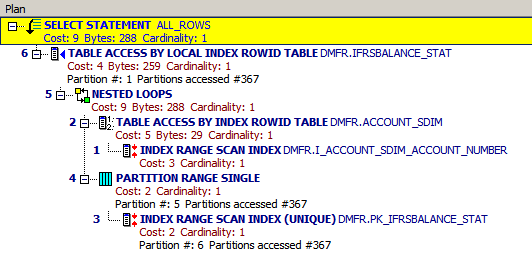
1. Соединение счетов и остатков по вложенному циклу:

select a.\* from dmfr.ifrsbalance\_stat a

inner join dmfr.account\_sdim b on

a.account\_uk = b.uk and B.ACCOUNT\_NUMBER = '47423810705660109619'

where a.value\_day = '01.01.2013'



Абсолютно корректный план, отрабатывает за доли секунды. Сначала по индексу ищется счет, затем он соединятся с остатками по вложенному циклу. Остатки также читаются по индексу.

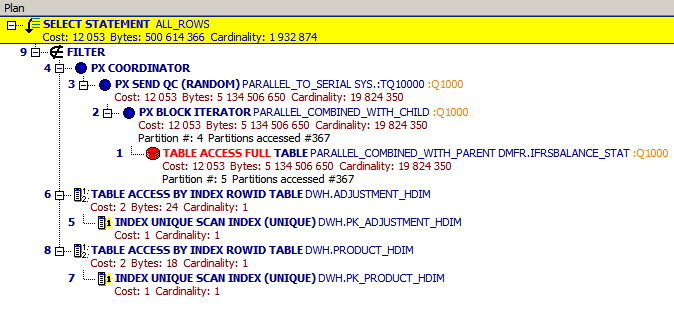
1. Фильтрация остатков:

select a.\* from dmfr.ifrsbalance\_stat a

where a.value\_day = '01.01.2013'

and (exists (select 1 from dmfr.adjustment\_sdim b where a.adjustment\_uk = b.uk and end\_date <= '01.01.2013')

**OR** (exists (select 1 from dmfr.product\_sdim c where a.product\_l2\_uk = c.uk and c.default\_flag = 'Y')))



В данном случае трансформация FILTER аналогична вложенному циклу. То есть на соединение со справочниками продуктов и корректировок посылается 20 млн записей! Такой запрос очень неоптимальный и помимо длительного выполнения создаст высокую нагрузку на базу.

Такой план получился из-за условия в фильтрации OR.

Запрос необходимо переписать следующим образом – произвести обычные левые внешние соединения со справочниками, затем на подзапрос наложить фильтрацию:

select \* from (

select a.\*, b.uk as b\_uk, c.uk as c\_uk from dmfr.ifrsbalance\_stat a

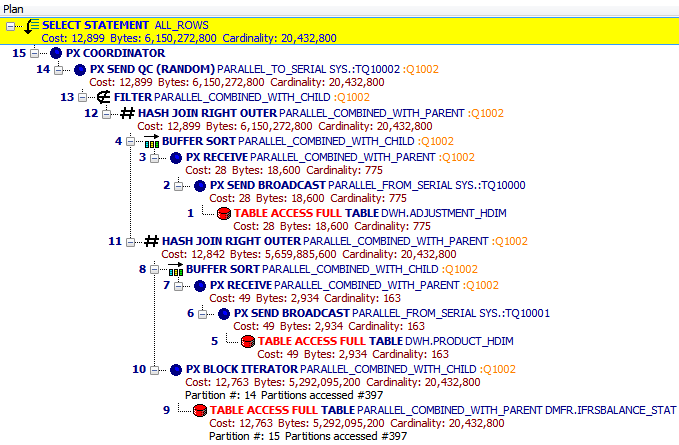
left join dmfr.adjustment\_sdim b on a.adjustment\_uk = b.uk and end\_date <= '01.01.2013'

left join dmfr.product\_sdim c on a.product\_l2\_uk = c.uk and c.default\_flag = 'Y'

where a.value\_day = '31.01.2013' and a.deleted\_flag = 'N')

where b\_uk is not null or c\_uk is not null;

Тогда план запроса становится таким:



1. Запрос в запросе:

select b.xk,

case when ( select default\_flag from dwh.account\_hdim a

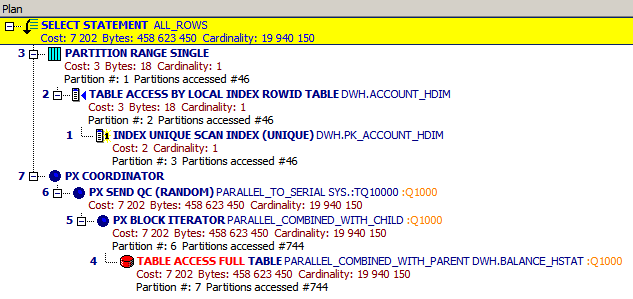
where a.validto = '31.12.5999' and b.account\_uk = a.uk) = 'N' then 1 else 0 end

from dwh.balance\_hstat b

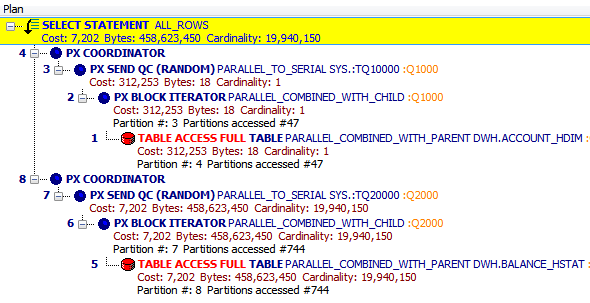
where value\_day = '01.01.2013'

Никогда нельзя использовать в селекте еще один селект! Как в данном примере для каждого остатка выполняется селект по счетам, то есть 20 млн раз! Это долго и очень высокая нагрузка на базу.

План запроса выглядит следующим образом:



При этом если бы на счетах не было индекса, то запрос выглядел бы так:



То есть большая таблица счетов полностью сканировалась бы 20 млн раз!

Запрос необходимо переписать на обычный джойн:

select b.xk,

case when a.default\_flag = 'N' then 1 else 0 end

from dwh.balance\_hstat b

left join dwh.account\_hdim a on

a.validto = '31.12.5999' and b.account\_uk = a.uk

where value\_day = '01.01.2013';

1. Лишнее чтение на примере проводок:

SELECT \*

FROM DWH.TRANSACTION\_HTRAN tr

JOIN

(SELECT DISTINCT pd.tk, value\_day, value\_date

FROM DWH.PAYMENTDOCUMENT\_HTRAN pd

JOIN DWH.CURRENCY\_HDIM cur

ON PD.CURRENCY\_PAYMENT\_UK = cur.uk

WHERE PD.VALUE\_DAY BETWEEN '20.05.2013' AND '31.05.2013'

AND pd.validto = '31.12.5999'

AND cur.validto = '31.12.5999'

AND CUR.CURRENCYTYPE\_UK = 3

AND PD.PAYDOCCLASS\_UK = 8657389640

UNION ALL

SELECT DISTINCT pd.tk, value\_day, value\_date

FROM DWH.PAYMENTDOCUMENT\_HTRAN pd

WHERE PD.VALUE\_DAY BETWEEN '20.05.2013' AND '31.05.2013'

AND pd.validto = '31.12.5999'

AND pd.PAYDOCTYPE\_UK = 8657389591 *--'01OOB00'*

AND PD.PAYDOCCLASS\_UK = 8657389640

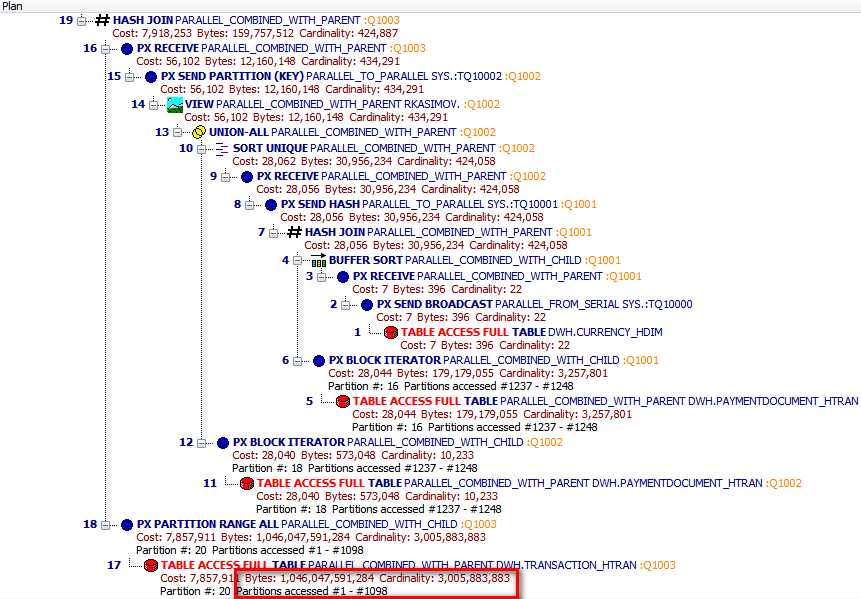
) pd

ON TR.PAYMENTDOCUMENT\_TK = pd.tk

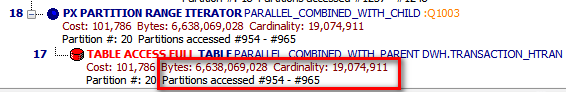
AND TR.PAYMENTDOCUMENT\_VALUE\_DAY = PD.VALUE\_DAY

AND **TR.VALUE\_DAY = PD.VALUE\_DATE**

WHERE tr.validto = '31.12.5999'



Видно, что на таблицу DWH.PAYMENTDOCUMENT\_HTRAN везде наложено ограничение по дате между 20 и 31 мая, соединение с таблицей DWH.TRANSACTION\_HTRAN содержит условие равенства дат, значит на таблицу DWH.TRANSACTION\_HTRAN также необходимо добавить условие фильтрации по датам **and tr.value\_day BETWEEN '20.05.2013' AND '31.05.2013'.** Тогда вместо 3 млрд строк мы прочитаем всего 19 млн:



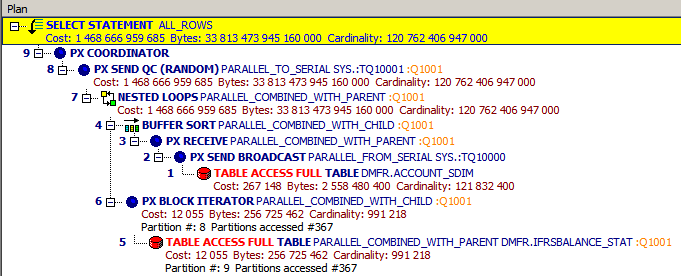
1. Отсутствие жесткого условия соединения:

select ifrs.\*

from dmfr.ifrsbalance\_stat ifrs

inner join dmfr.account\_sdim a on ifrs.account\_number like '%' || substr(a.account\_number, 3, 8) || '%'

where value\_day = '01.01.2013'



Если в условии соединения двух таблиц **нет жесткого условия соединения, то оно невозможно с помощью хэш-соединения**! В данном примере мы получили вложенный цикл. То есть партиция остатков будет читаться 121 млн раз!

1. Условие OR в соединении:

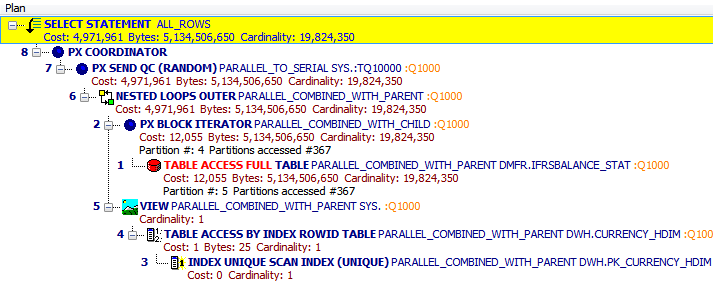
select ifrs.\*

from dmfr.ifrsbalance\_stat ifrs

left join dmfr.currency\_sdim cur on

cur.uk = ifrs.currency\_account\_uk and (cur.deleted\_flag = ifrs.deleted\_flag **OR** cur.deleted\_flag is null)

where value\_day = '01.01.2013';



Условие OR в джойне делает невозможны соединение по хэшу. Поэтому в данном примере получено соединение через вложенный цикл. В данном случае это будет выполняться очень долго и создаст высокую нагрузку на базу. Необходимо воспользоваться функцией decode:

select ifrs.\*

from dmfr.ifrsbalance\_stat ifrs

left join dmfr.currency\_sdim cur on

cur.uk = ifrs.currency\_account\_uk and **decode(cur.deleted\_flag,ifrs.deleted\_flag,1,null,1,0) = 1**

where value\_day = '01.01.2013';

