Санкт-Петербургский Государственный Университет

Математико-механический факультет

Кафедра системного программирования

Исследование оптимизации запросов в СУБД

Курсовая работа студента 445 группы

Нишневич Анастасии Юрьевны

Научный руководитель ……. Б.А.Новиков

Санкт-Петербург

2011

Оглавление

Введение

Системой Управления Базами Данных(СУБД) называется программный продукт, предназначенный для централизованного хранения данных.

Сервер базы данных - это совокупность компонент СУБД, находящихся в состоянии выполнения и способных обрабатывать данные. Данные обрабатываются по средствам запросов, которые записываются на языке запросов СУБД. Обычно это диалект SQL.

Запрос, сформулированный на языке SQL может быть преобразован в алгебраические выражения разными способами и следовательно для его выполнения могут быть построены различные планы. Все эти планы возвращают одинаковый результат, однако, совершают разные действия, а значит и скорость их выполнения разная. При этом скорости могут отличаться в тысячи раз. Такой разброс показывает, насколько важна оптимизация.

В большинстве систем (Oracle [1], MS SQL [2]) выбор оптимального плана осуществляет оптимизатор.

Оптимизатор выбирает план на основе функции стоимости.

Функция стоимости может быть задана двумя способами:

- неявно (Оптимизатор использует преобразования, заведомо приводящие к улучшению плана);
- явно (Используется функция, которая вычисляется на основе сложности алгоритма и статистических характеристиках хранимых данных).

В большинстве случаев оптимизатор выбирает лучший план.

Однако, задача оптимизации запросов не может быть решена точно, так как:

- вычисление функции стоимости основано на косвенных оценках;
- множество возможных планов очень велико.

По описанным выше причинам, в некоторых случаях необходимо понимать в каких случаях оптимизатор строит не оптимальный план и знать как преобразовывать запросы в таких случаях.

Цели и задачи

В рамках данной работы ставятся цели:

- поиск случаев, в которых оптимизатор не способен выбрать оптимальный план исполнения;
- воспроизведение этих случаев;
- преобразование запросов для построения более удачных планов.

Описание экспериментов

Эксперимент с Ms SQL

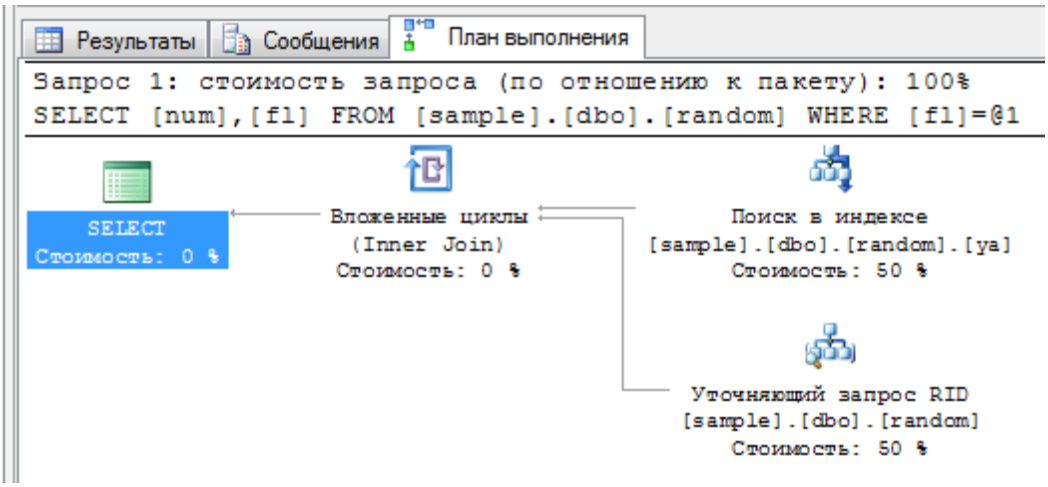
В этом эксперименте использовался Ms SQl Server 2005. Описание запроса взято с сайта http://stackoverflow.com/ [3].

Для воспроизведения была создана таблица, в которой в колонке FL значения от 1 до 21 встречались много раз, а 111 -- один раз. По FL был построен некластеризованный индекс.

Для запроса:

```
SELECT [num]
      ,[fl]
 FROM [sample].[dbo].[random]
  where fl = 111
```

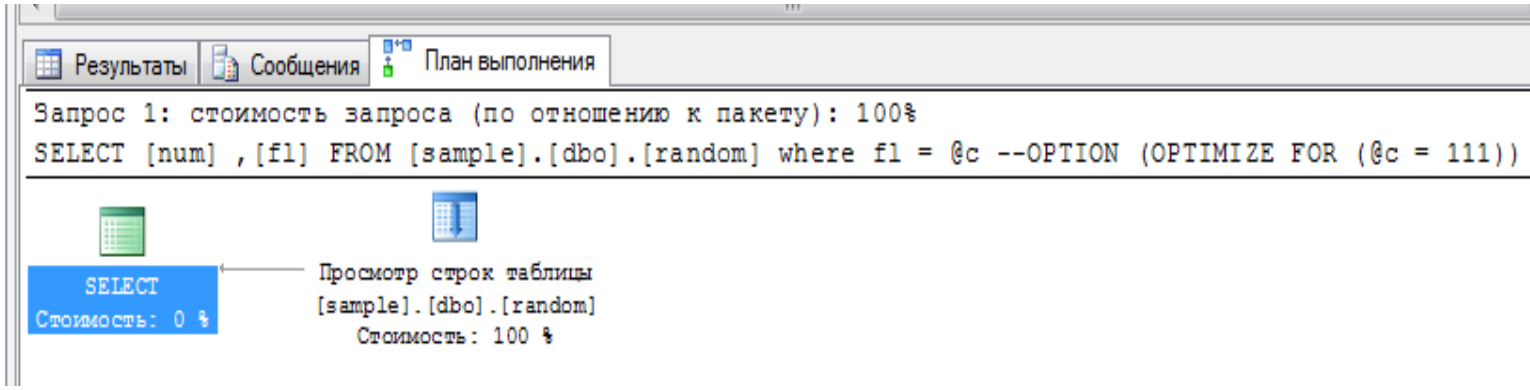
Оптимизатор построил нормальный план выполнения, с использованием индек



Однако, если в запросе точнее указание константы = 111 заменить на переменную:

```
declare @c float
set @c = 111
SELECT [num]
     ,[fl]
 FROM [sample].[dbo].[random]
  where fl = @c
```

Получим план с полным просмотром:



Таким образом, при одинаков результатах запросов на больших таблицах, второй запрос будет работать намного медленнее.

Если же использования переменной не удается избежать, можно использовать подсказку оптимизатору.

С ней запрос будет выглядеть так:

```
declare @c float
set @c = 111
SELECT [num]
     ,[fl]
 FROM [sample].[dbo].[random]
  where fl = @c
  OPTION (OPTIMIZE FOR (@c = 111))
```

А план выполнения совпадать с первым.

Эксперименты с Oracle

Для проведения экспериментов была использована версия Oracle 11.1.0.7.0

Использование “!=”

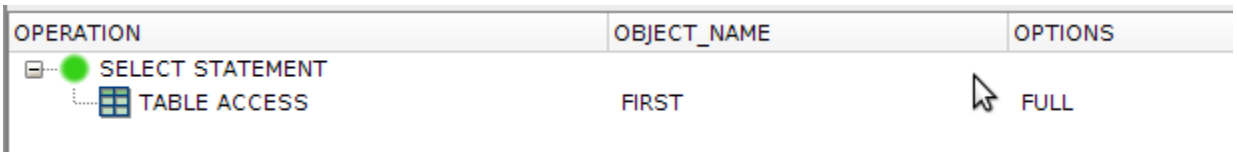
Oracle не использует индекс, если в условие выборки используется ‘not equals’.

Пусть в таблице FIRST поле RAND принимает значения 1, 110, 112 или 111. При этом в более чем в 95% случаев значение будет равно 1.

Составим запрос:

```
SELECT * from FIRST where rand != 1
```

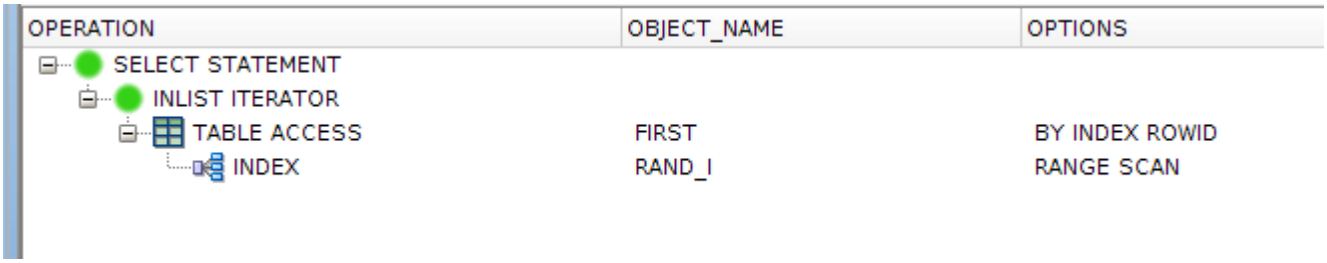
План выполнения будет использовать полный просмотр таблицы:



Однако, если переписать запрос так(заметим, что в данном случае это не изменит выдачу):

```
SELECT * from FIRST  where rand in (110,111,112)
```

План будет построен с учетом индекса:



Выборка с использованием NOT NULL

Этот и следующий примеры взяты из [4].

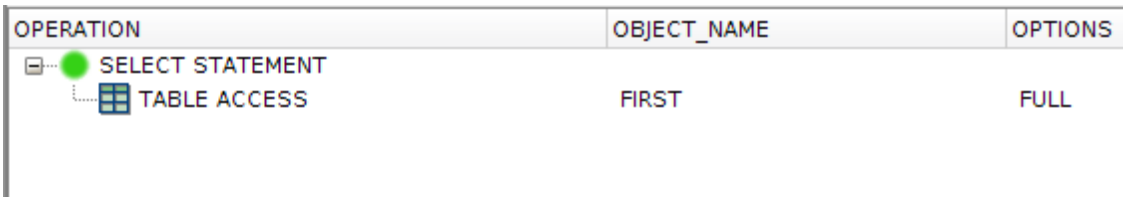
Индекс не может использоваться для поиска значений равных null.

В таблице FIRST в поле RAND один раз встречается значение null и на RAND есть индекс.

Однако, для запроса:

```
select * from FIRST where rand is null;
```

План выполнения такой:

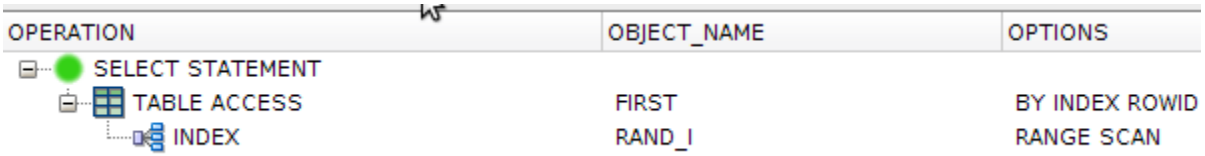


OPERATION	OBJECT_NAME	OPTIONS
SELECT STATEMENT		
TABLE ACCESS	FIRST	FULL

Заменим значения, в которых находится null , например на -1 и перепишем запрос:

```
select * from FIRST where rand = -1;
```

Хотя выдача не изменилась, в плане стал использоваться индекс:



OPERATION	OBJECT_NAME	OPTIONS
SELECT STATEMENT		
TABLE ACCESS	FIRST	BY INDEX ROWID
INDEX	RAND_I	RANGE SCAN

Выборка с использованием функции

В этом примере проводится соединение таблиц и выборка, в которой используются функции.

Рассмотрим запрос:

```
select
    *  from  first d, second i
where
    d.row_id = i.id
    and d.num in (111, 112)
    and i.num in (100, 101)
    and d.created between add_months(trunc(sysdate,'MM'), -1) and sysdate;
```

План:

```
SELECT STATEMENT
```

```
 NESTED LOOPS
     INLIST ITERATOR
      TABLE ACCESS BY INDEX ROWID SECOND
       INDEX RANGE SCAN SECOND_IDX2
 TABLE ACCESS BY INDEX ROWID FIRST
     INDEX UNIQUE SCAN FIRST_P1
```

Преобразуем запрос таким образом, чтобы значение функции вычислялось заранее:

```
select  *  from
    first d, second i
    (select add_months(trunc(sysdate,'MM'), -1) as startdate from dual) sd
where
    d.row_id = i.id
    and d.num in (111, 112)
    and i.num in (100, 101)
    and d.created between sd.startdate and sysdate;
```

План исполнения стал:

```
SELECT STATEMENT
 NESTED LOOPS
   TABLE ACCESS BY INDEX ROWID FIRST
    INDEX RANGE SCAN FIRST_IDX1
   INLIST ITERATOR
    TABLE ACCESS BY INDEX ROWID SECOND
     INDEX RANGE SCAN SECOND_IDX1
```

Изменилась последовательность действий в плане и стал использоваться индекс по первой таблице, что дало существенный выигрыш в скорости.

Неудавшиеся эксперименты

Еще два эксперимента для Oracle воспроизвести не удалось.

Оба они взяты из книги [5].

ПРимер с OR и Union

При использование в условии селекции операнда “OR” оптимизатор включает полный просмотр

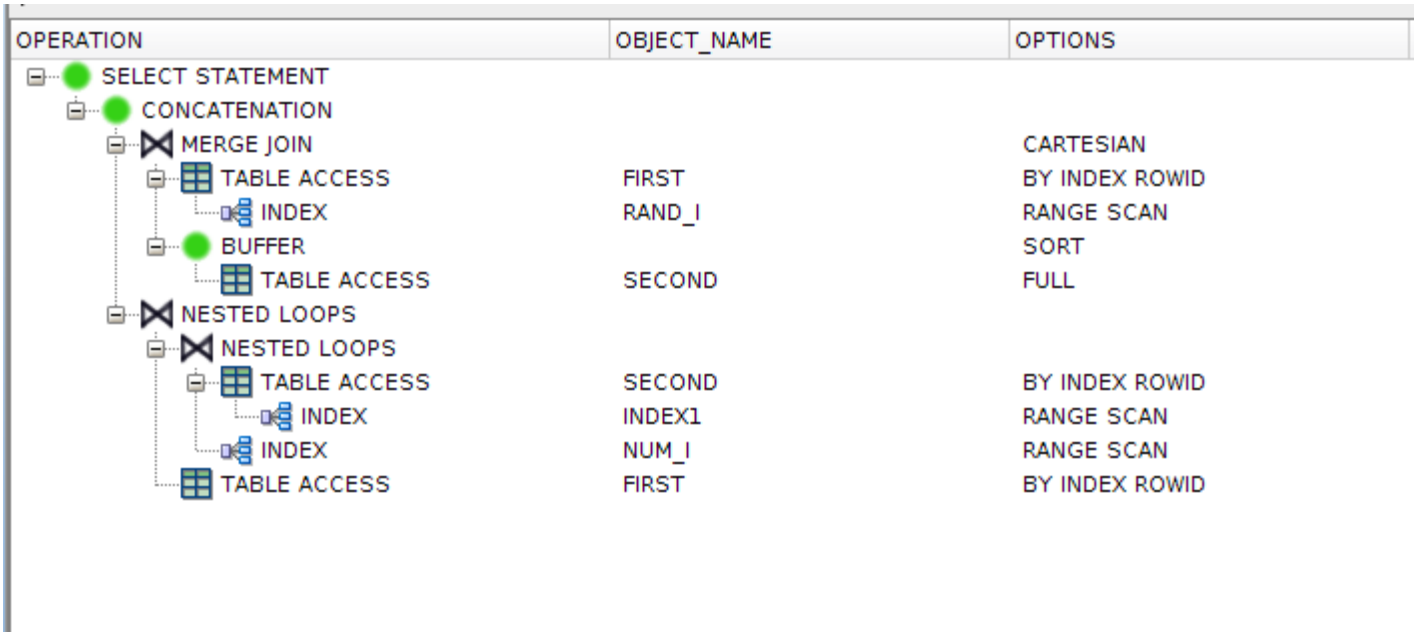
таблицы, несмотря на то что на колонках, по которым происходит выборка построен индекс. Предлагается заменять в таких случаях выборку с OR на UNION двух таблиц.

Для таблицы FIRST индекс стоит на полях NUM и RAND.

Для таблицы SECOND на поле RAND2.

Построим запрос и посмотрим на план для него:

```
SELECT * FROM FIRST F, SECOND S WHERE
F.NUM= S.NUM AND S.RAND2 = 110 OR F.RAND =
11
```



В плане используется индекс, а значит описанное выше преобразование запроса не нужно.

Пример с Not IN(...) и MINUS

Предлагается заменять условие NOT IN на теоретико-множественную операцию MINUS.

В таблице SECOND на поле RAND2 есть индекс и оно принимает значение 110 один раз.

Но более старые версии Oracle строили для запроса:

```
select rand from FIRST where rand not in(select rand2 from second where rand2 = 110) ;
```

план, в котором выполнялся два полных просмотра таблицы.

А при преобразовании:

```
select rand from FIRST  minus (select rand2 from second where rand2 = 110);
```

в плане второй операнд не вычисляется полностью, а использует индекс чтобы исключить строки из первого операнда.

В версии Oracle(11) , которая использовалась мной план для первого запроса был построен и использование индекса:

OPERATION	OBJECT_NAME	OPTIONS
SELECT STATEMENT		
MINUS		
SORT		UNIQUE
TABLE ACCESS	FIRST	FULL
SORT		UNIQUE NOSORT
INDEX	INDEX1	RANGE SCAN

так же как и для второго.

Результаты и выводы

В процессе работы были:

- найдены примеры не оптимального построения планов для Oracle и MS SQL;
- получено представление о работе оптимизаторов для Oracle и MS SQL;
- запросы из указанных выше примеров воспроизведены и преобразованы для ускорения работы.

Проделанная работа еще раз доказала наличие безусловных недостатков в работе оптимизаторов.

Во-первых, нерешенными остаются даже достаточно старые проблемы в работе оптимизаторов. Некоторые запросы для экспериментов были найдены в книге, изданной в 1997 году([]), а воспроизвести их медленную работу удалось на достаточно новом Oracle 11.1.0.7.0.

Во-вторых, оптимизатору не всегда удается удачно построить план даже для очень простых на первый взгляд запросов. Как в примере с выборкой по “!=”.

С другой стороны, некоторые примеры так и не удалось воспроизвести на новых версиях СУБД.

Что позволяет надеяться, что оптимизатор научился с ними работать.

В заключение, хочется еще раз отметить, что для эффективной работы с СУБД требуется наличие теоретической базы.

Список литературы

[1] Oracle //Офицальный сайт Oracle
URL: http://oracle.com
[2] MS SQL //Официальный сайт MS SQL
URL: http://www.microsoft.com/sqlserver
[3] stackoverflow.com// URL:http://stackoverflow.com/questions/510214/why-does-a-parameterized-query-produces-vastly-slower-query-plan-vs-non-parameter
[4] Oracle SQL High-Performance Tuning Guy Harrison 1997
[5] “Настройка Приложений баз данных” Новиков Б.А., Домбровская Г.Р.