\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
|  | ИНСТИТУТ ИНТЕЛЛЕКТУАЛЬНЫХ КИБЕРНЕТИЧЕСКИХ СИСТЕМ  Кафедра №42  «Криптология и кибербезопасность» |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Федеральное государственное автономное образовательное учреждение высшего образования*

**«Национальный исследовательский ядерный университет «МИФИ»»**

ЛАБОРАТОРНАЯ РАБОТА №2-3:

**«Индексирование и кластеризация данных.»**

Аверин Владислав

Группа: Б19-505

Октябрь, 2022

*Содержание*

[*1.* *Типичные для БД запросы* 4](#_Toc116042261)

[*2.* *Индексы (B-деревья и BIMAP)* 5](#_Toc116042262)

[*3.* *Хранение таблиц по индексу/в кластере* 11](#_Toc116042263)

[*Выводы:* 18](#_Toc116042264)

*Цель работы*

Ускорить выполнение запросов к базе данных за счёт оптимизации физической структуры данных и создания внешних индексов.

*Ход работы*

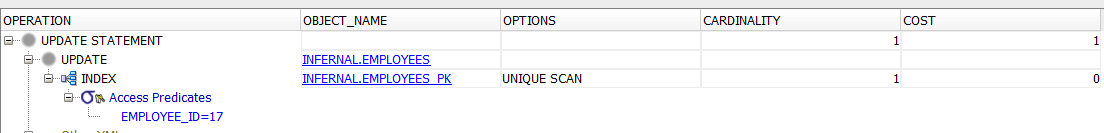
1. Если это ещё не было сделано для текущей схемы данных, разработать и проверить 5–10 запросов, которые, согласно легенде, являются типичными запросами к базе данных для разных категорий пользователей и регулярно используются при её эксплуатации;
2. Запросить у СУБД схему выполнения каждого из этих запросов. Для этого можно настроить функцию AUTOTRACE в SQL\*Plus или использовать функцию контекстного меню EXPLAIN в SQL Developer;
3. При необходимости, создать дополнительные индексы на основе B-деревьев. Обосновать их полезность. Убедиться в том, что индексы используются при выполнении запросов (план выполнения должен измениться). Если такая необходимость отсутствует, специально разработать дополнительный запрос, для которого применение индекса целесообразно;
4. Аналогично, создать индекс на основе битовых карт, обосновать его полезность и убедиться, что он используется;
5. Существуют ли в схеме данных таблицы, которые целесообразно организовать по индексу? Изменить организацию этих таблиц; сравнить новый план выполнения запросов с её участием с прежним;
6. Существуют ли в схеме данных пары (тройки,…) таблиц, которые целесообразно хранить в кластере? Если они есть, поместить эти таблицы в кластер; сравнить новый план выполнения запросов с её участием с прежним;
7. Целесообразно ли применение hash-кластера для ускорения работы с одной или несколькими таблицами? Если да, поместить её (их) в hash-кластер и сравнить новый план выполнения запросов с её участием с прежним;
8. Оформить отчёт.
9. *Типичные для БД запросы*

С учетом специфики выполняемой лабораторной, желательно выполнять те запросы, где явно можно увидеть использование индексов. Но чтобы показать их не полную власть над всеми запросами (пока мы сами не оптимизируем работу), в начале приведем примеры типичных запросов, где стандартных созданных при формировании индексов не хватает.

Медперсонал: изменить значение атрибута med\_rec у определенного сотрудника



План выполнения:



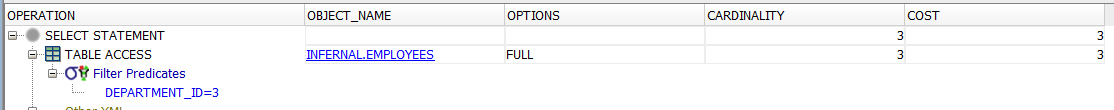
EMPLOYEES\_PK – это то название объекта, которое мы давали про создании таблицы в CONSTRAINT. То есть в данном случае индекс используется (что логично).

(Насчет UNIQUE SCAN и других методов доступа к данным будет описано далее.)

HR-отдел: найти всех сотрудников из одного отдела



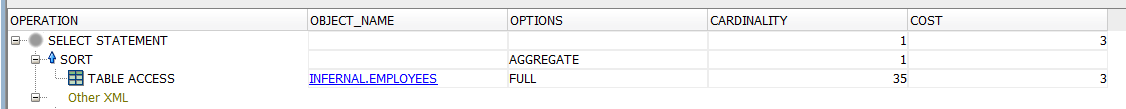
План:



Т.е. теперь СУБД просматривает все строки на наличие выполнения условия (т.к. на атрибут department\_id индекса у нас нет).

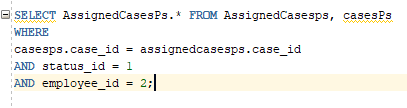
Бухгалтерия: найдем среднюю зарплату по участку



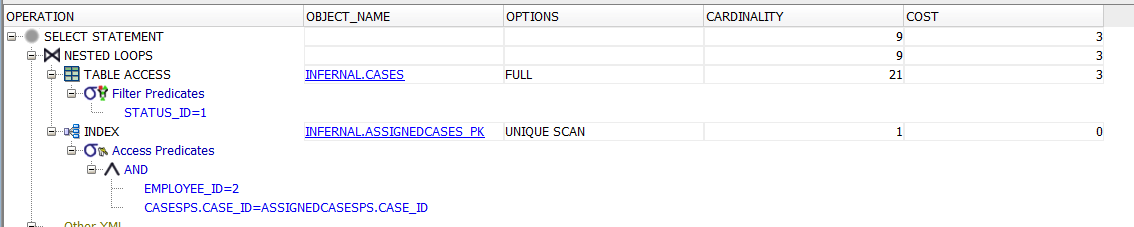


Опять же, поиск идет по всей таблице без использования индекса.

Отдел руководства: найти открытые дела, за которым закреплен определенный сотрудник



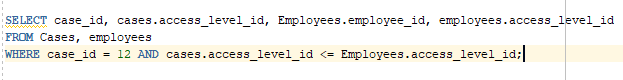
План:

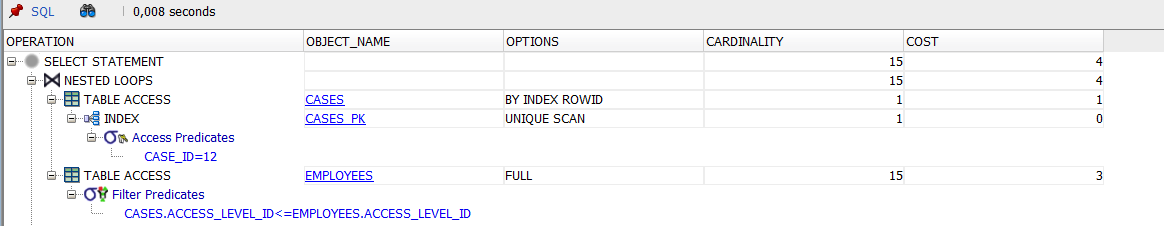


Oracle ищет в AssignedCases строки с определенным employee\_id по индексу, и делает полный перебор таблицы Cases для соединения по условию case\_id и нужного status\_id. Причем, поиск по индексу не занимает практически никакого времени, в отличие от того же TABLE FULL SCAN таблицы Cases. Поэтому неплохо было бы добавить новые индексы, но не бездумно на каждый из атрибутов всех таблиц, т.к. тогда БД начнет жутко тормозить при любом добавлении (все индексы будут балансироваться для добавления), а только те, что действительно смогут принести пользу.

1. *Индексы (B-деревья и BIMAP)*

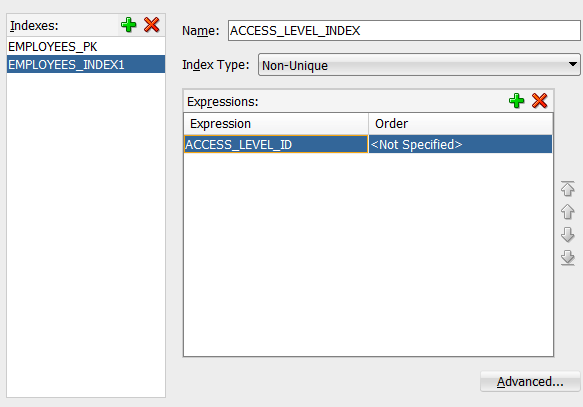
Насколько я понял, Oracle (как и в принципе все СУБД) по определению сразу создает для всех первичных ключей индексы, и индексы на основе B-деревьев (т.к. операция поиска по pk – основополагающая операция для баз данных). Поэтому практически все соединения выполняются максимально быстро, т.к. в подавляющем большинстве случаев соединения (не важно, какого типа) происходят по первичному ключу одной из таблиц. Моя БД не исключение: все потенциально типичные запросы проходят через первичные ключи (а следовательно, реализуются с помощью индексов). Самое простое решение – попробовать сделать соединение без использования первичного и внешнего ключей (гениально, знаю). В рассматриваемых таблицах Employees и Cases есть атрибут access\_level\_id – какой уровень допуска у сотрудника и требуемый уровень для доступа к делу соответственно. Попробуем найти всех сотрудников, которые могут быть допущены к работе с определенным делом:



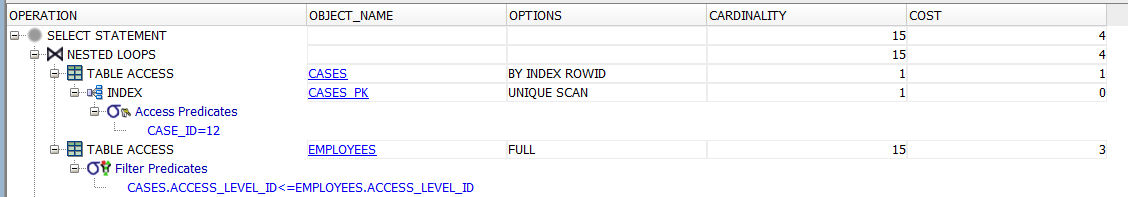


Как мы видим, в таблице Employees (для соответствия строки из Cases, где для case\_id есть индекс) используется FULL table access, т.е. перебираются все строки для поиска нужных по условию.

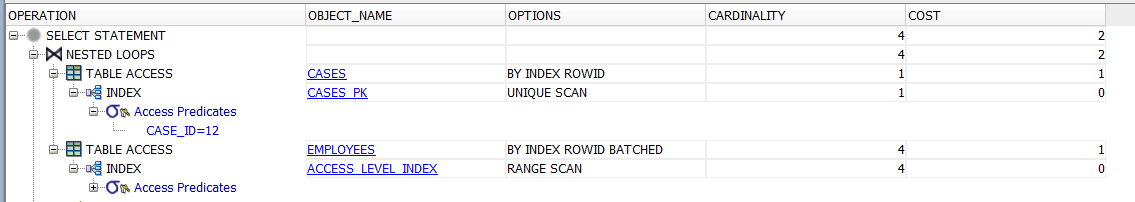
Теперь создадим индекс по Employees.access\_level\_id (из-за лени автора, был использован sql developer, не бейте):



Вызовем план еще раз, иии….

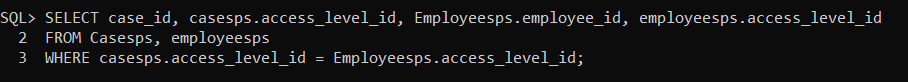


Ничего. Он все также ищет нужную строку в Cases или в самом индексе – BY INDEX ROWID, UNIQUE SCAN говорит о том, что вернется одно (или ноль) строк, а FULL как раз показывает, что СУБД не использовала наш индекс… Почему? Потому что используется неравенство, вместо равенства (видимо, для таких операций, где для нужно дополнительно обрабатывать все листовые блоки дерева, как для равенств, система считает это медленнее, чем TABLE FULL SCAN, но что-то как-то сомнительно). Но если мы сделаем равенство, то появится уже RANGE SCAN с использованием нашего индекса (т.е. субд не перебирает все листовые блоки):

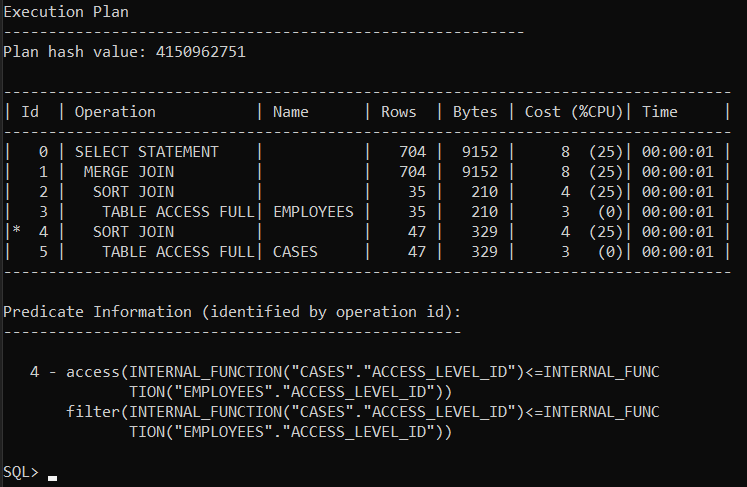


(ROWID BATCHED как я понял просто более оптимизированный ROWID, если нужная информация уже есть в блоке индекса, то обращаться к самому ряду таблицы не надо)

Можно еще расширить это дело и вывести таких сотрудников вообще для всех дел (согласен, немного странный запрос сам по себе, но может, кто-то захочет его как подзапрос использовать):

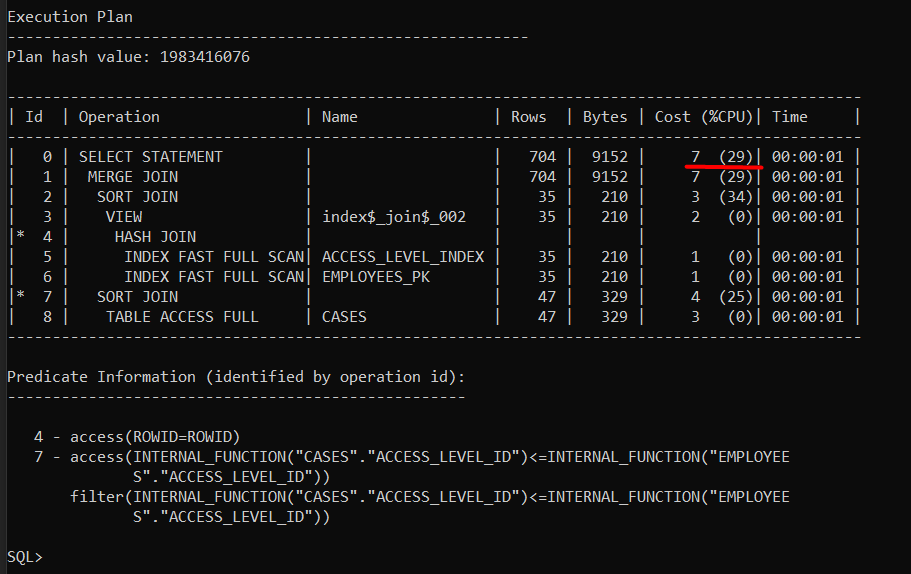


План без использования индекса по employees.access\_level\_id:



Он снова выполняет FULL FULL SCAN.

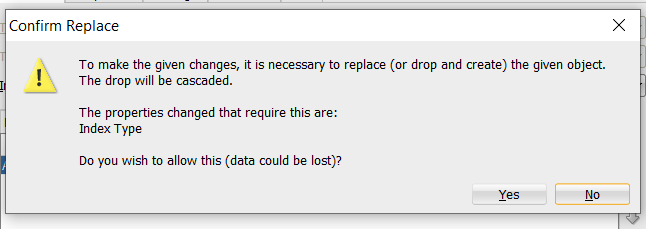
При включении индекса обратно, он использует уже INDEX FAST FULL SCAN (то же, что и INDEX FULL SCAN, но без обращения к самой таблице, т.к. нужные данные (access\_level\_id) уже есть в листе дерева):



Таким образом, прирост производительности наблюдается, но он достаточно мал (т.к. сами таблицы были размером не больше 40 строк, что достаточно мало).

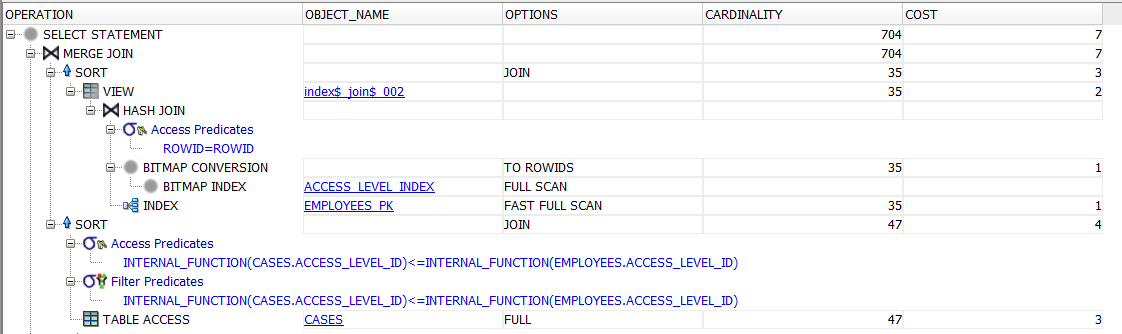
*Индексы на основе битовых карт.*

В силу специфики концепции битовых карт использовать их для навигации по первичным ключам не рационально (вместо линейного приращения размера матрицы получим квадратичное – n\*n), поэтому в данном случае ACCESS\_LEVEL\_INDEX также можно настроить на использований битовой карты:



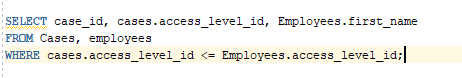
Выполним тот же запрос с соединением двух таблиц:



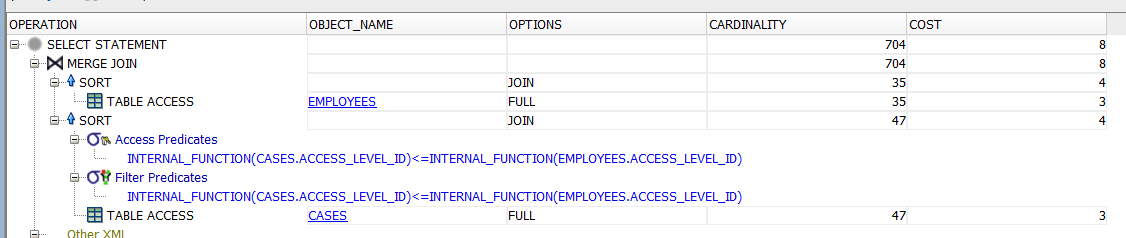


Насколько я понимаю, он точно так же, соединяет каждую строку из Cases (TABLE FULL SCAN) со строками, которые находит через BITMAP INDEX. Причем точно так же, обращения к самому блоку таблицы не идет, вся нужная информация (employee\_id и access\_level\_id) уже есть в карте.

Note: интересно, что если попытаться найти ту информацию, которой в битовой карте явно нет (т.е. если необходимо обращение к памяти):



то Oracle будет не будет использовать данный индекс, а будет делать просто полное соединение с одновременным отбором подходящих по условию строк:



(P.S. Мне чем-то эти индексы напомнили типы хранения графов в памяти: их тоже можно хранить как список или в матричном представлении. И в зависимости от вида графа тот или иной способ будет эффективнее по скорости, но проигрывает в размере. Как, впрочем, и всегда: нужно искать баланс между производительностью и требуемой памятью)

1. *Хранение таблиц по индексу/в кластере*

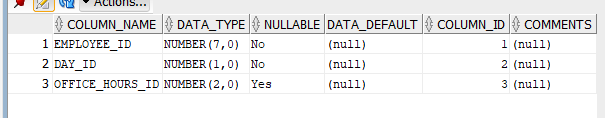
Для реализации таблицы через индекс она должна максимально удовлетворять следующим условиям:

- Редкое обновление (добавление и удаление строк);

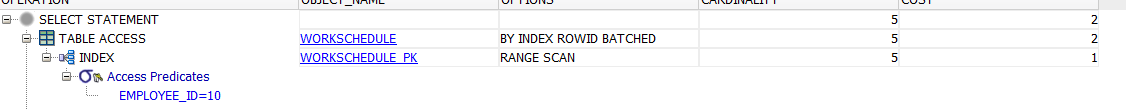
- Большой размер (кол-во строк);

- Хорошо бы чтобы еще и все данные ряда умещались в одном блоке (чтобы не выделять CHAINED BLOCK), но с этим вроде бы проблем быть не должно

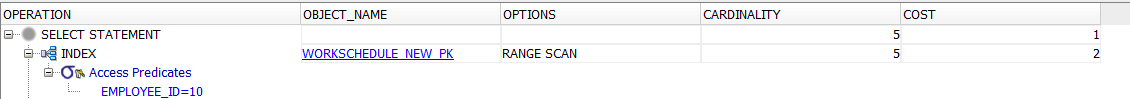
Под данное описание очень хорошо подходят таблицы измерений, которые создавались для нормализации таблицы; из всех таких таблиц наиболее большая это WorkShedule – в ней на данный момент 135 строк, с составным первичным ключом (из двух атрибутов), и изменяться она должна (официально) только тогда, когда сотрудник увольняется/нанимается.



Собственно, тогда обычный select с условием поиска по определенному сотруднику в первом случае будет использовать свой индекс для данного ключа:



А в случае с индекс-таблицей:



Т.е. таблица сама является индексом.

*Хранение таблиц в кластере.*

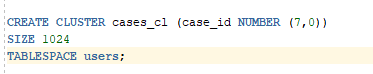
Изначально у меня была идея создать кластер из трех таблиц. Но не с одним и тем же ключом, а чтобы реализовать двойное соединение сотрудник-дело, организованное через дополнительную таблицу Assigned Cases. Но насколько я понял, так просто это не сделать. Может, и возможно создать кластер из двух кластеров (если каждое соединение организовать отдельно), но как-то мне не нравится то, что в таком случае будет происходить внутри. Поэтому обойдемся созданием одного кластера из этих двух: соединение AssignedCases с Cases.

Ключом кластера будет, как ни странно, case\_id.

Во-первых, привилегий на CREATE CLUSTER мы не выдавали, так надо выдать:



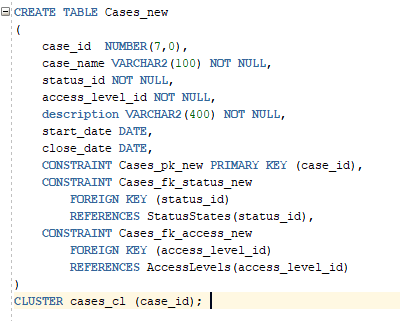
Кластер:



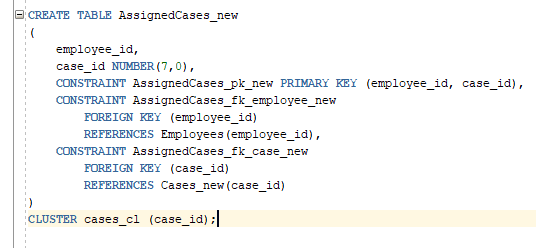


(SIZE 1024 по идее должно хватить, чтобы вместить всю информацию, с учетом полей VARCHAR2(100) и (400))

Создаем обе таблицы заново, с учетом уникальности имен индексов и связи с кластером (остальное создание взято из lab1-1):









Причем, понадобилось явно указать тип case\_id несмотря на то, что это внешний ключ; иначе Oracle не давал создать таблицу:



Экспортируем и импортируем все данные из старых таблиц в новые и выполним запрос на соединение:

……

А, или не импортируем :)

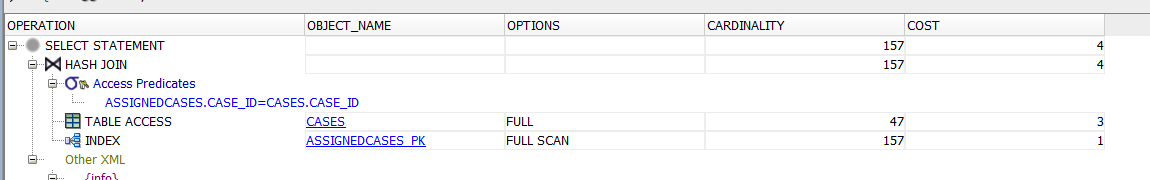


Значит, создадим (хотя непонятно, почему он автоматически не создается)

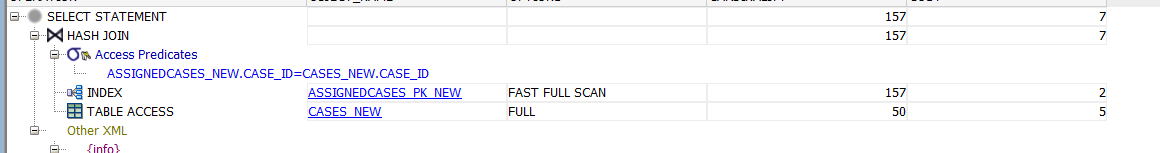




Для обычного соединения:



Для соединения кластеризированных таблиц:

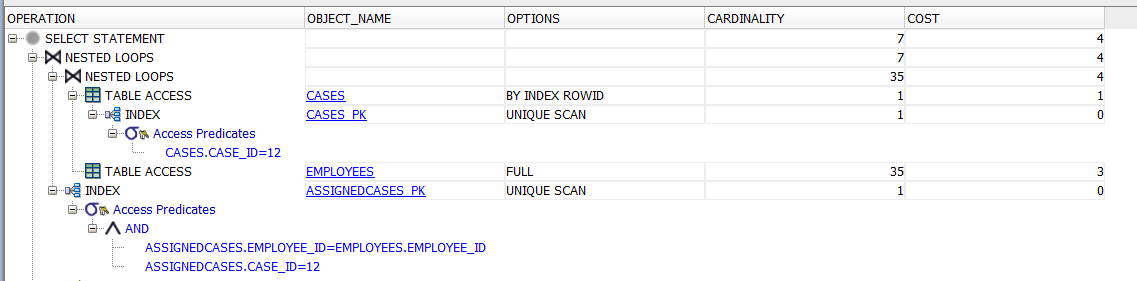


Ожидания не оправдались, оно работает медленнее…. Хотя во втором случае и идет FAST FULL SCAN засчет того, что вся информация сразу хранится в индексе.

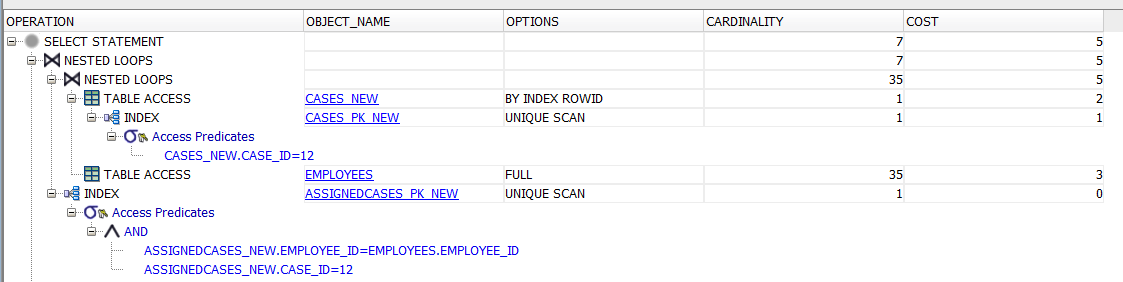
Проверено: даже если выполнить двойное соединение дополнительно с таблицей Employees, то оно все равно будет работать медленнее при работе с кластером.

Однако, если сделать выборочный запрос (с определенным id), использовав тем самым INDEX ROWID, то эффективность обоих методов одинакова, то в случае с кластеризацией выигрыш более существенен, чем при полном просмотре таблиц (в чем, как мы знаем, кластеры не могут помочь):

Не в кластере:

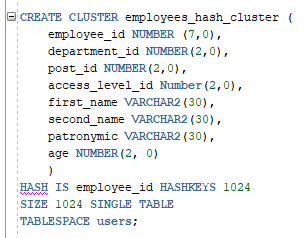


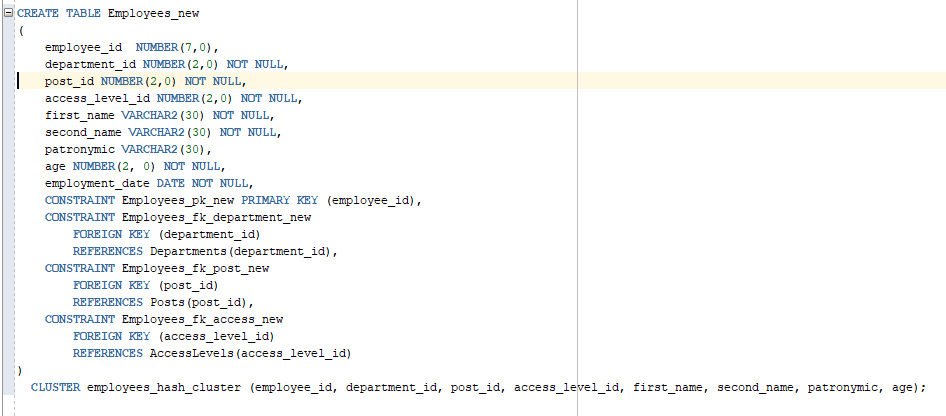
В кластере:



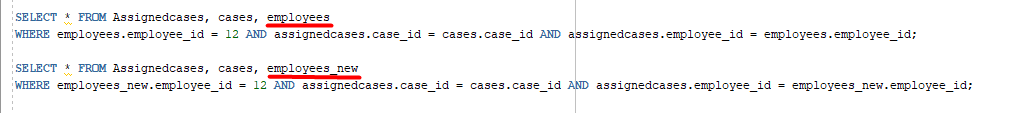
Hash-кластер:

Самым простым примером использования будет создание hash-кластера для таблицы Employees. Поставим ограничение, что в участке не могут работать более 1000 человек, что уже многовато. Создадим кластер и копию таблицы Employees:

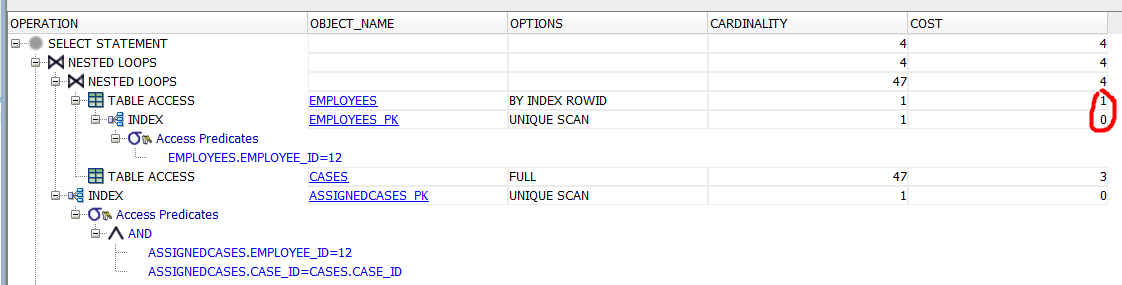


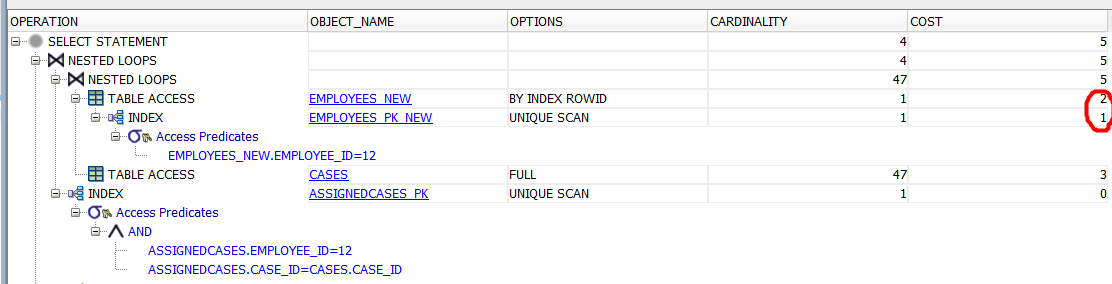


И выполним двойное соединение:



Однако вопреки моим ожиданиям, поиск подходящей по условию строки в случае с кластером так же, как и в кластеризированных таблицах, происходит медленнее, чем с обычным B-tree индексом:





Возможно, это связано со слишком малым размером таблиц, и использовать такую оптимизацию не рационально. Либо Hash-кластеризация для одиночной таблицы используется в каких-то других случаях. Хотя, вроде бы мы и должны были выиграть при операции чтения по индексу (которой у hash не будет).

# *Выводы:*

В результате данной лабораторной работы были изучены базовые механизмы работы поиска в СУБД: индексы. Рассмотрены их физическая и логическая структуры, добавлены дополнительные индексы на основе битовых карт и B-деревьев, а также проведены практические эксперименты с созданием кластеров таблиц, из которых был сделан вывод, что кластеризация в данной БД не сможет в достаточной мере ускорить ее (по крайней мере, пока она находится в исходном состоянии).

Интересное наблюдение: при создании индексов вообще на все атрибуты таблицы Cases и добавлении новой строки, СУБД действительно практически на одну секунду залагала, а сам коммит длился не как обычно 0,001 сек, а намного больше. Поэтому индексация (ровно как и кластеризация) не является панацеей, и своими действиями мы можем сделать только хуже (что я, судя по всему и сделал при создании кластеров :))