# Лабораторная работа № 10

**Oracle Database 11g: Настройка производительности - планы выполнения**

|  |
| --- |
| Настройка SQL. Необходимо включить режим ECHO и вывести протокол лаб. работы в файл *<Фамилия студента>.txt*. Этот файл является отчетом о проделанной лаб. работе.  SQL> set echo on  SQL> spool d:\spool. txt  ……………………………………..  Завершить протокол лаб. работы (команда spool off) и направить результаты преподавателю.  SQL> spool off |

**ORACLE DATABASE 11G НАСТРОЙКА ПРОИЗВОДИТЕЛЬНОСТИ.**

**Как читать планы выполнения запросов**

**Что такое план выполнения ?**

План выполнения - это вывод оптимизатора интерпретированный на языке движка выполнения. Он инструктирует движок выполнения об операциях, которые необходимо выполнить для получения данных, запрашиваемых выражением максимально быстро и эффективно.

Выражение EXPLAIN PLAN захватывает план выполнения выбранный оптимизатором для выполнения выражений типа SELECT, UPDATE, DELETE и INSERT. Шаги плана выполнения не выполняются в том порядке, в котором они указаны в плане. Между шагами существуют отношения типа родитель-потомок. дерево исходных строк  - основа плана исполнения. Оно содержит следующую информацию:

* Сортировка таблиц, на которые ссылается оператор
* Метод доступа для каждой таблицы, указанной в операторе
* Метод соединения для таблиц, применяемый операторами соединения в выражении
* Операции с данными, такие как фильтр, сортировка или  агрегирование

Дополнительно к дереву исходных строк (или дереву потока данных в параллельных операциях) таблица плана содержит следующие данные:

* Данные оптимизации, такие как стоимость и кардинальность каждой операции
* Данные секционирования, такие как набор партиций к которым выполнялся доступ
* Данные параллельного выполнения, такие как метод распределения операций соединения

Результаты выполнения EXPLAIN PLAN позволяют определить, выбирает ли оптимизатор конкретный план выполнения, например, использование вложенных циклов.

**Где найти планы выполнения?**

Существует много способов получить планы исполнения для выражений в БД, наиболее популярные перечислены ниже:

* Команда EXPLAIN PLAN позволяет вам просматривать план исполнения, который оптимизатор может использовать для выполнения выражения. Эта команда очень полезна, поскольку она строит план выполнения и записывает его в таблицу, называемую PLAN\_TABLE не сохраняя при этом SQL выражение.
* V$SQL\_PLAN предоставляет возможность просмотреть планы выполнения для курсоров, которые были недавно выполнены. Информация хранящаяся в V$SQL\_LAN очень похожа на информацию, которую выдает команда EXPLAIN PLAN. Однако Explain Plan показывает потенциальный план выполнения, а V$SQL\_PLAN хранит планы уже выполнявшихся запросов.
* V$SQL\_PLAN\_MONITOR содержит статистику мониторинга на уровне планов для каждого SQL выражения, найденного в V$SQL\_MONITOR. Каждая строка содержащаяся в  V$SQL\_PLAN\_MONITOR соответствует определенной операции плана исполнения.
* Инфраструктура AWR и Statspack хранят планы выполнения наиболее часто вызываемых SQL. Планы помещаются в представление dBA\_HIST\_SQL\_PLAN или STATS$SQL\_PLAN.
* Планы выполнения и источники строк записываются также в фалы трассировки, генерируемые DBMS\_MONITOR.
* SQL Management Base - это часть словаря данных, хранящегося в табличном пространстве SYSAUX. Здесь хранится журнальная информация об операциях, история планов выполнения и опорные линии так же как и профили для SQL выражений.
* Событие диагностики 10053 используемое для записи вычислений оптимизатора стоимости также может генерировать планы выполнения запросов.
* Начиная с версии 10.2 когда вы получаете dump состояния процесса, план выполнения также включается в сгенерированый файл трассировки.

**Просмотр планов выполнения**

Если вы выполняете команду EXPLAIN PLAN в SQL\*Plus, вы можете затем выбрать данные из таблицы PLAN\_TABLE и просмотреть сгенерированный планы выполнения. Наиболее простым способом просмотреть план выполнения является использование пакета DBMS\_XPLAIN. Пакет DBMS\_XPLAIN содержит пять доступных функций:

* DISPLAY: Используется для форматированного вывода плана выполнения.
* DISPLAY\_AWR: Используется для форматированного вывода плана выполнения SQL выражений, хранящихся в репозитории AWR.
* DISPLAY\_CURSOR: Используется для форматированного вывода плана выполнения из любого загруженного курсора
* DISPLAY\_SQL\_PLAN\_BASELINE: Используется для форматированного вывода одного и более планов выполнения SQL выражений идентифицируемых по заголовкам.
* DISPLAY\_SQLSET: Используется для форматированного вывода плана выполнения хранящегося в SQL Tuning set.

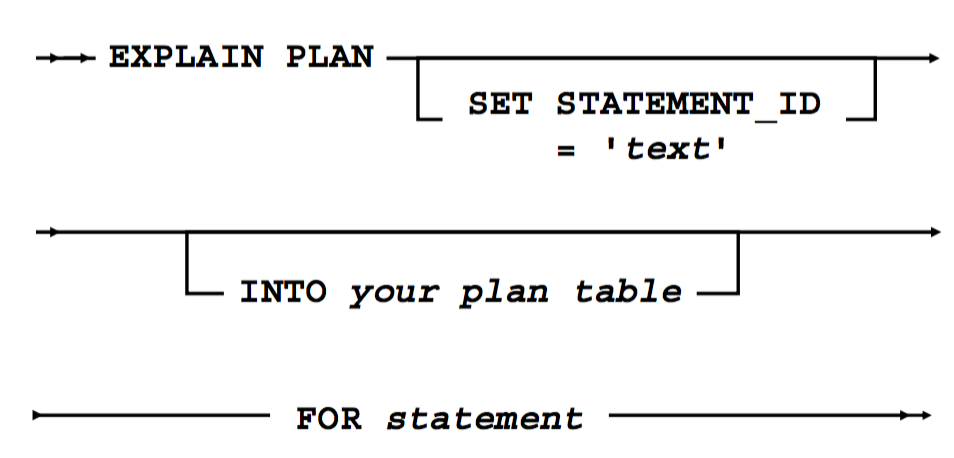
Преимущество использования пакета DBMS\_XPLAIN заключается в предоставлении возможности просмотра форматированного плана выполнения  SQL выражения в не зависимости от источника.

**Команда EXPLAIN PLAN**

* Команда EXPLAIN PLAN используется для генерации плана выполнения запроса.
* После того как план сгенерирован, его можно посмотреть, запросив информацию из таблицы PLAN\_TABLE

PLAN TABLE создается автоматически как глобальная временная таблица, используемая в последующем всеми пользователями для хранения планов выполнения. Вы можете создать собственную PLAN TABLE при помощи скрипта $ORACLE\_HOME/rdbms/admin/utlxplan.sql в случае необходимости длительного хранения планов выполнения.

**Структура команды EXPLAIN PLAN**



Команда EXPLAIN PLAN вставляет строку в PLAN TABLE для каждого шага плана выполнения.

**1. Пример выполнения команды EXPLAIN PLAN**

SQL> EXPLAIN PLAN

2 SET STATEMENT\_ID = 'demo01' FOR

3 SELECT e.last\_name, d.department\_name

4 FROM hr.employees e, hr.departments d

5 WHERE e.department\_id=d.department\_id;

Explained.

----------------Вывод содержимого PLAN TABLE

SQL> SET LINESIZE 130

SQL> SET PAGESIZE 0

SQL> select \* from table (DBMS\_XPLAN.DISPLAY());

SQL> select \* from table (DBMS\_XPLAN.DISPLAY(null,null,'ALL'));

SQL>select \* from table(dbms\_xplan.display\_cursor());

Данная команда вставляет план выполнения для выражения в PLAN TABLE и добавляет тэг demo01 для последующего обращения.

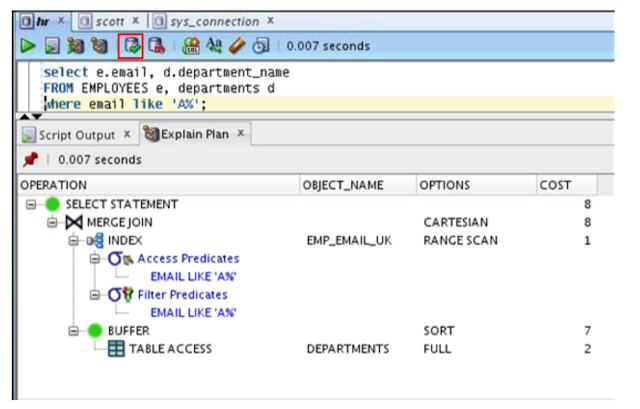
Существует множество способов получения плана выполнения. выше рассматривается метод с использованием команды EXPLAIN PLAN.  Данная команда генерирует план выполнения SQL выражения не выполняя его при этом, и помещает результат в PLAN TABLE. PLAN TABLE представляет древовидную структуру при помощи которой можно вернуть план выполнения для выражения используя колонки ID и PARENT\_ID и конструкцию CONNECT BY в выражении SELECT .

В примере приведенном выше используется ключ ALL для функции DBMS\_XPLAIN.DISPLAY, который позволяет просмотреть всю доступную информацию о плане выполнения, хранимую в PLAN TABLE. Данный вывод помимо стандартной информации, содержит дополнительную информацию такую как PROJECTION, ALIAS и информацию о REMOTE SQL, если операция распределенная.

Для более точного контроля выводимой информации могут быть использованы перечисленные ниже ключевые параметры. Каждое ключевое слово добавляет отдельный блок в вывод информации из PLAN TABLE. Ключевые слова должны отделяться друг от друга запятой или пробелом:

* ROWS если это уместно, показывает количество строк,  предположительно посчитанное оптимизатором.
* ROWS если это уместно, показывает количество байт,  предположительно посчитанное оптимизатором.
* COST если это уместно, показывает  стоимость,  предположительно посчитанную оптимизатором
* PARTITION если это уместно, показывает отброс патриций оптимизатором
* PARALLEL или это уместно, показывает информацию PX (метод распределения информации и информацию о очередях доступа к таблице)
* PREDICATE или это уместно, показывает информацию о предикате
* PROJECTION или это уместно, показывает секцию проекции

**2. Использование Explain Plan в SQL Developer**

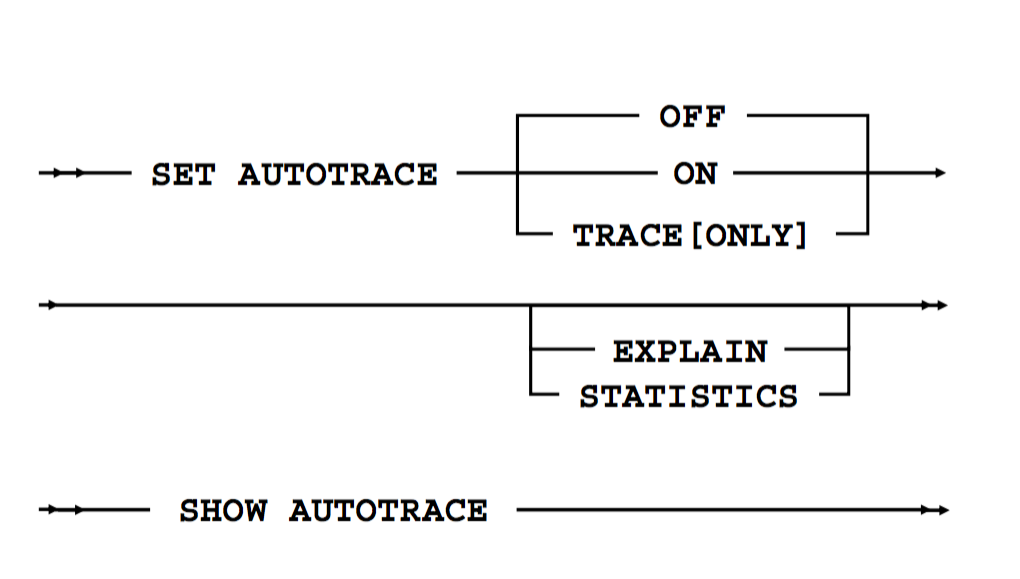


**AUTOTRACE**

Когда выражение  выполняется в SQL\*Plus или SQL Developer вы можете автоматически получать план выполнения и статистику выполнения выражения. Отчет генерируетсся автоматически после выполнения любых видов операций таких как SELECT, INSERT, UPDATE и DELETE. Данную информацию можно использовать для диагностики и настройки производительности SQL выражений.

Для использования AUTOTRACE в БД должна быть создана PLAN TABLE и пользователю, который выполняет AUTOTRACE должна быть выдана роль PLUSTRACE. Роль PLUSTRACE создается и выдается роли DBA при помощи скрипта $ORACLE\_HOME/sqlplus/admin/plustrce.sql

**Синтаксис AUTOTRACE**



Вы можете выполнять Autotrace используя синтаксис указанный на рисунке выше. Доступны для использования также следующие опции:

* OFF Отключает использование трассировки
* ON Включает использование автоматической трассировки
* TRACE [ONLY] Включает автоматическую трассировку и подавляет вывод SQL
* EXPLAIN Показывает план выполнения но не показывает сатистику
* STATISTICS Показывает статистику без плана выполнения

**3. Примеры использования AUTOTRACE:STATISTICS**

SQL> set autotrace on

SQL> select \* from hr.employees;

SQL> show autotrace

SQL> set autotrace traceonly statistics

SQL> select \* from hr.jobs;

SQL> set autotrace off

Статистика записываемая сервером при выполнении выражения отражает обьем системных ресурсов, затраченных на выполнение выражения и включает в себя следующие основные показатели:

* **recursive calls** - количество рекурсивных вызовов, сгенерированных на клиентской и серверной стороне.Oracle Database поддерживает таблицы, используемые для внутренней обработки. Когда Oracle Database необходимо внести изменения в этих таблицах, она генерирует внутренний оператор SQL, который, в свою очередь, порождает рекурсивный вызов.
* **db block gets** - количество раз, которое запрошен блок CURRENT
* **consistent gets** - количество раз, которое запрошена операция целостного чтения блоков данных.
* **physical reads**- количество блоков данных, прочитанных с диска. Это число представляет сумму значений physical reads direct и всех чтений из буферного кэша.
* **redo size**- общее количество генерированного redo в блоках
* **bytes sent via SQL\*Net to client**- общее количество байт переданных клиенту от фонового процесса.
* **bytes received via SQL\*Net from client** - общее количество байт, полученных от клиента Oracle\*Net
* **SQL\*Net roundtrips to/from client**- Общее количество сообщений Oracle NET отправленных и полученных от клиента.
* **sort (memory)**- количество операций сортировки, успешно выполненных в памяти и не потребовали записи на диск.
* **sort (disk)**- количество операций сортировки которое потребовало выполнения как минимум одной дисковой операции.
* **row processed**- количество строк, обработанных в процессе выполнения операции.

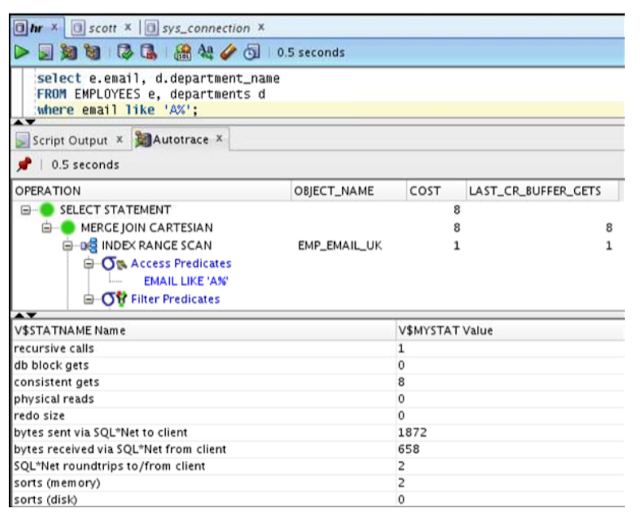
**db\_block\_gets**показывает операции чтения текущего блока из базы данных. **consistent gets** - это  операции чтения блоков которые должны удовлетворять конкретный номер SCN.  **physical reads**показывает чтение блоков с диска. **db\_block\_gets**и **consistent gets** - показатели статистики, которые постоянно мониторятся. Они должны быть низкими по сравнению с количеством извлекаемых строк. Сортировка выполняться в памяти, а не на диске.

**AUTO TRACE с использованием SQL\*Developer**

**Представление V$SQL\_PLAN**

Данное представление показывает планы выполнения для курсоров, которые все еще находятся в библиотечном кэше. Информация, хранящаяся в данном представлении во многом похожа на информацию из PLAN TABLE. Однако V$SQL\_PLAN содержит планы исполнения для выражений, которые уже были выполнены. План выполнения, полученный в процессе выполнения EXPLAIN PLAN может отличаться от фактического плана выполнения, хранящегося в курсоре. Так происходит потому, что параметры сессии и значения BIND переменных могут отличаться от текущих.

Другое полезное представление: **V$SQL\_PLAN\_STATISTICS**в котором представлена статистика выполнения для каждой операции в плане выполнения каждого кешированного курсора. Еще одно полезное представление**V$SQL\_PLAN\_STATISTIC\_ALL**объединяет в себе информацию выполнению из **V$SQL\_PLAN\_STATISTICS**и **V$SQL\_WORKAREA** c планом выполнения, хранящимся **V$SQL\_PLAN**.



**Описание основных столбцов представления V$SQL\_PLAN**

Представление содержит те же столбцы, что и PLAN TABLE и несколько дополнительных столбцов. Столбцы, представленные в PLAN TABLE и имеющие одинаковые значения:

* ADDRESS
* HASH\_VALUE

Столбцы ADDRESS и HASH\_VALUE могут быть использованы для связки с V$SQLAREA для добавления специфической информации по курсору.

PLAN\_HASH\_VALUE - это столбец, содержащий числовое представление плана SQL выражения для курсора. сравнивая значения столбца PLAN\_HASH\_VALUE можно быстро определить изменения ли план выполнения для одного и того же выражения, не сравнивая их построчно.

**Представление V$SQL\_PLAN\_STATISTICS**

Представление V$SQL\_PLAN\_STATISTICS предоставляет актуальную статистику по выполнению для каждой операции в плане выполнения, такие как количество обработанных строк или время выполнения. Вся статистика, за исключением количества строк накопленная. Например статистика по объединениям таблиц может включать в себя 3 операции объединения таблиц. Статистика, хранящаяся в V$SQL\_PLAN\_STATISTICS доступна для курсоров, которые были скомпилированы с параметром инициализации STATISTICS\_LEVEL = ALL или с использованием подсказки оптимизатору GATHER\_PLAN\_STATISTICS.

Представление V$SQL\_STATISTICS\_ALL содержит статистику использования памяти для всех исходных строк, использовавших память SQL (сортировка или HASH join) Данное представление объединяет информацию, хранимую в представлении V$SQL\_PLAN с статистикой выполнения из представлений V$SQL\_PLAN\_STATISTICS и V$SQL\_WORKAREA.

**4. ПРИМЕР**

Как правило оптимизатор запросов Oracle работает хорошо, но бывают ситуации когда он начинает чудить, использовать неоптимальный план, и некогда быстрые запросы начинают тормозить. Найдя проблемный запрос, ты правишь его хинтами, но понимаешь, что развернуть его перекомпилировав пакет на боевой среде ты не можешь. В этой ситуации на помощь может прийти SQL Plan Management в лице пакета dbms\_spm. Он позволяет зафиксировать обновленный план выполнения запроса, не меняя текста самого запроса хинтами.

Итак, у нас есть тестовая таблица с набором данных и двумя индексами:

create table t\_plan\_test (n1 number not null, n2 number not null);

create index idx\_t\_plan\_test\_1 on t\_plan\_test(n1);

create index idx\_t\_plan\_test\_2 on t\_plan\_test(n2, n1);

insert into t\_plan\_test

select rownum, rownum from dual connect by rownum <=1000;

commit;

Выполним запрос к этой таблице:

select t.\* from t\_plan\_test t where t.n1 = 100;

**Output**

100 100

Данные на месте. Проверим какой индекс используется для выборки данных:

explain plan for select t.\* from t\_plan\_test t where t.n1 = 100;

select \* from table(dbms\_xplan.display);

**Output**

Plan hash value: **2822388801**

————————————————————————————————-  
| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time |  
————————————————————————————————-  
| 0 | SELECT STATEMENT | | 1 | 26 | 1 (0)| 00:00:01 |  
| 1 | TABLE ACCESS BY INDEX ROWID| T\_PLAN\_TEST | 1 | 26 | 1 (0)| 00:00:01 |  
|\* 2 | INDEX RANGE SCAN | **IDX\_T\_PLAN\_TEST\_1** | 1 | | 1 (0)| 00:00:01 |  
————————————————————————————————-

Predicate Information (identified by operation id):  
—————————————————

2 – access(“T”.”N1″=100) Note —– – dynamic sampling used for this statement (level=2)

Вполне ожидаемо используется индекс idx\_t\_plan\_test\_1, но допустим что оптимизатор не должен был его использовать и мы хотим сделать так, чтобы в этом запросе использовался индекс idx\_t\_plan\_test\_2.  
Для начала узнаем sql\_id запроса, который мы хотим поменять. Для этого возьмем значение Plan hash value из вывода предыдущего запроса и подставим его в следующий запрос:

select s.sql\_text, s.sql\_id, s.plan\_hash\_value from v$sql s where s.plan\_hash\_value = **2822388801**;

**Output**

select t.\* from t\_plan\_test t where t.n1 = 100 **2w8k1yhuk8k1v** 2822388801  
explain plan for select t.\* from t\_plan\_test t where t.n1 = 100 2rcswv78fcp13 2822388801

**Обратите внимание на то, что нас интересует именно sql\_id оригинального запроса, а не запроса explain plan для него.**  
Запомнив sql\_id и plan\_hash\_value из запроса приступим к его оптимизации. Для начала добавим хинт на использование индекса idx\_t\_plan\_test\_2 и выполним его:

select /\*+INDEX(t idx\_t\_plan\_test\_2)\*/ t.\* from t\_plan\_test t where t.n1 = 100;

После этого проверим что запрос использовал именно этот индекс:

explain plan for select /\*+INDEX(t idx\_t\_plan\_test\_2)\*/t.\* from t\_plan\_test t where t.n1 = 100;

select \* from table(dbms\_xplan.display);

**Output**

Plan hash value: **3105628069**

————————————————————————————–  
| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time |  
————————————————————————————–  
| 0 | SELECT STATEMENT | | 1 | 26 | 5 (0)| 00:00:01 |  
|\* 1 | INDEX FULL SCAN | **IDX\_T\_PLAN\_TEST\_2** | 1 | 26 | 5 (0)| 00:00:01 |  
————————————————————————————–

Predicate Information (identified by operation id):  
—————————————————

1 – access(“T”.”N1″=100)  
filter(“T”.”N1″=100) Note —– – dynamic sampling used for this statement (level=2)

Всё верно, используется индекс idx\_t\_plan\_test\_2.  
В очередной раз взяв значение Plan hash value определим sql\_id нового запроса.

select s.sql\_text, s.sql\_id, s.plan\_hash\_value from v$sql s where s.plan\_hash\_value = **3105628069**;

**Output**

explain plan for select /\*+INDEX(t idx\_t\_plan\_test\_2)\*/t.\* from t\_plan\_test t where t.n1 = 100 4jnycbw4j9cav 3105628069  
select /\*+INDEX(t idx\_t\_plan\_test\_2)\*/ t.\* from t\_plan\_test t where t.n1 = 100 **43x83pbfc3x5n**3105628069

Теперь у нас есть все необходимые параметры для изменения и фиксации плана запроса.

**Примечание**

Следующие инструкции выполняются из под пользователя SYS. Выдайте необходимые привилегии вашему пользователю, если хотите выполнять их из под него.

Во-первых, необходимо запомнить план первого запроса, взяв его sql\_id и plan\_hash\_value:

begin

dbms\_output.put\_line(dbms\_spm.load\_plans\_from\_cursor\_cache(sql\_id => '**2w8k1yhuk8k1v**', plan\_hash\_value => **2822388801**));

end;

Во-вторых, необходимо определить sql\_handle нашего запроса и отключить этот план:

select sql\_handle, to\_char(sql\_text), plan\_name from DBA\_SQL\_PLAN\_BASELINES;

**Output**

**SQL\_cbead429bd574d05** “select t.\* from t\_plan\_test t where t.n1 = 100” SQL\_PLAN\_cruqn56ypfm8591102725

begin

dbms\_output.put\_line(dbms\_spm.alter\_sql\_plan\_baseline(sql\_handle => '**SQL\_cbead429bd574d05**',attribute\_name => 'enabled',attribute\_value => 'NO' ));

end;

В-третьих, необходимо зафиксировать план второго запроса, указав в качестве sql\_handle значение предыдущего плана и взяв sql\_id и plan\_hash\_value второго запроса:

begin

dbms\_output.put\_line(dbms\_spm.load\_plans\_from\_cursor\_cache(sql\_id => '**43x83pbfc3x5n**', plan\_hash\_value => **3105628069**, fixed => 'YES', sql\_handle => '**SQL\_cbead429bd574d05**'));

end;

Готово. Теперь, если посмотрите план выполнения первого запроса, увидите что используется индекс idx\_t\_plan\_test\_2 без хинта.

explain plan for select t.\* from t\_plan\_test t where t.n1 = 100;

select \* from table(dbms\_xplan.display);

**Output**

Plan hash value: **3105628069**

————————————————————————————–  
| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time |  
————————————————————————————–  
| 0 | SELECT STATEMENT | | 4 | 104 | 0 (0)| 00:00:01 |  
|\* 1 | INDEX FULL SCAN | **IDX\_T\_PLAN\_TEST\_2** | 4 | 104 | 0 (0)| 00:00:01 |  
————————————————————————————–

Predicate Information (identified by operation id):  
—————————————————

1 – access(“T”.”N1″=100)  
filter(“T”.”N1″=100)

Исполнение любого SQL предложения в Oracle извлекает так называемый "план исполнения". Этот план исполнения запроса является описанием того, как Oracle будет осуществялть выборку данных, согласно исполняемому SQL предложению. План представляет собой дерево, которое содержит порядок шагов и связь между ними.

Базовые правила плана исполнения:

1. План исполнения имеет корень, ветвь не имеющую родителя
2. Родительские ветки могут иметь одного и более потомков, и их идентификатор меньше чем идентификатор потомка
3. У потомка может быть только один родитель, это справедливо и для нескольких уровней вложенности

Пример плана исполнения:

SQL> EXPLAIN PLAN **FOR**

2 **SELECT** E.EMPNO, E.ENAME, D.DNAME **FROM** EMP E, DEPT D **WHERE** E.DEPTNO = D.DEPTNO **AND** E.DEPTNO = 10;

Explained.

SQL> **SELECT** \* **FROM** TABLE(dbms\_xplan.display(**null**,null,'basic'));

PLAN\_TABLE\_OUTPUT------------------------------------------------  
Plan hash value: 568005898------------------------------------------------  
/ Id  / Operation                    / Name    /------------------------------------------------  
/   0 / SELECT STATEMENT             /         /  
/   1 /  NESTED LOOPS                /         /  
/   2 /   TABLE ACCESS BY INDEX ROWID/ DEPT    /  
/   3 /    INDEX UNIQUE SCAN         / PK\_DEPT /  
/   4 /   TABLE ACCESS FULL          / EMP     /------------------------------------------------

Если следовать перечисленным выше правилам, то:

* Операция 0 - корень дерева, и у нее один потомок, операция 1
* У операции 1 два потомка - операции 2 и 4
* У операции 2 один потомок - операция 3

Если прочитать это дерево, то получим следующее, в порядке выполнения: Для выполнения операции 1, необходимо выполнить операции 2 и 4. Операция 2 выполняется первой. Во время выполнения операции 2, необходимо выполнить операцию 3. Для выполнения операции 4, необходимо выполнить операцию 2.

* Операция 3 получает доступ к таблице DEPT, используя INDEX UNIQUE SCAN и предоставляет ROWID операции 2
* Операция 2 возвращает все строки из таблицы DEPT для операции 1
* Операция 1 выполняет операцию 4, для каждой строки, которую вернула операция 2
* Операция 4 выполняет полностью просматривает таблицу (TABLE ACCESS FULL) и применяет фильтр E.DEPNO=10 и возвращает строки для операции 1
* Операция 1 возвращает финальный результат в операцию 0