|  |
| --- |
| EPAM Systems, RD Dep. |
| MTN.BI.07 Oracle Join Methods |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| REVISION HISTORY | | | | | |
| Ver. | Description of Change | Author | Date | Approved | |
| Name | Effective Date |
| 1.0 | Initial status | Anastasiya Khilko | 06-NOV-2017 |  |  |

Contents

[1. Join Methods 3](#_Toc443491697)

[1.1. Task 2: Nested Loop Join 3](#_Toc443491698)

[1.2. Task 3: Sort-Merge Join 3](#_Toc443491699)

[1.3. Task 4: Hash Join 3](#_Toc443491700)

[1.4. Task 5: Cartesian Join 3](#_Toc443491701)

[1.5. Task 6: Left/Right Outer Joins 3](#_Toc443491702)

[1.6. Task 7: Full Outer Join 3](#_Toc443491703)

[1.7. Task 8: Results 3](#_Toc443491704)

# Join Methods

The main task is to try to get different types of different join methods. You can create your own needed objects, or use existing (in sample schemas). If you are creating tables or indexes, please, attach a script.

## Task 2: Nested Loop Join

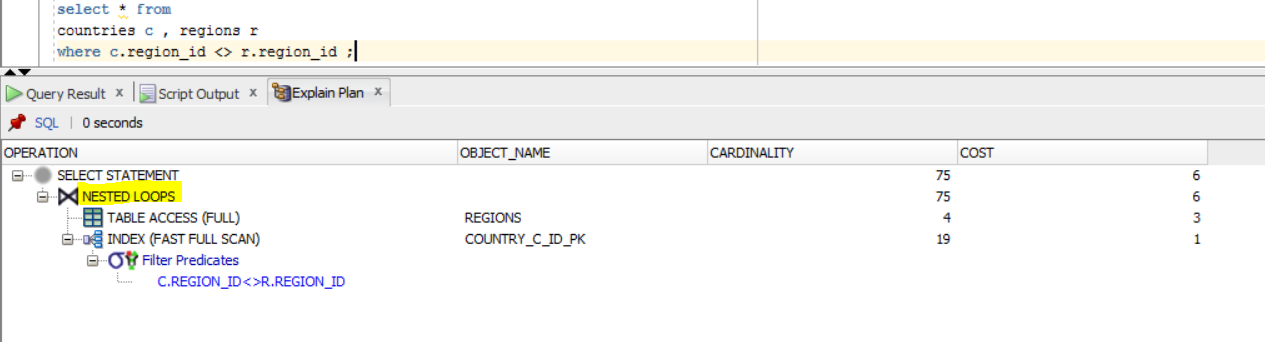
Метод соединения данных вложенными циклами. Сначала сканируется источник 1, затем для каждой найденной записи источника 1 в цикле ищется удовлетворяющий условию список строк из источника 2, затем организуется цикл по источнику 3 и т. д. В случае **наличия индексов** осуществляется **индексный поиск данных** в источниках по заданным условиям, **иначе** выполняется **полное сканирование** таблиц-источников. В случае отсутствия индексов время выполнения соединения может оказаться неприемлемо высоким.

Так как все строки должны подвергнуться сравнению, это означает, что скорость работы такого алгоритма будет зависеть от размера соединяемых таблиц. Чтобы уменьшить стоимость такого запроса, можно ограничить выборку основной таблицы, строки которой будут сравниваться со строками из driving tale.

select \* from

countries c , regions r

where c.region\_id <> r.region\_id ;



В данном примере объединяются таблицы с маленьким количеством данных. **Nested Loop** поддерживает все предикаты соединения, включая предикаты эквивалентности и предикаты неравенства.

Только на таблице Regions существует индекс по полю region\_id. Работа запроса начнется с полного просмотра таблицы Regions. И так как region\_id включен в предложение where, то метод доступа fast full scan.

## Task 3: Sort-Merge Join +

**Note:** You can instruct optimized to use sort-merge join with the hint: USE\_MERGE.

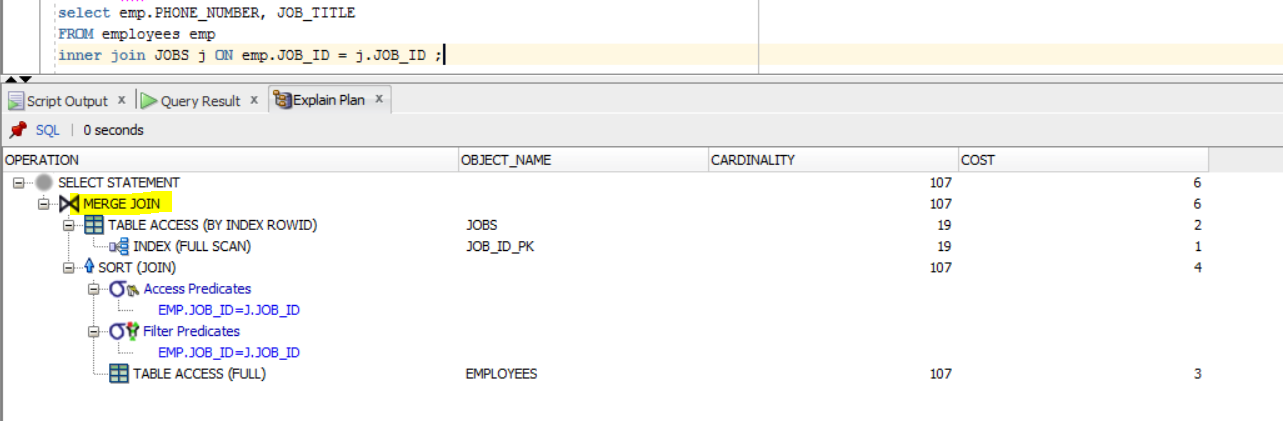
Выполняется соединение данных нескольких источников при помощи предварительной сортировки данных исходных таблиц. Часто очень неэффективный алгоритм из-за потенциально больших накладных расходов на сортировку данных перед выполнением соединения.

Стоимость такого запроса будет зависеть от суммы строк двух таблиц, полученных на входе. При этом таблицы будут просматриваться один раз.

select emp.PHONE\_NUMBER, JOB\_TITLE

FROM employees emp

left join JOBS j ON emp.JOB\_ID = j.JOB\_ID ;



В нашем примере объем данных относительно мал. Данное соединение относится к экви-соединению, что является одним из условий срабатывания merge join. По полю job\_id существует индекс в таблице Jobs, поэтому доступ производится на основе индекса и сортировка не нужна, так как индекс уже изначально упорядочен. Данные из второй таблицы будут сортироваться. И Только потом выполнится merge join.

## Task 4: Hash Join +

**Note:** You can instruct optimized to use hash join with the hint: USE\_HASH.

Как правило, хэш-соединение используется при наличии тяжелых запросов или же при необходимости соединения больших наборов данных.

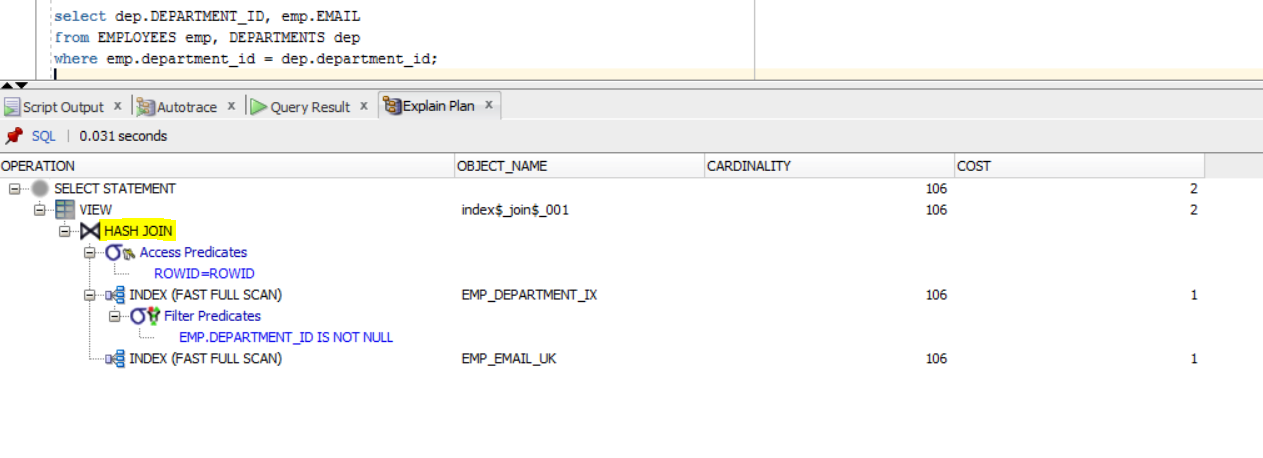
Сначала осуществляется чтение всех строк первой таблицы, эти строки хэшируются по ключам соединения и в памяти создается хэш-таблица. Второй этап алгоритма: чтение всех строк второй таблицы, эти строки хэшируются по тем же ключам соединения , затем происходит поиск соответствий строк в хэш-таблице.

Хэш-соединение блокируют вывод до окончания компоновки. То есть вначале должны быть полностью прочитан и обработан поток компоновки, и только потом могут быть возвращены какие-либо строки.

select dep.DEPARTMENT\_ID, emp.EMAIL

from EMPLOYEES emp, DEPARTMENTS dep

where emp.department\_id = dep.department\_id;



Хэш-соединение имеет много общего с соединением слиянием. В данном примере то же экви-соединение. Метод доступа – индексный(на обеих таблицах индекс по полю department\_id) и эти же поля включены в предложение where , что приводит к fast full scan.

## Task 5: Cartesian Join

Implement Cartesian join with:

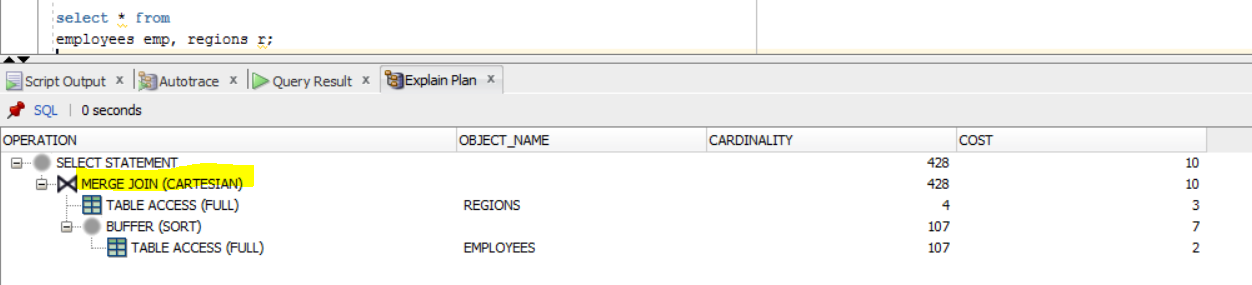
* ORDERED hint and wrong predicate order.
* W/o join conditions.

Декартово произведение возникает обычно в том случае, если для нескольких источников не указаны никакие условия соединения. В итоге каждая строка из источника 1 соединяется со всеми строками источника 2, затем то же самое для каждой строки источника 2 по отношению к источнику 3 и так далее. Декартово произведение способно генерировать огромное количество строк. Однако иногда декартово произведение может применяться намеренно.

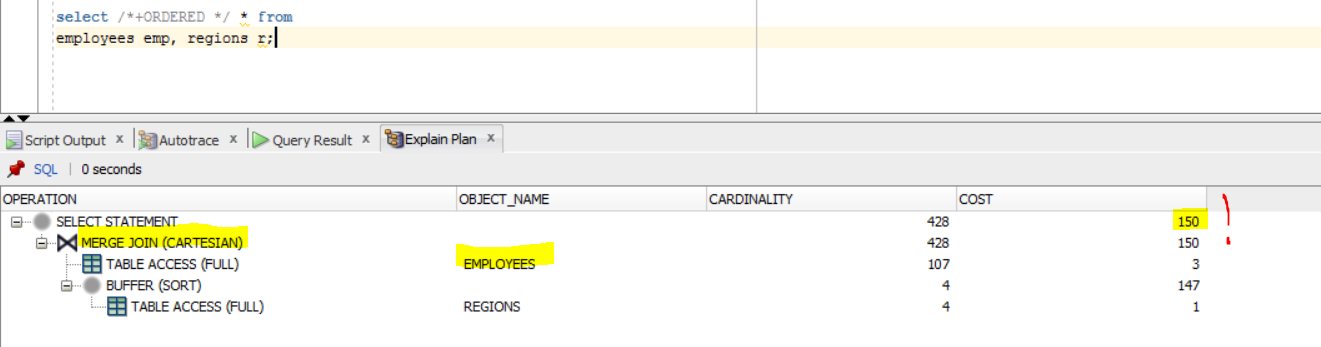
Хинт **ordered** применяется для доступа к таблицам в предложении FROM в указанном порядке. На плане выполнения данного запроса видно, что первой просматривается меньшая таблица Regions, а затем большая таблица – Employees.

select \* from

employees emp, regions r;



Применив хинт мы изменили порядок следования таблиц, что привело к резкому увеличению стоимости запроса.



## Task 6: Left/Right Outer Joins

Left outer join построен таким образом, чтобы в результат вошла каждая строка таблицы, даже если для этой строки не имеется соответствия во второй таблице. Не найденные соответствия заполняются NULL.

Для Right outer join выполняется тот же принцип, только для таблицы , которая находится справа от оператора join.

select emp.PHONE\_NUMBER, JOB\_TITLE

FROM employees emp

left join JOBS j ON emp.JOB\_ID = j.JOB\_ID ;

select emp.PHONE\_NUMBER, JOB\_TITLE

FROM employees emp, JOBS j

where emp.JOB\_ID(+) = j.JOB\_ID;

Результат этих двух запросов будет одинаковым – 107 строк.

## Task 7: Full Outer Join

select emp.PHONE\_NUMBER, JOB\_TITLE

FROM employees emp

full join JOBS j ON emp.JOB\_ID = j.JOB\_ID ;

select emp.PHONE\_NUMBER, JOB\_TITLE

FROM employees emp, JOBS j

where emp.JOB\_ID(+) = j.JOB\_ID

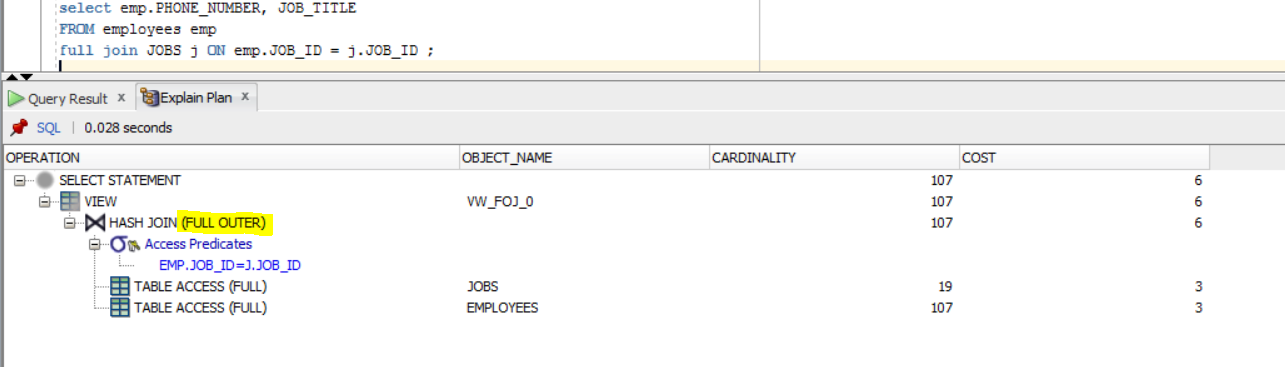
union all

select emp.PHONE\_NUMBER, JOB\_TITLE

FROM employees emp, JOBS j

where emp.JOB\_ID = j.JOB\_ID(+);

Данный тип соединения означает объединение двух типов соединений: **left outer join** и **right outer join**. Отображает все возможные комбинации строк из соединяемых таблиц.



## Task 8: Results

* Screenshots, SQLs, descriptions for Tasks 2-7.
* Make a list of combination of different tables (see examples below) and write a description for the join between them. Express your opinion on why the selected join type was chosen by an optimizer. Create at least 10 variants.

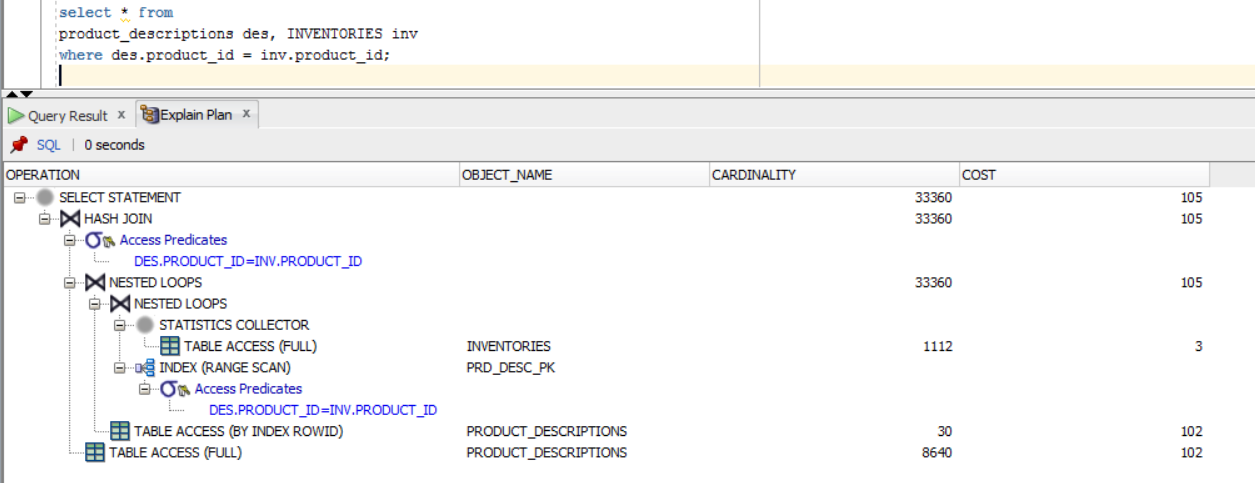
|  |  |  |  |
| --- | --- | --- | --- |
|  | Table “A” | Table “B” | Join type description |
| 1 | Большая таблица с индексом по ключу соединения | Средняя таблица с индексом по ключу соединения | Метод соединения – Hash Join. Данные в обоих полях упорядочены за счет наличия индексов, плюс экви-соединение, так что Hash Join наилучший вариант для такого типа запроса. |
| 2 | Маленькая таблица | Маленькая таблица | Условие соединения отсутствует. Наличие индексов или отсутствие не играет никакой роли. Всегда будет Merge Join Cartesian. |
| 3 | Маленькая таблица с индексом по ключу соединения | Маленькая таблица с индексом по ключу соединения | Маленький размер таблиц, упорядочивание по индексу, ограничение выборки предложением where в сумме дают Nested Loop. Наличие индексируемого поля в условии = доступ по row\_id |
| 4 | Большая таблица без индекса по ключу соединения | Средняя таблица без индекса по ключу соединения | Условие соединения – екви, в предложении where дополнительное условие отбора where t1.t\_pad > t2.t\_pad, что сильно нагружает запрос, стоимость запроса неприлично большая. Снова Hash join, все большие нагрузки валятся на него. При этом ставит таблицу t2 на первое место, так как она меньше. |
| 5 | Большая таблица без индекса по ключу соединения | Маленькая таблица без индекса по ключу соединения | Обычное экви-соединение. Размеры таблиц заметно отличаются. В результатет соединения получается right outer , так как таблица t2 меньше чем t1 и поэтому ставится на первое место. Merge join не особо прокатит, так как явная сортировка только навредит. Поэтому Hash Join. |
| 6 | Средняя таблица | Среднаяя таблица | Нет индекса. И хотя одним из основных условий Merge Join является наличие екви-соединения, данный пример все равно попадает под действие данного метода. Плюс второе условие ограничивает выборку, что снижает стоимость запроса. |
| 7 | Средняя таблица с индексом по ключу соединения | Большая таблица без индекса по ключу соединения | Или же наоборот. Средняя таблица будет без индексированного ключа,а большая - с . По мере повторения экспериментов с соразмерностью таблиц, было замечено, что предпочтение отдается Hash Join в тех случаях, если имеется большая разница между размерами таблиц. |
| 8 | Средняя таблица с индексом по ключу соединения | Средняя таблица с индексом по ключу соединения | Соединение происходит по ключу. При этом размеры таблиц примерно одинаковые.Может накладываться ограничение на выборку. Но здесь уже эффективнее использовать Merge join, чем Nested Loop |
| 9 | Маленькая таблица без индексированного ключа | Средняя таблица с индексированным ключом | В зависимости от условия соединения может выбираться Nested Loop (например, если поставить условие >), при >= , <= будет работать Merge Join, как более чувствительный к условию равенства и сортировкам(наличие индекса). При обычном екви будет работать Hash Join(скорее всего, для того чтобы работа запроса была максимально быстрой, хотя и другие методы справились бы хорошо в данном случае) |
| 10 | Маленькая таблица с индексированным ключом | Средняя таблица с индексированным ключом | Nested Loop хорош тем, что работает с различными предикатами соединений. Индекс дает индексный доступ, что в целом не тормозит работу запроса. И плюс ограничение выборки. |

1. .

select \* from

product\_descriptions des, INVENTORIES inv

where des.product\_id = inv.product\_id;

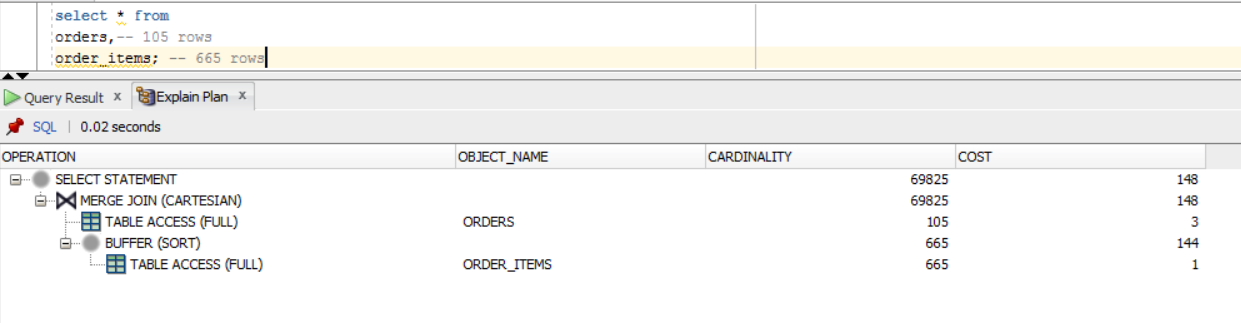


1. .

select \* from

orders, -- 105 rows

order\_items; -- 665 rows

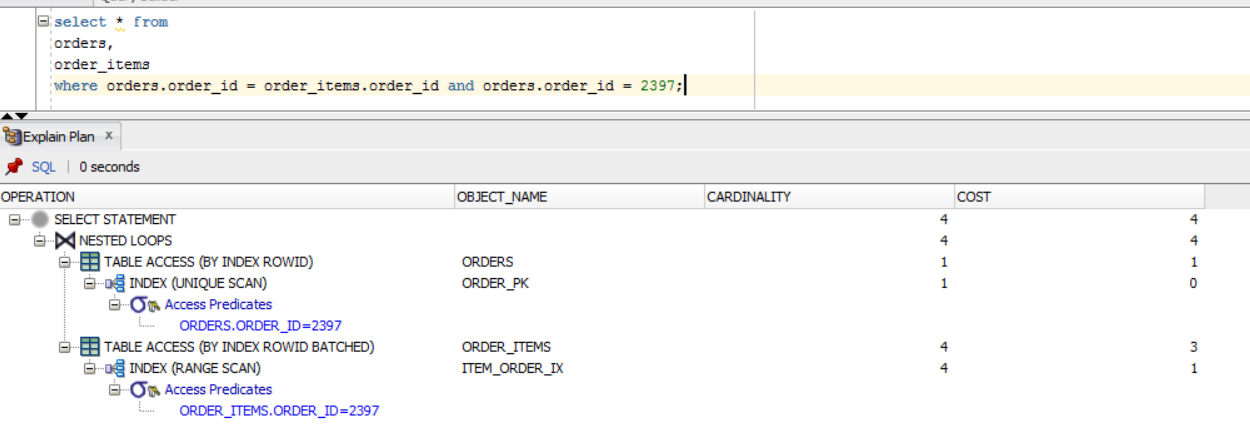


select \* from

orders,

order\_items

where orders.order\_id = order\_items.order\_id and orders.order\_id = 2397;



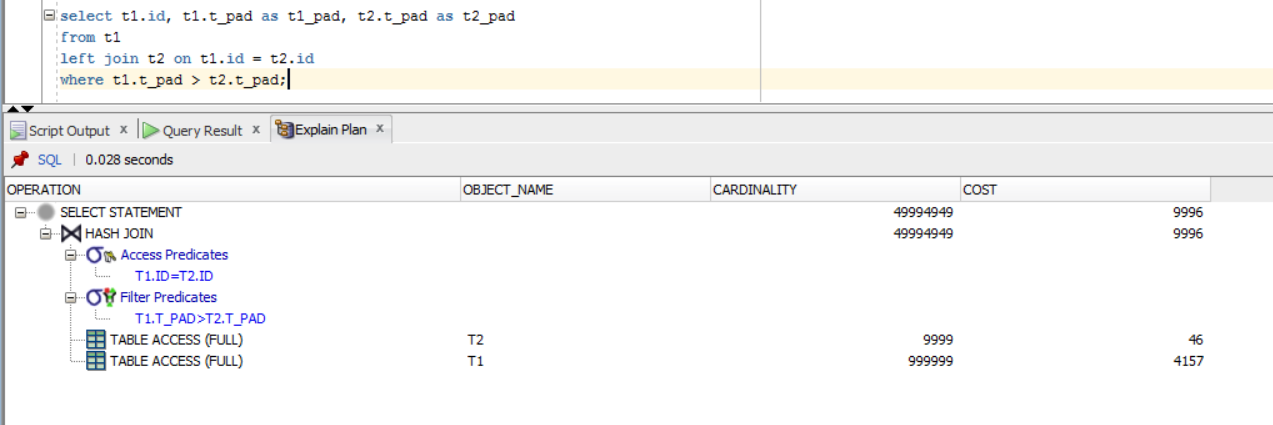
1. .

select t1.id, t1.t\_pad as t1\_pad, t2.t\_pad as t2\_pad

from t1

left join t2 on t1.id = t2.id

where t1.t\_pad > t2.t\_pad;

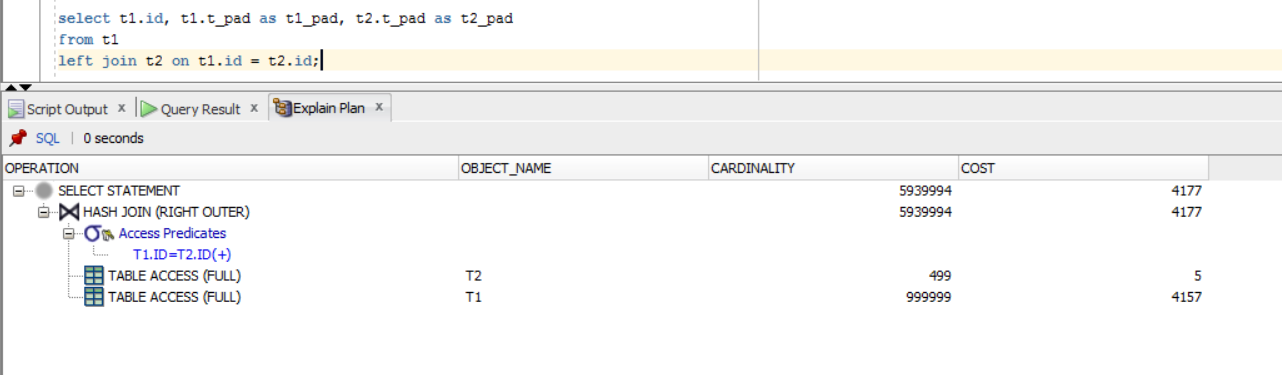


1. .

select t1.id, t1.t\_pad as t1\_pad, t2.t\_pad as t2\_pad

from t1

left join t2 on t1.id = t2.id;

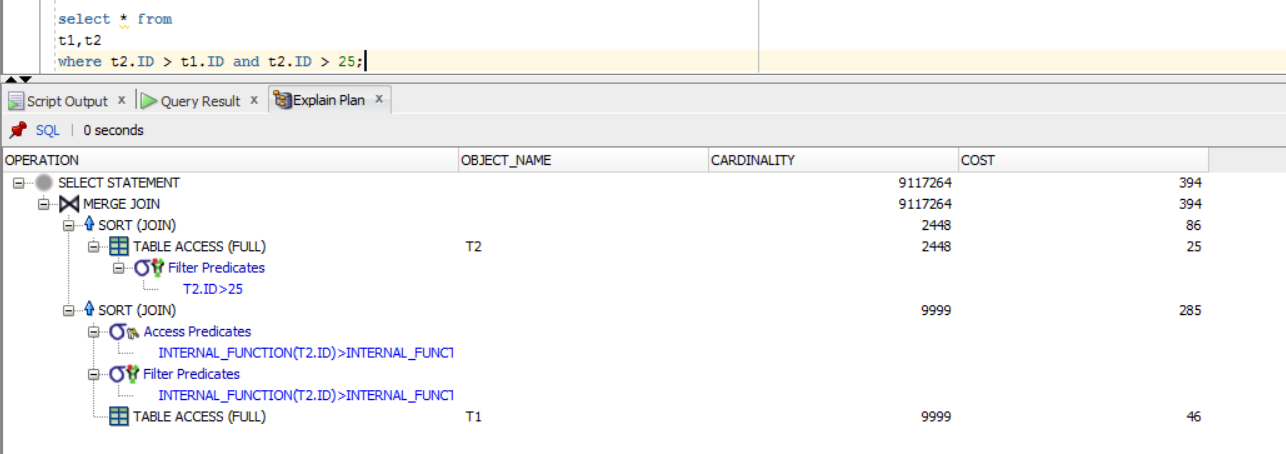


1. .

select \* from

t1,t2

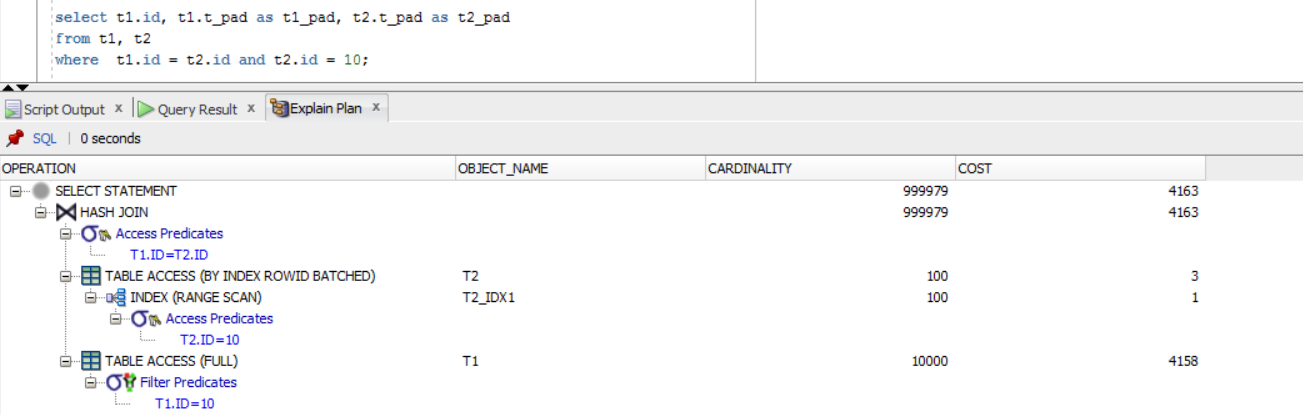
where t2.ID > t1.ID and t2.ID > 25;



select t1.id, t1.t\_pad as t1\_pad, t2.t\_pad as t2\_pad

from t1, t2

where t1.id = t2.id and t2.id = 10;



10) .

select t1.id, t1.t\_pad as t1\_pad, t2.t\_pad as t2\_pad

from t1, t2

where t1.id <> t2.id and t2.id = 10;

