|  |  |
| --- | --- |
|  | **Rochester Institute of Technology**  Due: \_\_\_\_ / \_\_\_\_ / \_\_\_\_  **Golisano College of Computing and Information Sciences**  **Department of Information Sciences & Technologies** |

Lab 4

By Rahul Sakore

Investigate Query Execution Plans Basics

Overview

The execution plan for a query provides detailed information about how a specific query should be executed. In the best of all possible worlds, database users will properly write their SQL code to avoid unnecessary execution costs. However, in the real world there will always be poorly written SQL. In addition, for various reasons, the plan produced for a given query by a DBMS’s query optimizer may or may not be optimal. Therefore, many DBMSs provide a way to view – or externalize – query execution plans so that DBAs can evaluate and tune query performance.

**At the end of this exercise you should be:**

* Able to retrieve and display the plan for a given SQL query using several DBMS-provided tools, in this case tools based on the Oracle statement EXPLAIN PLAN.
* Able to compare some of the options for plan externalization for a specific DBMS – i.e. Oracle.
* Able to read the plan for a query and explain the access steps used for the execution of a query.

To complete this exercise you will need access to your IPEDS database on your IST Database server from a machine in one of the IST database labs.

**Deliverables:**

1) This completed Lab #4 document

2) Lab #4 Report

Note: As you do this lab, it will be helpful to keep a record of what happens in each activity. Your instructor(s) encourage you to keep a running copy/paste log or a set of images with the results of your investigations – both for yourself and to include with your lab submission.

**Activity #4-0 – *Connect to Your IPEDS Database***

At one of the Database Lab computers, startup SQL\*Plus and connect to your IPEDS database as the IPEDS DBA, instDBA. SQL\*Plus will provide a good interface for your investigations; however, you can use the SQL client, SQL Developer, if you prefer.

**Activity #4-1 – *Investigate the Query Plan Display Process***

As discussed in lecture, you can request that a query’s execution plan be displayed using a DBMS’s plan-display utility. Oracle provides EXPLAIN PLAN, which is an Oracle SQL statement that will “externalize” – i.e. retrieve and store – the execution plan (access path steps) chosen for a query by the query optimizer into a special plan-storage table that is accessible to users. This can be requested as follows:

**EXPLAIN PLAN SET statement\_id = ’<*identifier string here*>’ FOR**

**<*some SELECT | INSERT | UPDATE | DELETE statement here*>;**

A specific execution plan can be identified by specifying an identifier string with the SET clause. A DBA can then review the plan for possible database and/or SQL tuning. **Check out the documentation available for EXPLAIN PLAN on Oracle’s TechNet website.**

* In Oracle, where does EXPLAIN PLAN store execution plans?

EXPLAIN PLAN store execution plans in PLAN\_TABLE.

PLAN\_TABLE is the default sample output table into which the EXPLAIN PLAN statement inserts rows describing execution plans.

* How does a (i.e. *any*) plan produced by EXPLAIN PLAN relate to the *actual* plan used to execute a query? Explain.

Any plan produced by EXPLAIN PLAN relate to the *actual* plan used to execute a query by same Schema and Cost.

* Does Oracle provide any views for query execution plans? Answer “yes” or “no” and explain.

Yes, V$SQL\_PLAN views to display the execution plan of a SQL statement. The advantage of V$SQL\_PLAN over EXPLAIN PLAN is that you do not need to know the compilation environment that was used to execute a particular statement

**Activity #4-2 – *Investigate the Plan-Storage Table***

Your IPEDS DBA, instDBA, will want to be able to access query plans. So, let’s investigate.

* Does instDBA have this special plan-storage table in his/her schema? How did you find out?

**No**

!! Don’t give up yet !!!.Try describing the EXPLAIN PLAN plan-storage table.

* What happens? Explain your results.

When I describe the plan table, there is a column named: object\_owner which tells if the current object owner have plan table or not. In this case there were no rows returned.

* Who “owns” this plan-storage table?

Since oracle 9, by default plan-storage table is owned by sys.

Finally, before looking at a query execution plan, let’s investigate the plan-storage table itself.

* How many columns does this table have? \_\_36\_\_\_\_\_

**Table 1** lists some of the more important attributes from the EXPLAIN PLAN plan-storage table from our perspective. Look them up in the Oracle documentation and *clearly* explain (in YOUR OWN words and include a reference) what information each attribute stores about a query plan.

**Table 1.** *Some* *Important Columns from the Plan-Storage Table*

|  |  |
| --- | --- |
| **Column Name** | **Description** |
| STATEMENT\_ID | Value of the optional STATEMENT\_ID parameter specified in the EXPLAIN PLAN statement. |
| OPERATION | Name of the internal operation performed in this step. In the first row generated for a statement, the column contains one of the following values:   * DELETE STATEMENT * INSERT STATEMENT * SELECT STATEMENT * UPDATE STATEMENT |
| OPTIONS | A variation on the operation described in the OPERATION column. |
| OBJECT\_NAME | Name of the table or index. |
| OBJECT\_TYPE | Modifier that provides descriptive information about the object; for example, NON-UNIQUE for indexes. |
| OPTIMIZER | Current mode of the optimizer. |
| ID | A number assigned to each step in the execution plan. |
| PARENT\_ID | The ID of the next execution step that operates on the output of the ID step. |
| COST | Cost of the operation as estimated by the optimizer's query approach. Cost is not determined for table access operations. The value of this column does not have any particular unit of measurement; it is merely a weighted value used to compare costs of execution plans. The value of this column is a function of the CPU\_COST and IO\_COST columns. |
| CPU\_COST | CPU cost of the operation as estimated by the query optimizer's approach. The value of this column is proportional to the number of system cycles required for the operation. For statements that use the rule-based approach, this column is null. |
| IO\_COST | I/O cost of the operation as estimated by the query optimizer's approach. The value of this column is proportional to the number of data blocks read by the operation. For statements that use the rule-based approach, this column is null. |
| ACCESS\_PREDICATES | Predicates used to locate rows in an access structure. For example, start or stop predicates for an index range scan. |

**Activity #4-3 – *Investigate a Simple Query Execution Plan***

Let’s start with a very simple query that accesses a single table in your database:

SELECT \*

FROM DEGREE\_GRANT

WHERE degGrantID IN (2, 3);

As discussed in lecture, you can request that an execution plan be externalized for this query as follows (use the plan identifier label shown):

**EXPLAIN PLAN SET statement\_id = 'Lab4Plan' FOR**

SELECT \*

FROM DEGREE\_GRANT

WHERE degGrantID IN (2, 3);

Run the plan-externalization request query.[[1]](#footnote-1)

* What happens?

Plan is explained.

In the above query, oracle database internally creates a query execution plan in order to fetch desired data from tables. Instead of accessing full table, database is only reading rows based on degGrantID therefore less cost to execute the plan. It is range based query where is rows of degGrantID is fetched of either 2 or 3.

Let’s see what your query execution plan looks like! Start with a brute force display of the execution plan, as stored in your plan-storage table.[[2]](#footnote-2) Run the following (assuming that your plan ID is ‘**Lab4Plan**’):

SELECT \*

FROM PLAN\_TABLE

**WHERE STATEMENT\_ID = 'Lab4Plan';** 🡨 using the previous identifier

What do you see?

* How meaningful is this output format? Clear? Helpful? \_\_\_ YES \_\_\_ **NO** Explain.

The output is not well formatted, I tried to change pagesize and linesize but it was not presentable.

It is very *important* to have plan information displayed in a meaningful format. So try something a little more informative. There are several ways to display plans from the plan-storage table. Some are better than others. So, let’s try using an Oracle-recommended query approach (**Figure 1**) to retrieve your plan.[[3]](#footnote-3) While this is only one possible way to display a plan, it shows some things that are key to interpreting the steps in a text-based query plan. First, convince yourself what this query does and then run it. Make sure to save the results to answer questions in report. You may want to read report questions as well before starting.

col statement\_id format a10;

col operation format a20;

col object\_name format a15;

col options format a15;

set linesize 100;

select /\* + GATHER\_PALN\_STATISTICS \*/statement\_id "PLAN LABEL", id "ROW ID", parent\_id,

lpad(' ',level-1) || operation "OPERATION",

object\_name, options

from plan\_table

connect by prior id = parent\_id

and prior statement\_id = statement\_id

start with id = 0

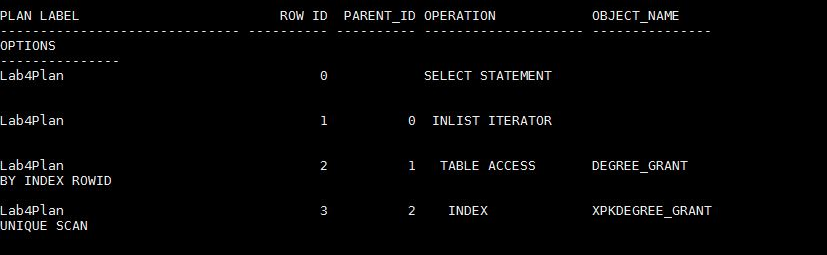
and statement\_id = '**Lab4Plan**'

order by id;

**Figure 1.** *A “Standard” Oracle-Type Plan Retrieval Query*

Document (i.e. paste) your query results here. Be sure it is **formatted for readability**.

(paste)



* What does *each row* in the plan-storage table with your label represent?
* Each step of the execution plan returns a set of rows that either is used by the next step or, in the last step, is returned to the user or application issuing the SQL statement. A set of rows returned by a step is called a row set.
* The numbering of the step Ids reflects the order in which they are displayed in response to the EXPLAIN PLAN statement. Each step of the execution plan either retrieves rows from the database or accepts rows from one or more row sources as input.
* Can you see any “relationship” between the data in the rows in the plan table? Explain.

Yes, based on row id and parent id, there is relationship between the data in the rows. For example Rowid = 0 is the root node or parent element.

Obviously the key to interpreting a query execution plan is understanding *how* a plan will be executed. So, based upon what you’re seeing …

* What tells you *how* a plan – as stored in the plan-storage table – will be executed?

There are pacakages for SQL/PLSQL to display the output from plan-storage table, for example

DBMS\_XPLAN.DISPLAY

Following SQL scripts or PL/SQL package provided by Oracle tells how a plan will be executed.

UTLXPLS.SQL

Fill in **Table 2** to show the *order* in which the steps in your plan will be executed. Explain what each step does *in detail*; and specify the name of the database object(s) that are *accessed* in each step.

**Table 2.** *Interpretation of Plan Step Execution* (add/remove rows as needed)

|  |  |  |  |
| --- | --- | --- | --- |
| **Execution Order** | **ID** | **Operation & Options Explanation** | **Object Accessed** |
| 1st | 2 | Which access all the rows of table | Degree\_grant |
| 2nd | 3 | Select unique value based on index | XPKDEGREE\_GRANT |
| 3rd | 1 | Filter using IN criteria |  |
| 4th | 0 | Displays the resultset |  |

**Activity # 4-4 – *Using a Traditional Plan-Display Utility***

EXPLAIN PLAN was part of the Oracle database well before the current version. So there are several earlier utilities for query plan investigation. For example, the UTLXPLS and UTLXPLP SQL scripts in the /rdbms/admin directory of your Linux Oracle installation. There is also a PL/SQL package called DBMS\_XPLAN that is currently recommended for displaying execution plans.

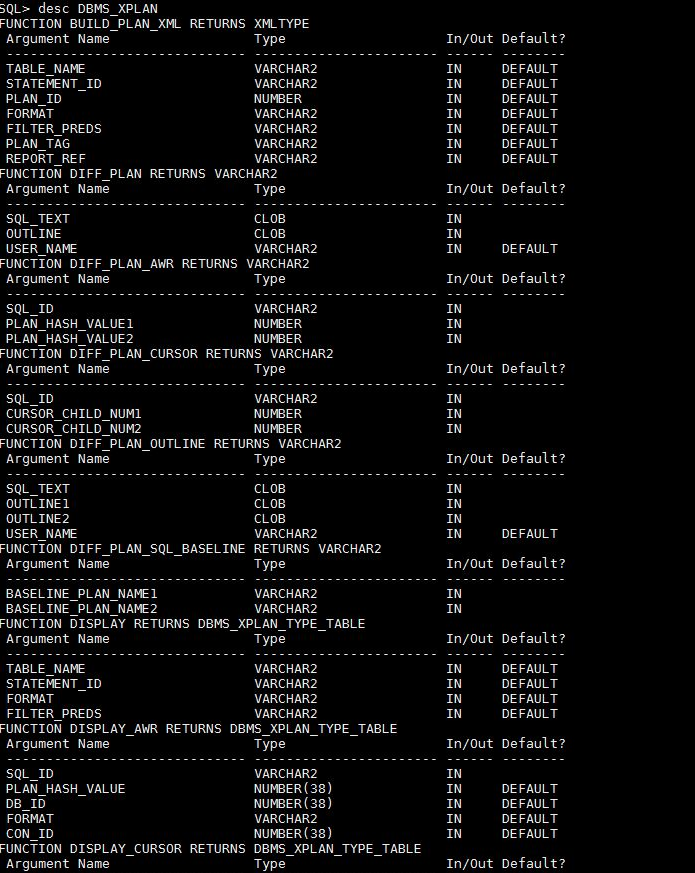
* What is a PL/SQL package?

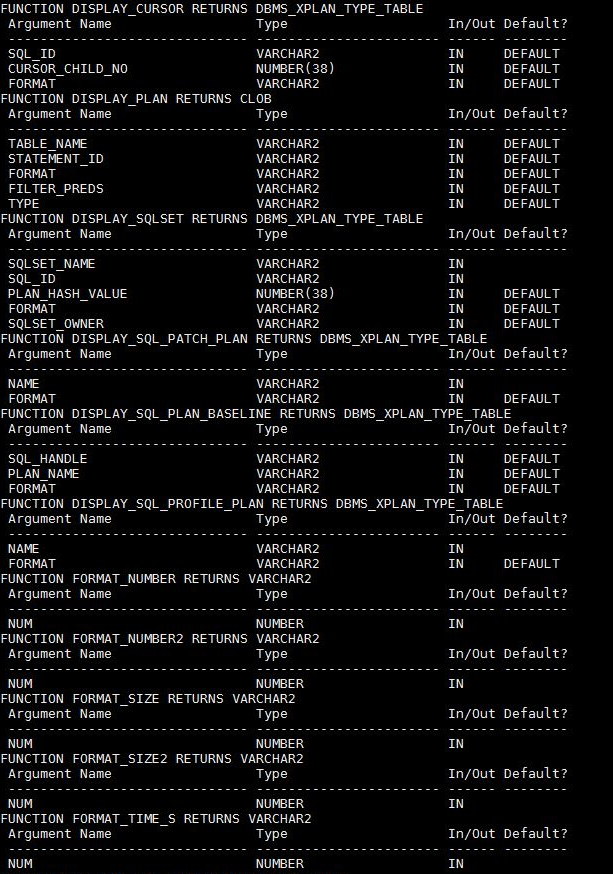
A **package** is a schema object that groups logically related PL/SQL types, items, and subprograms. Packages usually have two parts, a specification and a body, although sometimes the body is unnecessary. The **specification** (**spec** for short) is the interface to your applications; it declares the types, variables, constants, exceptions, cursors, and subprograms available for use. The **body** fully defines cursors and subprograms, and so implements the spec.

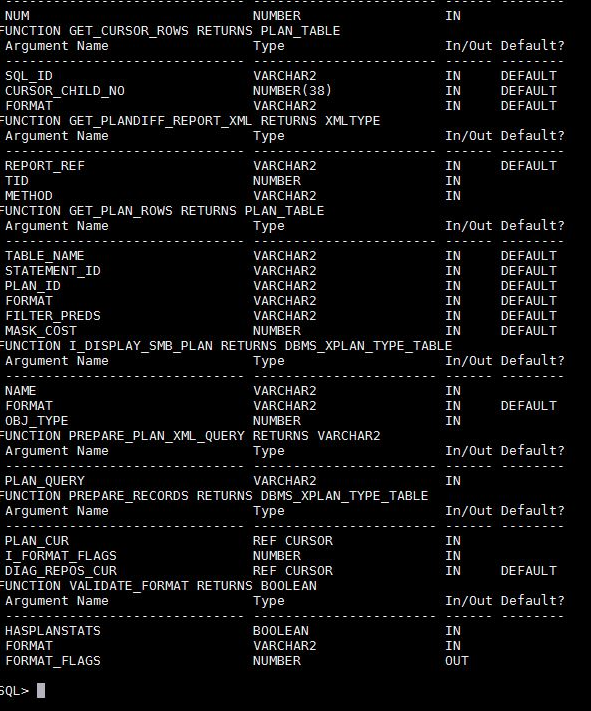
To take a look at the documentation for the DBMS\_XPLAN package; describe it.

* What information is displayed? Record your results below.

(paste)







As you can see, the package contains multiple functions. We’re interested in the DISPLAY() function which has four (4) parameters. Fill in **Table 3** to explain these parameters. Clearly *relate* each parameter to the value that you would use to extract the plan for your example query (Activity #4-3). If there are multiple argument value options, explain them.

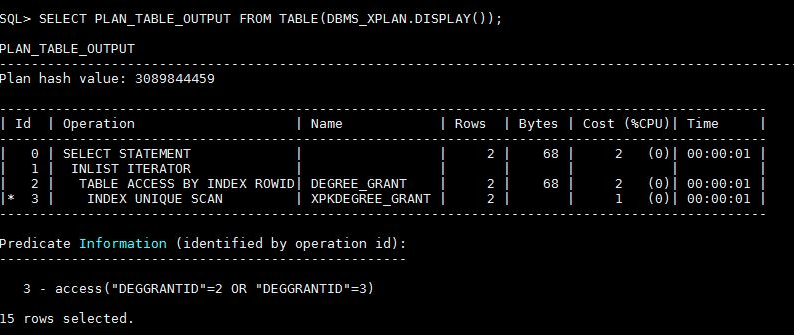
**Table 3.** *Explanation of* *DISPLAY() Function Parameters*

|  |  |  |
| --- | --- | --- |
| **Argument Name** | **Argument Meaning** | **All Possible Values** |
| Table Name | Specifies the table name where the plan is stored. | PLAN\_TABLE |
| Statement\_ID | Specifies the statement\_id of the plan to be displayed. | DEFAULT IS NULL |
| Format | Controls the level of details for the plan | BASIC, TYPICAL, ALL, SERIAL |
| Filter\_Preds | SQL filter predicate(s) to restrict the set of rows selected from the table where the plan is stored. | DEFAULT IS NULL |

Using this package, write a query to fetch your plan again, showing *default* information.

* Show your code below.

(paste)



* Paste your output below. (Be sure to format it to align and fit on the page.)

(paste)

Refer above screen shot.

* What does the asterisk (\*) in the Id column of row ID #3 in the plan mean?

The asterisk operator is use to denote the Index

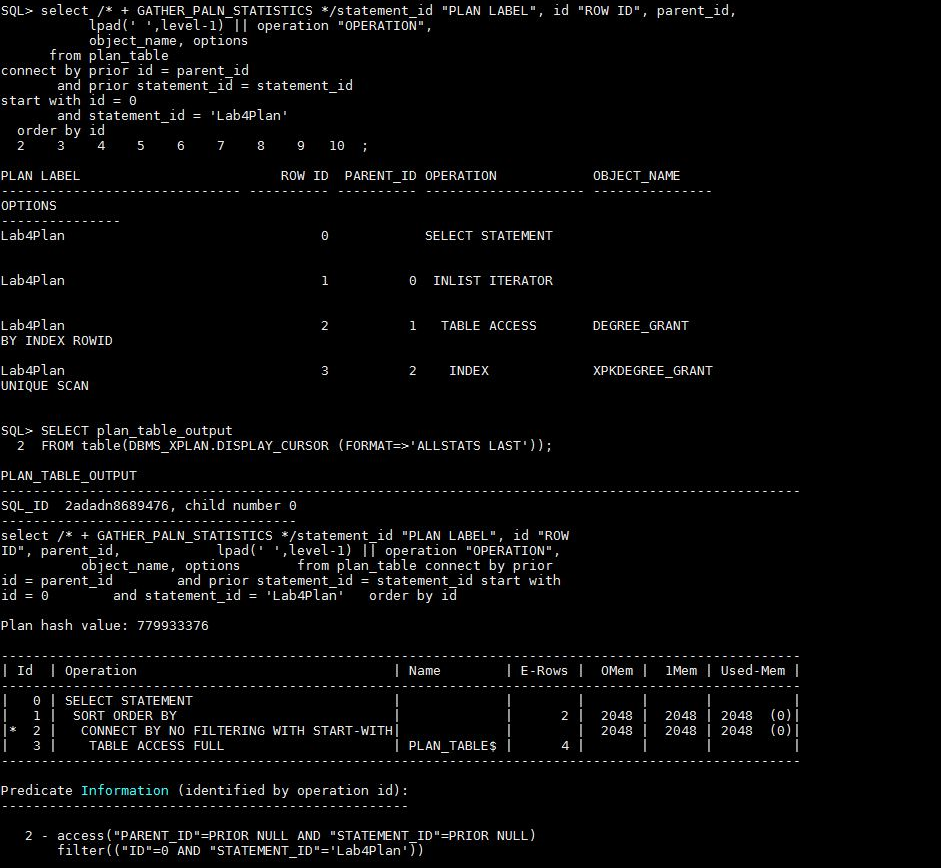
* What *additional* information do you see in this plan versus the plan from Figure 1? (wt 2)

We see Cost and time taken by the statement to execute the query.

Write another query to re-fetch your plan using DBMS\_XPLAN. This time show *all* possible information. Show your command and results.

(paste command)

(paste results)



* What additional information do you get? Explain in detail.

Alternatively you can use the GATHER\_PLAN\_STATISTICS hint in the SQL statement to automatically collect more comprehensive runtime statistics. This hint records the actual cardinality Oracle Optimizer. This execution time (or run time) cardinality can then be displayed in the execution plan, using DBMS\_XPLAN.DISPLAY\_CURSOR, with the format parameter set to 'ALLSTATS LAST'. An additional column called A-Rows, which stands for actual rows returned, will appear in the plan.

Let’s summarize what you’ve found out so far...

**Activity #4-5 – *Summarize Query Plan Basics***

We will get into the interpretation of execution plans and execution plan statistics more in the next lab. For now, answer the questions below to summarize what you’ve found out so far about query plan execution in a typical commercial DBMS.

For a query plan:

* Query plans are executed from [ **the bottom** | the top ] of the plan display results

[**Upwards** | downwards].

For a query plan displayed through EXPLAIN PLAN: (weight 3)

* Plan execution starts from the [ left-most out-dented | **right-most indented** ] step shown in the query plan and works [ rightward | **leftward**  ] out through the plan steps.
* The steps in a query plan are executed in a [non-hierarchical | **hierarchical**] order.
* How does a text-based query execution plan – as shown in a tool like Oracle’s EXPLAIN PLAN – relate to the query trees that we’ve been discussing in class? Explain.

Text-based are a bit harder to read, but more information is immediately available. There are three text plan formats:

* **SHOWPLAN\_ALL**: a reasonably complete set of data showing the Estimated execution plan for the query
* **SHOWPLAN\_TEXT** : provides a very limited set of data for use with tools like **osql.exe**. It too only shows the Estimated execution plan
* **STATISTICS PROFILE:**similar to **SHOWPLAN\_ALL**except it represents the data for the Actual execution plan

Whereas, for tree based is the row source tree is the core of the execution plan. It shows the following information:

* An ordering of the tables referenced by the statement
* An access method for each table mentioned in the statement
* A join method for tables affected by join operations in the statement
* Data operations like filter, sort, or aggregation

In addition to the row source tree, the plan table contains information about the following:

* Optimization, such as the cost and cardinality of each operation
* Partitioning, such as the set of accessed partitions
* Parallel execution, such as the distribution method of join inputs

It’s hard to imagine how things can possibly get any *more* exciting. However, now it’s time to do some investigating on your own! ☺

**🡪 Be prepared to discuss your findings in class.**

**Activity #4-6 – *Terminate Session***

Before you leave the lab, logout of all applications, disconnect any users, and exit any sessions. Your instructor(s) recommend that you leave your database running.

**Activity #4-7 – *Optional Bonus (+2 weights)***

Refer back to your query plans in Activity #4-4. Notice the INLIST ITERATOR step?

* What is INLIST ITERATOR? (+1)

An INLIST ITERATOR operation appears in the EXPLAIN PLAN output if an index implements an IN list predicate. Basically it’s an OR operation.

* Per Oracle, what condition significantly impacts the performance of the INLIST ITERATOR operator? (+1)

Per Oracle, Index column condition significantly impacts the performance of the INLIST ITERATOR operator. Because rather than doing full table scan it only selects rowID based on index, hence less cost.

1. If an error occurs when you try to use the EXPLAIN PLAN command, your database or user may not be properly setup for its use. It should work for instDBA, however. [↑](#footnote-ref-1)
2. Be aware that EXPLAIN PLAN just adds rows, with execution plan information, to the plan-storage table at each execution; it does not delete them. So if you run it more than once for a given query, you will just store the same plan again – unless you truncate the plan-storage table (as sys). [↑](#footnote-ref-2)
3. Note: The spacing shown in the query code is only for readability. However, the formatting is necessary for plan readability. [↑](#footnote-ref-3)