**Purpose:** To provide you with the opportunity to gain additional insight into aspects of query performance.

**Activities:**

1. Logon to your general user account and use the supplied script (contained in lab6.sql) to create a sequence, create two tables, and to populate both tables with several rows of data. (Note: neither table has a primary key or an index). Think of the *lab6\_parent* table as a strong entity (e.g., *Employees*) and the *lab6\_child* table as a weak entity (e.g., *Dependents* of the employee). You may recall a weak entity inherits its parent's primary key and adds this inherited key to a local partial key to create a composite primary key. In our situation the *lab6\_child* table will eventually (i.e., several steps later) have a composite primary key of (ee#, fname).   
     
   Note: To ensure that you are able to create a sequence, connect as SYS and *grant create sequence to* your general user. Then connect as your general user and run the lab6.sql script.   
   1. From your general user account:
      1. Enter: *EXPLAIN PLAN FOR*

*Select ee#*

*From lab6\_parent*

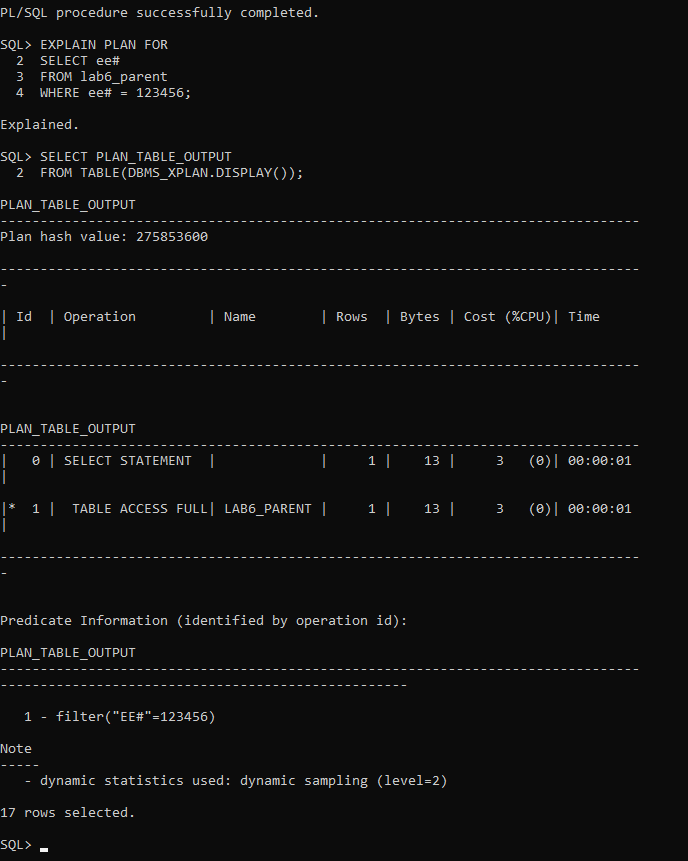
*Where ee# = 123456;*

Enter: *SELECT PLAN\_TABLE\_OUTPUT*

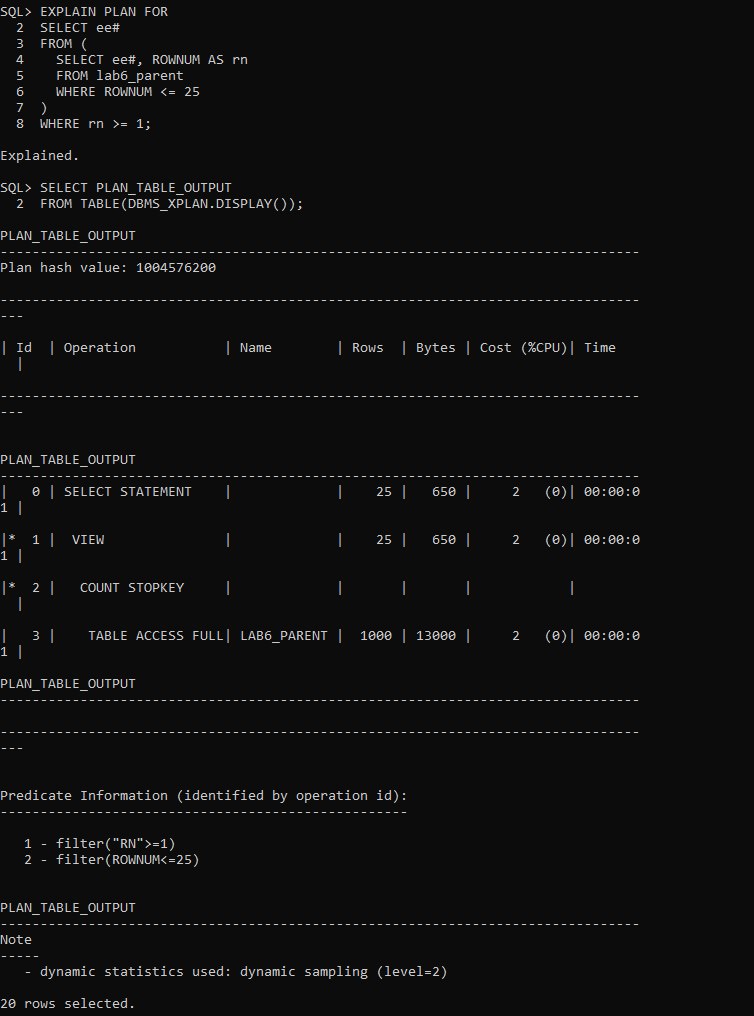
*FROM TABLE(DBMS\_XPLAN.DISPLAY());*

Note: The “EXPLAIN PLAN….” command above creates some rows of text as output, and then you access it using the “SELECT PLAN\_TABLE\_OUTPUT … “command. The number of rows shown is just the number of rows in the plan output – NOT the number of rows in the result set of the original query that was part of the EXPLAIN PLAN command.

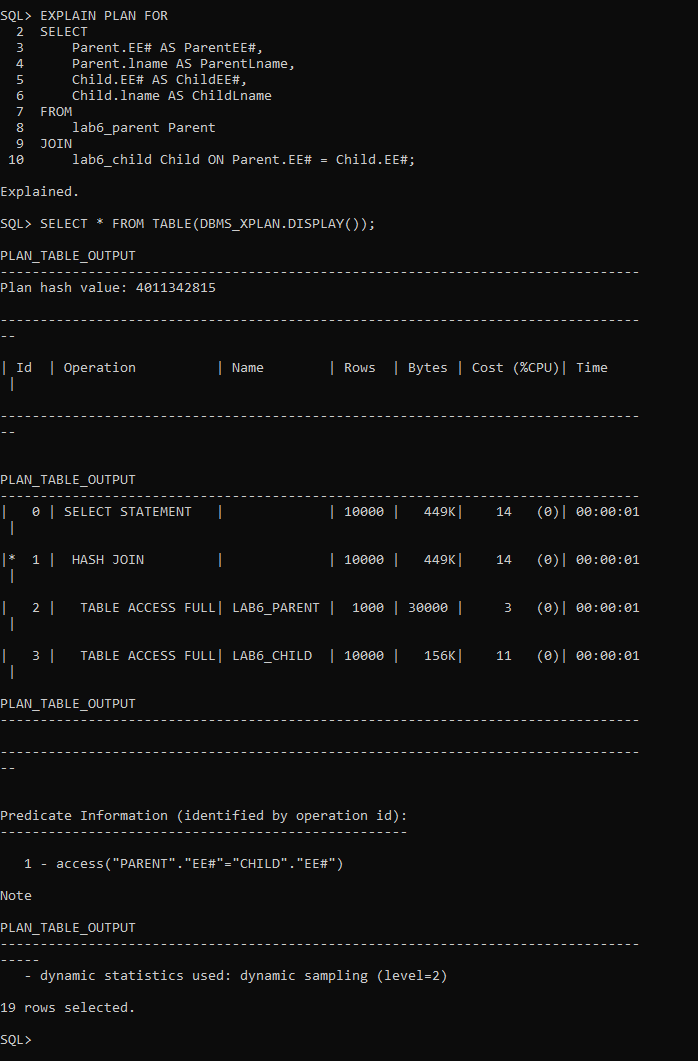
* + 1. Paste a screen snapshot of the Explain Plan query and the Plan\_Table\_Output into the area below.



* + 1. Change the WHERE clause in *1ai* so that the query returns a range of approximately 25 ee#s. Copy your Explain Plan and Plan\_Table\_Output below.

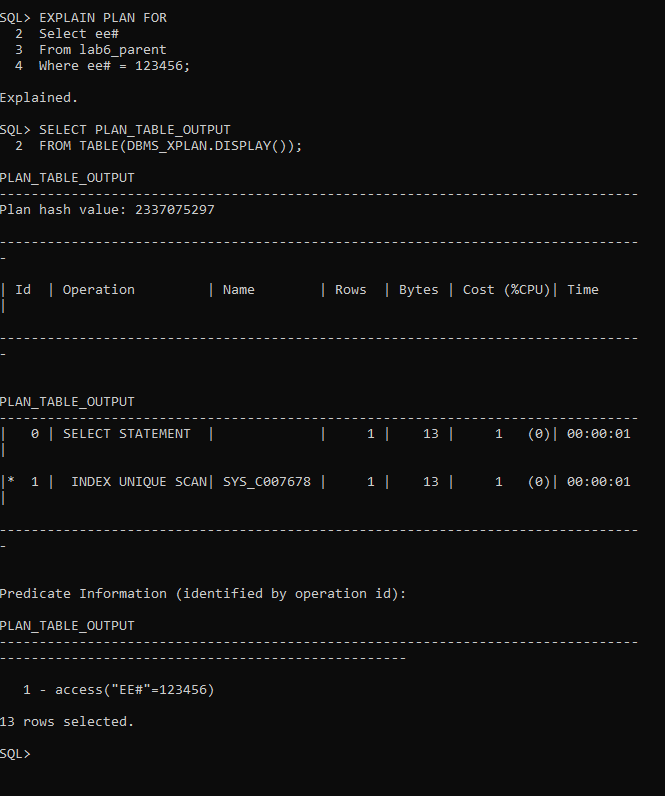


* 1. Write an Explain Plan for a query that displays Parent.EE#, Parent.Lname, Child.EE# and Child.Lname. Take a screenshot of the Explain\_plan and the Plan\_Table\_Output



* 1. The main purpose of the preceding activities was to reinforce the idea that a certain type of retrieval – generally considered relatively slow - will be the default approach to retrieving data. Look at the explain plan – it likely shows that each row is retrieved and accessed to check the “where” clause.

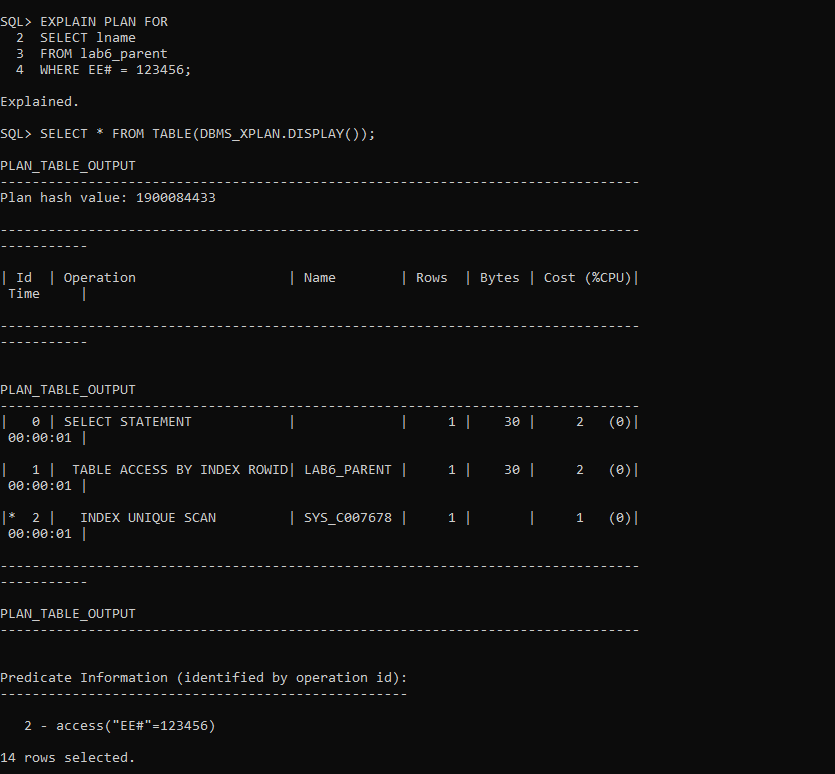
1. Add a primary key to both the parent table (ee#) and the child table (ee#, fname). Then rerun *1ai*. The idea is that with the key, there will be an index, which could allow the table to be accessed more efficiently for queries that use ee#. If you obtain the same result as *1ai*, then connect to sys as sysdba and enter: *ALTER SYSTEM FLUSH SHARED\_POOL;* and bounce the database (shutdown, startup), reconnect to your general user and rerun the EXPLAIN PLAN query. That will ensure the cached data is flushed.
   1. Your Plan\_Table\_Output should now be significantly different from the non-indexed version. Take a screenshot of the Explain Plan query and the Plan\_Table \_Output and add it to your submission document.



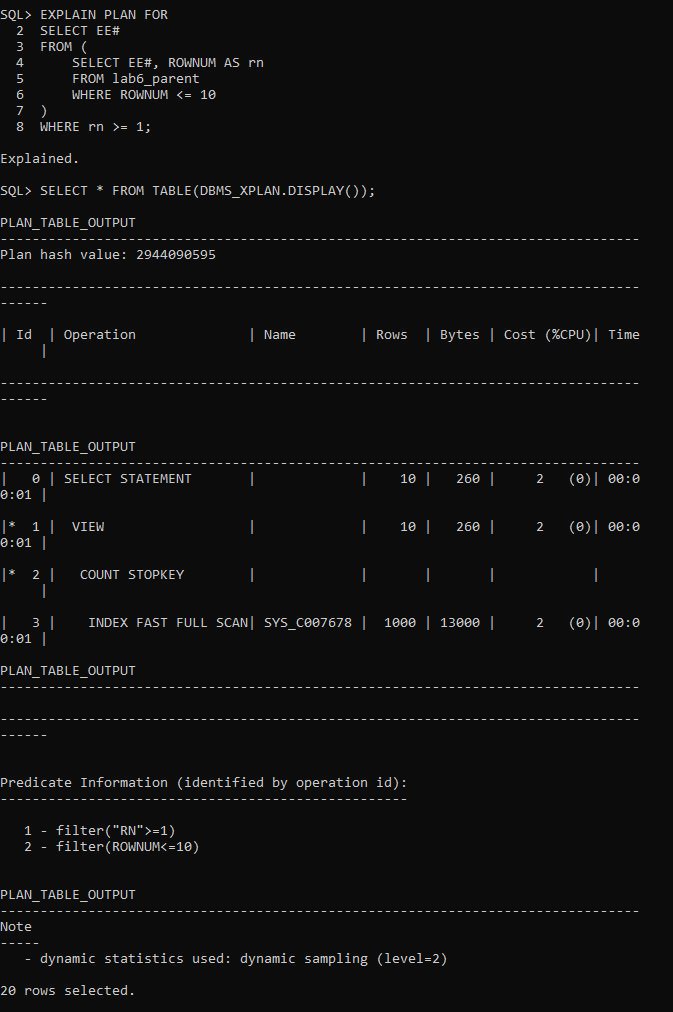
Explain how your actions in 2 changed the query plan (if your plan doesn't change, explain what you expected to occur. You can also try different where clauses to get the index to be used….). \_

By adding primary keys, which in turn automatically create indexes, queries targeting indexed columns should become more efficient. Factors such as table statistics, data distribution, and specific SQL operations will influence the adjustments to the execution plan. By closely monitoring and understanding changes to this plan, you can gain insights into the database's query handling and how alterations to the database schema might influence performance

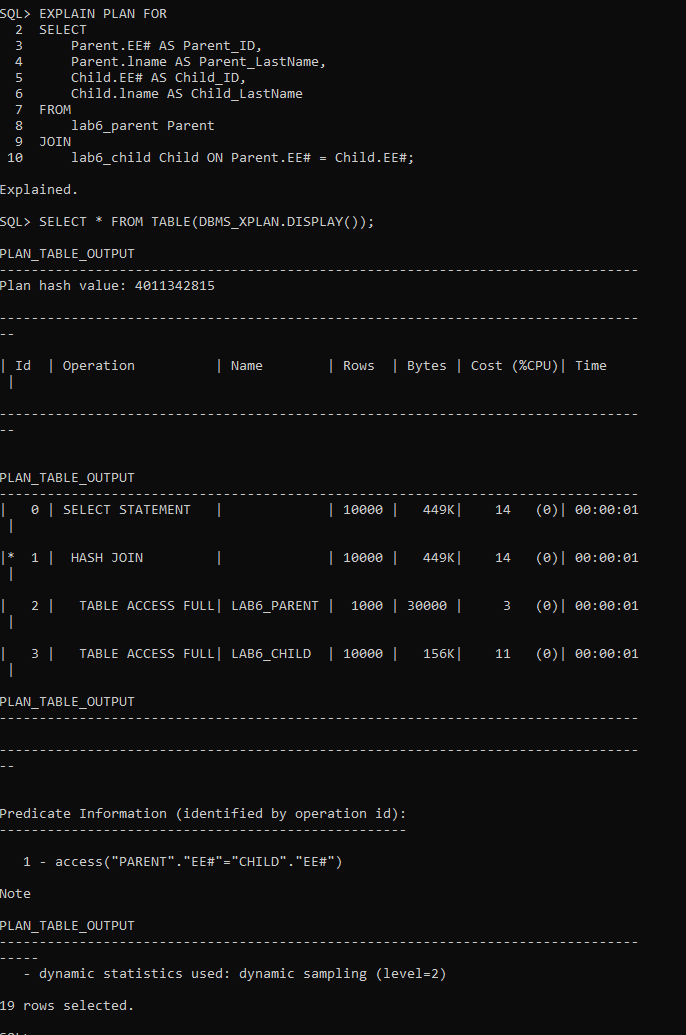
* 1. Now that you have added primary keys to both tables, try each of the following scenarios and include screenshots of the explain plan and plan table output for each scenario
     1. a single ee#



* + 1. a range of ee#s

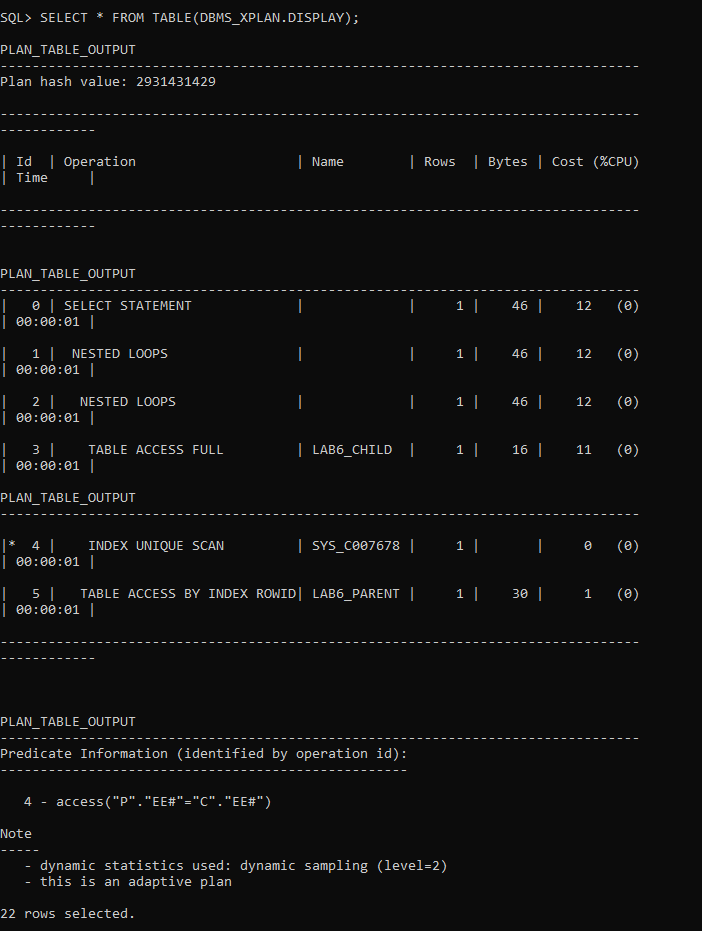


* + 1. a table join on ee# similar to Question 1. Include a screenshot of the explain plan and plan table output.

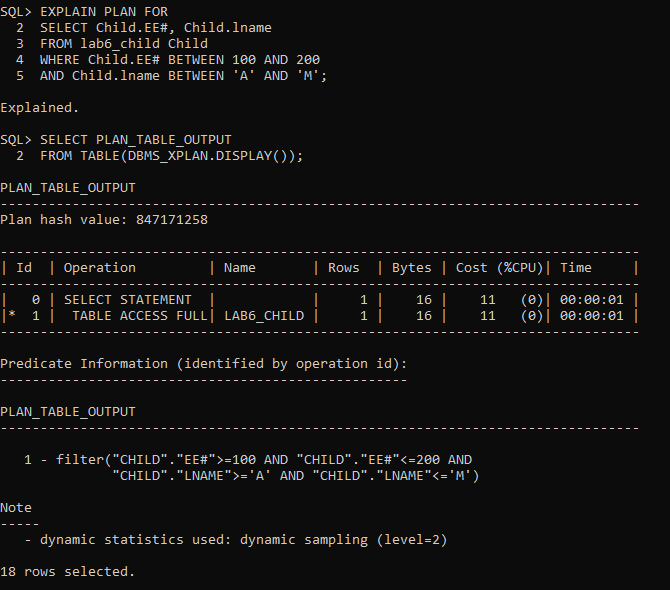


1. Add a foreign key to the child table referencing the parent table. Rerun the table join case from 2 b iii. If your results did not change from the pre-foreign key table join results then, from SYS, bounce the database and retry. Include a screenshot of the explain plan and the resulting plan table output. IF there is still no difference, try changing the where clause to include multiple criteria.   
     
   Describe what you expected to happen (and why) and show your actual results below.

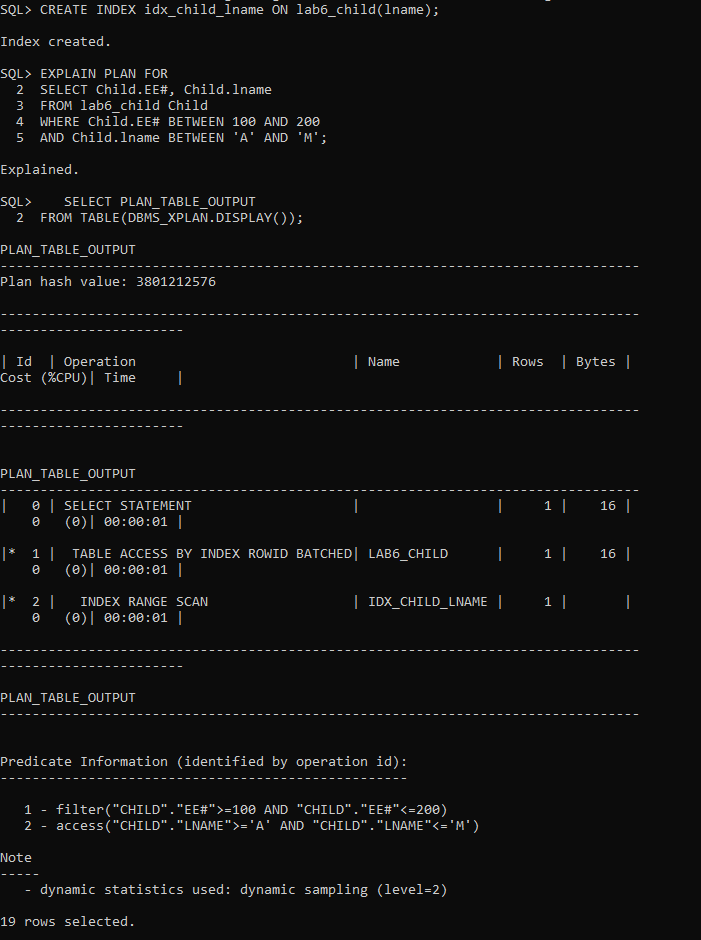
Before adding the foreign key, the optimizer's decisions on join methods were based on the available statistics and indexes. There might have been a full table scan for the EE# column in the lab6\_parent table if no primary key or index was present. The cost and cardinality were determined by the available table statistics at that time After adding the foreign key, the optimizer's decisions could be influenced because foreign keys utilize indexed primary keys. This can potentially improve performance by streamlining query operations, as the database knows about the relationship between the tables and can make optimized decisions based on it.



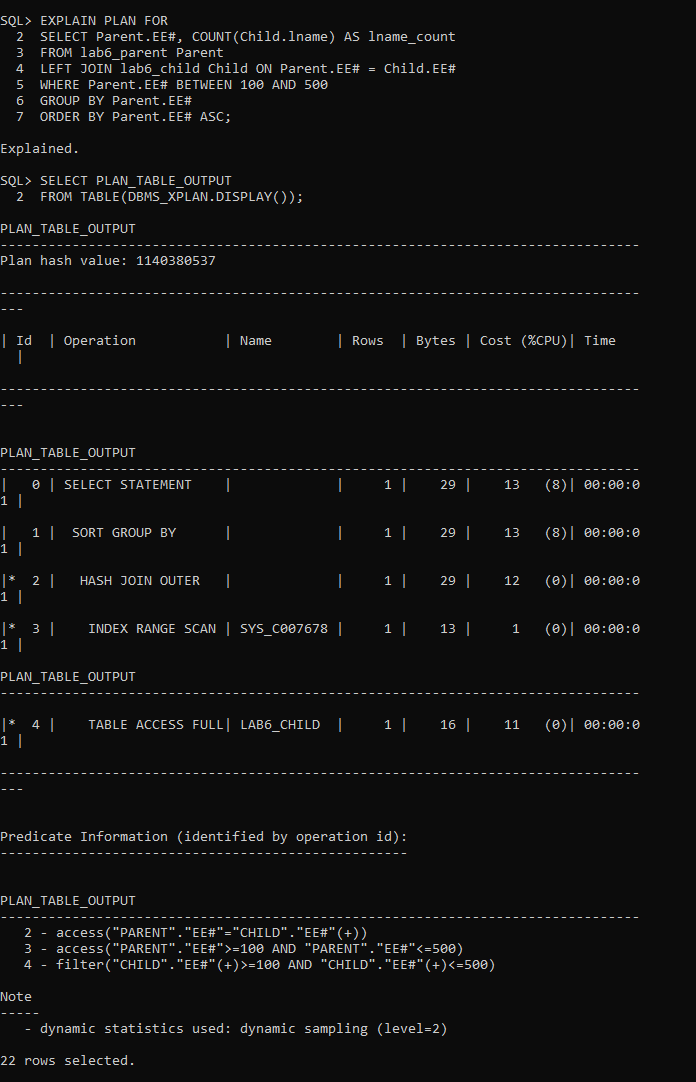
1. Write an Explain Plan query that contains a child ee# range and a child lname range selection criteria.
   1. Copy the explain plan and plan­ table output.



* 1. Add an index to child.lname. Rerun the previous child ee# and child lname range query. Copy the explain plan and plan­ table output. Again, if you don't see any change in results, bounce the instance and rerun.



1. Write an Explain Plan query that contains parent ee#, count(lname) an ee# range selection criterion and an ordering of the results by ee# in ascending order. Copy the explain plan and plan­ table output.

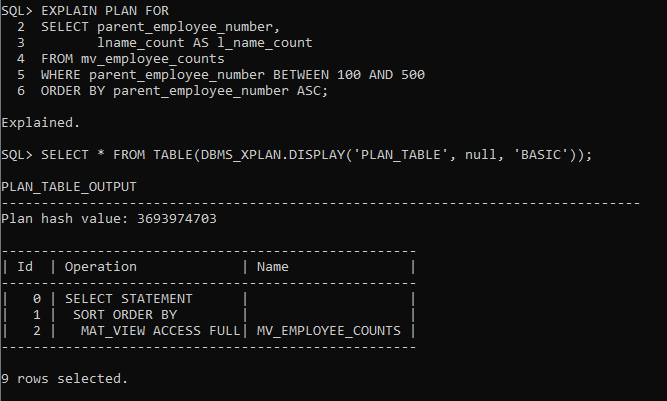


1. Materialized views are very important performance enhancing objects in large databases and/or data warehouses that either need to join multiple tables or perform significant aggregation from remote instances. The materialized view avoids reexecuting the query for each access, even across restarts of the database. The Oracle Data Warehousing Guide, located here, <http://docs.oracle.com/database/121/DWHSG/basicmv.htm#DWHSG008> , provides a good description of this feature.

What does a materialized view contain that a regular view does not?

A materialized view stores a snapshot of actual data, so it uses disk space. We need to update this data periodically. It's great for faster query results when the original data or calculations are complex. A regular view doesn't store data. It's a saved version of a query. It doesn't need updates like a materialized view, also it might not be as fast for certain data or calculations.

1. Connect to SYS as SYSDBA and grant your generalized user the privilege of being able to *create materialized view*.
   1. Connect as your general user and create a materialized view using a query that returns parent ee#, count(fname) and count(lname). Show your work below.
   2. Edit the query from Q5 to use the newly created materialized view instead of the underlying tables. Execute the new query, and copy the explain plan and plan­ table output.



You're finished. Please submit.