Performance Improvement using Indexes and ABAP Hints

Applies to:
Release SAP 4.6B and greater. For more information, visit the Performance homepage.

Summary
This article describes the techniques that can be used when faced with problem of long background job runs or when the overall system performance is low. It explains how one can conclude the need for Secondary Index or ABAP hints. It then explains in detail how a Secondary Index and ABAP hint can be created.

Author: Ranjeeta Nair
Company: Larsen & Toubro Infotech Limited
Created on: 13 July 2010

Author Bio
Ranjeeta Nair is a SAP Consultant in Larsen & Toubro Infotech Limited. She has four years of SAP experience. She has worked extensively in ABAP and SCM.
# Table of Contents

Overview........................................................................................................................................3

Secondary Index .................................................................................................................................3
  Need of Secondary Indexes ..............................................................................................................3
  How to create Secondary Index .......................................................................................................4

Database Hints ..................................................................................................................................6
  Need of Database Hints ....................................................................................................................6
  How to create Database Hints ..........................................................................................................7

Points to remember while using Indexes and Hints ........................................................................7

Related Content ................................................................................................................................10

Disclaimer and Liability Notice .........................................................................................................11
Overview

Optimized performance of background jobs are a prerequisite in live production environments. No matter how optimized the ABAP code is, over a period of time due to accumulation of large amounts of data in tables, programs can go into a long run.

In such instances, index definitions needs to be checked. If the query is such that it cannot use the primary key, creation of a secondary index can be considered.

There are instances when the SAP Cost Based Optimizer (CBO) over-rides the secondary index and continues to do a sequential read on the table. At such times, coding an ABAP hint that would force the CBO to use the secondary index can be used.

Secondary Index

Need of Secondary Indexes

By default the Primary key forms the primary index of database tables. Whenever a SELECT query is executed, the system checks if the fields in selection or the filter are part of the primary key. If yes, then the cost of executing such a query is less. Therefore, it is always advisable for users to consider the primary key while coding SELECT queries.

It might not always be possible to use the Primary key fields in the queries. Programs having SELECT queries with non-key fields are vulnerable to go into a long run. This can happen if the table contents increase over a period of time.

At such times, one can create a secondary index, which would include the fields that are part of the selection/ filter of your SELECT query. At execution time, when the system uses the Secondary Index while fetching data from table, the cost of such a hit would be less.

Given below is an example on how one can go about deciding the need for a Secondary Index. The issue explained here occurred on an APO production system:

1. The background jobs gradually took long to complete. Therefore, a trace of the execution was done
2. It pointed to the query that gave the performance issue. In the example explained here, the program used to run for 600000+ seconds that eventually terminated due to lack of memory space.

Further investigation of the job and a trace revealed that almost 27 hours of execution is spent reading from table /SAPAPO/COMP.
Execution plan of the query is as shown below:

```
SELECT
  "ACTID"
FROM
  "/SAPAPO/COMP"
WHERE
  "MANDT" = :A0 AND "LOGCOMP" LIKE :A1 AND "IOIND" = :A2#
```

Follow is the SQL Statement at system level:

```
SELECT
  "ACTID"
FROM
  "/SAPAPO/COMP"
WHERE
  "MANDT" = :A0 AND "LOGCOMP" LIKE :A1 AND "IOIND" = :A2#
```

The sequential reads on this table was slow. Upon reviewing the execution plan of this SQL statement, it was noted that a full table scan was being done of this table, which performance wise is pretty intensive.

Normally such a SQL statement should use an index, which would optimize the reads. The indices defined against this table were checked and none of them were adequate for this SQL statement. This was because the order of the fields in the indices did not match the order of the predicates in the SQL statement. Hence it did the full table scans. In such cases it is advisable to create a secondary index.

For this example, the order of the fields would be: MANDT, LOGCOMP and IOIND

**How to create Secondary Index**

Given below are the steps to create a Secondary Index. Standard SAP APO table /SAPAPO/COMP is considered for explanation.


   ![](image)

   On the pop-up that appears, click on the Create (F5) button
2. Enter an appropriate Index Name e.g. ZLC (Z* for Custom Secondary Indexes) and click OK.

3. Make note of the Information messages (if any) and click on OK button.

4. Enter the Short Description and fields that you need to be included in the Secondary Index.
5. Save and Activate the Index. This will change the status to Active.

6. After the Index is created, stats update needs to be done. This is usually done by the Basis team.

7. Also, a point to be noted is that the index can be imported to production environment and the index can be activated in oracle at low activity period as this might have a performance impact. This activity should be done with consultation with the DBA/ Basis team

8. When the job is then executed, it gave better performance than before the introduction of Secondary Index.

Database Hints

Need of Database Hints

In another instance, though the secondary index existed for the custom table, the Oracle Cost Based Optimizer (CBO) chose to ignore it and did a full table scan instead. This could be because the CBO found the cost of full table scan lower than the cost of using the secondary index. This was most likely caused by oracle profile settings which make full table scan more favorable. In such cases, in order to optimize performance we need to force the system to use the existing Secondary Index. This can be done by using Database Hints. Hints force the CBO to use the index defined in the query.

In case of the example used below, the Secondary Index alone did not improve performance. Further analysis on these runtimes revealed that oracle did not make use of the Secondary index and preferred doing full table scans. The job run times was 35000+ seconds.

An SQL trace of the job while it ran revealed that the query took an average of 3.4 seconds to execute and this multiplied by the number of iterations explained the long runtimes of the job.
How to create Database Hints

In order to have the report to consider the secondary index created, a hint would need to be added in the SQL query of the ABAP:

```
SELECT <fields> INTO <work area> FROM <database table>
  WHERE <logical expression>
  AND <logical expression>
  %_HINTS <database name> 'INDEX("<database table>" "Index name")'.
```

The Hint will be specific to one database system. Appropriate value needs to be put for <database name> viz. ORACLE, DB2 etc. Once the index is created, import the modified report which includes the hint for the index usage. Once this index is activated in oracle, Basis team will update the stats on the table.

Points to remember while using Indexes and Hints

Following points are to be noted while working with Indexes and Hints:

1. The ABAP syntax check does not indicate an error if the Index name mentioned in the hint is incorrect.

2. If the hint is not valid (e.g. wrong index, syntax error), the query is handled by the CBO like there would be no hint specified, and oracle can take the wrong "access path" based on wrong stats.

In other words, always make sure that the Index name in the data dictionary (SE11) and the Index created in the Oracle database matches. In case the Oracle index name differs from that of SE11, either change the index name for the hint in ABAP to the one created on oracle or add the index name created in oracle as a second index for the hint. With two indexes specified, the hint should be able to use index which got created on oracle.
Refer the following screen shots for better understanding:

1. Query execution plan showing it's doing full table scans. For this custom table a secondary index was created with the relevant fields:

```
SELECT /*
  INDEX('/ZXX/XXX/XXX/XXX' , '/ZXX/XXX/XXX/XXX-001')
*/
  LOCNO , NATNR , RUN_DATE , AGR_DATE , QUANTITY_ORD , QUANTITY_CHG , DELTA_QTY ,
  PLANNER , CHANGE_DATE , PLANNER_PPS
FROM /*
  /ZXX/XXX/XXX/XXX*/
WHERE
  LOCNO = A1 AND 'NATNR' = A1 AND 'AGR_DATE' = A2 OR 'LOCNO' = A3 AND 'NATNR' = A4 AND 'AGR_DATE' = A5 OR 'LOCNO' = A6 AND 'NATNR' = A7 AND 'AGR_DATE' = A8 OR 'LOCNO' = A9 AND 'NATNR' = A10 AND 'AGR_DATE' = A11 OR 'LOCNO' = A12 AND 'NATNR' = A13 AND 'AGR_DATE' = A14
```

2. Index definition in SE11, shows different name for the index than that in oracle. ABAP code is pointing to the Data Dictionary name of the index, but this one doesn't exist on Oracle. So the hint is not used and job is doing full table scans.

- Index Name: [ZXX][XXX][XXX][XXX] - 091
- Short description: Secondary index created for runtime enhocmt of PD1610
- Original language: EN English
- Status: Active
- Saved
- Package: [ZXX][XXX][XXX][XXX]00

Index [ZXX][XXX][XXX][XXX]00 exists in database system ORACLE

- Non-unique index
- Index on all database systems
- For selected database systems
- No database index

- Unique index (database index required)

**Index fields**

<table>
<thead>
<tr>
<th>Field name</th>
<th>Short Text</th>
<th>DT/Len</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCNO</td>
<td>Location</td>
<td>CHAR</td>
</tr>
<tr>
<td>NATNR</td>
<td>Product ID</td>
<td>CHAR</td>
</tr>
<tr>
<td>AGR_DATE</td>
<td>Date</td>
<td>DATS</td>
</tr>
</tbody>
</table>
3. Hint in ABAP is pointing to the Database Dictionary name for the index:

```
SELECT * FROM /ZXX/XXX_XXX
INTO CORRESPONDING FIELDS OF TABLE <internal table>
FOR ALL ENTRIES IN <internal table2>
WHERE locno = <internal table>-locno
AND matnr = <internal table>-matnr
AND agr_date = <internal table>-agr_date
     /*_HINTS ORACLE 'INDEX("/ZXX/XXX_XXX_XXX" "/ZXX/XXX_XXX_XXX-001")' _RNN091-
     "RNN091+"
```

4. Index name on oracle level

```
SQL> select index_name from cba_indexes where table_name = '/ZXX/XXX_XXX_XXX';

<table>
<thead>
<tr>
<th>INDEX_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>/ZXX/XXX_XXX_XXX ~0</td>
</tr>
<tr>
<td>/ZXX/XXX_XXX_XXX_00</td>
</tr>
</tbody>
</table>
```

5. The ABAP was modified to use the Oracle Index name and the program’s performance was considerably improved.

```
SELECT * FROM /ZXX/XXX_XXX
INTO CORRESPONDING FIELDS OF TABLE <internal table>
FOR ALL ENTRIES IN <internal table2>
WHERE locno = <internal table>-locno
AND matnr = <internal table>-matnr
AND agr_date = <internal table>-agr_date
     /*_HINTS ORACLE 'INDEX("/ZXX/XXX_XXX_XXX" "/ZXX/XXX_XXX_XXX-001")' _RNN091-
     "RNN091+"
```

For the above given example, the runtime of the changed program came down to approximately 3700 seconds from 35000+ seconds.

Please note that creation of Secondary Indexes and ABAP Hints should be done with clear communication with the Database Administrator and Basis team.
Related Content

http://help.sap.com/saphelp_nw04/helpdata/en/cf/21eb47446011d189700000e832d00/frameset.htm
https://websmp130.sap-ag.de/sap(bD1lbiZjPTAwMQ==)/bc/bsp/spn/sapnotes/index2.htm?numm=129385

For more information, visit the Performance homepage.
Disclaimer and Liability Notice

This document may discuss sample coding or other information that does not include SAP official interfaces and therefore is not supported by SAP. Changes made based on this information are not supported and can be overwritten during an upgrade.

SAP will not be held liable for any damages caused by using or misusing the information, code or methods suggested in this document, and anyone using these methods does so at his/her own risk.

SAP offers no guarantees and assumes no responsibility or liability of any type with respect to the content of this technical article or code sample, including any liability resulting from incompatibility between the content within this document and the materials and services offered by SAP. You agree that you will not hold, or seek to hold, SAP responsible or liable with respect to the content of this document.