Infor LN Performance, Tracing and Tuning Guide for Oracle

Publication Information

Release: Infor LN 10.x

Publication date: November 26, 2012

Document code: B0078B

Contents

About this guide

This document provides guidelines to improve the performance of an Infor LN environment on an Oracle database by tracing and tuning the environment. The following chapters describe the processes to improve the database and the Infor LN application.

All information is based on the use of the Infor LN software.

Note: This document is a comprehensive compilation; however there may be instances wherein relevant information or procedures may have been omitted. Therefore, we strongly recommend verifying the proposed changes in a test environment before moving to production. The information provided may not hold true for future versions of the Oracle database.

Intended audience

This document is intended for intermediate to expert Infor LN and database Administrators and Technical Consultants in order to get optimal performance out of an Infor LN system.

Related documents

Certain sections in this document are described in more detail in other documents. These documents help to extend the knowledge in particular areas

- *Infor LN - Performance, Tracing and Tuning Guide (U9357 US)*
- *Infor LN - Sizing guide (B0045 US)*
- *Infor Enterprise Server - Technical Reference Guide for Oracle Database Driver (U7076 US)*

You can find the documents in the product documentation section of the Infor Xtreme Support portal, as described in "Contacting Infor" on page 8.

A good reference to Oracle documentation with regards to tracing and tuning is important. This site is recommended: http://docs.oracle.com.

Contacting Infor

If you have questions about Infor products, go to the Infor Xtreme Support portal at www.infor.com/inforxtreme.

If we update this document after the product release, we will post the new version on this Web site. We recommend that you check this Web site periodically for updated documentation.

If you have comments about Infor documentation, contact documentation@infor.com.

Chapter 1 Tuning Oracle for Infor LN

1

The performance of Infor LN also depends on the database performance. This chapter details the most important tuning areas of Oracle.

Introduction

There is no standard way to improve your system performance. This chapter describes the required actions based on the performance issues you have identified. Every new version of a database comes with new settings, tools, and so on. New parameters are also introduced in each version, and some defaults are modified. Most applications run well with an out-of-the box Oracle database but some do not. For these customers, tuning is necessary for efficient performance. Oracle is a dynamic database, nevertheless tuning is required for optimal performance. For the Oracle database to perform at optimum, knowledge of Oracle database is required.

As part of easing the process of database tuning, the following topics are discussed in detail:

- I/O setup
- SGA Setup
- Locking
- Creating statistics
- Stored Outlines/Profiles
- Keep data in buffer cache
- Oracle parameters
- Oracle compression

For generic Oracle tuning, the Oracle documents on http://docs.oracle.com are a good starting point. The basic tuning of Oracle depends on the version used. Some important quick reference guides can be found at the following Oracle knowledge base articles:

- 390374.1: Oracle Performance Diagnostic Guide
- 248971.1: Query Tuning Best Practices
- 68735.1: Diagnostics for Query Tuning Problems
- 67983.1: Oracle Net Performance Tuning

The recommended Oracle performance tools are the Oracle Enterprise Manager, statspack, AWR, and ADDM. Also the Oracle ADDM process creates a default snapshot every hour, which makes comparisons (for a period of time) easier. However, by default snapshots older than a week are deleted hence, it is recommended to change the setting with the following command:

```
BEGIN
  DBMS_WORKLOAD_REPOSITORY.modify_snapshot_settings(
    retention => 43200,      -- Minutes (= 30 Days)
    interval  => 30          -- Minutes.
  );
END;
```

I/O Setup

The *Infor LN - Performance, Tracing and Tuning Guide (U9357)* provides guidelines for a correct I/O setup. For Oracle, the following additional guidelines are important:

- Allocate adequate size for the Undo tablespace in order to prevent latch: undo global data concurrency. The minimum recommended starting value is 10 GB. In case, the latch “undo global data” gets visible in the AWR report the size of the Undo tablespace should be increased even when there seems sufficient free space in the Undo tablespace.
- For Infor LN, the minimum recommended size is 6 redo log files of at least 1 GB, to prevent log writer wait events. For large environments redo logs of 10 GB are not uncommon.

SGA Setup

The following rules provide a general guideline for the Oracle SGA size:

- If the system is only used for the Oracle database, 40% of the physical memory can be allocated to Oracle shared memory.
- If the system is used for Infor LN and the Oracle database, 25% of the physical memory can be allocated to Oracle shared memory.

For more guidelines about the expected memory usage of the Infor LN system, see *Infor LN - Sizing Guide (B0045 US)*.

The Oracle (shared) memory must be divided into different areas. In general, if more control is provided to Oracle on the memory parameters, the more resources it will cost.

It is possible to use the memory_target parameter for automatic memory tuning, but for performance reasons, Oracle suggests using sga_target and pga_aggregate_target instead.

Locking

Although locking has little to do with Oracle tuning, the performance of the end user can be restricted by locked records or tables. Ensure that the correct Infor LN settings are used.

Oracle detects deadlocks and will let you know which user is affected by using the Oracle trace files. This information can be used when validating the issue. The Oracle Enterprise manager can give comprehensive information such as who is blocking who, and also scripts that can help you. Usually, long duration of locking points to an application issue. When locking occurs on a table that starts with ttccom050, ensure that for the first free numbers, caching is enabled in Infor LN. See *Infor LN - Performance, Tracing and Tuning Guide (U9357 US)* for more information about First Free Number caching.

Creating statistics

Statistics on Oracle tables and indexes can be generated with the dbms_stats packages. Oracle support knowledge base article 749227.1has links to all types of documents that explain this feature in detail. Described below is an example on how to generate tables and indexes:

```
SQL> execute dbms_stats.gather_table_stats('INFORLN','TTTTXT010000', cascade => true);
```

Or to create statistics for the whole schema:

```
SQL> execute dbms_stats.gather_schema_stats('INFORLN', cascade => true);
```

The estimate percentage helps to overrule the standard percentage of blocks that will be scanned. The higher this percentage, the better the statistics, However it will take longer to generate these statistics. Adding “cascade => true” extends the statistics with index statistics. Statistics can be generated during the creation of the tables, automatically by a job or manually. The automatic update of statistics differs from the manual process, which can affect the performance. For example, statistics are updated by a script during the weekend and the performance declines during the week. It is recommended to disable the Oracle job that creates these statistics and run only the manual script at a regular interval. The job can be found through the Enterprise Manager. Go to the Server Tab -> Scheduler Jobs -> All -> MGMT_STATS_CONFIG_JOB.

When manually creating statistics, Oracle recommends the usage of automatic sample_size for statistics as several algorithms are turned off when the sample size is explicitly set.

SQL Plan Management

It is possible to revise the standard Oracle execution plan by using the Oracle Enterprise manager or SQL*Plus. This option is very useful when some of queries restrict the performance. In these cases, only these queries need optimization, to ensure a stable environment. For more information about SQL Plan Management, refer to Oracle support knowledge base articles 456518.1.

Keep data in buffer cache

It is possible that the Oracle buffer cache gets trashed by bad queries, import or export sessions, or batches that are not well scheduled. To retain information in the cache, the buffer_pool keep option must be used:

```
ALTER TABLE <tablename> STORAGE (BUFFER POOL_KEEP);
ALTER INDEX <indexname> STORAGE (BUFFER_POOL KEEP);
```

The tables to keep in memory are:

- Frequently used static tables and indexes.
- Small tables that are frequently accessed.

Examples of small tables are currencies, units and business partners.

Oracle parameters

The below table lists the Oracle parameters for which benchmarks and customer experience indicate improved performance. Some of are optional, others are required. All parameters are explained in this chapter. Note that the underscore parameters are very data dependent. For information on changes to Oracle Hidden parameters, refer to Oracle support knowledge base article 315631.1.

Database driver data retrieval

Infor LN is mainly an OLTP oriented application. For many queries, Infor LN stops retrieving rows after the first rows are fetched. During this process, Oracle generates an execution plan for the whole result set. Generally, this execution plan is not suitable for Infor LN. For most queries, the NESTED _LOOPS algorithm is preferred over HASH joins, SORT's, and VIEW's because this algorithm retrieves the first set of records faster. To enable the Oracle optimizer to prefer NESTED_LOOPS over the others, you can use parameters like db_file_multiblock_read_count, optimizer_index_caching, _optimizer_sortmerge_join_enabled and _hash_join_enabled. These parameters will be discussed in more detail below.

Parameter	Value	Importance	Remarks
lock_sga	TRUE	Recommended	Refer to the platform specific documentation.
use_large_pages	ONLY	Recommended	Refer to the platform specific documentation.
_enable_NUMA_support	TRUE	Recommended on NUMA systems	Tested on Oracle 11.2.

Parameter	Value	Importance	Remarks
filesystemio_options	SETALL	Recommended	
recyclebin	OFF	Recommended	Disable when not used.
db_file_multiblock_read_count		Recommended	Leave to default value
optimizer_index_caching	0 or a value between 10 and 90	Recommended	Test with both values.
optimizer_index_cost_adj	10	Recommended	
_complex_view_merging	FALSE	Optional	
_hash_join_enabled	FALSE	Optional	
_optimizer_sortmerge_join_enabled	FALSE	Optional	
_optim_peek_user_binds	FALSE	Optional	

Caution: Before implementing any of the above parameters, it is advised to study the behavior in the Oracle documentation.

Setting these parameters in the database can affect other products or other Infor LN sessions in the same database. However, it is possible to set these parameters only for the INFOR LN users, by defining the parameters in the db_resource with the ora_alter_session option, as below:

```
ora_alter_session:set "_optim_peek_user_binds"=false optimizer_index_cost_adj=false
```

For testing new settings, the ora_alter_session is the best solution, because also non-dynamic parameters can be easily validated without restarting the database. If the tests show improvements, the values can be moved to the db_resource file of all users; if the database is only used for Infor LN, the parameters can be set as Oracle parameters. A specific test db_resource file can be created and tested by setting USR_DBS_RES. For more information about this variable, refer to the “Infor LN Performance, Tracing and Tuning guide” (solution 22881401).

The listing of the changed parameters is always logged in the alert.log file, and can be viewed using the Enterprise Manager. The current used parameters can be listed in SQL*Plus using:

```
SQL> show parameter
```

All available normal and hidden parameters can also be listed with the following query:

```
col parameter for a50
col description for a70
col value for a20

set linesize 160
set pagesize 999

select i.KSPPINM parameter,
       v.KSPPSTVL value,
       i.ksppdesc description
from   x$ksppi i, x$ksppcv v
where  i.indx = v.indx
```

```
order by ksppinm;
```

Infor support has written two solutions specific for Infor LN together with an Oracle database:

- solution 725078 - Important Oracle 9, 10 and 11 performance Parameters
- solution 813406 - Bad performance with UNICODE and MLE with Oracle database

HP-UX_SCHED_NOAGE

To allow Oracle Database to use the SCHED_NOAGE scheduling policy (using the Oracle initialization parameter HPUX_SCHED_NOAGE=178, which is default from Oracle 11g onwards), the OSDBA group (typically, the dba group) must have the RTSCHED and RTPRIO privileges to change the scheduling policy and set the priority level for Oracle processes. The following steps must be followed to grant the dba group the required privileges:

1. Add line "dba RTPRIO RTSCHED" to "/etc/privgroup" file.

   Note: The group dba is assuming the oracle owner's ID is part of the dba group.

2. As root, run the following command:

   ```
   /etc/setprivgrp -f /etc/privgroup
   ```

3. Startup Oracle.

Lock SGA into memory

To improve performance and to prevent the Oracle SGA paging out of memory, the SGA can be locked into memory. To lock SGA in memory, special OS permissions are required for the Oracle user. LOCK_SGA=TRUE is platform specific and therefore, it is strongly recommended to refer to the platform specific documentation. We recommend using LOCK_SGA in combination with large/huge pages. To be sure all large pages can be allocated while starting the database we recommend setting also USE_LARGE_PAGES='ONLY'.

AIX

To support LOCK_SGA=TRUE on AIX, run the following command as user root:

```
chuser capabilities=CAP_BYPASS_RAC_VMM,CAP_PROPAGATE oracle
```

HP-UX

On HP VM, the virtual address mappings and teardowns, along with the emulation of the translation table can get expensive, so on a HP VM the Oracle SGA must be locked into memory.

To support LOCK_SGA=TRUE on HP-UX, follow these steps:

1. Add line "dba MLOCK" to "/etc/privgroup" file.

   Note: The group dba assumes that the oracle owner's ID is part of the dba group.

2. As root, run the following command:

```
/etc/setprivgrp -f /etc/privgroup
```

3. Startup Oracle.

Windows

To enable large pages in Windows, ensure that the Windows registry keys ORA_LPENABLE and ORA_LPSIZE are defined as a string (REG_SZ) datatype. Here is the Oracle Windows registry entry for ora_lpenable to enable large page support in Windows:

```
"ORA_LPENABLE" = "1"
```

Oracle Solaris

Setting parameter LOCK_SGA=TRUE is not supported on Solaris. For more information see Oracle support id 121983.1.

filesystemio_options

You can use the filesystemio_options initialization parameter to enable or disable asynchronous I/O or direct I/O on the File System files. This parameter is platform-specific and has a default value that is best for a particular platform.

During benchmarks the best results were obtained when using the value SETALL. To set filesystemio_options, run the following command and restart the database:

```
SQL> alter system set filesystemio_options=SETALL scope=spfile;
```

Disable Oracle recycle bin

The Oracle recycle bin can give performance or administration issues. If you are using the recycle bin and face performance issues, the recycle bin can be disabled:

```
alter system set recyclebin=OFF scope=both;
```

db_file_multiblock_read_count

The db_file_multiblock_read_count parameter is one of the parameters you can use to minimize I/O during table scans.
The value of db_file_multiblock_read_count can have a significant impact on the overall database performance and it is not easy for the administrator to determine its most appropriate value. It is advised not to change this parameter but leave it to its default value.

optimizer_index_caching

The optimizer_index_caching parameter indicates the percentage (between 0-100, where 0 is the default) of index blocks the optimizer should assume are in cache. The other percentage will be physical reads. Setting optimizer_index_caching to a higher value makes nested loop joins look less expensive to the optimizer. In that case, the optimizer will be more likely to pick nested loop joins over hash or sort merge joins. Setting the value too high can lead to selecting poor selective indexes that are not recommended.

optimizer_index_cost_adj

The optimizer_index_cost_adj parameter can be used to adjust the cost of index probes. The range of values is 1 to 10000. The default value is 100, which means that indexes are evaluated as an access path, based on the normal costing model. A value of 10 indicates that the cost of an index access path is one-tenth the normal cost of an index access path. Most Infor LN environments benefit if this parameter is set to a lower value. The value 10 is commonly used for Infor LN environments.

_always_semi_join

When Oracle queries that have an EXISTS/IN that has performance issues and the execution plan shows a HASH optimization in the execution path, the reason could be the setting of _always_semi_join. The default value for this parameter is CHOOSE and can be changed to NESTED_LOOPS for better performance.

_complex_view_merging

Oracle can move to an inline view when the _complex_view_merging parameter is set to TRUE. If this occurs, the parameter can be set to FALSE to disable this behavior. For Infor LN installations using Multi Language Enabling (MLE), this parameter should be set to TRUE.

_hash_join_enabled

Enables hash joins when set to TRUE. In hash joins, a hash table is created on the join key of the smallest table; it then joins the other tables to find the match. For Infor LN, it is recommended to set this value to FALSE because Infor LN breaks a lot of queries after a few fetches. For Cognos, the TRUE value can result in a better performance as it processes large ranges of data.

_optimizer_sortmerge_join_enabled

Sort merge joins perform better than nested loop joins for large data sets. However, it is recommended to set this value to FALSE because is most cases Infor LN only takes the first few rows of the resultset. When set to FALSE, the Oracle optimizer will choose nested loops instead of sort merge joins.

_optim_peek_user_binds

When set to TRUE, the optimizer uses the value of the bind variables when the query execution plan is compiled. This only occurs when no plan is available in memory. When this occurs with a full-range setting, the performance will be satisfactory for that query, but the next time a closed range is selected, the previously generated execution plan might not be most optimal. It can be the case that sessions that run fine, at times run slow with the same selection. To avoid bind variable peek, _optim_peek_user_binds can be set to FALSE. Note: The tkprof output can be misleading because EXPLAIN PLAN FOR is not reading what is actually in the cache for that query. For more information refer to the Oracle support knowledgebase article 387394.1.

Oracle Compression

Data can grow faster than expected, particularly when too much history or financial logging has been turned on. Implementing these parameters correctly will help reduce the data increase. However, existing data can be reduced by Oracle compression. Oracle compression has been available since Oracle 9.2 and improved in the latest Oracle 11g release. It is available starting from the Enterprise Edition level. The new ‘Advanced compression feature’ in Oracle 11g is licensed separately in addition to the Enterprise Edition. Tests have been performed and are documented in solution 22894468. These tests were based on the larger tables and indexes. It is recommended to keep Oracle compression to the specified tables only. According to Oracle, compression is far more about reducing I/O than it is about saving space.

To use Oracle compression, complete the following steps.

1 Determine used space and find indexes of table:

```
SQL> select segment_name, bytes
  2  from user_segments
```

```
  3  where segment_name like 'TTFGLD482100%';
```

This is always needed even if you do not want to compress indexes as indexes become unusable after compression.

2 Determine compression per index:

```
SQL> validate index ttfgld482100$idx1;
SQL> select opt_cmpr_count
  2  from   index_stats
  3  where  name = ‘TTFGLD482100$IDX1’;
```

3 Compress table:

```
SQL> alter table ttfgld482100 move compress FOR ALL OPERATIONS;
```

4 Compress indexes.

When compressing indexes, the output of step 2 is required:

```
SQL> alter index ttfgld482100$idx1 rebuild compress <#fields>;
```

5 Measure again.

Use the query as used in step 1.

Chapter 2 Tuning Infor LN for Oracle

2

This chapter describes the most important Infor LN performance settings and parameters.

Important Oracle db_resource parameters

In the db_resource file for Oracle database you must add these parameters:

```
ora_init:0101000
ora_max_array_fetch:2
ora_max_array_insert:1
oracle_home:/app/oracle/product/11g
oracle_sid:INFORLN
ora_default_tablespace:dataspace
ora_temporary_tablespace:temp
#following parameter should only be set on a new database or after
#converting the data to varchar
ora_use_varchar:1
```

In addition to the above parameters, other parameters can also be used. For example: Unicode specific NLS parameters need to be set.

ora_init

The resource ora_init defines several driver behaviors. Multiple behaviors can be selected by adding the octal values. For older porting sets, the default is 0011000. For porting set 8.4b onwards, it is 0101000. The activated and de-activated value descriptions are:

- **0001000** Use fixed char type (compulsory for Level 2).
- **0010000** Explicitly lock during inserts. This value is turned off to force implicit inserts. This saves queries, time, and space in the SGA.
- **0100000** Enable array interface. Enabling the array interface allows you to set the number of rows fetched and inserted in one block, by setting the variables ora_max_array_fetch and ora_max_array_insert.

Other values are:

- **0020000** Explicitly lock for updates (action: SELECT FOR UPDATE).
- **0040000** Explicitly lock for deletes (action: SELECT FOR UPDATE).

ora_max_array_fetch

If the array interface is enabled (by setting the ora_init 0100000 flag) the ora_max_array_fetch variable defines the maximum number of rows fetched immediately from the database to the driver. For best OLTP performance, it is recommended to set ora_max_array_fetch to 2 to avoid unnecessary process switches and network traffic.

A higher value of ora_max_array_fetch can be useful for some batch sessions or when making a dump of tables. The value of this variable must be the same as the value of rds_full, which is only applicable in case of running non-combo mode.

ora_max_array_insert

If the array interface is enabled (by setting the ora_init 0100000 flag), the ora_max_array_insert variable defines the maximum number of rows immediately inserted in the database from the driver. For optimal OLTP performance, it is recommended to set ota_max_array_insert to 1. A higher value of ora_max_array_insert can be useful for batch sessions or when uploading data. Uploading data with a value of 100 or higher can be up to 10% faster compared to a value 1.

retained_cursors

The resource retained_cursors sets the number of inactive cursors that must be retained in the list for reuse. After all rows have been fetched, the driver has a facility to put inactive cursors in Cancel state in a Cancel list, so that the inactive cursors can be assigned to a different query. However, a number of inactive cursors, in this list, are not available, and are defined by the resource retained_cursors, which defaults it to 50. If more than 50 cursors are in the cancel list, and a request for a new cursor is issued, the cursor that has been inactive the most is used. This cursor is disassociated from the original query and assigned to a new query, which does parsing and binding on this cursor. When the original query is running a re-execute, the driver detects the cursor has been associated with another query. It then gets a new cursor and re-parses and binds the query again. Increasing the value of retained_cursors leads to lesser re-parsing and rebinding of queries, which reduces usage of CPU resources. However, the number of open cursors and memory increases.

Usually, increasing retained cursors can help increase performance, but only a small effect has been observed for batch sessions. Therefore, this variable can be left to the default value for OLTP usage. An example for a batch setting is:

```
retained_cursors:300
```

ansi_outer_join

You can force the Infor LN Oracle driver to generate ANSI syntax queries, instead of the proprietary Oracle syntax, by setting the resource setting ansi_outer_join. According to Oracle, ANSI joins are

the future. This syntax is required when running with Multi Language Enabling (MLE). Also, running reporting tools might require the use of ANSI queries. When setting ansi_outer_join to 0, the proprietary syntax will be generated; when set to 1, the ANSI syntax will be generated. The difference can be easily seen when the following 3GL query is implemented:

```
SELECT tdsls401.orno
FROM   tdsls401, tcibd001
WHERE  tdsls401.item REFERS TO tcibd001
AS SET WITH 1 ROWS
```

This results in the following (Non-ANSI) Oracle query:

```
SELECT /*+FIRST_ROWS(1)*/ a0.t$orno
FROM   inforln.ttdsls401300 a0,inforln.ttcibd001300 a1
WHERE  a1.t$item (+) = a0.t$item
```

When running the query in ANSI mode, the following Oracle query is produced:

```
SELECT /*+FIRST_ROWS(1)*/ a0.t$orno
FROM  ( inforln.ttdsls401300 a0
LEFT JOIN inforln.ttcibd001300 a1 ON a1.t$item = a0.t$item)
```

This parameter can have a significant impact on performance. In general, customers running without MLE benefit from the proprietary syntax, setting ansi_outer_join to 0, while MLE customers benefit from the current default ANSI syntax. From portingset 8.4a onwards, the default setting for ansi_outer_join is 1.

first_rows_hint

The Infor LN Oracle driver generates for multiple queries, the FIRST_ROWS hint. Since Oracle 9i, it is recommended to use the FIRST_ROWS(N) hint where N is a number that specifies the amount of records expected to be returned. The first_rows_hint has been introduced to switch between 3 flavors:

```
first_rows_hint:1    old style FIRST_ROWS hint will be generated
first_rows_hint:2    new style FIRST_ROWS(N) hint will be generated

first_rows_hint:0    FIRST ROWS hints will be suppressed. Note: the benchmark failed
                     with timeouts on this setting.
```

Initially, the default was set to 1 for backward compatibility. This resource has been introduced in 6.1c.07.09, 7.1d.11, 7.6b, and 8.2b.

From porting sets 6.1c.07.14, 8.3a.02, and 8.4a onwards, the default is set to 2.

Note: this parameter can have a significant impact on performance. However, the use of this parameter varies according to customer requirements.

ora_timeout

The resource ora_timeout sets the timeout value in seconds when trying to set a lock. If the lock does not succeed or fails within this time, the driver aborts the query and the Infor LN application goes back to the db.retry point. The timeout can be set for 5 types of locks:

- Select for update
- Insert
- Update
- Delete
- Lock table

During implementations and in most live environments, a timeout of 30 seconds is sufficient. In some situations a higher value is recommended, such as, when batches face too many locking but need to finish and must not stop by error 107 or “Max retries (10) exceeded”. However, in these situations, increasing the ora_timeout values is not the best solution; it is better to find and solve the locking issue. From porting set 8.4a.01 and onwards, the default of this variable is set to the following:

```
ora_timeout:{30,30,30,30,30}
```

The variable ora_timeout can be set separately from lock_retry, but it is recommended to use this parameter and set lock_retry to 0. This variable must not be used on porting sets older than 8.4b on Windows database servers because this leads to a deadlock.

ora_use_varchar

This resource is available since porting set 8.6a. It is highly recommended to use varchar to reduce data(grow): By default, string columns in the tables will be created with a fixed data, when using varchar, the VARCHAR2 or NVARCHAR2 datatype will be used. Using varchar heavily reduces the amount of data storage and data growth of Infor LN in combination with Oracle. If you have an existing database, the database has to be rebuilt to take advantage of varchar. After converting the database or before creating a new database, the resource can be set to 1 to specify that varchar is used:

```
ora_var_char:1
```

It is highly recommended to use Oracle varchar with porting set 8.6a.01 and onwards, as an important change has been made in the 8.6a.01 porting set.

Converting from fixed length strings cannot be executed by only changing the datatype. Trailing spaces must also be removed and empty strings replaced with single space. Also, indexes, views and triggers must be recreated.

Oracle environment variables in the db_resource

Instead of specifying Oracle variables in the database definition, many of these variables can be added to the db_resource file. This saves space and memory, and is easier for an overview of used variables and maintainance. The following Oracle variables can be used:

- oracle_home
- oracle_sid
- two_task
- nls_lang
- nls_sort

Database storage and driver files

Database storage files contain parameters that influence the creation of tables and indexes. Only performance topics are discussed in this chapter. For more information on these files, consult the driver manuals.

Oracle storage file

Oracle data and indexes are stored in the lib/ora/ora_storage_param file. This file has the following format:

<table/module specification>:<company number>:<object type>:<storage parameters>

Usually, the storage parameters indicate the tablespace used for storage. Because Locally Managed Tablespaces are common, nothing more than specifying a tablespace is used for this file:

```
*:*:*:TABLESPACE LNSPACE
```

From a performance point of view, it does not matter if tables and indexes are stored in the same or different tablespaces. The disk layout is more important: is the I/O evenly spread across the disks, and is there an I/O bottleneck?

Oracle driver parameter file

The Oracle driver parameter file is located at lib/ora/ora_driver param. The layout of this file is:

<table/module specification>:<company number>:<object type>:<group>: <table/index optimization>:<refresh time>

By default, this is:

```
*:*:*:group:014:0:
```

Usually, these defaults should not be changed. However, an exception could be if you want to disable the creation of the statistics after the creation of the table. This can be done by changing the content into:

```
*:*:*:group:004:0:
```

Chapter 3 Tracing Oracle

3

In addition to the Infor LN trace possibilities, Oracle has other powerful tools to trace an application.

Monitoring the database via Oracle Enterprise Manager

Since Oracle 10 the Oracle Enterprise Manager (OEM) has been extended and improved to such a level that it is worth to use this for monitoring and tracing Oracle. The following tasks within OEM are worth in-depth investigation:

- The Performance tab in OEM shows a comprehensive overview of the system and database performance
- The Top Activity shows the most expensive queries and the total load generated by the queries.
- SQL Tuning Adviser helps to find a better execution plan.
- Snapshots. AWR and ADDM reports can be generated easily from here to find performance characteristics of a time window.
- SQL Monitor shows the most expensive queries, actual and estimated plan and costs.

The above tools are just the most used tools; also other powerful functions are built in OEM.

Tracing with SQL trace and TKPROF

The SQL trace facility displays all used queries and the output in a formatted way. In addition to the used queries, it is also possible to view the used execution plans.

The most important benefits of using SQL trace over ORAPROF in Infor LN are:

- Smaller dumps.
- Output includes evaluation plan.
- Output can be easily sorted in several ways.
- Number of used database blocks specified.

However, the disadvantage is that only Oracle database query times are displayed. If the network or connection between the driver and database is a bottleneck, it cannot be found with a SQL trace.

More information about this tool can be obtained from the Oracle knowledge base article 980711.1: *How to use SQL Trace and TKPROF for performance issues.*

Preparations for SQL trace

Before SQL traces can be started, you must check some information:

Statistics level

The Oracle setting STATISTICS_LEVEL specifies the level of collection for database and operating system statistics. Setting this value to ‘ALL’ gives more trace information. It is advised to set this to ‘ALL’ only on session level (with e.g. the resource ora_alter_session:all) or temporary on system level.

The trace destination directory

The directory where the traces will be placed can be specified with the parameter USER_DUMP_DEST. This parameter is normally specified in the active configuration file. It is advised to use a directory with enough free space. This parameter can also be changed online with the following command:

```
SQL> ALTER SYSTEM SET USER_DUMP_DEST = <newdir>;
```

The maximum trace size

The maximum trace size can be specified in system blocks in the variable MAX_DUMP_FILE_SIZE. The default value for this parameter is 500. For example, if the block size of the USER_DUMP_DEST is 2 Kb, the maximum dump size will be 1 Mb, not sufficient to trace a batch job. Therefore, it is recommended to increase this parameter to 5000 or UNLIMITED. If the trace file is truncated, it indicates that this parameter must be increased or that the USER_DUMP_DEST disk is full. This parameter can be changed online with the following command:

```
SQL> ALTER SYSTEM SET MAX_DUMP_FILE_SIZE = <new size>;
```

File permissions

A trace file is created with Oracle user as the owner. On UNIX systems, it will not be possible to read these files, if the user is not a member of the same group. Therefore, you need permissions to read these files. An alternative is to set the following parameter in the active INIT.ORA file:

```
trace_files_public = true
```

This parameter ensures permissions will be extended such that the file can be read by everybody. This is only required if users that do not have the right permissions to access the trace files.

Create EXPLAIN TABLE

During the formatting of the trace output, the table PLAN_TABLE will be created if the *Explain* option is used. Therefore, a valid Oracle user who is able to create a table is required, for example: the owner of the ERP tables. The PLAN_TABLE can be created by running the UTLXPLAN.SQL script, which is usually located in the rdbms/admin directory of the ORACLE_HOME.

Statistics on tables

Oracle statistics must be generated with the dbms_stats package. More information on statistics can be found in the section “Creating statistics”,

Permissions to start tracing

The Oracle user that is used for the tracing requires the “ALTER SESSION” privilege. This can be granted with these commands:

```
SQL> GRANT ALTER SESSION TO bsp;
```

In the above case the privilege is granted to the user bsp.

Or:

```
SQL> GRANT ALTER SESSION TO r_inforln;
```

Grants the privilege to the role for the user inforln.

Tracing with SQL trace

Usually, a SQL trace will be started at the beginning of the application. This can be done in several ways:

- Set SQL_TRACE=true in the command line such as:
  ```
  -set SQL_TRACE=true
  ```
- Set sql_trace:true in the db_resource. This is not recommended because all users will be traced.
- Set SQL_TRACE=true in the database. This is not recommended because all users will be traced.

It is recommended to trace the sessions one by one, and to close the bshell because the entire output will be in one trace file which makes it difficult to interpret.

If the startup results in a BDB error 2031, with reflects ORA-0131, this indicates that the user is not allowed to run ALTER SESSION in Oracle. This can be resolved by an Oracle DBA with sufficient rights who can grant the user the required rights as mentioned in the above paragraph.

Tracing running processes is also possible. This can be done with the following command:

```
SQL> EXECUTE dbms_system.set_sql_trace_in_session(SID, SERIAL#);
```

Here, the SID and SERIAL# can be gathered from V$SESSION:

```
SQL> SELECT sid, serial#
  2> FROM   v$session
  3> WHERE  osuser = 'bsp';
```

If more information of the Oracle user is known, for example: the Process ID, the query can be extended with that information.

It can be difficult to find the right process. The following info will help in selecting the best process:

- If an Infor LN session stops, this command indicates if any SQL activity is being run.
- If an Oracle connection consumes most of the CPU power.
- If part of a session must be traced.

To format the trace output

When finishing a trace, the output is placed in the value of the USER_DUMP_DEST parameter. The file has the following name:

- ORA_<Process ID>.trc on UNIX systems.
- ORA<Tread ID>.trc on Windows systems.

The trace file can be formatted with the TKPROF utility. In most situations, the following options will display required information of this tool:

```
tkprof <trace file> <output file> explain=<user>/<password> sort=<sort option>
```

tracefile

The name of the tracefile, which can be specified without “.trc” extension.

outputfile

The name of the file, where the output must be written. If no extension is given, the ".prf" extension is added.

explain

Determines the execution plan for each SQL statement in the trace file and writes these plans to the output file. TKPROF determines execution plans by issuing the EXPLAIN PLAN statement after connecting to Oracle with the *user* and *password* specified in this parameter. The specified *user* must have CREATE SESSION system privileges. TKPROF takes longer to process a large trace file, if the EXPLAIN option is used.

It is possible that the shown execution plan is different to the one really used due to settings that are changed by logging on (ALTER SESSION statements etc). To see the correct plan, it is advised to monitor with the SQL Monitor option in Oracle Enterprise Manager.

sort

Sorts traced SQL statements in descending order of specified sort option before listing them into the output file. If more than one option is specified, the output is sorted in descending order by the sum of the values specified in the sort options. Most user sort options are:

- FCHELA Elapsed time spent on fetches.
- EXEELA Elapsed time spent on executes.
- PRSELA Elapsed time spent on parsing.

Other sort options can be found by running the tkprof command, without the options.

table

When the default explain table does not exist, tkprof can use another table for storing the explain plan.

The output shows the expected application queries and other system queries; these queries are needed to prepare the application query. When looking for a bad performing application query, the system queries can be suppressed with the following action in the tkprof command:

```
sys=no
```

An example of running tkprof is:

```
$ tkprof ora_08456 tkprof_08456 sort=(prsela,exeela,fchela) sys=no
```

To interpret the formatted output

The most difficult part of the trace is the interpretation of the trace output. However, the tkprof utility makes it much easier to interpret the output. Before locating the problem, you must understand how tkprof formats the output.

This figure shows a part of an example trace:

```
SELECT /*+ FIRST_ROWS INDEX(b ttipcf310120$idx1)  */ a.t$preq,a.t$expl,
  b.t$mitm,b.t$pono,b.t$sern,b.t$cnsc,b.t$sitm,b.t$dsca,b.t$leng,b.t$widt,
  b.t$noun,b.t$qana,b.t$scpf,b.t$cwar,b.t$opno,b.t$cpha,b.t$exin,b.t$nnts,
  b.t$ltmo,b.t$indt,b.t$exdt,b.t$pqan,b.t$pper,b.t$txta
FROM
 inforln.ttibom010120 a,inforln.ttipcf310120 b WHERE b.t$mitm = :1 AND (b.t$mitm =
 :2 AND b.t$pono = :3 AND b.t$sern > :4) ORDER BY 3,4,5


call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse       53      0.01       0.02          0          0          0           0
Execute    287      0.63       0.73          0          0        136           0
Fetch      287    198.00     238.58     930456    1070896       1632         816
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      627    198.64     239.33     930456    1070896       1768         816

Misses in library cache during parse: 1
Optimizer goal: FIRST_ROWS
Parsing user id: 613

Rows     Row Source Operation
-------  ---------------------------------------------------
      0  FILTER
      0   MERGE JOIN CARTESIAN
      2    TABLE ACCESS BY INDEX ROWID TTIPCF310120
      2     INDEX RANGE SCAN (object id 178354)
      0    SORT JOIN
      0     TABLE ACCESS FULL TTIBOM010120
```

In the figure above, the query is displayed first. Below the query, the header supplies common information on how certain columns must be interpreted. Several rows are made for *Parse*, *Execute*, and *Fetch*.

The functions of the rows are:

- **Parse**
  Prepares the SQL statement. It checks if the table(s) and the required column(s) exists and makes the execution plan.
- **Execute**
  Performs the statement. For `INSERT`, `UPDATE`, and `DELETE` statements, this modifies the data. For `SELECT` statements, this identifies the selected rows.
- **Fetch**
  Retrieves rows returned by a query. Fetches are only performed for `SELECT` statements.

The columns for Parse, Execute, and Fetch are:

- **Count**
  The number of times a SQL statement has been parsed, performed, or fetched.
- **Cpu**
  The total CPU time in seconds for all parse, execute, or fetch calls for the statement.
- **Elapsed**
  The total elapsed time in seconds for all parse, execute, or fetch calls for the statement.
- **Disk**
  The total number of data blocks physically read from the data files on disk for all parse, execute, or fetch calls.
- **Query**
  The total number of buffers retrieved in consistent mode for all parse, execute, or fetch calls. Buffers are usually retrieved in consistent mode for queries.
- **Current**
  The total number of buffers retrieved in current mode. Buffers are retrieved in current mode for statements such as `INSERT`, `UPDATE`, and `DELETE`.
- **Rows**
  The total number of rows processed by the SQL statement.

If the columns *cpu* and *elapsed* are always zero, it is advised to check if *timed_statistics* was set to *true*. If yes, then the most probable cause is the minimum resolution of timing being 1/100 of a second.

Below the columns is the execution plan, as shown in this figure.

```
Rows     Execution Plan
-------  ---------------------------------------------------
      0  SELECT STATEMENT   GOAL: HINT: FIRST_ROWS
      1   FILTER
      0    NESTED LOOPS (OUTER)
      1     NESTED LOOPS (OUTER)
      1      NESTED LOOPS (OUTER)
      2       TABLE ACCESS   GOAL: ANALYZED (BY INDEX ROWID) OF
                  'TFMFOC201570'
      0        INDEX   GOAL: ANALYZED (RANGE SCAN) OF
                   'TFMFOC201570$IDX6' (UNIQUE)
      0       TABLE ACCESS   GOAL: ANALYZED (BY INDEX ROWID) OF
                  'TFMFMD100570'
      0        INDEX   GOAL: ANALYZED (UNIQUE SCAN) OF
                   'TFMFMD100570$IDX1' (UNIQUE)
      0      TABLE ACCESS   GOAL: ANALYZED (BY INDEX ROWID) OF
                 'TFMFOC200570'
      0       INDEX   GOAL: ANALYZED (UNIQUE SCAN) OF
                  'TFMFOC200570$IDX1' (UNIQUE)
      0     TABLE ACCESS   GOAL: ANALYZED (BY INDEX ROWID) OF
                'TTCIBD001570'
      0      INDEX   GOAL: ANALYZED (UNIQUE SCAN) OF
                  'TTCIBD001570$IDX1' (UNIQUE)
```

The execution plan is the plan the Oracle optimizer used to locate the result set. The output must be read from the most right statement to the most left statement. From the first indent statement to the right, the tables are mostly written below that point and the standard way of using the data is written above that point:

When searching through the trace file, follow these guidelines:

- It is important to look for queries that are used often.
- When a query is parsed several times, it is important to check if the shared pool is large enough.
- Reading many blocks is not recommended, and reading blocks from disk must be avoided. However, if many blocks need to be read from disk, check if the buffer cache hit ratio is high enough. For more information, refer to database specific manuals.

Sometimes Oracle jumbles the trace file and the wrong table names are mentioned in the execution plan. This can be the case as several sessions write to the same tracefile. If this occurs on a query that requires more investigation, that query can be extracted from the large trace using the tkprof and trcsess command:

Add session id's to the trace file

```
$ tkprof erp1_ora_20077 no_aggregate aggregate=no
```

Find the problem query in the new trace file and write down the session ID. Make a tracefile for only this session ID:

```
$ trcsess session=1381.609 output=partly_trace.trc no_aggregate.trc
```

Format the trace file as you would the original tracefile:

```
$ tkprof partly_trace partly_trace sort=exeela,prsela,fchela
```

Tracing with event 10046

When you have to trace a session using a query of another user, such as a long running batch job, or to get more trace information, event 10046 will help. This paragraph describes this event on a higher level. More information can be obtained using Oracle Support.

Event 10046 has the following trace levels:

Level	Description
1	Emit statistics for parse, execute, fetch, commit, and rollback. Same as when setting SQL_TRACE=true
2	Unused.
4	Emit values for SQL bind variables.
8	Emit statistics for wait events as listed in v$event_name

These levels can be combined, but every non-zero value will include Level 1 tracing. So, Level 12 is the same as Level 1, 4 and 8. For only query tracing, Level 1 supports sufficient information, and the information of Level 4 and 8 will not be summarized correctly by tkprof. To interpret information from Level 4 and 8 traces, scripts must be written; otherwise the raw output has to be interpreted.

Level 4 output can be useful to reproduce bad queries by Oracle tools as SQL*Plus. The output of Level 8 is very useful to check locking contention and weak areas of general performance.

Activating and deactivating the trace

The 10046 trace can be activated from the Infor LN UI command prompt:

```
-set ORA_ALTER_SESSION="set events '10046 trace name context forever, level 8'"
```

Or the trace can be activated by finding the SID and SERIAL# from V$SESSION of the session you want to trace. For these values, the following commands must be used:

```
exec sys.dbms_system.set_bool_param_in_session(SID, SERIAL#, 'timed_statistics', true)
sys.dbms_system.set_int_param_in_session(SID, SERIAL#, 'max_dump_file_size',
2147483647)
exec sys.dbms_set_system.set_ev(SID, SERIAL#, 10046, 1, '')
...
exec sys.dbms_set_system.set_ev(SID, SERIAL#, 10046, 0, '')
```

Output of this trace is available at the same location and format as the SQL_TRACE trace.

Tracing with event 10053

Tracing with event 10053 provides information on how Oracle decided to choose the execution plan as found by tkprof. All optimizer permutations are displayed, including the cost and other information. This event has only 1 level: setting it to 1 activates it, setting it to 0 turns it off. The previous paragraph explains how to activate or deactivate the tracing.

For more information refer to the Oracle support knowledge base 225598.1: *How to obtain tracing of optimizer computations (EVENT 10053).*

Appendix A Useful scripts

A

To format the output of ORAPROF

The output of the Oracle level 2 driver can be large, and is not readable with every editor. The script below formats the output in a readable format. The output has the following columns:

- **QID**: The query ID used in this script, makes searching for the related query later in the output, easier.
- **Table**: The first (mostly the main) table in the query.
- **Command**: Lists the following type of queries:
  - select
  - for update
  - update
  - insert
  - delete
- **Count**: The number of times the query has been parsed, executed, or fetched.
- **Time**: The total amount of time the query has been used for the parse, execute, or fetch.
- **Start Time**: The first time the query has been found in the trace file.

In the original output, each query is placed in single lines. To make the output more readable, the queries are wrapped, if the output becomes longer than 65 characters. An example of the output is:

```
                                Parse        Exec         Fetch
QID Table            Command Count  Time  Count  Time  Count  Time Start Time
  7 ttfacp610600 Select       989   1.24  1978   1.44  1978   1.94 11:35:30.750
  5 ttfacp610600 For Update     1   0.00   990   1.03   990   0.65 11:35:30.680
  4 ttfacp610600 Select         1   0.00     3   0.00     3   0.02 11:35:30.650
  1 tttadv999000 Select         0   0.00     2   0.00     2   0.01 11:35:23.450
  3 tttadv999000 Select         1   0.00     2   0.00     2   0.01 11:35:23.520
  2 tttadv112000 Select         0   0.00     1   0.00     1   0.00 11:35:23.480
  6 ttfacp610600 Update         1   0.00   990   1.00     0   0.00 11:35:30.690

Query : 1
SELECT /*+ index(a tttadv999000$idx1) */ t$pacc,t$keyr,t$desc,t$Refcntd,
t$Refcntu FROM inforln.tttadv999000 a WHERE t$pacc=:1 AND t$keyr=:2

Query : 2
SELECT /*+ FIRST_ROWS INDEX(a tttadv112000$idx1)  */
```

```
a.t$pacc,a.t$cpac, a.t$sequ,b.t$mess,b.t$expi,b.t$za_mtyp FROM inforln.tttadv112000
a,inforln.tttadv450000
b WHERE b.t$clan = :1 AND b.t$cpac = :2 AND b.t$cmes = :3 AND b.t$vers
= a.t$vers AND b.t$rele = a.t$rele AND b.t$cust = a.t$cust AND a.t$pacc
= :4 AND a.t$cpac = :5 ORDER BY 1,2,3
```

An example of the script is:

```
if [ $# -ne 2 ];then
   echo "Usage $0: <inputfile> <outputfile>"
   exit
fi

Awk=awk
[ -x /usr/xpg4/bin/awk ] && Awk=/usr/xpg4/bin/awk
[ -x /usr/bin/nawk ] && Awk=/usr/bin/nawk


fold -w 800 $1 |\
$Awk 'BEGIN{
   MaxQID=0
   MAXLENTGH=65
}
{
   if (substr($1,1,5)=="------")
      getline
   if (substr($1,1,1)=="<") {
      split($0,t0,":")
      split(t0[2],t1,"[")
      split(t0[4],t2,"]")
      StTime=t1[2] ":" t0[3] ":" t2[1]
      getline
   }
   if ($2=="(parse)") {
      Time=$4
      line=NR
      getline
      getline
      getQID()
      ParseTotal[QID]+=Time
      ParseCount[QID]++
      next
   }
   if ($2=="(multi_exec)") {
      Time=$4
      getline
      getline
      getQID()
      ExecTotal[QID]+=Time
      ExecCount[QID]++
      next
   }
   if ($2=="(multi_fetch)") {
      Time=$4
      getline
      getline
      getQID()
```

```
      FetchTotal[QID]+=Time
      FetchCount[QID]++
      next
   }
}
function getQID()
{
   i=0
   while (substr($0,1,5)!="-----" && $0!="" ) {
      Line[++i]=$0
      getline
   }
   for (j=MaxQID;j>0;j--) {
      for (k=i;k>0;k--)
         if (Line[k]!=InLine[j,k])
            k=-1
      if (k==0)
         j=-j
   }
   MaxInLine=i
   if (j==0) {
      MaxQID++
      MaxLength=MAXLENTGH
      k=1
      for (j=1;j<=i;j++) {
         TmpLine=Line[j]
         InLine[MaxQID,j]=Line[j]
         while (length(TmpLine)>MaxLength) {
                SQLLine[k]=SQLLine[k] substr(TmpLine,1,MaxLength)
            TmpLine=substr(TmpLine,MaxLength+1)
            l=index(TmpLine," ")
            m=index(TmpLine,",")
            if (m==0) m=999
            if (l==0) l=999
            if (m==999 && l==999) {
               SQLLine[k]=SQLLine[k] substr(TmpLine,1)
               TmpLine=""
            }
            else if (m>l) {
                  SQLLine[k]=SQLLine[k] substr(TmpLine,1,l)
               TmpLine=substr(TmpLine,l+1)
            }
            else {
                   SQLLine[k]=SQLLine[k] substr(TmpLine,1,m)
               TmpLine=substr(TmpLine,m+1)
            }
            MaxLength=MAXLENTGH
            k++
         }
         if (length(TmpLine)!=0) {
            SQLLine[k]=SQLLine[k] substr(TmpLine,1)
            MaxLength=MAXLENTGH-length(SQLLine[k])
         }
      }
      NrLines=k
      QID=MaxQID
```

```
      for (j=1;j<=NrLines;j++)
         SQL[QID,j]=SQLLine[j]
      Stat="Select"
      s=substr(SQLLine[1],1,6)
      u=substr(SQLLine[NrLines],length(SQLLine[NrLines])-5)
      if (s=="INSERT") Stat="Insert"
      if (s=="UPDATE") Stat="Update"
      if (s=="DELETE") Stat="Delete"
      if (Stat=="Select") {
         i=1
         while (index(SQLLine[i],"FROM")==0 && i<=NrLines)
            i++
         a=index(SQLLine[i],"FROM")
         if (length(SQLLine[i])==a+4) {
            b=SQLLine[++i]
         }
         else
            b=substr(SQLLine[i],a+4)
         split(b,c)
         split(c[1],d,".")
      }
      else if (Stat=="Insert") {
         split(SQLLine[1],c)
         split(c[3],d,".")
      }
      else
      {
         split(SQLLine[1],c)
         split(c[6],d,".")
      }
      if (u=="UPDATE") Stat="For Update"
      Length[QID]=NrLines
      MainTable[QID]=d[2]
      Status[QID]=Stat
      StartTime[QID]=StTime
      for (j=1;j<=NrLines;j++)
         delete SQLLine[j]
   }
   else
      QID=-j-1
}
function swap(A, i) {
   t=A[i-1]
   A[i-1]=A[i]
   A[i]=t
}
END {
   for (i=1;i<=MaxQID;i++) {
      Q[i]=i
      for(j=i;j>1 && FetchTotal[j-1]<FetchTotal[j];j--) {
      swap(MainTable, j)
      swap(Status, j)
      swap(ParseCount, j)
      swap(ParseTotal, j)
      swap(ExecCount, j)
      swap(ExecTotal, j)
      swap(FetchCount, j)
```

```
      swap(FetchTotal, j)
      swap(StartTime, j)
      swap(Q, j)
      }
   }
   print "             Parse          Exec      Fetch"
   print "QID Table     Command Count  Time  Count  Time  Count  Tim
e Start Time"
   for (i=1;i<=MaxQID;i++) {
      printf("%3d %s %-10s %5d %6.2f %5d %6.2f %5d %6.2f %s\n",
         Q[i], MainTable[i], Status[i], ParseCount[i],
         ParseTotal[i], ExecCount[i],  ExecTotal[i],
         FetchCount[i], FetchTotal[i], StartTime[i])
   }
   for (i=1;i<=MaxQID;i++) {
      print
      print "Query :",i
      for (j=1;j<=Length[i];j++)
      print SQL[i,j]
   }
}' $1
```