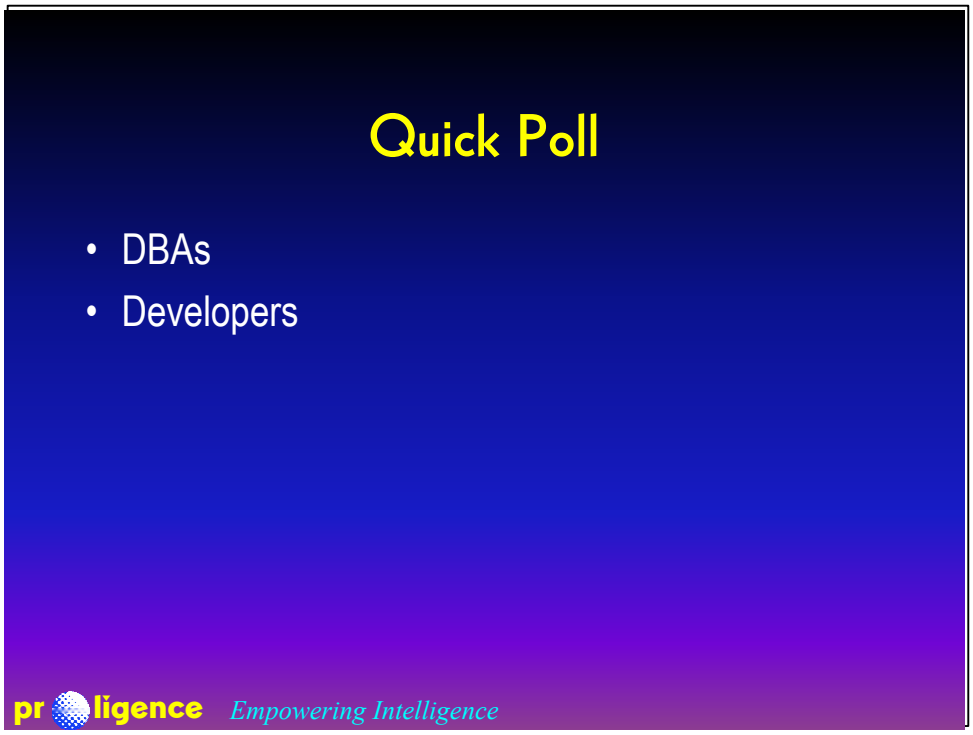
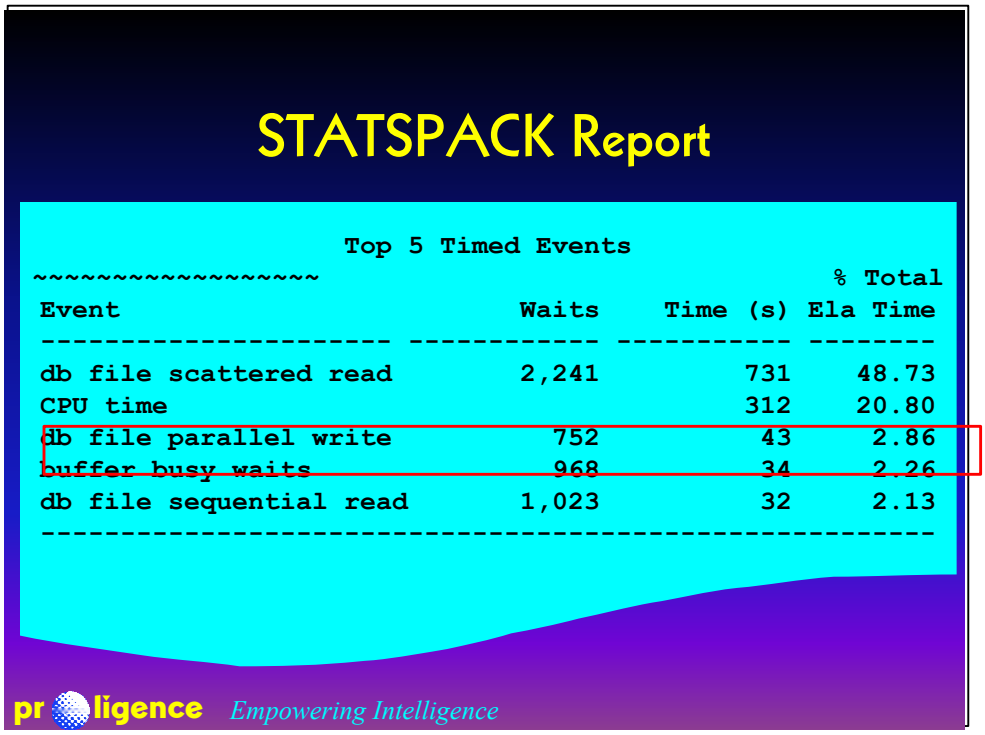
Digging Deeper

Arup Nanda
Proligence, Inc.

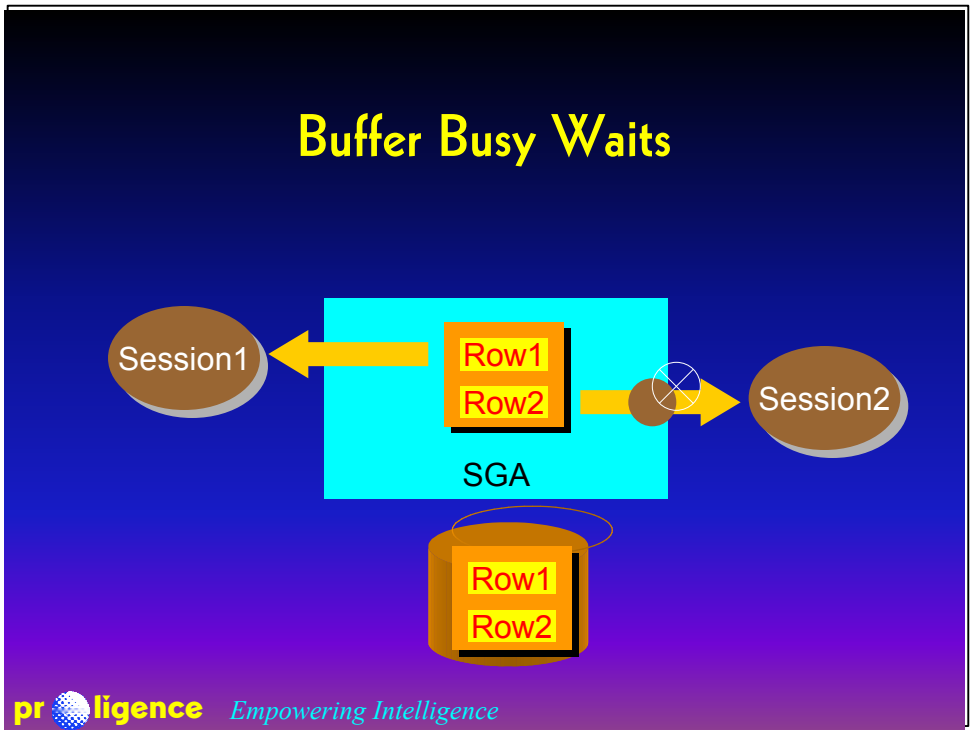
Empowering Intelligence



This is primarily for DBAs for some specific areas are of interest to developers, too.



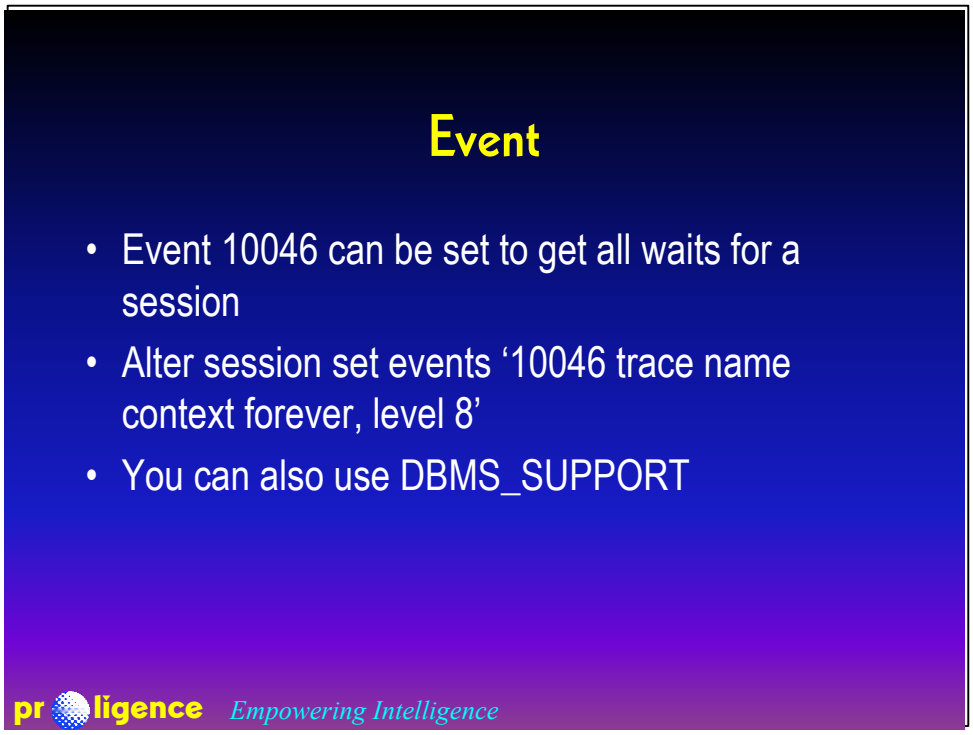
The STATSPACK report does provide a good snapshot of problems in the database, but at a very macro level, not at the segment level. For instance the excerpt above shows the wait events summarized for the whole instance. It does not show which segments suffered this.



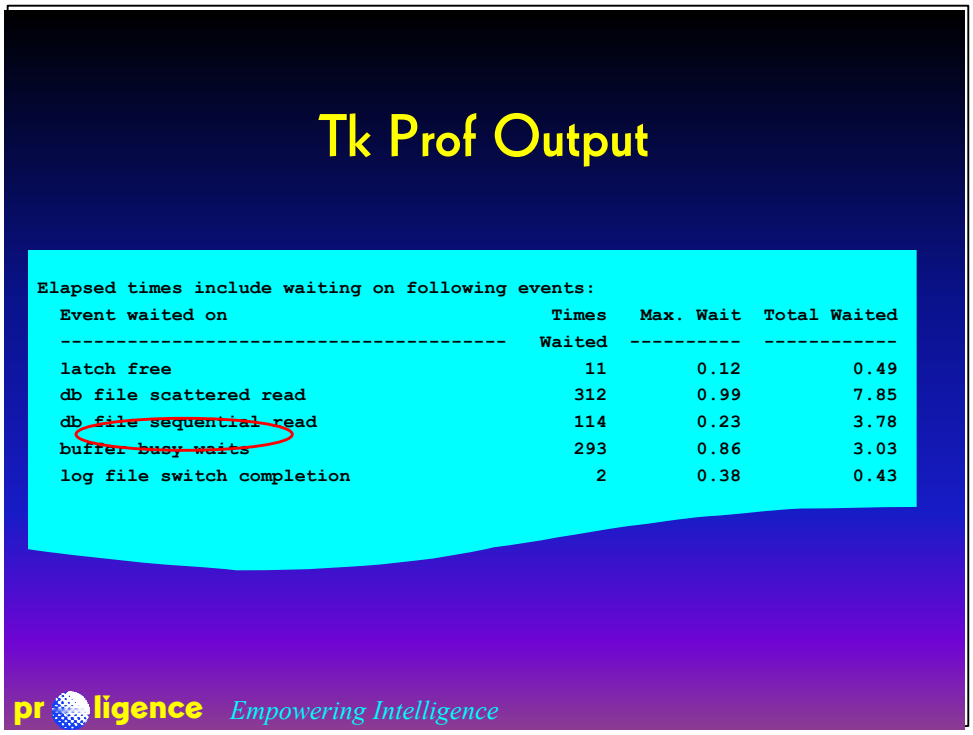
Let's examine one of the statistics, buffer busy waits. The common cause of the wait event is the simultaneous access of the same buffer by two sessions.

Session 1 requests the update of Row1. The entire block containing the row Row1 is loaded into the SGA. The block also contains the row Row2. At the same time as the Session1, another session Session2 is trying to update Row2. Since the block containing Row2 is already in the SGA, the server process of the session will try to update the block. But since the session Sesssion1 is modifying the block, it has to wait, creating a buffer busy wait scenario.

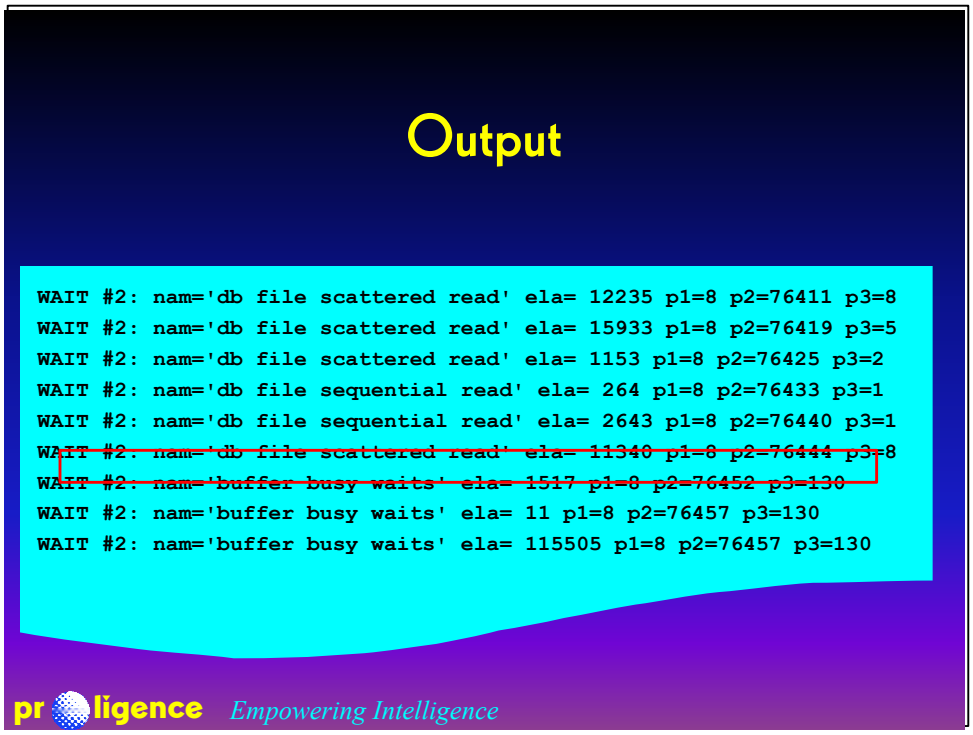
The problem can be alleviated by packing the segment less so that the a buffer contains less number of rows, making the it less likely that the buffer will be a contention point for access. The resolutions is thus to reorganize the segment that contributed to the problem. But, what segment? STATSPACK report does not provide the details of the segments.



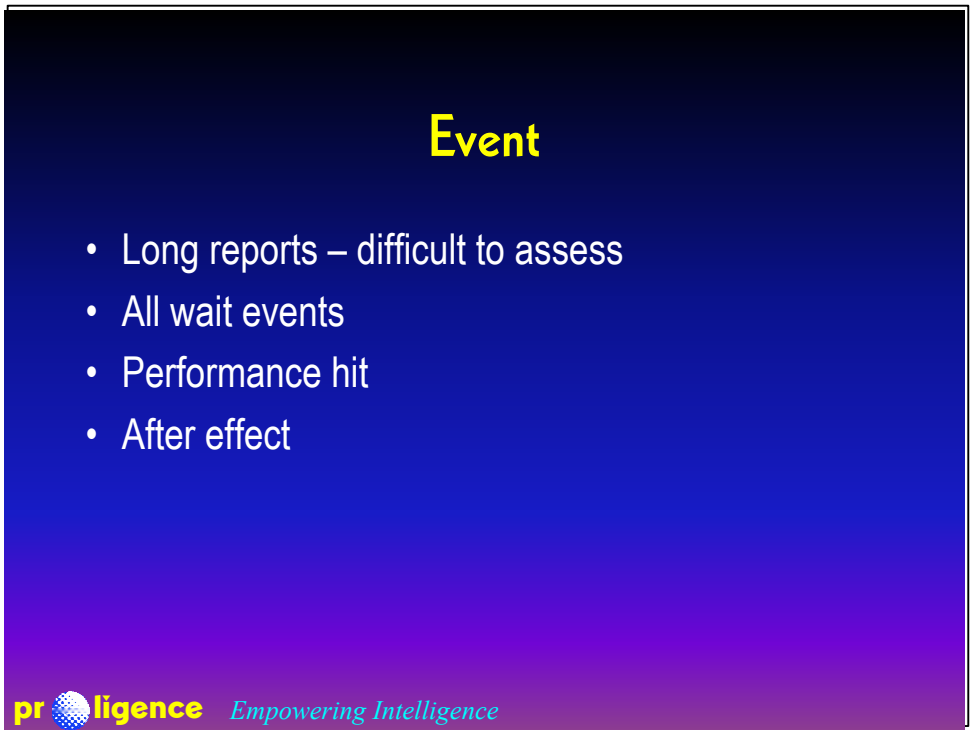
The segments can still be obtained through some other techniques, e.g. by setting an event 10046. This can be done via alter session and via dbms_support.



The trace file can be fed through tkprof to produce a report; but the report still does not report the individual segments.



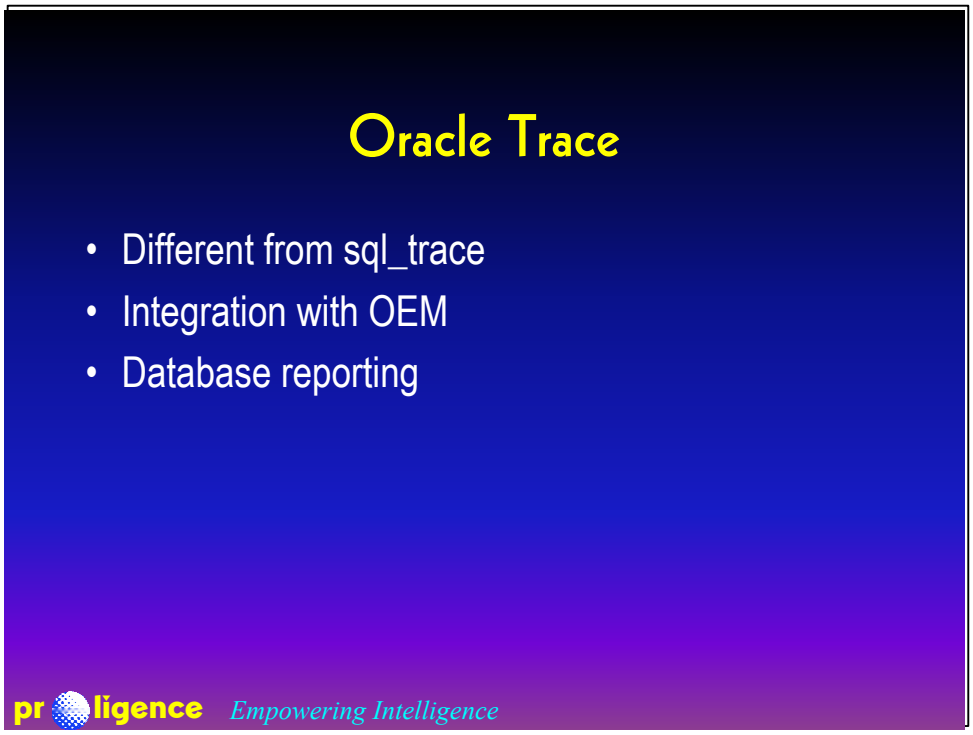
This event produces a very long text file in the user_dump_destination directory. A small excerpt of the file is shown here. This shows the wait event and the corresponding p1 and p2 values that can be used to get the segment name.



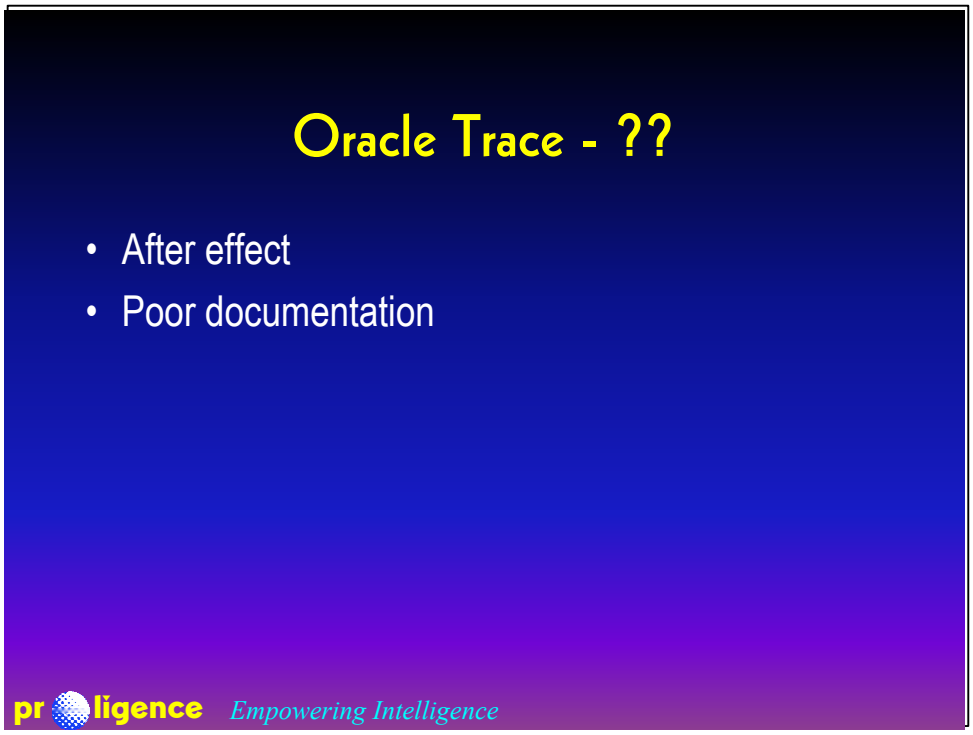
So, the event does provide the segment names but does not provide in a manner that is useful in diagnosis. It's very long and a quick assessment is difficult.

Also, this report contains information on all the wait events the session has experienced, not just the ones we are interested in. This makes the report somewhat indeterministic.

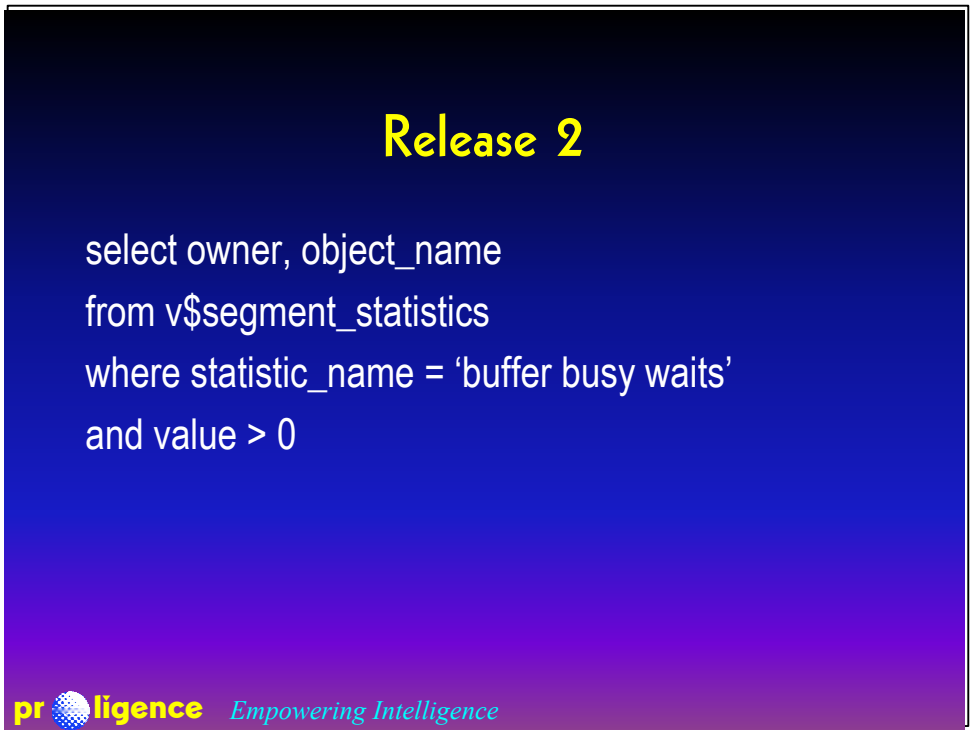
The report can be run after a problem has occurred. When the problem did occur the report was non-existent. The use of the event assumes that the problem could be reproduced at will; not necessarily a valid one.



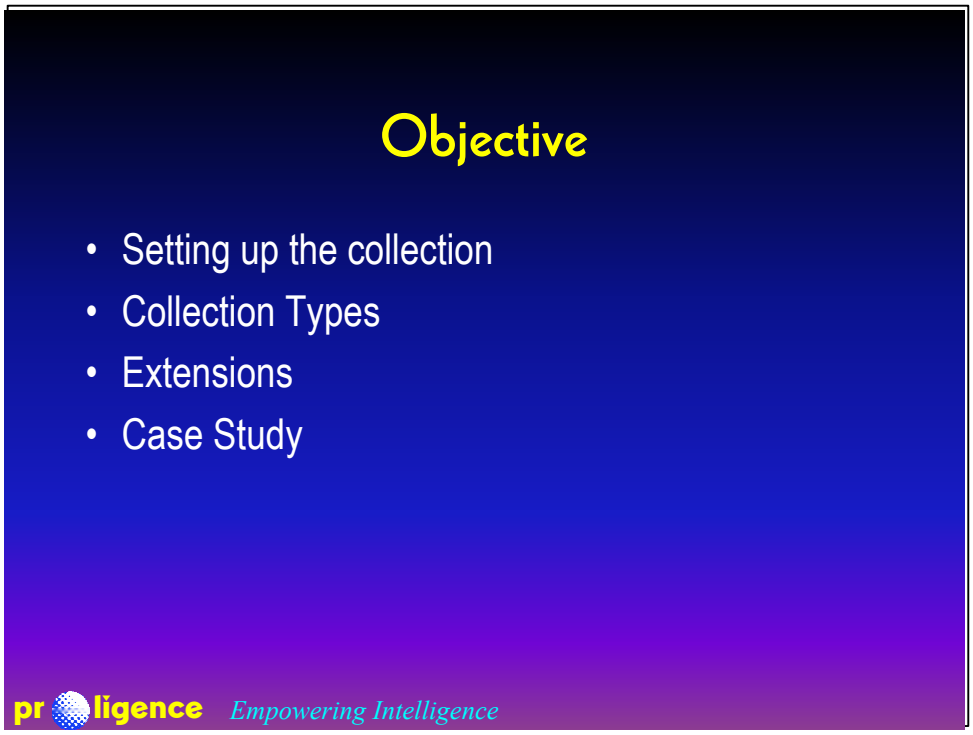
The other option is Oracle Trace. This is different from SQL Trace. This can be set at the database level to capture wait events that can be formatted to the database tables or reports. The stats can be analyzed via OEM. A very useful tool, but this is obscured by lack of documentation and general ignorance.



Still, this assumes that the effect can be reproduced, may not be true.

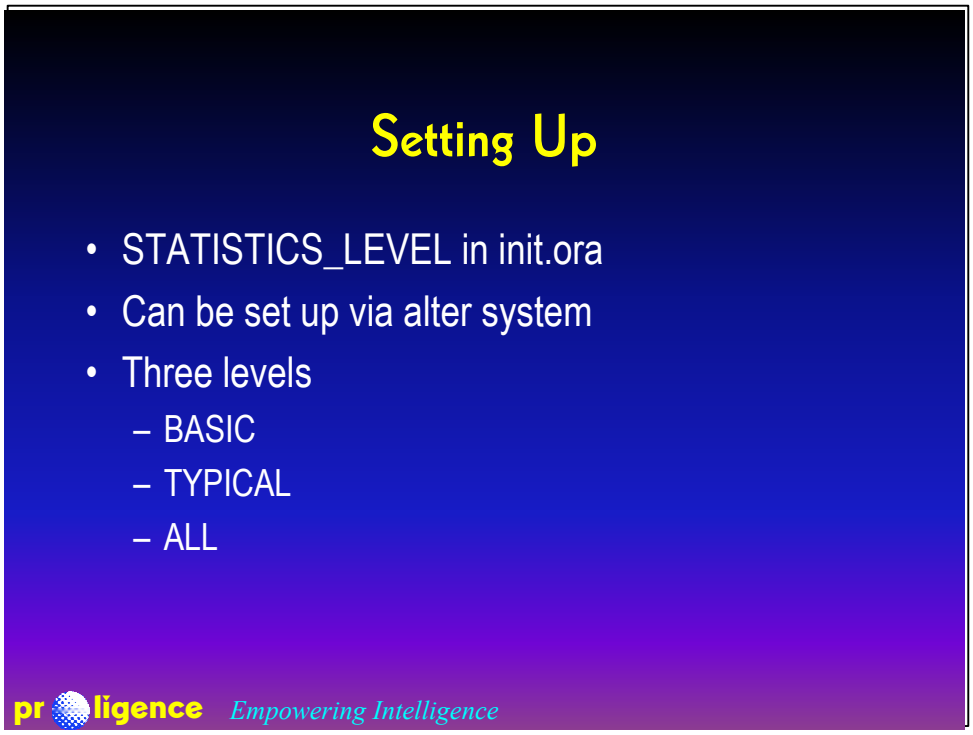


In Oracle 9i Release 2, the solution is simple. In order to find out the segments that experienced the buffer busy waits, just issue this query.



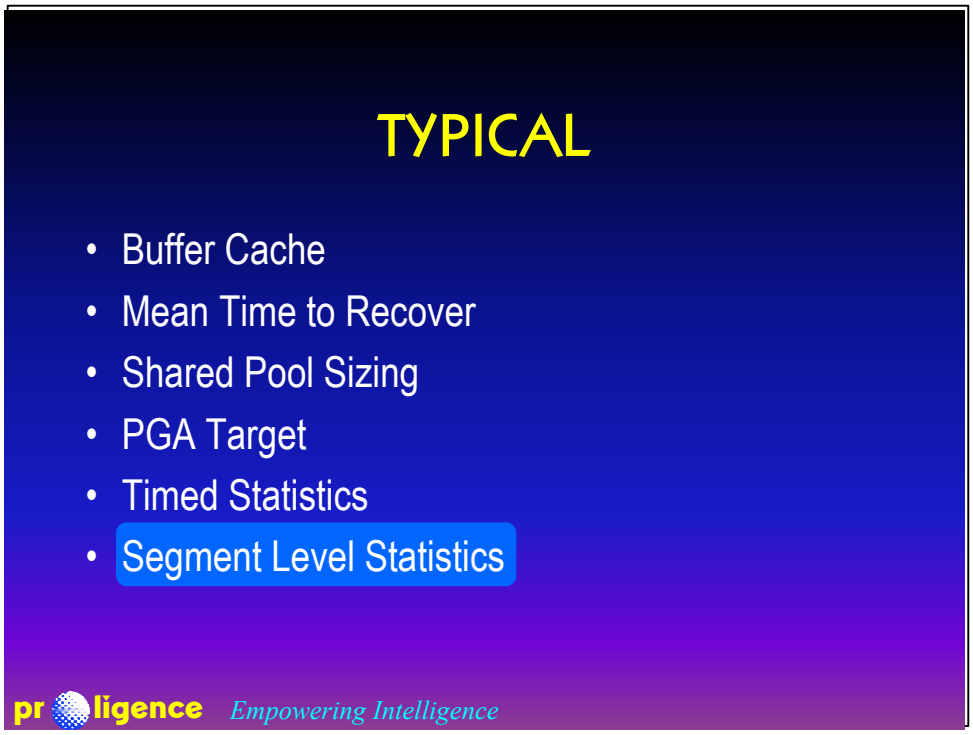
In this presentation, we will set up the collection and study the results to identify the problem objects.

The author has extended the scope of the view that Oracle provides. We will study that. Finally, we will cement our learning via a case study.

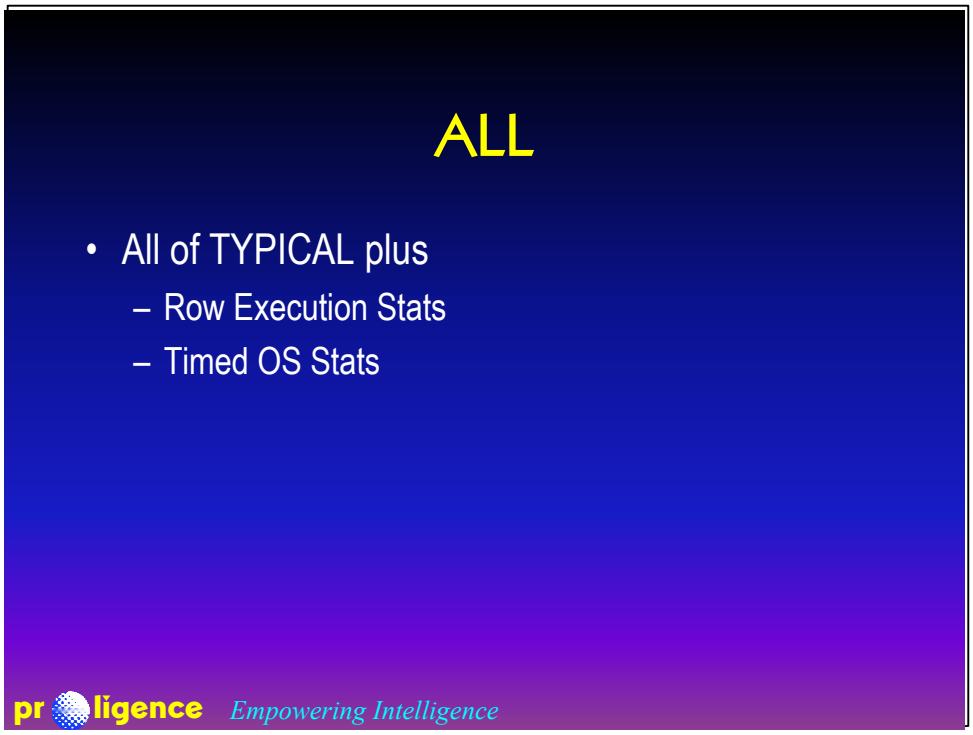


An init.ora parameter called STATISTICS_LEVEL needs to be set for collection of the stats to occur. There are three values for the parameter – BASIC, TYPICAL and ALL. This parameter can also be set via ALTER SYSTEM.

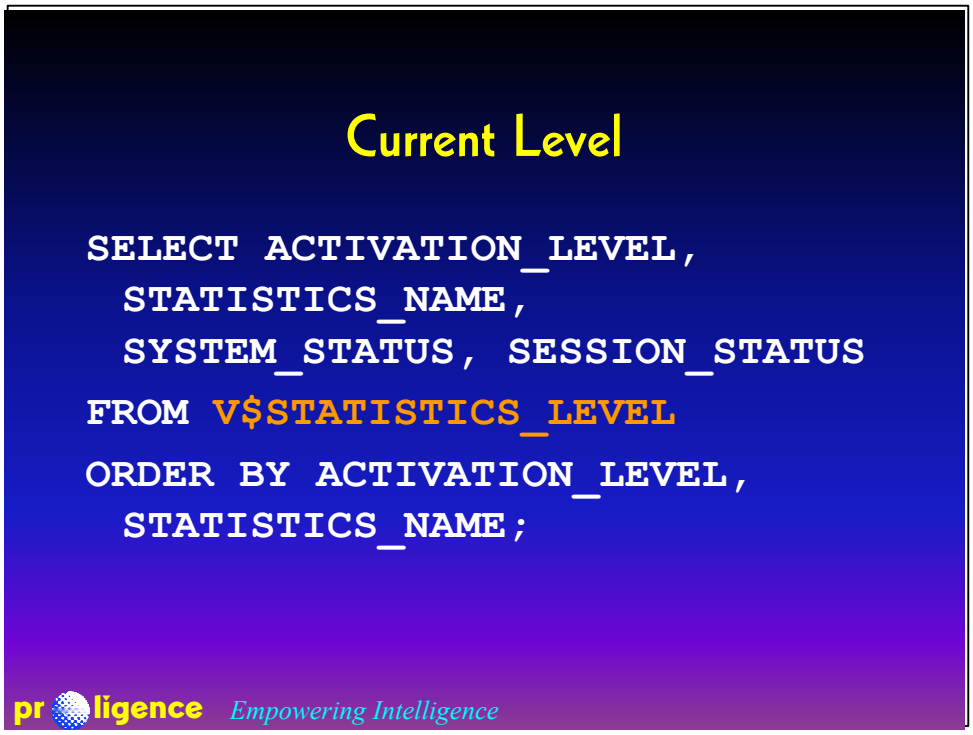
At the setting of BASIC, nothing is collected at all.



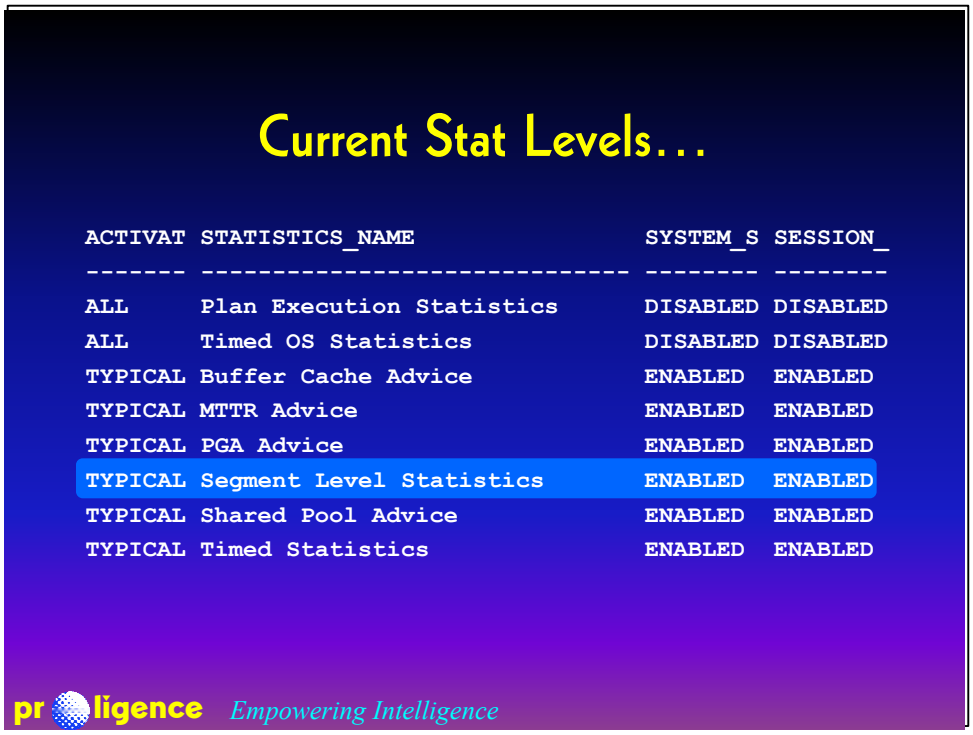
In the TYPICAL setting the above stats are collected.



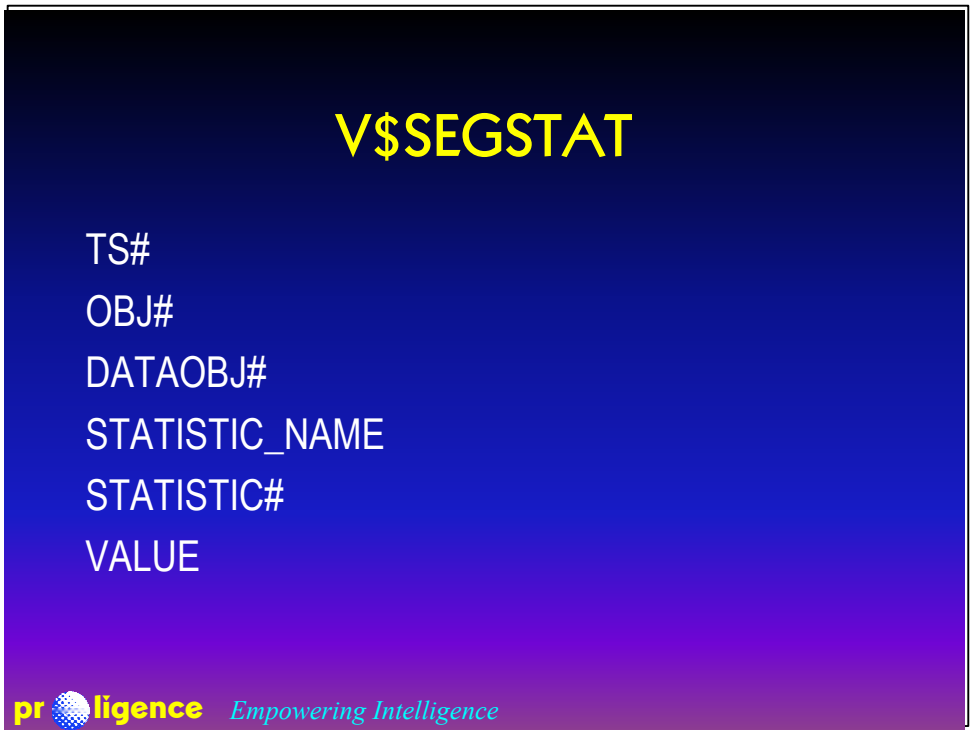
In ALL setting, all of TYPICAL and the above two are collected.



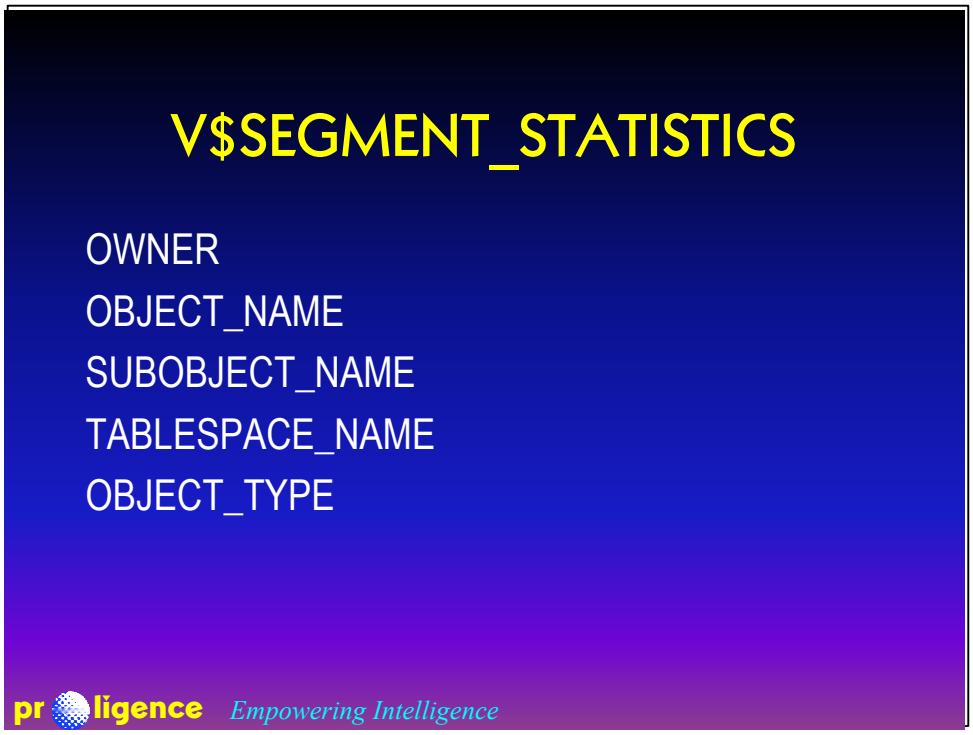
How do you find out the current setting of the parameter? Just issue the following query.



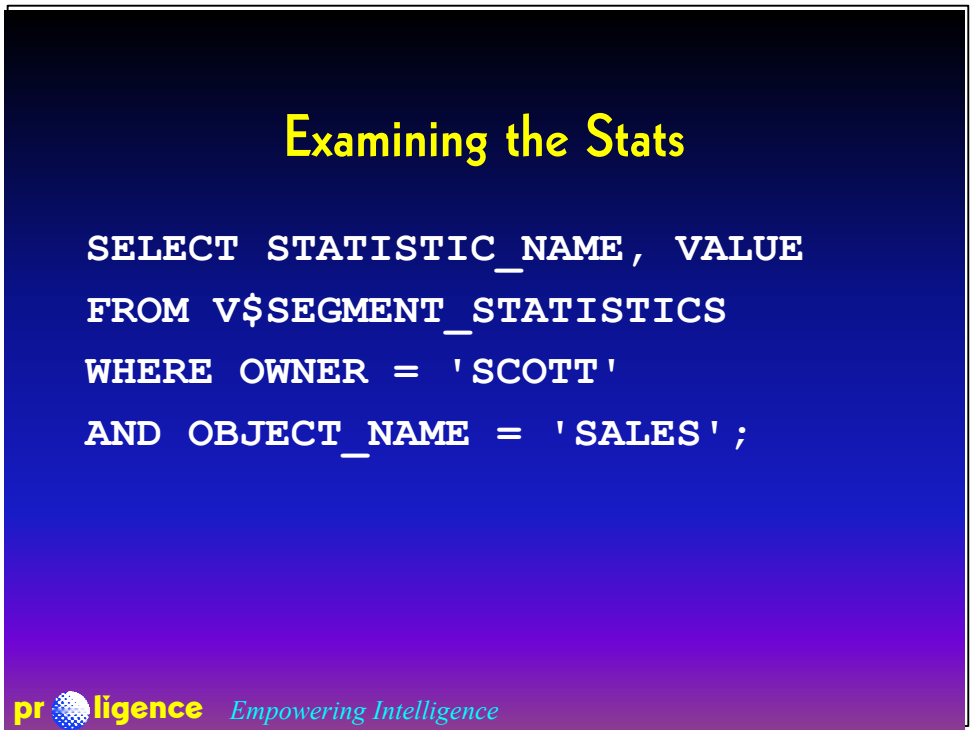
Here are the results of the query. Note that the Segment Level Stats are colelcted. This is where we would be interested in. Since the individual stats can be stopped in a session whereas the group setting may be on, there are two columns for the system level and the session level status.



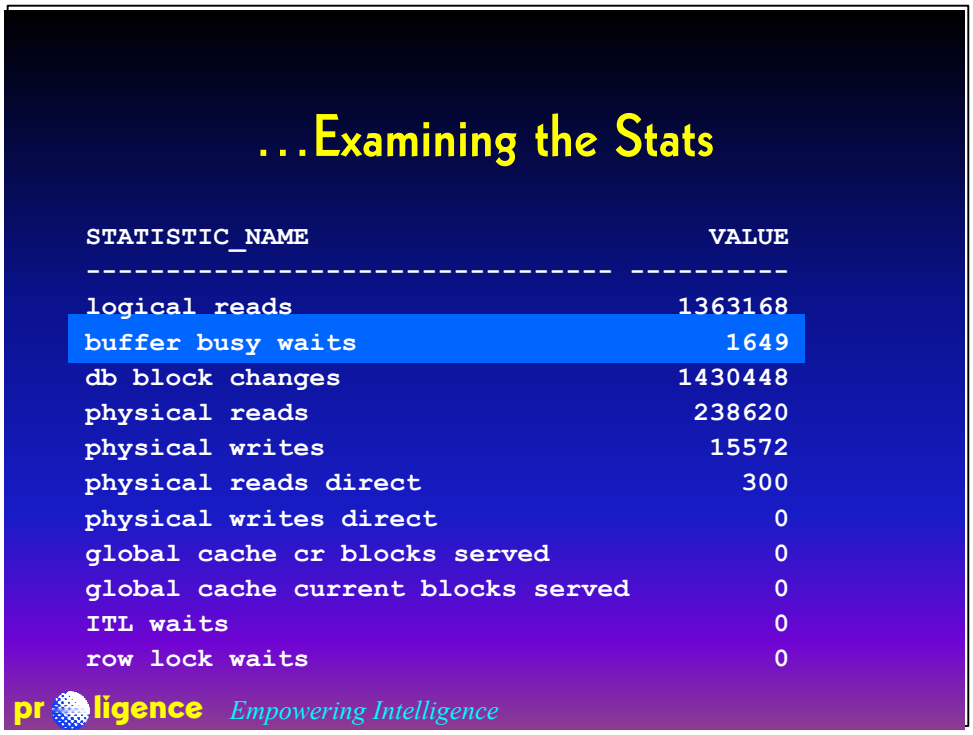
The view that contains the stats collected is called v$segstat owned by sys. These are the columns of the view. Most of these are self explanatory.



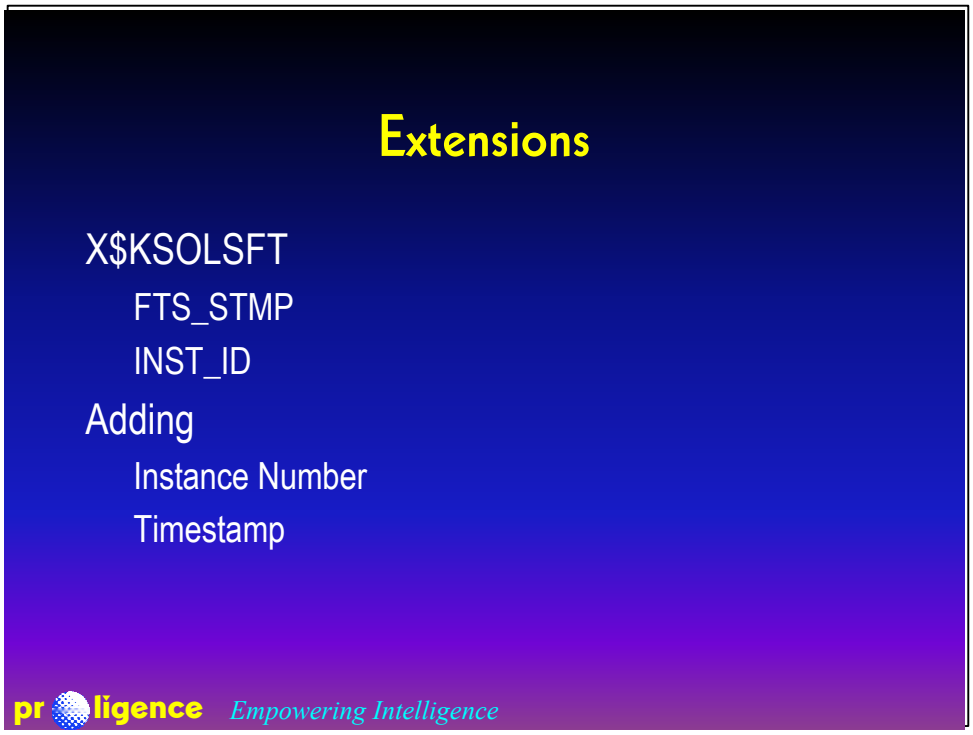
However, due to the cryptic nature of the view, Oracle has provided another view called v$segment_statistics with more meaningful fields names, in addition to the ones in v$segstat.



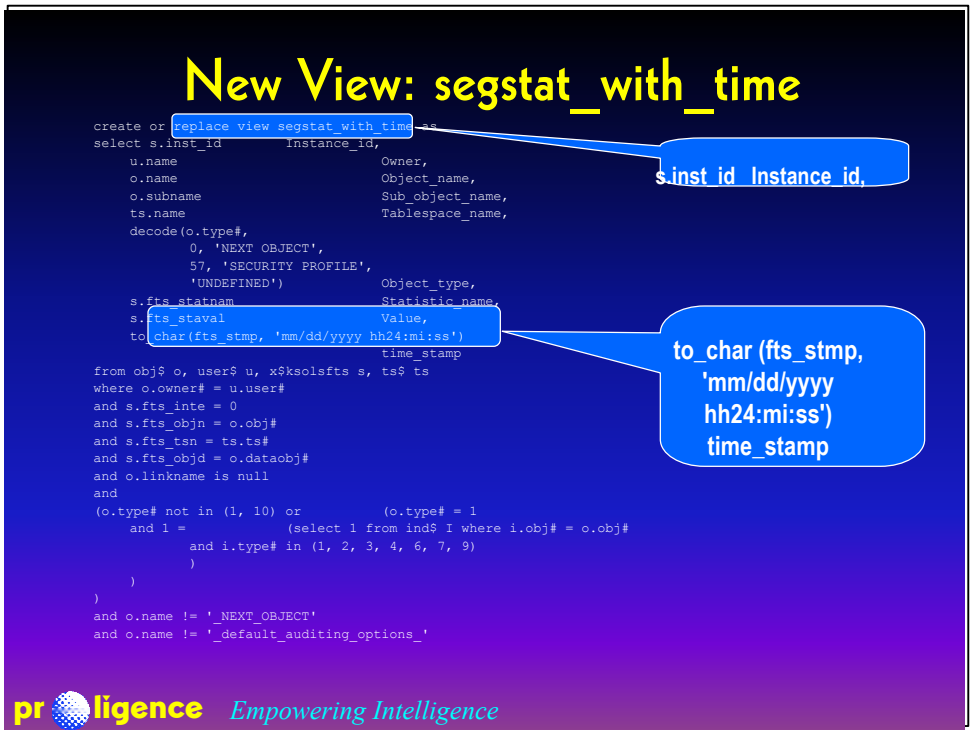
If you want to find out the stats for an object, this is the query.



Here is the output of the query. Note the buffer busy waits have a value of 1649. This number is cumulative, not a snapshot.



Examining the view definition of v$segment_statistics we find that the main table is X$KSOLSFT, an internal table that contains two fields important to us – the Instance Number and Timestamp. The field FTS_STMP is that value. We can use this to our advantage.



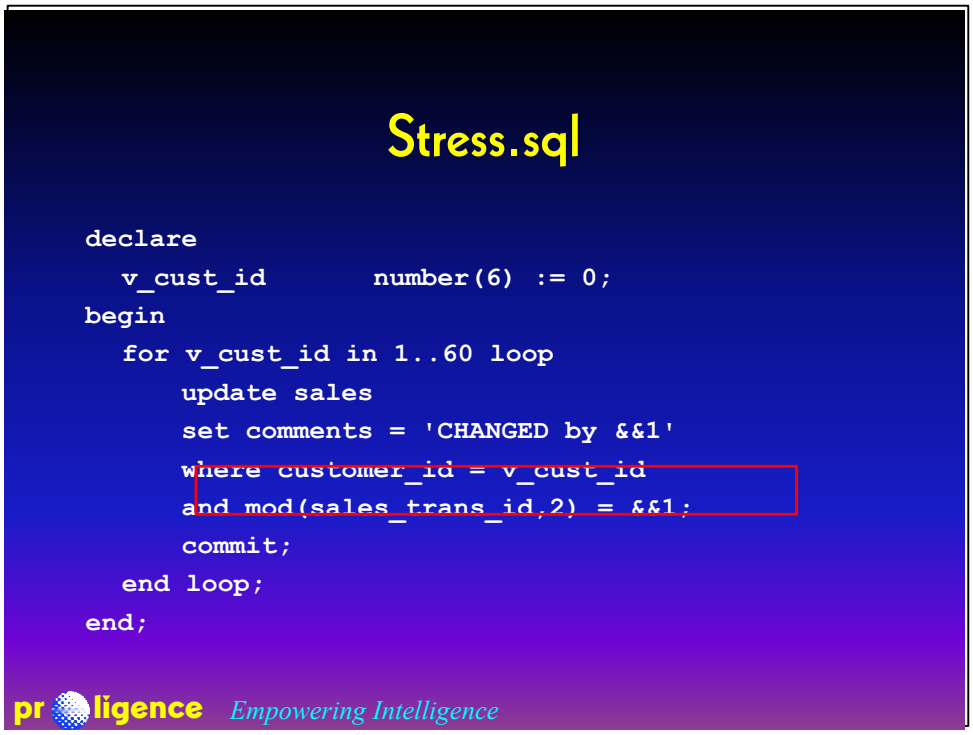
Note how we have created a new view called segstat_with_time that has two new fields for instance number and timestamp. The instance number is important in RAC environments.

Finally, the case study. We have an OLTP system with a table called SALES.

Table: SALES

```
SQL> desc sales
 Name                      Null?    Type
 ------------------------- -------- ------------
 SALES_TRANS_ID            NOT NULL NUMBER
 CUSTOMER_ID                        NUMBER(2)
 PRODUCT_ID                         CHAR(10)
 PRICE                              NUMBER(10,2)
 QUANTITY                           NUMBER(5)
 COMMENTS                           VARCHAR2(20)
```

The sales table has a column named sales_trans_id as PK.



The stress.sql is a script that updates the rows. However, based on the input (1 or 2) the updates happen to either odd or even numbered sales_trans_id.

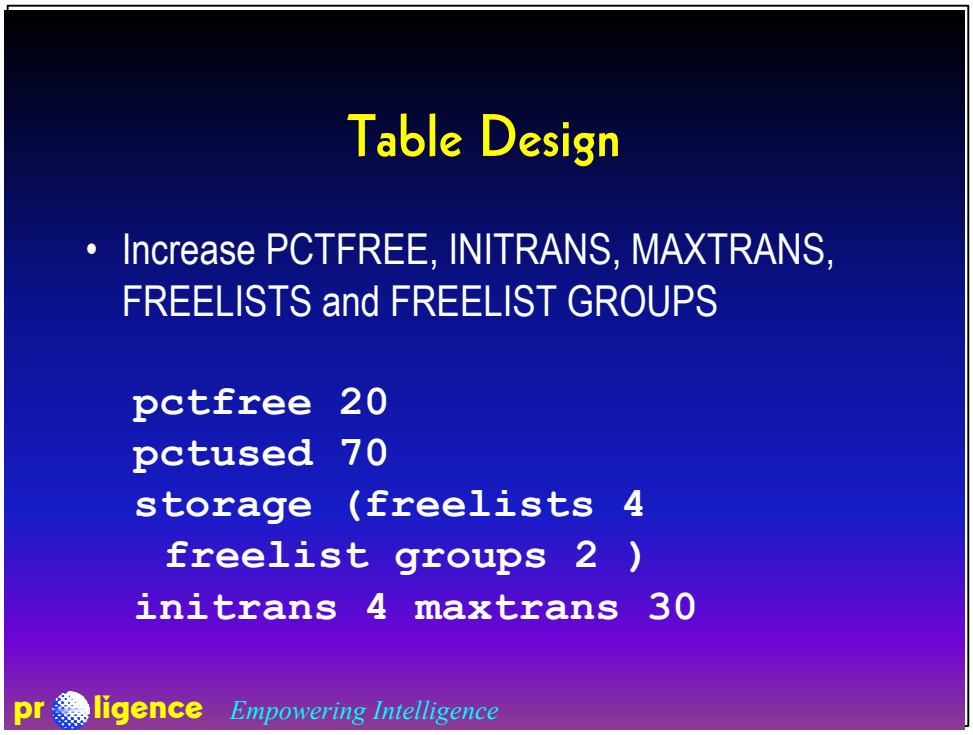
If stress.sql is run from one session with a parameter 1 and from another with parameter 2 at the same time, they will not contend with each other for the same row (since they will operate on different rows) but they will operate on the same block, creating a buffer busy waits event.

Stressing

- Running from SQL*Plus Concurrently
  - SQL> @stress 0
  - SQL> @stress 1

We saw the segment stats for the SALES table from v$segment_statistics and also from the statspack reports and we have identified that the buffer busy waits occur on the SALES table.

As a solution, we will now pack the sales table less and increase the other parameters that will contend less.



We recreated the table with new parameters as above and run the same stress.sql again in the same manner. We saw that the buffer busy conditions are now severely reduced.

The new view collects only a few stats. It will useful to have some more stats there.

More Information

- Oracle Documentation

  http://otn.oracle.com/docs/products/oracle9i/doc_library/release2/server.920/a96533/instance.htm#34509

- Proligence Website

  http://www.proligence.com

- Contact Me

  arup@proligence.com

proligence *Empowering Intelligence*


Thank you!
proligence
www.proligence.com
proligence
Empowering Intelligence