TECHNICAL NOTE

PARTITION, COMPRESS, ARCHIVE & PURGE - KEEP YOUR SYSTEM ON THE ROAD

Prepared By David Kurtz, Go-Faster Consultancy Ltd.

Technical Note

Version 0.02

Tuesday 23 April 2013

(E-mail: david.kurtz@go-faster.co.uk, telephone +44-7771-760660)

File: Partition.Compress.Archive.Purge.docx, 23 April 2013

Contents

Introduction

This paper deals with the challenges of keeping a mature on-line transaction processing system of that size and complexity on the road, and what can be about it. It discusses some work I have been doing on systems at a number of PeopleSoft customers, though I am mainly going to talk about one customer in particular. Normally, I have to talk about my customers anonymously, but this is an example of doing the right thing, and getting good outcomes. Morrisons is a particularly challenging environment because they are a large organisation, and their HR system is complex because majority of their staff are hourly paid and work to variable schedules. So there are some significant engineering challenges.

This story happens to be about PeopleSoft systems, but the priciples certainly apply to any OLTP system. It happens that the PeopleSoft systems all included Global Payroll, but the principles apply to any product.

Business Drivers

There are two business drivers for this data management work.

- The first is performance. The challenge is not only to get sufficient performance for the system with the resources available but ensure that you continue so to do as the volume of data builds up over time.
    - Although it is easy to measure how a system performs, it is not straightforward to calculate the saving obtained by reducing the data volumes involved. It is much more difficult to ascribe a financial value to improvement in performance.
- The second driver is database size. If you can compress data, or even better purge it from the database altogether you database can be made smaller, and backup, recovery, and cloning is faster and easier.
    - Ironically it is easier to quantify this saving and thus easier to articulate its benefit to management because it turns out that while disk is cheap, disks in your high performance corporate SAN isn't.

Throw Away

The challenge in many systems, whether mature or not, is that while the working set of data, the data that one is interested in at any time, is fairly constant (not withstanding changes in the size of a business and seasonal variations), the amount of data that has to be scanned and discarded in order to find the stuff in which you are interested increases with size of tables. Repeatedly fetching data just to throwing it away again is expensive in terms of both time and system resources. The techniques that I am going to describe mainly work by seeking to reduce thrown away.

Caveats

All of the ideas discussed in this document relate to the Oracle database, but there is no reason why they are not equally valid in other databases. However the specific implementation will be different.

- ~~Partitioning is a licenced feature of the Enterprise Edition of Oracle RDBMS. SQL Server and DB2 also support partitioning, but the specific implementation is different on each database.~~
- Where I mention compression, I am talking about the basic block compression available in Oracle 10g that is not a separately licenced feature. I am not referring to either Advanced Compression, which is a licenced feature of 11g, or Hybrid Columnar Compression.

Case Study: HR, T&L and Global Payroll at Morrisons

Although I have also used some of these techniques on other sites, I am allowed to talk about the system at Morrisons, let me start by giving you an idea of the size and complexity of the system..

PeopleSoft has been rolled out in phases through all Morrisons stores, of which there are now over 450 with a little over 100,000 employees. Two years ago a further 30,000 employees in manufacturing and distribution were addid. It now pays all 130,000 people every 4 weeks.

Store-based employees and scheduled to work specific hours. They clock in and out, and those clock swipes are loaded and compared to schedules by T&L. There is a custom piece that handles schedule generation. Manufacturing and distribution have a legacy T&L system that feeds approved time directly into T&L. T&L feeds payable time into Payroll. The system also does HR, statutory payroll and pension auto-enrollment processing. There are various MIS reports run on this system.

This system has been live since 2008. By January 2013 the database size had exceeded 4TB because nothing had been archived. The rate of growth over the previous year had been around 1Tb/year.

~~This paper deals with the challenges of keeping an on-line transaction processing system of that size and complexity on the road, and what we are doing about it.~~

Objectives

I want to look at 4 techniques for reduce I/O to and from live application tables during BAU processes. Either by

- giving Oracle a way to discard it without scanning it,
- compressing it,
- putting it elsewhere in the database, or
- getting rid of it complemently.

The fastest way to do anything is not to do it at all.

Partitioning

A partitioned database table or index is divided in smaller pieces, such that certain data values in specific columns are stored in certain pieces. Thus, if you are looking for certain data values, Oracle may be able to infer that the data can only be in certain partitions. In which case, it can eliminate other partitions from a query without needing to scan them.

Database partitioning is often positioned as a method for improving performance in very large databases, but it also has application in OLTP databases. It does so by reducing throw away with partition elimination.

The challenge is then to find a selective column that is used in most queries. If a query does not use a predicate on the column by the table is partitioned, Oracle cannot eliminate any partitions and must inspect all of them.

- PeopleSoft Global Payroll has a concept of concurrent processing it calls 'streams'. The employee population is broken groups of contiguous ranges of employee ID. Each range or 'stream' is processed by a separate and usually concurrent process. So, it is usual to range partition the payroll result tables by the employee ID to match the definition of the payroll 'streams'.
    - Here partition elimination ensures that each stream doesn't have to scan through data being used in other streams.
    - In this case there are no 'historical' partitions, so this partitioning strategy is no use for the data management concepts that I am discussing in this paper. However, I have no choice as to how to partition these tables, because the partitioning needs to match the predicates in the application SQL. The SQL contains 'WHERE EMPLID BETWEEN ...' predicates, so I have to range partition by EMPLID. This has forced me to archive these tables with DML.
- The largest Global Payroll result tables are also list sub-partitioned by the ID payroll period. Most payroll processes are run for a specific pay period, and so Oracle can eliminate all list sub-partitions other than the one that contains the data for the period in question.
    - CAL_RUN_ID is a string, not a number or a date. However, each value does correspond to a period of time, so we do have 'historical' sub-partitions that can be compressed and purged.
- In Financials, General Ledger reporting is by period (in PeopleSoft that is the combination of fiscal year and accounting period). The queries in PeopleSoft nVision reports always include an equality condition on fiscal year and another condition on accounting period. It makes good sense to range partition the ledger tables on the combination of these two columns.
    - I generally create single period partitions for the current and previous fiscal year. This supports current period and year-to-date reports that compare to same period last year.
    - Quarterly or annual partitions for previous years.
- In other Financials modules it can be more difficult to pick a partitioning column because some tables are accessed in different ways, so partitioning of a particular table may be appropriate for one process, but cause difficulties for another.

Partitioning is a pre-requestite for much of what comes next. Partitioning creates a link between the logical value of a piece of data, and its physical location in the database. Tables that have been partitioned by a date column, or an ID column that corresponds to a period of time, have current, future and historical partitions. In the following sections I will discuss how we used partition-wise operations to manage data rather than working row by row. Partition-wise operations are generally faster, and generate less redo logging on the database.

Historical data can be:

- Compressed by compressing individual historical partitions. This does have the penalty of rewriting all the data, but it is a one time cost.
- Archived by exchanging historical partitions out of the table with another (empty) table
- Purged by dropping historical partitions. Space is released back to the database tablespace by dropping a whole partition (segment), and if you are careful about laying out your partitons and tablespaces you can release that space from the datafiles back to the file system.
    - NB: This is not the case when you delete individual rows of data. That space is freed up in the data block, but the block remains a part of the segment. It is still read during a full scan.

I wouldn't want to try doing any of this without partitioning because you could only do it by rebuilding whole tables. This would take longer and require more/longer outages to the system.

Partitioning at Morrisons

We introduced partitioning primarily as way of managing performance. However, the initial design always had an eye to the possibility of partition-wise archiving at some point in the future. Our data is grouped into a number of logical areas

- Global Payroll is calculated by a COBOL program. The Morrisons payroll is not simple. It takes 2 – 3 seconds per employee, of if you prefer around 1000-1500 employees per hour. A single process could run for 5 days. So we run 32 processes concurrently. That degree of parallelism is determind by the number of CPUs we can bring to bear. PeopleSoft calls this parallel processing 'streaming[1]', it is an intrinsic part of the product. Each stream processes a contiguous range of employee IDs. So we have partitioned all the payroll tables and some of the HR tables (eg. JOB) into 32 corresponding range partitions.
    - The two main result tables (GP_RSLT_ACUM and GP_RSLT_PIN) are also list sub-partitioned such that each UK Lunar pay period is held in a separate sub-partition. Historically UK weekly and Gibraltar payroll data is held in the same tablespaces, but the volume of Gibraltar data is so low (there is only a single store), tht we are going to leave all of it in the default list partition.
    - There are also a number of other GP processes written in Application Engine that also run in streams. There is only one streaming definition, and so they must take their lead from the needs of the Global Payroll calculation.
    - At another customer I partitioned TL_PAYABLE_TIME as a payroll table because it is updated by the payroll calculation. I made the mistake of carrying that descision forward into Morrisons. That tabke is populated by the main T&L process (TL_TIMEADMIN). The table will be partitioned weekly when we implement T&L archiving, though many of the non-unique indexes will continue to be partitioned by EMPLID.
- Schedules: The stores work on a weekly cycle. Weekly schedules, weekly absence processing, 4-weekly payroll. Stores generate weekly schedules for their staff 4 to 6 weeks in advance. It made good sense to partition the schedule tables into weekly range partitions. 52 or 53 partitions per year.

---

[1] It is no related to Oracle database streams. It is coincidence.

- T&L: Reported Time, Exceptions. This is also weekly partitioned for very much the same reasons as scheduling.
    - The sheer volume of reported time to be processed by TL Time Admin is also a challenge. So we sub-partitioned TL_RPTD_TIME on EMPLID. We were and still are using Oracle 10g. So having range partitioned a table, the only options were list or hash sub-partition it. So we chose to sub-partition each range partition in to 16 hash-subpartitions. We use 16 concurrent instances of TL_TIMEADMIN to process this data and we distribute employees to processes by the same hash function so that each partition is only referenced by a single process. As with Global Payroll, we are just keeping concurrent processes away from each other. This avoids making the database do lots of work to maintain read consistency.
        - If we used Oracle 11g we could range sub-partition a range partitioned table, and I would range-partition TL_RPTD_TIME on EMPLID.
    - The T&L partitions reside in the same tablespaces as the schedules. This was reasonable when there was no purge policy.
- Audit: We have a lot of audit tables, mainly for fraud detection. Most have all been range partitioned by the audit timestamp into calendar monthly range partitions. Some of the smaller ones are not range partitioned.

Tablespaces

The initial policy was that all data was retained indefinitely, so monthly tablespaces were created to hold tables for each logical area, with corresponding tablespace for indexes. There is no performance benefit to this, but it does help you to see where the database is growing.I think is a better option that just allowing the delivered tablespace to grow. Even with this approach our PSINDEX tablespace had reached 175Gb, requiring 7 files.

When we come to archiving, the tablespace strategy can help you eliminate free space from the database.

Managing Parititoning in PeopleSoft

The partitioning strategies that I have chosen are mostly determined by the application, though some of the detail is determined by the way Morrisons uses them, and their data. I have been working with Global Payroll since its first 2001. I have implemented this partitioning strategy at 14 different customers. The number of streams varies, and the approach to subpartitioning the large result tables has varied.

I have implemented partitioning in GL in Financials in a number of places. At Morrisons I have extended partitioning into Audit, Scheduling and TL.

One of the challenges of partitioning in PeopleSoft is that the Application Designer does not generate partition DDL, mainly because it is different on different databases. Partitioning does not invalidate your support. Various Red Papers have encouraged its use.

From PeopleTools 8.51, Application Designer will generate DDL to preserve existing partitioning, but AFAIK that only works on Oracle. I found it was quirky. You could lose partitioning on an index if you changed the keys.

If you want to introduce partitioning into PeopleSoft, you are on your own. My solution has been to build a utility that simulates the script generation in Application Designer but to add my own meta-data tables to describe the partitioning[2]. Wherever I introduce partitioning in PeopleSoft on an Oracle database I use this utility to generate the DDL. It is a lot more efficent and reliable that crafting the scripts by hand.

Licencing

Various databases supported by PeopleSoft, support partitioning. Oracle, SQL Server and DB2 also support partitioning, but the specific implementation is different on each database.

With Oracle, partitioning is a licenced feature that is only available with Enterprise Edition of Oracle RDBMS.

[2] See Managing Oracle Table Partitioning in PeopleSoft Applications with GFC_PSPART Package: http://www.go-faster.co.uk/gfc_pspart.manual.pdf

Compression

There are various types of compression in Oracle. Most of the systems I am working with are still running Oracle 10g so the only option is what Oracle often calls 'simple' or 'basic' block compression. However, it has the merit of not being a licenced option.

The technique can be used to compress both tables and indexes, but there are some different limitations.

Table Compression

How does it work? Simply, data values in a column that are repeated in different rows stored in the same data block are only stored once.

For example, you might have 80 rows of a payroll result table in an 8KB data block. They might releate to just one or two employees (because the rows are inserted in employee ID order). The employee ID might be 8 characters, so instead of storing the same two employee IDs repeatedly and consuming 640 bytes, each employee ID is stored one, and there is a pointer to those two values, saving 546 (which 78 * 7) bytes.

This technique enables Oracle to store more rows in one block, and you need fewer blocks for the same data. There is some additional CPU overhead to process the compressed blocks, but far more significant you do less physical I/O because you visit fewer blocks.

There is a catch. There is always a catch. You can only compress table blocks when you create the object compressed, or when you populate them using a direct path mode operation into a compressed segment. Normal insert/update/merge statements issued during an application do not result in compressed blocks. Update/delete/merge statement that alter a compressed block, have the effect of decompressing the data block. So this form of compression can only be applied to historical partitions in table which are never going to be updated.

The amount of space saved by compression varies, depending upon the data. I have typically seen a 50-90% reduction is size – or if you prefer, the number of rows per block increases by a factor of between 2 and 5.

Table compression does not affect the size of indexes. The index leaf block still stores index key values and a row ID for every row.

Index Compression

Index compression is similar in concept to table compression, but it applies to the whole index. You cannot compress just some partitions in an index. The index leaf blocks will still be compressed if the data is inserted during a normal DML operation, but there is an overhead during insert.

So you might compress some partitons of a live table, but not compress the index.

What do I compress?

It depends on how you have partitioned, and upon your application. Any table that is partitioned by date or by something that corresponds to a data is a candidate. Partitions can be compressed as they become static.

- Audit Data is partitioned into monthly partitions on the audit timestamp. These can be compressed as soon as the month is closed.
- Payroll result data is static as soon as the pay period is closed. We have list sub-partitioned the two largest payroll result tables by payroll calendar group ID. So there is a separate partition for each monthly payroll.
- Report Time can be corrected up to six months after the date, so we wait six months before compressing.

When can I compress?

Table compression can be introduced into application tables long before it is possible to archive or purge data. We have a batch process that runs weekly that determines which partitions can be compressed. It issues a series of ALTER TABLE … (SUB)PARTITION … MOVE … COMPRESS commands.

GOTCHA: ALTER TABLE MOVE is supposed to be an on-line operation. However, we have found with Oracle 10g that the locally partitioned indexes momentarily become unusable during the move operation, so we only schedule this process during an outage.

Archive & Purge

I will take these two subjects together but I mean different things by these terms.

- Archiving involves relocating data from the table referenced by the application to another table from where it is still accessible. In this paper I will only discuss the case of keeping that table in the same database.
- Purging data is removing it from the database so that it will not be accessible from anywhere.

For each table, or set of tables, it is necessary to define an archive policy and/or purge policy for each table. Our approach has evolved over a period of time. We started by archiving T&L Schedule data because we needed to improve the performance of some of the T&L processing. Now we have extended that to Schedules, Audit and Global Payroll. The policy for schedules has changed. Initially we archived the data and kept the archive data indefinitely. The business has now said it doesn't actually need archived schedules, so we are going to change that the process to just purging the data.

Archive by Partition Exchange

In general, where we archive a table, the archive table has the same partitioning strategy. Audit tables and their respective archive tables are all partitioned by the audit timestamp into month partitions. Partitions are archived by partition exchange. An empty table of the same structure is required, because you can only exchange a partition with a table not the partition of another table. So we do two exchanges to move a full partition from the application table to the archive table. The empty partition must exist on the archive table. The exchange table will still be empty at the end of the process. The archived partition in the live table will be empty, and it can be dropped.

- Locally partitioned indexes are also exchanged with the table partition, and so must also exist on both objects in the exchange.
- Global indexes are only maintained in the UPDATE GLOBAL INDEXES clause is specified (which we always do), otherwise they would become unusable and have to be rebuilt. We do not build those indexes on either the archive table or the exchange table.
    - However, the maintenance of global indexes during partition management is effectively done by DML and can be time consuming. Oracle scans the partition and maintains the corresponding index entries.
    - Global index maintenance does not impact dropping an empty partition that has always been empty because there are no index entries to maintain,
- Any additional indexes required on the archive table to support processing must be built as global indexes.
- Partitioned table segments and local indexes whose partitions are being exchanged must also have the same compression as the table and indexes with which they are being exchanged.
    - Our exchange utilities check this and adjust the compression if necessary. We change the compression on whichever segment is empty because that is fast to rebuild.

Performing Partition-wise Operations

All of the partition-wise operations are done with DDL, specifically some sort of ALTER TABLE command.

We have extended the meta-data that drives the partitioning utility to specify whether a partition should be purged by dropping, or archived by exchange. We have developed further

PL/SQL utilities to generate and issue the DDL. This utility can be invoked from SQL*Plus, or from an Application Engine program.

Exceptions to the Archiving Rules

We partitioned the schedule data by the effective date of the schedule. That has been fairly effective for performance, although we still have some challenges around the current effective date sub-queries.

Schedules also have an end date. Some schedules with very early EFFDTs are still active. In 4 tables we have a few rows, about 1% of the volume of each partition, that must not be archived. We move those rows back into the application table by DML during the archiving. We call this the no-archive rule and we have built it into the PL/SQL utility that we have built.

Arching Row by Row

Sometimes, it is not appropriate for an archive table to have the same partitioning strategy as the application table, in which case data cannot be moved by partition exchange. For example, the payroll result tables are range partitioned to match the stream processing. We don't need that approach in the archive table. Instead we will eventually purge whole tax years when they are 8 years old.

We have range partitioned the payroll result archive tables by the calendar group ID (CAL_RUN_ID) such that there is a partition for each Tax Year. The two largest tables are also list sub-partitioned on CAL_RUN_ID to create monthly sub-partitions. We will be able to purge whole tax years by dropping partitions.

Pay periods are archived on a rolling basis when their end data is 2 years old. We have to copy the data with DML. We do an INSERT … SELECT on each table. The data is relocated into new tablespaces. However, we can perform the INSERT in direct path mode by using the APPEND hint, and so we can also compress the archive data during this operation. The indexes on these archive tables are also compressed.

The data must be deleted from the application tables with a DELETE statement from the result tables, although we can drop sub-partitions from the 2 largest result tables. This does not recover any space, but free space in the object can be used by the next payroll to insert new data in preference to extending the object.

There is no question that row-by-row approach is slower, and generates a lot more redo. However, there is no risk of any problems with partition DDL failing with DDL locks,

- We had the opportunity to identify just a sub-set of the result data to be archived, the rest could be purged. However, this proved too difficult, and we are just retaining everything.

Impact on Tablespace Design

Initially most partitioned tables were put into monthly tablespaces because we were keeping data indefinitely. The exception was the payroll tables that were only range partitioned to match the streamed processing, We had a pair of tablespaces per stream for tables and indexes.

We have an annual exercise in adding new partitions, and this also includes creating new tablespaces.

We realised that once you have a purge policy for a table, you effectively have a fixed upper limit to the size of the table (not withstanding changes in the size of the company). If you purge on a regular rolling basis you have a roughly constant volume of data. So now, instead of having pairs of monthly tablespaces for the audit tables, we have 3 pairs of tablespaces (for the three purge policies). Space is freed by dropping partitions and consumed again as current partitions expand. This arrangement will be extended to Schedules.

Having monthly tablespaces mean you can see how the database grew over time, but moving to a fix set of tablespaces has removed a bit of administration. There is no significant performance benefit to this other than the database spends slightly less time updating datafile headers during a checkpoint, but this is too small for me to have been able to measure.

Recommendation: Keep objects with different purge policies in different tablespaces.

In the early days of the project we partitioned both Schedule and T&L data into weekly partitions, and co-located those partitioned in the same month tablespaces. Not surprisingly Schedule data is created before T&L data. However, now we purge schedule data after 56 weeks, but we purge T&L data after 108 weeks. When they were coresident the schedule data would tend to be nearer the start of the tablespace and the T&L data further along. When we archive the schedule data we leave free space at the front of the tablespace that cannot be trimmed.

Backup

A word about backup. Everyone uses RMAN to back-up their Oracle databases. RMAN backs-up every block that has been used. It is used if it contains an SCN. It doesn't have an SCN until it has been used to store a database object, such as a table or index. When you drop that table or index, the block still has an SCN, so it still gets backed-up during a full back-up and will do so forever. So the only way to shrink your backup is either to drop data files or to trim free space from your data files that used to contain database objects.

Flashback Database

Testing this process is a challenge. You need to know that the data is still referentially integeral after you have archived and purged, and that processes and reports still function correctly. You also need to test how user will access the archive data. Oracle Flashback Database has been extremely effective. If something goes wrong you can roll back the entire database to a restore point or a point in time.

We usually set a guaranteed restore point before we start a process.

Some things to bear in mind:

- I usally set a guaranteed restore point so that flashback is retained beyond any retention time limit.
- Dropping a data file or reducing its size by resizing will break database flashback. You cannot flashback through that operation, so we do that separately when we are satisfied with everything else.
- We enable Flashback Database in production when we perform these actions so that if anything goes wrong we can flash everything back rather than do a restore. We don't normally run in production with flashback because we encountered performance issues (that are probably specific to our version/environment), and so we have to restart the database to enable flashback.
- When have completed archiving/purging we have a go/no-go point. It is then that we can drop empty tablespaces, trim free space from data files, and defragment tablespaces.

Accessing the Archived Data

Once data has been moved into a separate table it is not visible in the PeopleSoft application. Our general approach has been that the archive tables are available via PeopleSoft Query. There are a couple of payroll components and reports that have been modified to optionally retrieve archived data.

We have deliberately not made it easy to access this data. We want to save space and I/O for the greater good of the system. If the business can articulate a need for something better than

that we will of course respond to it. In practice the opposite has happened with the schedule data, they have accepted that they really don't need it and we are not allowed to purge it.

I think all of us would be naturally conservative about letting go of data that is perceived as valuable, and there is a process of getting used to the idea of archiving and ultimately purging data from the system.

PeopleSoft Archiving Utility

We did examine the PeopleTools archiving utility, and we were not impressed.

1. It is a DML based approach.
    a. So it would be a relatively slow process.
    b. It would do nothing to reduce tables' high water marks, so we wouldn't save space and I/O unless we could drop partitions
2. We would still be responsible for partitioning, and would still have to do something about dropping empty partitions, and merging sparsely populated partitions.
3. PeopleSoft has added a sub-record with 2 key columns to each archive table.
    a. So the data gets larger when you archive it.
    b. The subrecord fields become a part of the unique key on the table. It is perfectly possible to have multiple copies of the data in the archive table.
    c. The subrecord is at the front of the index and gets in the way of querying the data. The only thing for which the resultant unique index is useful is recovering the archived data back into the live tables.
    d. The archive process, as delivered, cannot compress the archive data as it archives it.

However, when we built our own process to perform DML archiving of the GP tables, we based in on the PeopleSoft process

1. We used the same PeopleSoft meta-data that describes the tables to be purged, and the archive tables to which data will be moved.
2. We used the same tables that PeopleSoft has delivered for the purpose of holding archived data – but we did remove the sub-record.

Other Matters

Defragmentation

Over time, as we have and continue to introduce the archive and purge processing we have reduced the size of the database. In some cases whole tablespaces have become empty and we have been able to drop the whole tablespace and release that space back to the ASM file system.

Sometimes we have been able to reduce the size of a data file by trimming free space from the end of it. However, we are often left with free space in the middle of a tablespace that cannot be trimmed and because it used to be part of an object, the data blocks will still be backed up by RMAN.

Essentially a tablespace can start to look like a piece of Swiss cheese (specifically Emmentaler). This problem is much less severe since tablespaces started to use bitmap spare maps (Oracle 9i, and default in 10g), but it can still occur. Our solution has been to write a PL/SQL package to rebuild the object that occupies the highest block in the data file so that it is rebuilt lower down the tablespace, and trim off the free space by resizing the data file. We rerepeatedly do that in each tablespace until the free space in the middle of the data file is consumed. This is something that must either be done during an outage or at least when nothing is accessing the object being rebuilt.

In 11g you can rename tablespaces, so you have to option to handle fragmentation by rebuilding all the objects in a new tablespace, drop the old tablespace, and rename the new one.

11g Opportunities

There are some new things in 11g, when we get to it, that I am looking forward to.

Automatic partition creation

At the moment we have to create new partitions manually in advance of them being used. However, in 11g it is possible to have Oracle create the new partitions automatically where a table is range partitioned on a number or date. However, you have limited control over tablespace and naming.

Range-Range Partitioning

We will be able to replace some of the range-list partitions with range-range. It will improve some of the T&L processing. We range partition on period begin date, and we can range sub-partition on period end date.

Advanced Compression

This is the licenced feature, but it works with normal DML, although there is a performance.

Active DataGuard

DataGuard is an Oracle database technology to maintain one or more replica of a production database. Changes applied to the primary production database are shipped and applied to the replica in near or actual real-time – Oracle calls this managed standby recovery. Active DataGuard is a licenced feature of Oracle 11gR2 that permits a standby database to be simultaneously open for read-only activity and be in managed recovery mode. Otherwise the database is either in one more or the other but not both.

PeopleTools 8.51 supports Oracle Active DataGuard. Active DataGuard allows you to move reporting activity off the production dataase to the standby thus eliminating the contention between reporting and transaction processing.

NB: With PeopleSoft you need your standby database to be close enough to the primary that there is not a significant network latency Otherwise, the performance of the application server when connected to the active standby will affected.

Miscellaneous

Transportable tablespaces

Something we haven't looked at but might if we had needed to transfer archive data to another 'archive' database is using transportable tablespaces. Effectively, datafiles from one database can be copied to and read by another database.