
# Rapid Database Builds: Experiences from Oracle8™ Testing

*An Oracle Technical White Paper*

*June 1997*

ORACLE®

Enabling the Information Age™

# Rapid Database Builds:
# Experiences from Oracle8 Testing


## ABSTRACT: THE NEED FOR RAPID BUILDS

As databases grow larger, the need to minimize the total time to build each stage of the database increases. Each stage of the build—whether tablespace creation, data loading and generation, index building, object analysis, or aggregation creation—needs to be completed in a parallel format whenever possible, while also taking advantage of any general optimizations available. This white paper explores the optimizations learned at each stage of the build cycle, while scaling a customer system for use as a testing tool for Oracle8 to 500+ gigabytes. This paper includes optimizations possible in Oracle7™ and new ones available in Oracle8.


## TABLESPACE CREATION

The time to create tablespace is minimal for small databases, but if improperly planned, can easily become a long serial operation for large database builds. To avoid long tablespace creation, the following optimizations can be executed:

- Create tablespaces in parallel.
- Add all datafiles in parallel.
- Lay out datafiles properly to minimize contention between them.

The key to rapidly creating tablespace is understanding where the Oracle database functions serially and where it functions in parallel. Each datafile creation process is a serial task where the entire datafile needs to be scanned. Therefore the larger the datafile, the more time it takes to create the datafile. Since the datafile creation process usually is an input/output (I/O) intensive operation, even a single CPU machine should allocate multiple datafiles concurrently.

Until Oracle8, creating tablespaces was also a serial process because of an internal locking mechanism on the row cache. Since any “alter tablespace add datafile” had to wait for the create to finish, this function was also held up. In contrast, “alter tablespace add datafile” works in parallel in both Oracle7 and Oracle8.

In Oracle7 Release 7.3, the cost of serialization could be limited by creating very small initial datafiles and then larger datafiles during the alter that could run in parallel.

Since tablespaces in Oracle8 can now be created in parallel, all the create tablespaces can run concurrently followed by parallel "alter tablespaces add datafile" for all the remaining datafiles. Since "alter tablespace add datafile" can not be executed until the initial tablespace creation has been completed, the initial datafile still should be kept small to minimize total create time. With sufficient disk and CPU resources and no usage contentions, the time for the creation process should now only be bound by the size of the initial datafile plus the size of the largest added datafile.

Proper layout of these datafiles is critical to later stages of the database build and operation. Not only should these files be well striped across disks, but if the system is running in a parallel server environment, they also should be manually striped across each of the nodes in the cluster by creating separate datafiles for each node.[1] This will ensure that no node becomes a bottleneck at later stages of the build process. Disk striping even can be done in cases where the tablespace only would be used by a subset of the nodes after build.

## DATA LOADING

Data generation and loading can be the most time-consuming processes of the build for large systems. Fortunately these portions also tend to be the easiest to parallelize and they scale well as more resources are added. Several optimizations were built into Oracle8 to optimize the loading process. These include:

- Generating data for all tables concurrently.
- Loading data for all tables concurrently.
- Generating data for each table in parallel.
- Loading data for each table in parallel.
- Using pipes as an intermediary format between data generation and loads instead of files.[2]
- Loading with both the direct and no-logging options.

Many large database systems have significant amounts of data not created internally, but are generated externally into some kind of flat file that must be loaded into the system using SQL*Loader®.

If possible, the generation mechanism is the first area to optimize. As long as the generation mechanisms permit, data generation can be executed both concurrently for all objects being generated as well as in parallel within each object. Then if data can be loaded at the moment it is generated, the intermediary format can be switched between generation and load to pipes instead of files—this saves both space and I/Os.

Now that data is generated concurrently for all existing tables and in parallel within tables, a load stream can be set up to match each of these generation streams.

1 Automatic striping across multiple nodes, called HSD striping on the IBM SP2, removes any knowledge of the data locality from the Oracle database. This prevents the database from taking advantage of data locality and makes the job of designing an application that avoids pinging more difficult.

2 Pipes in UNIX can be created by using the mkfifo command. Once a pipe is created, it can be treated as a file for both the generation process and SQL*Loader. Both the writer and reader will wait for each other.

For example, in the testing system, five tables were loaded concurrently as shown in Figure 1.

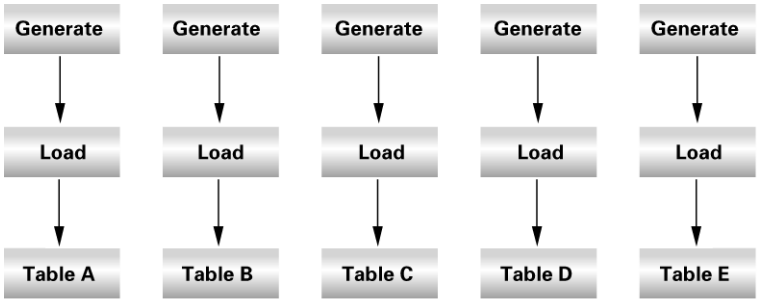


*Figure 1: Concurrent Loads*

Since this process was too slow, each of the tables were partitioned ten ways.[3] Loads and data generation then were parallelized ten ways for each table (one load per partition[4]) and ran concurrently for all tables, as shown in Figure Two.

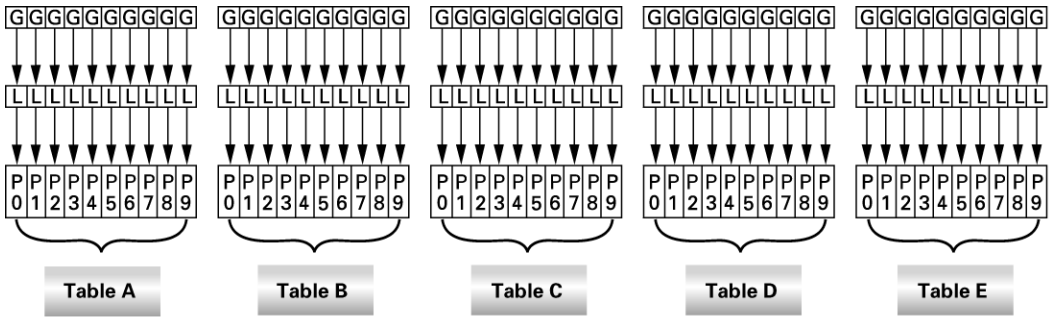


*Figure 2: Concurrent and Parallel Loads*

These changes gave a total of fifty generation/load process pairs, which was significantly more than the 20 CPUs available on the machine, but less than the number of disks used. By changing from five generation/load pairs (one per table) to fifty, the total load process increased speed by 7.5 times. Further optimization could be accomplished to parallelize larger objects to a greater degree and smaller ones to a lesser degree, allowing all the loads to finish about the same time.

3 The partition can be completed either with Oracle8 Partitioned Objects or by manual partitioning in Oracle7.3 using separate tables and partitioned views.

4 One load process per partition was executed for flexibility purposes of the code, as it allowed testing of index builds both before and after loading. Parallel direct loads are not permitted to tables that have an *existing* local index, but concurrent direct loads can run to each partition. Without pre-existing indexes, parallel loads can be completed to any degree, either to the tables or to the partitions.

With parallel server systems, the generation and load process should be spread out to work locally on each node. The load should be executed to a datafile that exists locally on that node. This will ensure maximum throughput as local disks are accessed the fastest. Spreading out the work in this manner also prevents pinging from occurring between the different instances during the load.

All pinging could be avoided on these datafiles if each were separately assigned Parallel Cache Management (PCM) locks and the datafiles were not used by any non-load process. When properly divided independently on each node, parallel server systems scale better across degrees of parallelism than do single-node systems. This is because parallel server systems are able to conduct this type of work in their own workspace with little contention or coordination necessary.

Finally, extra overhead loading can be avoided by using both the direct and no-logging[5] options whenever possible.

### INDEX BUILDS

Four aspects were tested during the build to minimize index-creation time. These include:

- Creating all indexes concurrently.
- Creating indexes with the no-sort option.
- Building indexes in parallel.
- Pre-creating the indexes before the load.

As each index is completely independent, the first optimization was to create all of the indexes concurrently. Indexes also can be created significantly faster if the Oracle database does not need to sort the data beforehand. A system-level sort routine to presort the data often is even faster. Control over the data generation enables the data to be generated in a sorted fashion. However, the no-sort option generally is unusable for large amounts of data because this option lacks a non-serializing, non-race-producing method to ensure that multiple-loading processes load data in a predetermined order. However, Oracle8 Partitioned Objects allows a workaround.[6]

Create a number of partitions equal to the number of desired load processes. Then a generation/load run pair can run concurrently that will generate only within the appropriate range for that partition, maintaining a sorted order and parallelism, as shown in Figure Three.

5 No-logging is *unrecoverable* in Oracle7.

6 Disclaimer: According to ANSI standards, data does not have to be stored in the order that it is inserted (thus allowing for parallel inserts and other optimizations). So even though the inserts currently keep the order with one insert process per partition, it is not guaranteed to do so in the future; nor is this method officially supported.

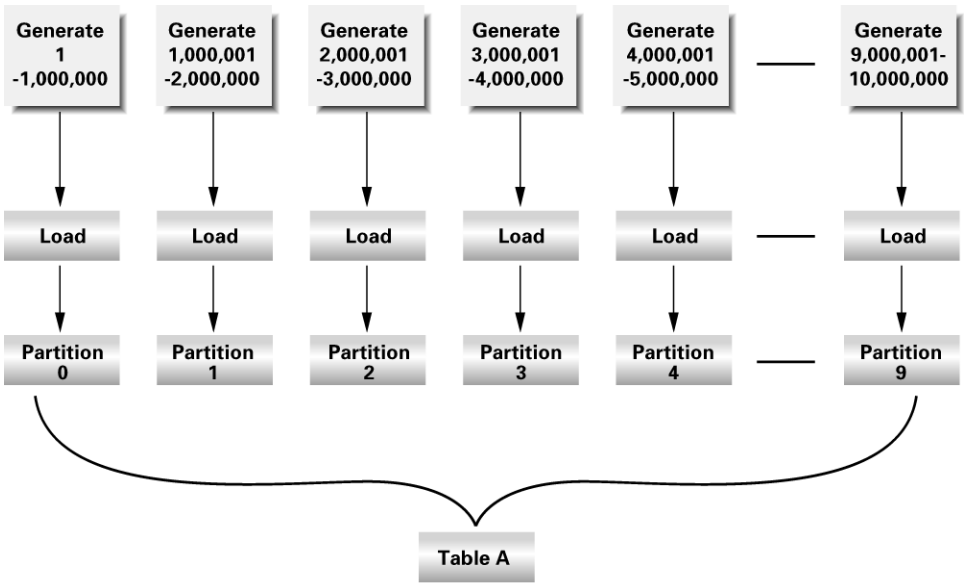


*Figure 3: Parallel In-Order Loading*

Finally, whether to create the indexes beforehand or afterwards in parallel needs to be determined. With Oracle7, since pre-existed loads could not be executed in parallel and direct mode, it was better to create the indexes afterwards in parallel. With Oracle8, a separate load process for each partition can be executed using a direct load. Comparative testing showed that creating indexes beforehand and doing the load afterwards took about the same time as loading and creating them afterwards, with the same degree of parallelism in the load as in the index creation.

## CALCULATING STATISTICS FOR OBJECTS

Three key aspects exist to minimize the time spent calculating statistics:

- Calculate the lowest level of statistics necessary to get an accurate picture for each object.
- Partition all objects that require significant analyze time.
- Calculate statistics concurrently both across objects and partitions.

Overcalculating statistics can be a significant resource waste for very large databases. While a 30% estimation for tables and calculation for indexes and small tables is sufficient for most tables, testing shows it can be inappropriate for very large tables that differ little statistically. In these cases, a wide range of statistical estimation/calculations should be tried to find the least-consuming method accurate enough for the existing data.

Oracle8 Partitioning helps with statistics in several ways. Partitioning now allows this calculation process to be parallelized by using a separate estimate/calculate process for each of a table/index's partitions instead of processing serially for the whole table/index. Partitioning also permits statistics to be recalculated for those objects that have changed significantly, instead of recalculating the whole table.

## AGGREGATION CREATION

The final step in the build process is creating aggregates. Aggregation consists of restructuring or creating summary tables of the base data that is loaded, and it is an important part of the building process of data warehouses.

A few simple ways exist to expedite creating aggregates:

- Create aggregates concurrently.
- Do insert as select using the parallel, no-logging, and direct options.
- Limit the amount of data accessed by partitioning both the base object and the aggregate to what will be processed.

Aggregate creation should maximize both CPU and disk usage. Creating aggregates often can be an I/O bound process, so it is best to both create multiple aggregates concurrently as well as creating each one in parallel.

New options exist in Oracle8 that can be useful for creating aggregates. This includes the ability to alter the table to no-logging so that redo is not produced for it, and executing an append with the insert statement which is the equivalent of using direct in SQL*Loader.

Aggregation runs much faster if it only has to hit the rows needed by the aggregation procedure. For example, when running a weekly aggregate, it is much better to access a table with only one week of data instead of three months of data. If the base objects being aggregated are partitioned at the granularity of aggregation (in this case weekly), partition elimination correctly limits the scope of the data being accessed. Similarly, if the aggregate object is partitioned at the same range, inserts can be established to occur as if they were to a new table. Figures Four and Five illustrate the non-partitioned and the partitioned cases, respectively.

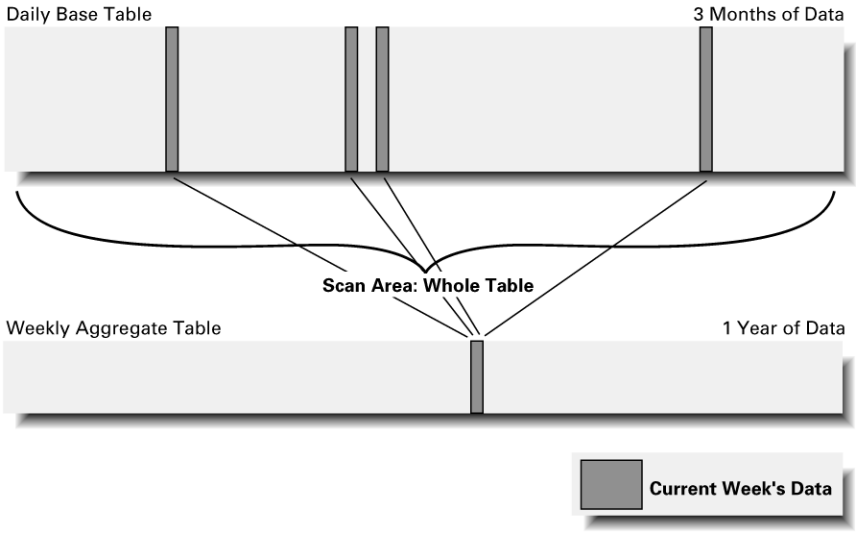


*Figure 4: Aggregation without Data Partitioning*

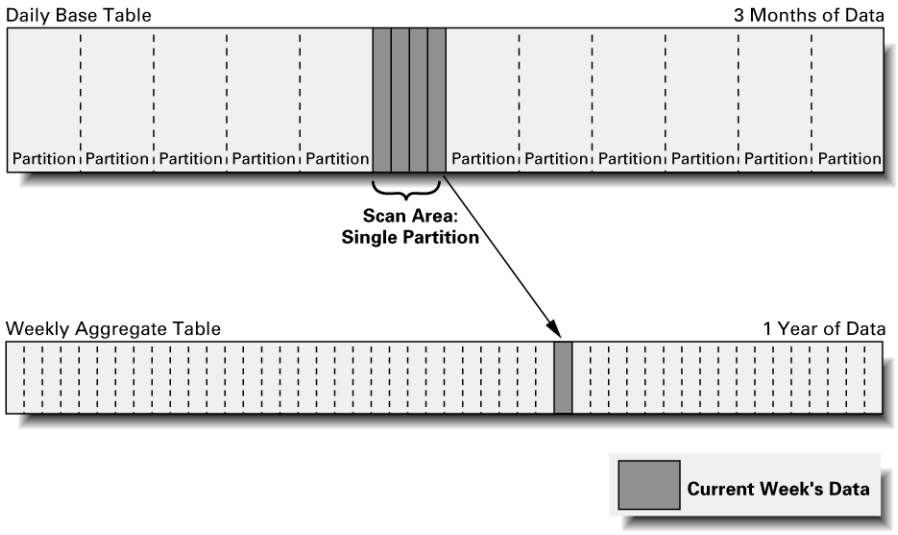


*Figure 5: Aggregation with Data Partitioning*

Use of partitions for aggregation also allows faster parallel inserts, along with creating the index for the partition in parallel, instead of being serialized by using the global index. Other advantages of using this method are a reduction in pinging, a reduction in overall fragmentation as each partition can be managed separately, and faster deletes of a partition by truncating an individual partition.

## CONCLUSION

Proper tuning of the database build process can significantly reduce the time required to build, populate, and analyze large database systems. Concentration of the tuning efforts should be done on those areas that are the greatest consumers of the build time. Other areas, although they could be tuned, will not return as many benefits. Not all systems need this amount of tuning. Those that do, such as test environments, can significantly take advantage of the shorter turnaround time made possible by following these three simple rules:

1. Avoid unnecessary work:
   - Using direct and no-logging options whenever possible
   - Limiting statistical calculation to what is really needed
   - Using pipes as an intermediary file format
   - Sorting data where it is the most efficient to do so
2. Parallelize and concurrently conduct work as much as possibly allowed by the resources of the system.
3. Use Oracle Partitioning to take advantage of partition independence, partition elimination, and data locality.

ORACLE®

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
+1.415.506.7000
Fax +1.415.506.7200
http://www.oracle.com/