PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 9/46, 12/00, 17/30

A1

(11) International Publication Number: **WO 00/31635**

(43) International Publication Date: 2 June 2000 (02.06.00)

(21) International Application Number: PCT/US99/27835

(22) International Filing Date: 24 November 1999 (24.11.99)

(30) Priority Data:
09/200,254 25 November 1998 (25.11.98) US

(71) Applicant: COMPUTER ASSOCIATES THINK, INC. [US/US]; One Computer Associates Plaza, Islandia, NY 11749 (US).

(72) Inventor: MIRZADEH, Rosita; 18440 Hatteras Street #37, Tarzana, CA 91356 (US).

(74) Agents: FLIESLER, Martin, C. et al.; Fliesler, Dubb, Meyer and Lovejoy LLP, Suite 400, Four Embarcadero Center, San Fransisco, CA 94111–4156 (US).

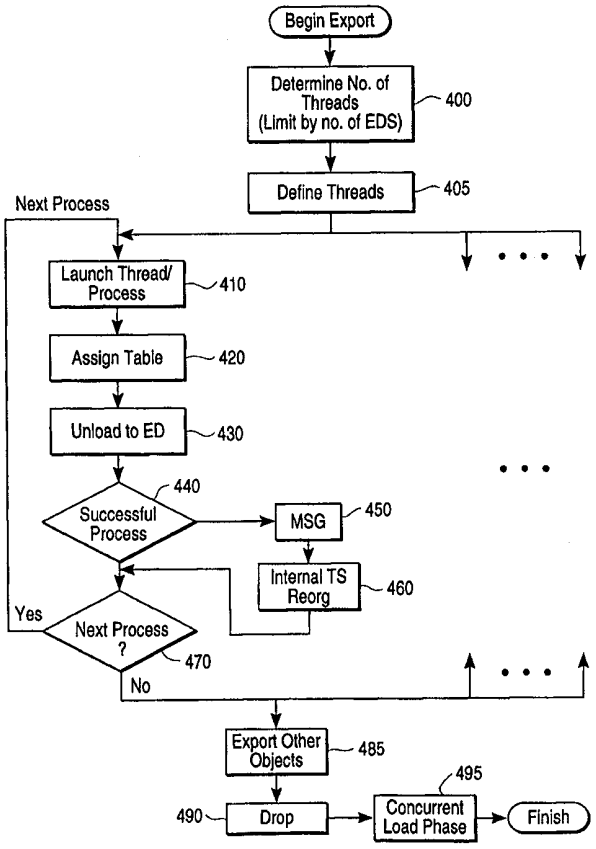
(81) Designated States: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: METHOD AND APPARATUS FOR CONCURRENT DBMS TABLE OPERATIONS

(57) Abstract

Database tables are unloaded by launching a number of threads (400) corresponding to a number of export directories (320) located on separate storage devices (310) that maintain the database tables. Each thread is assigned a database table to unload (420). Data unloaded from each database table is stored in a corresponding export directory (320). The data is unloaded from each database table by reading blocks of data from each table and storing the data logs in the export directory (320). Each thread is handled by a separate process in a Symmetric Multi-Processing (SMP) environment. The process is repeated until each database table has been unloaded. The data is then loaded into database tables by first creating a number of temporary tables corresponding to the number of threads (600), reading a set of data stored in the export directory (320) and storing the data in the temporary tables.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

Method and Apparatus for Concurrent DBMS Table Operations

Background of the Invention

Field of the Invention

This invention relates to a process of unloading and loading a database table. The invention is more particularly related to the application of parallel processing techniques to increase the speed at which database tables are unloaded and loaded. The invention is still further related to parallel processing techniques applied to unloading and loading database tables in a Symmetrical Multi-Processing (SMP) environment.

Discussion of the Background

Modern database management systems are increasingly called upon to maintain larger stores of data. In addition to the increased size of databases, the structure within modern databases is increasingly complex.

Typically, a database maintains data in the form of tables, each table maintaining one or more rows of related data. As an example, a basic database table may maintain plural rows having, for example, name, social security number, address and telephone number of individuals belonging to an organization.

The database would be increased in size as the organization adds new members, and would increase in both size and complexity as additional information about members is included. For example, a larger and more complex database could maintain, in addition to the above information, a map, perhaps in graphical format, showing the club members' residence. The

database could again be increased in size and complexity by including a work address and an additional graphical map showing the location of the work place.

The database table may again be increased in complexity by maintaining pointers to other tables or rows of the database. For example, a pointer to a set of coworkers maintained in another table, pointers to nearby organization members, or a pointer(s) to any number of items to where additional member data may be maintained.

Conventional Database Management Systems (DBMS) provide space for building database tables by allocating blocks. Once a table is defined, the DBMS will allocate blocks necessary for storing rows of the related data. For example, if a table is to be built to contain 100,000 rows, and the rows of the table were defined such that 100 rows could fit per block, the DBMS would allocate 1,000 blocks to build the table.

Generally, DBMS systems allocate blocks in sets of contiguous blocks. A contiguous set of allocated blocks is commonly referred to as an extent. As a general rule, extents vary in size. Using the above example, the DBMS may utilize a single extent of 1,000 blocks, 2 extents of 500 blocks, or any other combination of extent sizes to allocate the required 1,000 blocks to build the table. Once the required blocks are allocated, the data is then stored in rows in the table utilizing the blocks allocated.

Over the course of time, additional data may be added to the table and the DBMS will allocate additional blocks as required. For example, if a user adds 250 rows to the table, using the above parameters, an additional 3 blocks would need to be allocated.

Also over the course of time, information in the database may be deleted. In this case, deletion of rows occurs through the use of SQL to delete rows from the table. For example, a user may delete fifty rows from block 1, 40 rows from block 20, and 30 rows from block 60. When this occurs, it causes there to be more blocks in the table than required to maintain the data present in the table.

In addition, data within the database will be updated. For example, using the above-described database tables, a organization member who has not yet entered the workforce would have a row in the table allocated that maintained his/her name, address, social security and telephone number. Upon entering the workforce, the row would be updated to include the work address and related information. However, if a substantial amount of information is added as a result of the update, the original row may not have enough blocks of data allocated to maintain the updated information.

Row migration occurs when a row gets updated and the original row does not have enough space to hold all of the updated information. When this occurs, the row is moved to a location with more space, and a pointer is placed in the block where the original row resided, the pointer being set to point to the location of the moved (migrated) row. A large amount of row migration is caused if there are substantial updates to a table and an inadequate amount of space was allocated for the original rows.

More often than not, there is insufficient space within a block to hold an updated row. In this case, the row is migrated to an entirely different block than where the original row resided, and the pointer is

placed in the original row position points to the updated row in the different block.

Anytime a row is migrated it causes what is known as fragmentation. Fragmentation causes greatly increased retrieval time of database information because in addition to reading a block, a pointer must be read and interpreted. When a row is migrated to another block, at least two blocks (the block containing the pointer, and the block containing the migrated/fragmented row) must be read to retrieve row specific information. Other structural changes within the database tables also cause fragmentation and related efficiency problems (row chaining, for example).

From time to time, the Database Administrator (DBA) will perform an analysis on the DBMS tables that provides information regarding the condition of the tables. For example, the database administrator may look at information regarding the number of deleted rows to ascertain efficiency information with respect to how many blocks are holding deleted rows. As another example, the Database Administrator may look to see how many rows of a table have been migrated or fragmented by other processes.

If a lot of fragmentation has occurred, it indicates that block size and row space could be more efficiently allocated and that table data is not being retrieved efficiently. When this occurs, the database administrator will likely decide to rebuild the tables.

When creating a table, the DBA makes a decision regarding the structure of a database table by setting a percentage of blocks free (PCTFREE) or percentage of blocks used (PCTUSED). As the DBMS fills up each block with row or table information, it will keep a

percentage of a block free at least equal to the percentage PCTFREE.

The DBA sets the PCTFREE variable depending on how the database table is to be used. For example, if a table is to have frequent updates, additional PCTFREE would be established so that enough space is available to allow any necessary row migration to occur within the same block. As discussed above, row migration within the same block does not cause a table to become fragmented. Migrated, but non-fragmented rows are retrieved with a single block read rather than the cumbersome process of reading a block, interpreting a pointer, and reading a second block (or more) as occurs when migrated rows are fragmented. Therefore, appropriate PCTFREE settings allow DBMS performance to be maintained although the database tables may be modified.

PCTUSED is another parameter that allows the DBA to control the structure of a DBMS table. The DBMS prevents additional rows to be placed in a block unless the percentage of that block has fallen below PCTUSED. PCTUSED is different from PCTFREE in the sense that although a block may be used if there is an update, it will not be used to insert a new row unless the percentage used in the block is below PCTUSED.

A DBMS table involved in heavy OLTP activity (inserts, updates and deletes) over time will likely experience row migration, fragmentation, row chaining, etc. Furthermore, various database tables may not necessarily have appropriate settings (PCTFREE, PCTUSED, for example) when first built, or the needs of the database table may have changed, resulting in additional migration, deletion or fragmentation of

tables. This in turn results in a degradation of data retrieval performance and space usage.

The DBA will perform an analysis to determine whether the tables are storing data efficiently. As a result, one or more of the DBMS tables may be determined to being inefficient in storing and retrieving data. Reorganization (rebuilding) of the table is a solution to this problem. In order to achieve maximum performance the table needs to be rebuilt (i.e., the data unloaded into secondary space and a fresh instance of the table rebuilt). This process gets rid of many of the unwanted effects mentioned above because the fragmented rows are unloaded and stored without fragmentation in the rebuilt table.

Structurally sound databases make efficient use of disk space. They require less time to access data, reduce the time required for normal transactions, and provide better response time to the user. Even though Oracle and other modern database systems use efficient logic for data placement, normal activity over time causes the physical layout of the data on disk to degrade and space requirements to grow. This results in excessive space usage and extra time needed to perform table scans, database backups, and other functions. Partial empty pages and unused extent space contribute to the additional space usage. Also, nonsequential rows and extent interleaving seriously degrade performance if they are not resolved periodically.

One method to ensure that databases stay efficient (increasing productivity) is to regularly perform reorganizations on the databases' data. Currently, products are available to reorganize DBMS tables.

However, even with automated reorganization tools, reorganization of database tables can require substantial amounts of time. The time required to perform a reorganization can have substantial effects on revenue or productivity of a shop that requires database access. For example, some shops can incur costs of approximately $100,000 an hour when a database is off-line (See Dec. 1997 issue of Oracle magazine, "Reorgs in a Non-Stop Shop," for example). Therefore, any improvement in efficiency or speed at which a reorganization is performed would increase competitiveness and profitability.

Summary of the Invention

The present inventor has realized the need to provide fully parallel operations supporting database table unloading and loading which will increase the speed of any database operations performing either one or both of table unloading and table loading.

Accordingly, it is an object of the present invention to provide a parallel processing technique that allows for parallel database table unloads.

It is another object of this invention to provide a method for unloading and loading database tables utilizing parallel processing techniques in a Symmetric Multi-Processing (SMP) environment.

It is yet another object of this invention to provide a method for preventing bottlenecks in parallel unloading and loading of database tables.

It is still yet another object of the present invention to increase the speed at which a reorganization of database tables is performed by utilizing the parallel processing techniques of the present invention.

These and other objects are accomplished by a method for performing parallel unloading of database tables, including the steps of launching a number of threads to process the database tables, assigning a respective one of the database tables to a corresponding of said threads and unloading each respective database table by a process of the corresponding thread. The method allows the table unload process to take advantage of a Symmetric Multi-Processing (SMP) environment to significantly improve the speed of database table unloads.

The invention includes a method of parallel loading of table data including the steps of loading data into database tables, including the steps of determining X threads for loading data into database tables, creating X temporary tables, each temporary table corresponding to a set of data stored in an export directory, launching an SQL*LOADER™ process for loading each temporary table, and loading each temporary table with the data stored in the corresponding export directory via the corresponding SQL*LOADER™ process.

Brief Description of the Drawings

A more complete appreciation of the invention and many of the attendant advantages thereof will be readily obtained as the same becomes better understood by reference to the following detailed description when considered in connection with the accompanying drawings, wherein:

Figure 1 is a flow chart illustrating a database fast unload/load(PDL)procedure;

Figure 2 is a flowchart illustrating the iterative nature of non-concurrent database unload procedure;

Figure 3 is a flowchart illustrating high level processes for concurrent load and unload of database tables according to the present invention;

Figure 4 is an illustration of parallel processes performed on a single export disk;

Figure 5 is an illustration of parallel processes performed using multiple export disks according to the present invention;

Figure 6 is an illustration of plural storage devices maintaining database tables and export directories according to the present invention;

Figure 7 is a flowchart illustrating export (unload) process according to the present invention;

Figure 8 is an illustration of a Destination Tab Property Page according to the present invention;

Figure 9 illustrates a Parallel Query Tab page according to the present invention;

Figure 10 is a flowchart illustrating a parallel load process according to the present invention;

Figure 11 illustrates a Concurrent Table Reorganization with a parallel export phase and a parallel direct load according to the present invention;

Figure 12 is a flowchart illustrating a fail safe/fault recovery system for the unload/load processes according to the present invention; and

Figure 13 is a flowchart illustrating high level table name and loading processes according to the present invention.

Detailed Description of the Invention

CONCURRENT UNLOAD/LOAD OPTION IN TS REORGANIZATION

The present inventor has developed a new Concurrent Unload/Load methodology. This methodology will be utilized in an Unload/Load option to be introduced in Platinum TS reorg V2.1.0 to enhance the speed of tablespace reorganization. Previously, the fastest way to reorganize a tablespace in TS reorg was to use Fast Unload in conjunction with the Parallel Direct Load option.

Referring now to the drawings, wherein like reference numerals designate identical or corresponding parts throughout the several views, and more particularly to Figure 1 thereof, is a flowchart illustrating a Fast Unload/Load process. At step 100, a single Fast Unload process unloads each table, and steps 100a . . 110n illustrate parallel direct loads utilized to load tables.

Figure 2 is flow diagram detailing the Fast Unload process. At step 200, a Fast Unload is performed on a table. The Fast Unload reads and stores data from the table. At step 210, it is determined whether a next table in the database(s) is to be unloaded and the process is repeated until each table is unloaded.

As illustrated by the process in Figures 1 and 2, in Parallel Direct Load, the tables are unloaded sequentially using one single process on the unload and multiple SQL*LOADER™ (an ORACLE utility) processes on the load, while working on only one table at a time. For example, when unloading a tablespace containing a large number of tables, TS reorg must wait for one

table to be completely unloaded before it can process the next table.

Figure 3 illustrates the Concurrent Unload/Load option of the present invention that enables a user to take advantage of multi-CPU machines to unload (step 300) and load (step 305) multiple tables concurrently. This method is particularly efficient when the tablespace contains a combination of both large and small tables. Then, while TS reorg unloads and loads one large table, it can also unload and load several small tables simultaneously. The principles of the present invention are also extended to the concurrent building of multiple indexes and/or constraints.

The Concurrent Unload/Load process of the present invention are best practiced in a computing and database environment having the following characteristics:

(1) Symmetric Multi Processing (SMP) computing environment - Appropriate computing hardware contains multiple CPUs and must accommodate a symmetrical multiprocessing environment. The higher the symmetrical multiprocessing capacity of a host machine, the higher performance potential.

One type of multiprocessor computer is a *symmetric* multiprocessor (SMP) computer. An SMP computer usually has between 2 and 16 processors, all of which share the computer's single memory source and shared storage devices. The SMP capacity depends on the number of processors available. With more processors, the present invention allows more queries and loads to be performed in parallel.

(2) Datafiles partitioned across multiple disks - The datafiles of the tablespace to unload reorganization must be stored on different physical

disks. This requirement is needed to obtain full benefit from the Concurrent Unload/Load processes of the present invention.

(3) Defined export directories on disks - A number of threads used for the Concurrent Unload/Load option processes will be less than or equal to the number of export directories. These directories are best utilized when residing on separate physical disks.

Even with a high SMP capacity and efficient parallel operations, data movement can experience a *bottleneck*, or a halting reduction in speed, because of the limited bandwidth of physical storage disks. One of the few limitations of SMP occurs when physical disks cannot quickly accommodate the simultaneous read and write requests made by the numerous CPUs.

Another source of interrupted data, especially in parallel processing, is *disk contention*. When multiple processes attempt to access and change a database, the disk head can serve the request of only one process at a time, causing the other operations to wait. This situation of two processes simultaneously trying to access the same disk is called disk contention. The result of disk contention is clogged data, or a bottleneck.

Since parallel processes use multiple CPUs to move data between memory and disks, it is important to have multiple disks defined, so that the numerous CPUs can quickly move the data without interruptions or waiting.

One way to eliminate bottlenecks is by *partitioning* the data to multiple disks. Partitioning data is the process of physically spreading the data across multiple disk drives to reduce the limiting effects of disk I/O bandwidths and disk contention. The more disks you define (partition) for data export,

the more you reduce I/O bottlenecks, which results in faster parallel operations.

Figure 4 illustrates parallel processes using only one export disk. Even though the server's query coordinator breaks the query into two separate operations, the disk head can serve the request of only one scan at a time, causing the other operations to wait, or bottleneck. In this example, the single disk's I/O limitations defeat the purpose of the server's parallel process features. Resolving this problem depends on proper disk allocation.

Figure 5 illustrates one embodiment of a proper disk allocation, and parallel processes using numerous export disks (partitioned data). The parallel processes run simultaneously, rather than one process waiting for the other.

Using Concurrent Threads in TS reorg

The Concurrent Load/Unload processes of the present invention utilize a number of defined threads that determines a number of tables to be unloaded/loaded at the same time. In other words, the number of concurrent threads is equal to the number of processes that TS reorg launches in the unload and load phase. Each of these processes works on one table.

The number of defined threads is limited by a number of export directories. Figure 6 illustrates an SMP computing environment having three separate disks, 310a, 310b, and 310c, each disk storing datafiles and each having an export directory 320a, 320b, and 320c. In the illustrated environment the Concurrent Load/Unload would define three threads (one per export directory) for unloading tables.

Concurrent Unload Phase

Figure 7 is a flowchart illustrating the Concurrent Unload phase of the present invention. At step 400, a number of threads to be utilized in a current computing environment is determined. The number of threads to utilize is provided by a user via a GUI or other input device. However, the number of threads is also limited to a maximum equivalent to a number of export directories residing on separate disk drives. For example, TS reorg now utilizes a destination Tab Property Page to enter, change or delete the export directory paths designating the location for the unloaded data and other export files during a tablespace reorg (see Fig. 8).

During the export phase of a tablespace reorg, the reorg utility unloads the Data Description Language (DDL) and table data from the tablespace and exports it to a specified directory. The Host Directory Path(s) for Export is the directory path designating the location for the unloaded DDL and data during a reorg.

The Destination tab property page is utilized to do the following:

- Add a host export directory.
- Delete a host export directory.
- Modify a host export directory.

If insufficient space available in the export directory or the export file size reaches the *ulimit* while the reorg utility is unloading data, the reorg utility exports the remaining data to the next specified export directory. If another export directory is not specified, the TS reorg will not perform the reorganization.

This removes the 2-gigabyte limit common on most UNIX platforms. Since a compressed file cannot be

split, this does not apply to a compressed export method (described hereinafter).

At step 410, TS reorg launches a same number of Fast Unload processes as the number of defined threads. This is referred to as the first set of Fast Unload processes.

At step 420, a table is assigned to each process. Each Fast Unload process is executed by the corresponding thread and unloads the assigned table into one of the separate export directories (step 430).

To search for a directory on which to unload data, each thread checks all of the export directories and chooses the smallest available directory that can accommodate the created export file. Such a method is known as finding the *best fit* for the export directories. Once a thread finds a *best fit* directory, it claims that directory, so the next thread must choose another.

In the event that other applications are using space on the same file system, a thread might not load all its data into its chosen *best fit* directory. If this occurs, the reorg utility splits the export file into multiple export files and attempts to write it on another export directory. If there is no alternate export directory, or if the disk is full, the thread terminates and one of the remaining threads unloads the remaining data.

A user may estimate the size of the user's export files in order to assign enough space in an export directory for those files. To estimate the size of an export file, divide the current allocation of an object by the number of threads to be utilized.

The reorg utility utilizes a Parallel Query tab property page, as shown in Figure 9, to fill in fields

along with any existing parallel parameters for the object. For example, a table's parallel parameters might exist if that table was initially created or revised using parallel parameters. Those parameters are retained within the table's Data Description Language (DDL) and the Data Dictionary. If parallel parameters already exist for an object, the reorg utility splits the data query automatically, and a user need not specify values in the Parallel Query tab property page. Otherwise, the reorg utility splits the query according to the parameters set in the Parallel Query tab property page.

For full advantage of the present invention, the number of export directories is equal to or greater than the number of threads. Also, each export directory should reside on a separate physical disk to avoid disk I/O bottlenecks. As discussed above, I/O bottlenecks will likely occur if more than one export directory is located on a same disk because of the possibility of more than one process writing (unloading) to the same disk at the same time.

When the first Fast Unload process of the first set of threads finishes, TS reorg checks if the process was successfully terminated (step 440), if that was the case, the next Fast Unload process to unload a next table is launched (step 470).

In this manner, each table is unloaded by a process executed in it's respective thread. Each thread is independent, therefore if one thread has a process assigned to a large table, the other processes need not wait for the large process to finish before launching the process for the next table.

Otherwise, if a problem occurred during the unload of the table, TS reorg displays a reason why the

process failed (step 450), along with eventually the Fast Unload logfile in the job logfile. The table that a problem occurred is unloaded using an internal unload program of TS reorg. This unload program is not launched as a separate process and therefore it has to be finished before TS reorg can continue with the concurrent unload of the rest of the tables.

Once all the tables have been successfully unloaded, the unload process is completed. A performance meter (estimated, or interactive) may also be implemented to display the speed of unload/load operations and an amount of disk space being consumed.

Once the unload process is completed, TSreorg continues with the next step of the tablespace reorganization which is the export of other objects of the tablespace (step 485).

Export of other objects includes the creation of a files for storing the DDL of the tablespace including all indexes, triggers, constraints, and primary or unique keys (everything except the table and clusters).

Finally, TS reorganization performs a drop on the unloaded tables (step 490), and then performs a concurrent load (step 495) loading the unloaded data into fresh tablespace.

Concurrent Load Phase

After dropping all of the tablespace objects, TS reorg begins the load (import) phase of the reorganization. The concurrent load phase is illustrated in the flowchart of Figure 10. At step 600, TS reorg begins the load phase by creating tables to store the data. TS reorg sequentially creates as many tables as the number of threads specified. The

specified number of threads is equivalent to the number of threads in the unload phase.

After the first set of tables is created, TS reorg simultaneously launches the SQL*LOADER™ processes used to load the data of each corresponding table. The number of processes launched is equal to the number of threads. Each SQL*LOADER™ process loads data for one corresponding table, reading table data from an export directory and loading that data into the corresponding table (step 610).

The reorg utility recreates the objects, eliminating fragmentation and optimizing storage parameters, using the DDL in the export files. The reorg utility coordinates multiple CPUs in parallel threads to load the data back into the objects, resulting in a reorganized tablespace.

After one of these processes is completed, TS reorg checks if the data was successfully loaded back into the table and if the number of rows inserted was correct (Step 620). If an error occurred during the load or the number of rows inserted by SQL*LOADER™ was not correct, TS reorg switches automatically to the internal TS reorg load function to load sequentially the data before continuing with the next table (step 630). If there was no error with the SQL*LOADER™ process, or after the internal load terminated without error, TS reorg creates the next table (step 650) and launches concurrently still another SQL*LOADER™ process to load this table's data (repeating step 610).

As illustrated in the concurrent Table Reorganization diagram shown in Figure 11, a concurrent direct load invokes multiple CPUs, each of which executes a separate SQL*LOADER™ session to synchronously load data from the export files back into

the object's datafiles. When SQL*LOADER™ writes to these datafiles, the reorg utility uses the best fit method, which selects those partitioning directories that have enough space to accommodate the incoming data. The reorg utility then continues using the best fit method within that selected list of directories and chooses the smallest datafile that can accommodate all of the incoming data for the thread. Each thread loads its data into as much free space in a datafile as it can.

If a fatal error occurs during the load phase, or if the reorganization job is canceled or killed for any reason, the failed job becomes a job that needs recovery. As shown in Figure 12, after a job has failed (step 1105), TS reorg automatically skips all the tables that have been already successfully created and loaded before the failure (step 1110) and only loads concurrently the non-existing tables or the tables that were not completely loaded (step 1115).

As shown in Figure 13, during the load phase, TS reorg first creates the tables under a temporary name (step 1300). After the data has been reloaded the temp table is renamed to the original name of the table (step 1320). Finally, the primary constraints and indexes on the table are created (step 1330). This allows TS reorg to recognize the tables that were not completely imported before the failure and to drop all temporary tables and restart the load for those tables (see process 1115A, Fig. 11).

When using the Concurrent Unload/Load option, TS reorg automatically selects the option: Create all Indexes/Constraints after all of the Tables have been Created. The indexes and constraints of this tablespace are created after all of the tables are

successfully created and their data reloaded into the tables.

CONCURRENT INDEX CREATION IN TS REORG

Concurrent index creation is an option can be used in a table or a tablespace reorganization to increase the speed of index creation during the reorganization. In a regular table or tablespace reorganization, the indexes are created one after another after the table is created and the data loaded back into the table.

In a tablespace reorg, the user may select a concurrent index creation option. In this case, TS reorg will create the indexes concurrently (sequential index creation is the default behavior). This will allow TS reorg to gather all the indexes and constraints in one export file and when all the tables of the tablespace (in case of a tablespace reorganization) or the table to reorganize (in case of the table reorganization) has been created, it starts to create these indexes and constraints concurrently.

When selecting this option, the user has to specify a number of threads which would be used as the number of process to launch concurrently during the index/constraint creation phase.

When the import phase for indexes begins, TS reorg launches the same number of Index creation processes as the number of specified threads. When the first index process of the first set of threads finishes, a next index creation process is launched and subsequent index creation processes are similarly launched upon completion of other threads until all the indexes and constraints are created.

If an error occurs on the creation of one index or constraint, TS reorg logs the error and writes the DDL of the failed index into a file. The user can then manually edit and fix the problem.

The present invention has been described with reference and in terms consistent with an implementation in conjunction with a Oracle database. However, the processes described are equally applicable to other known database products and custom database installations. For example, instead of utilizing the Oracle utility, SQL*LOADER™, another program capable of reading table data stored in an export directory and loading the data into fresh table space may be utilized.

The present invention may be conveniently implemented using a conventional general purpose or a specialized digital computer or microprocessor programmed according to the teachings of the present disclosure, as will be apparent to those skilled in the computer art.

Appropriate software coding can readily be prepared by skilled programmers based on the teachings of the present disclosure, as will be apparent to those skilled in the software art. The invention may also be implemented by the preparation of application specific integrated circuits or by interconnecting an appropriate network of conventional component circuits, as will be readily apparent to those skilled in the art.

The present invention includes a computer program product which is a storage medium (media) having instructions stored thereon/in which can be used to program a computer to perform any of the processes of the present invention. The storage medium can include,

but is not limited to, any type of disk including floppy disks, optical discs, DVD, CD-ROMs, microdrive, and magneto-optical disks, ROMs, RAMs, EPROMs, EEPROMs, DRAMs, VRAMs, flash memory devices, magnetic or optical cards, nanosystems (including molecular memory ICs), or any type of media or device suitable for storing instructions and/or data.

Stored on any one of the computer readable medium (media), the present invention includes software for controlling both the hardware of the general purpose/specialized computer or microprocessor, and for enabling the computer or microprocessor to interact with a human user or other mechanism utilizing the results of the present invention. Such software may include, but is not limited to, device drivers, operating systems, database engines and products, and user applications. Ultimately, such computer readable media further includes software for performing the present invention, as described above.

Included in the programming (software) of the general/specialized computer or microprocessor are software modules for implementing the teachings of the present invention, including, but not limited to, retrieval of user inputs and the determination of a number of threads for parallel processing, launching threads, unloading database tables, determining success, initiating internal reorganization processes, exporting database objects, compressing unloaded data, monitoring the processes of the present invention, and setting up a concurrent load environment utilizing a table load utility, and the display, storage, or communication of results according to the processes of the present invention.

Obviously, numerous modifications and variations of the present invention are possible in light of the above teachings. It is therefore to be understood that within the scope of the appended claims, the invention may be practiced otherwise than as specifically described herein.

CLAIMS

What is claimed is:

1. A method for unloading database tables, comprising the steps of:

launching a number of threads to process the database tables;

assigning a respective one of said database tables to a corresponding of said threads; and

unloading each respective database table by a process of the corresponding thread.

2. The method according to Claim 1, wherein said step of launching includes the steps of:

retrieving a number X of threads input by a user;

identifying a number of export directories located on separate storage devices that maintain said database tables;

limiting X to the number of export directories identified; and

utilizing X as said number of threads.

3. The method according to Claim 2, wherein said storage devices are disk drives.

4. The method according to Claim 1, wherein said step of unloading comprises the steps of:

reading blocks of data from a respective database table, and storing the data blocks read in an export directory.

5. The method according to Claim 4, wherein said step of unloading further comprises the steps of:

determining success of completion said steps of reading and storing; and

(1) stopping each of said threads, if said determining success step indicates non-completion,

(2) performing an internal TS Reorganization on the table.

6. The method according to Claim 1, further comprising the step of:

repeating said steps of assigning and unloading until each of the database tables is unloaded.

7. The method according to Claim 1, further comprising the steps of:

exporting other objects related to each of said tables; and

dropping said tables.

8. A method of loading data into database tables, comprising the steps of:

determining X threads for loading data into database tables;

creating X temporary tables, each temporary table corresponding to a set of data stored in an export directory;

launching an SQL* Loader process in conjunction with each thread for loading each temporary table;

loading each temporary table with the data stored in the corresponding export directory via the corresponding SQL* Loader process.

9. The method according to Claim 8, further comprising the steps of:

determining success of the loading step for a respective database table; and

if said step of loading was unsuccessful, performing the steps of:

(1) stopping each of said threads, if said determining success step indicates non-completion,

(2) performing an internal TS Reorganization on the table.

10. The method according to Claim 8, further comprising the steps of creating, launching and loading until each table is loaded.

11. The method according to Claim 8, further comprising the steps of:

recognizing a fault in said step of loading, and performing the steps of:

(1) recognizing unsuccessfully loaded tables; and

(2) performing an internal TS Reorganization on the unsuccessfully loaded tables.

12. The method according to Claim 8, wherein said step of determining includes the steps of:

retrieving a number of threads input by a user;

determining a number of export directories; and

establishing a number of threads equal to the lesser of the number of threads retrieved and the number of export directories.

13. A computer readable medium having computer instructions stored thereon that, when loaded into a computer, cause the computer to perform the steps of:

launching a number of threads to process the database tables;

assigning a respective one of said database tables to a corresponding of said threads;

unloading each respective database table by a process of the corresponding thread.

14. The computer readable medium according to Claim 13, wherein said step of launching comprises the steps of:

identifying a number X of export directories located on separate storage devices that maintain said database tables; and

utilizing X as said number of threads.

15. The computer readable medium according to Claim 14, wherein said storage devices are disk drives.

16. The computer readable medium according to Claim 13, wherein said step of unloading comprises the steps of:

reading blocks of data from a respective database table, and storing the data blocks read in an export directory.

17. The computer readable medium according to Claim 16, wherein said step of unloading further comprises the steps of:

determining success of completion said steps of reading and storing; and

(1) stopping each of said threads, if said determining success step indicates non-completion,

(2) performing an internal TS Reorganization on the table.

18. The method according to Claim 13, further comprising the step of:

repeating said steps of assigning and unloading until each of the database tables is unloaded.

19. The method according to Claim 13, further comprising the steps of:

exporting other objects related to each of said tables; and

dropping said tables.

20. A computer readable medium having computer instructions stored thereon that, when loaded into a computer, cause the computer to perform the steps of:

determining X threads for loading data into database tables;

creating X temporary tables, each temporary table corresponding to a set of data stored in an export directory;

launching an SQL* Loader process for loading each temporary table;

loading each temporary table with the data stored in the corresponding export directory via the corresponding SQL* Loader process.

21. The computer readable medium according to Claim 20, wherein said instructions stored thereon, when loaded into a computer, further cause the computer to perform the steps of:

determining success of the loading step for a respective database table; and

if said step of loading was unsuccessful, performing the steps of:

(1) stopping each of said threads, if said determining success step indicates non-completion,

(2) performing an internal TS Reorganization on the table.

22. The computer readable medium according to Claim 20, wherein said instructions stored thereon, when loaded into a computer, further cause the computer to perform the step of repeating said steps of creating, launching and loading until each table is loaded.

23. The computer readable medium according to Claim 20, wherein said instructions stored thereon, when loaded into a computer, further cause the computer to perform the step of:

recognizing a fault in said step of loading, and performing the steps of:

(1) recognizing unsuccessfully loaded tables; and

(2) performing an internal TS Reorganization on the unsuccessfully loaded tables.

24. The computer readable medium according to Claim 20, wherein said step of determining includes the steps of:

retrieving a number of threads input by a user;

determining a number of export directories; and

establishing a number of threads equal to the lesser of the number of threads retrieved and the number of export directories.

25. An apparatus for unloading database tables maintained in a system, comprising:

means for launching a number of threads to process the database tables;

means for assigning a respective one of said database tables to a corresponding of said threads; and

means for unloading each respective database table by a process of the corresponding thread.

26. The apparatus according to Claim 25, wherein said means for launching further includes means for retrieving a number of threads to launch from a user.

27. The apparatus according to Claim 25, wherein said means for launching includes means for limiting the number of threads launched to a number of export directories located on separate storage devices of said system.

28. The apparatus according to Claim 25, wherein said means for unloading comprises:

means for reading blocks of data from a respective database table, and storing the data blocks read in an export directory associated with the corresponding thread.

29. The apparatus according to Claim 25, further comprising means for repeatedly assigning respective database tables to corresponding threads and unloading each respective table until all of said database tables have been unloaded.

30. An apparatus for unloading database tables, comprising:

a launching device configured to initiate a number of threads to process said database tables;

an assignor configured to assign each of said database tables to corresponding of said threads launched; and

an unloader installed on each corresponding thread, each respective unloader configured to unload database tables assigned to the corresponding thread of the unloader.

31. The apparatus according to Claim 30, wherein said launching device includes:

a user interface configured to retrieve a number of threads to launch from a user; and

a limit device configured to limit the number of threads to launch to a number of export directories located on separate storage devices of said system.

32. The apparatus according to Claim 30, wherein each respective unloader comprises:

a read device configured to read blocks of data maintained within tables assigned to a same thread as the respective unloader is installed; and

a write device configured to store the data blocks read into an export directory corresponding to the same thread as the respective unloader is installed.

33. The apparatus according to Claim 30, wherein said assignor assigns a database table to each of said threads, and, upon completion of one of said threads, assigns another of said database tables to the completed thread until each database table has been assigned.

34. The apparatus according to Claim 32, further comprising:

a loader, loaded and executed on each of said threads after each database table has been unloaded, each loader configured to,

read data blocks stored in an export directory corresponding to the thread executing the loader, and

save the datablocks in fresh tablespace.

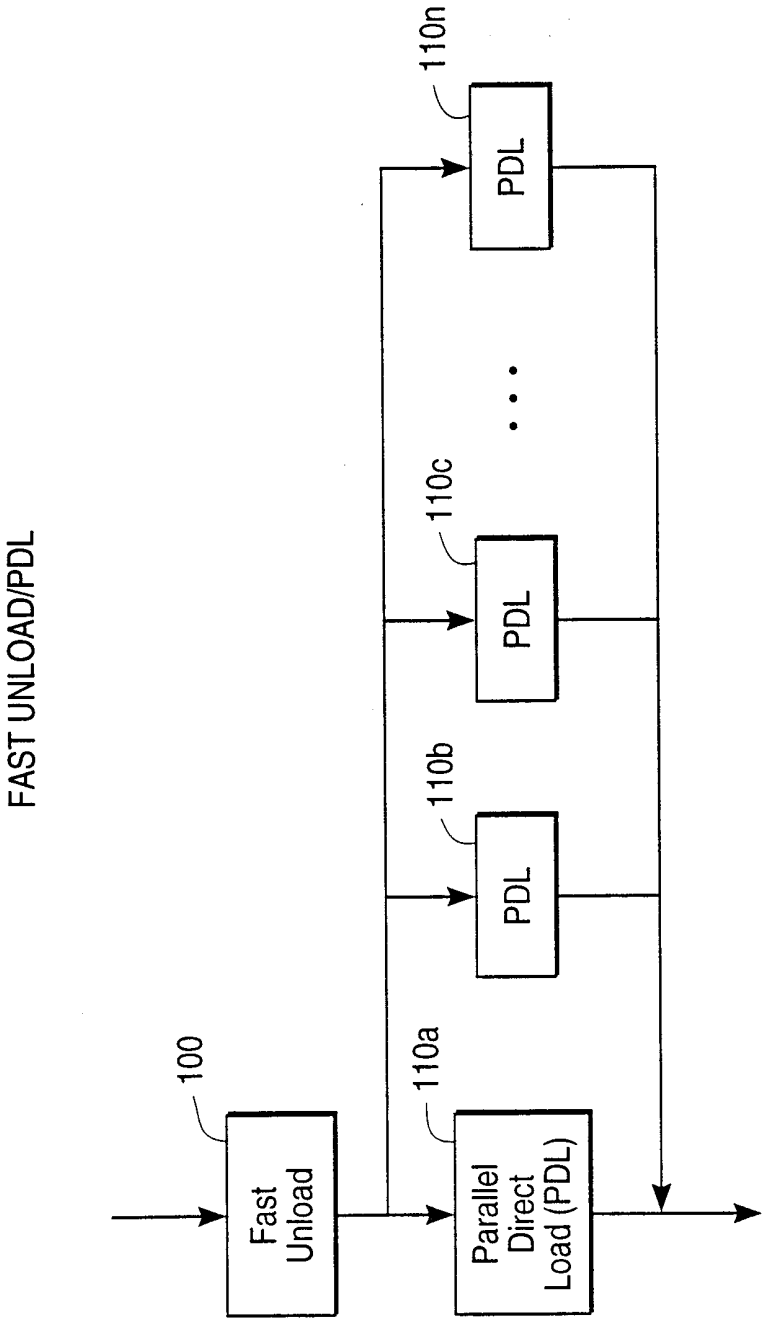

FAST UNLOAD/PDL
100
Fast Unload
110a
Parallel Direct Load (PDL)
110b
PDL
110c
PDL
110n
PDL

FIG. 1

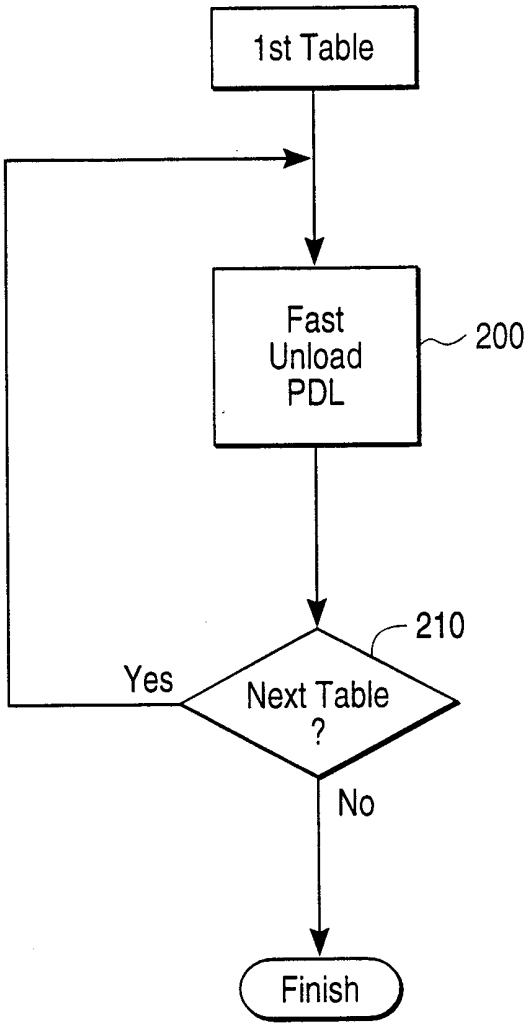


FIG. 2

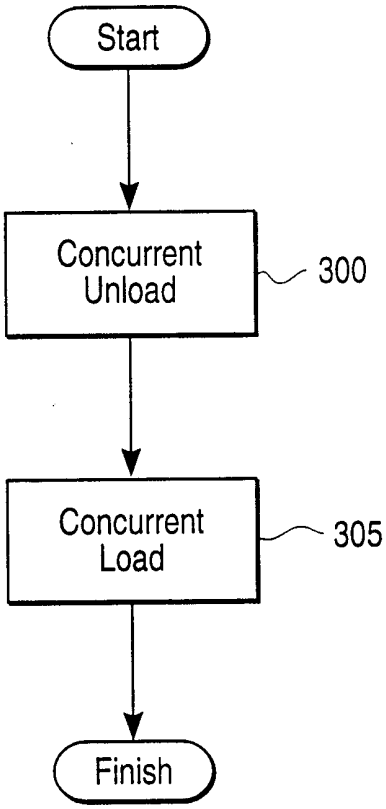


FIG. 3

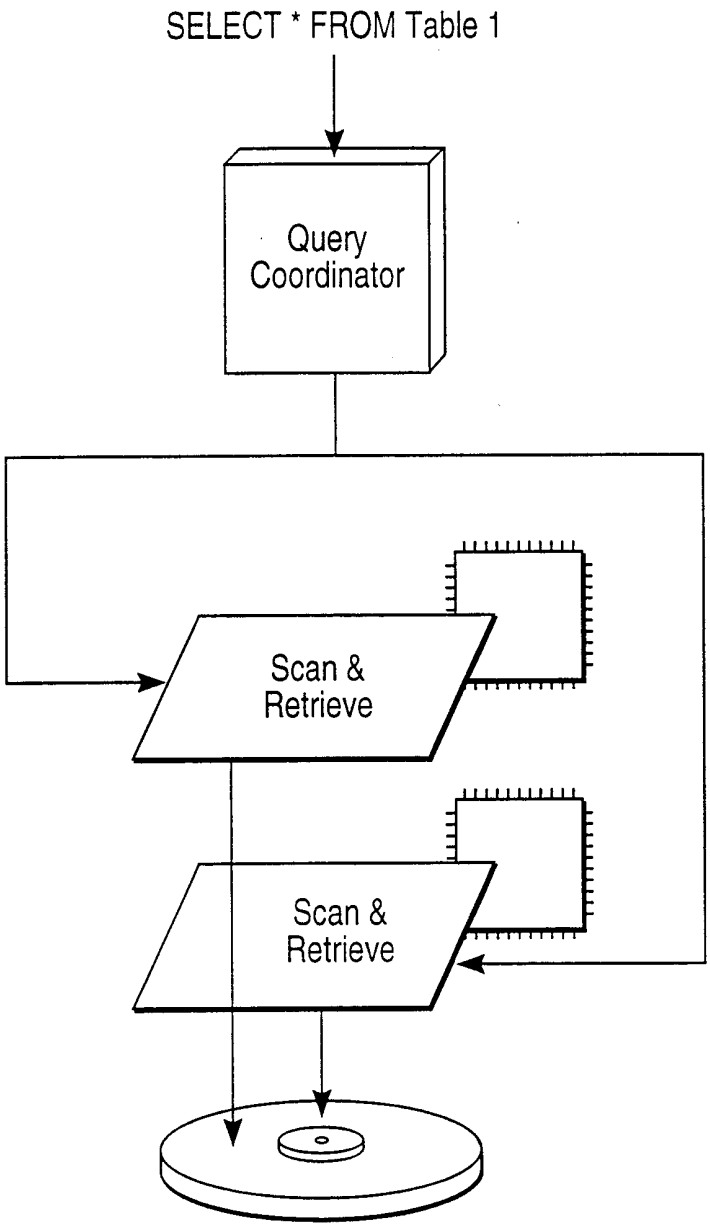

SELECT * FROM Table 1
Query Coordinator
Scan & Retrieve
Scan & Retrieve

FIG. 4

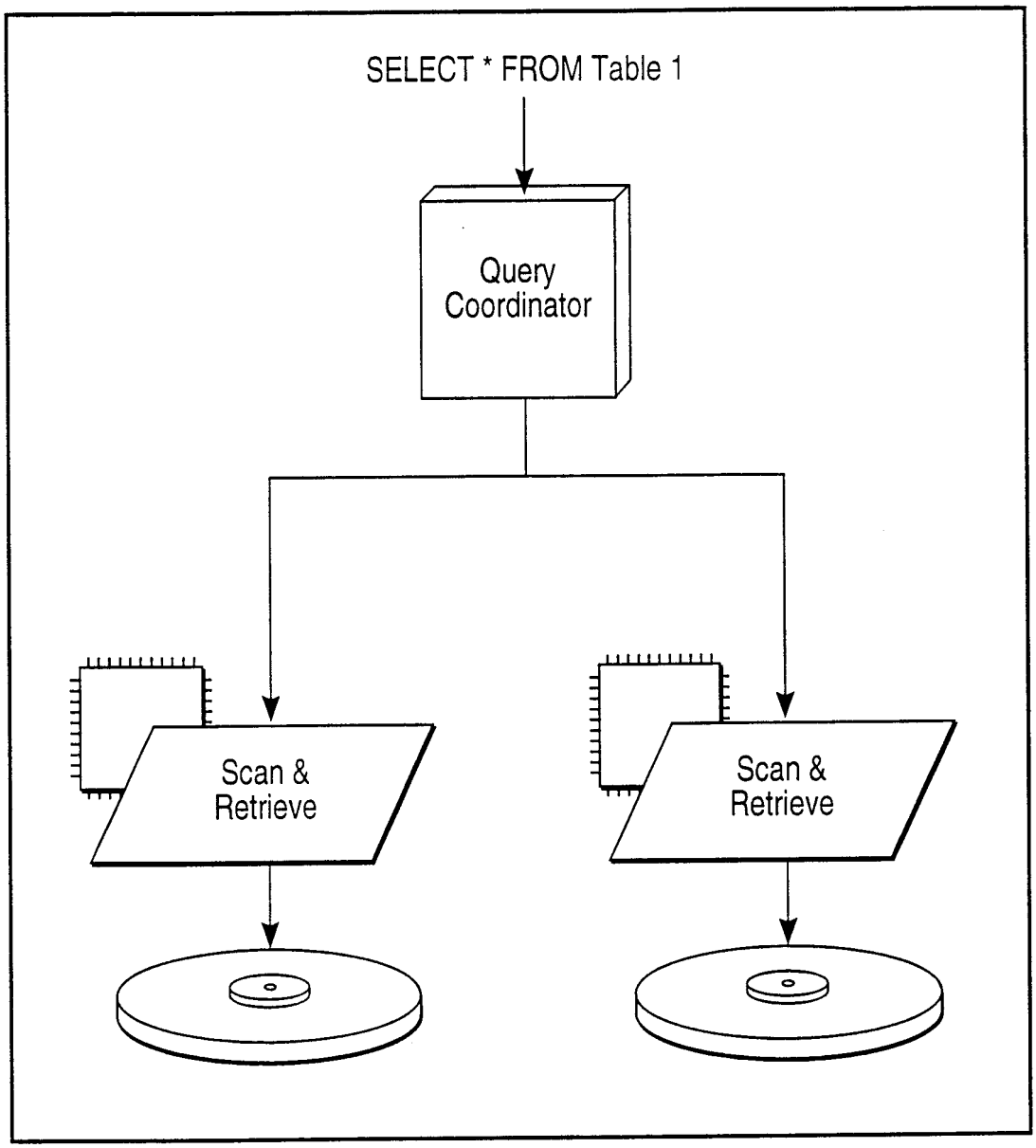

SELECT * FROM Table 1
Query Coordinator
Scan & Retrieve
Scan & Retrieve

FIG. 5

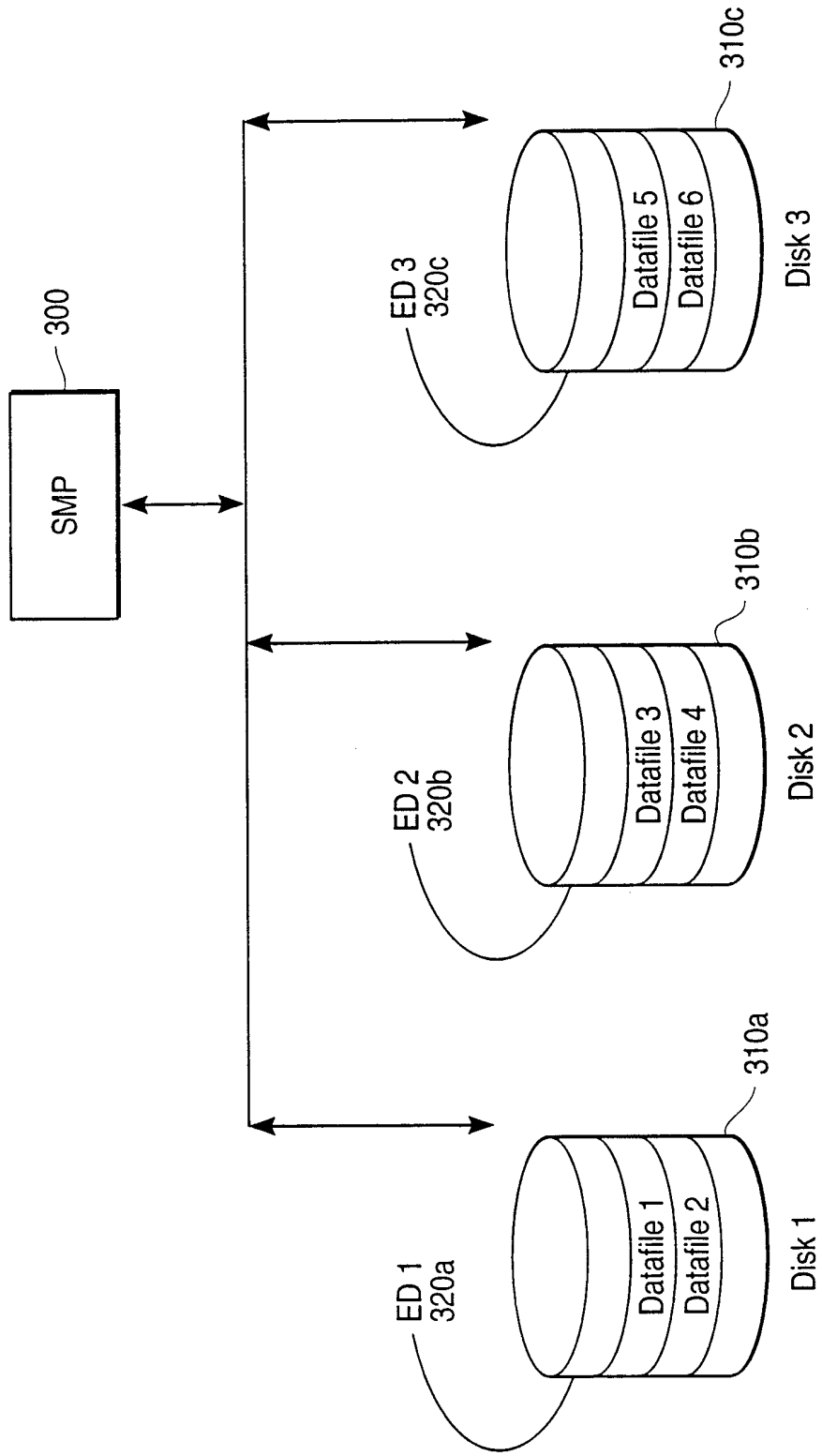

300
SMP
ED 1
320a
ED 2
320b
ED 3
320c
Datafile 1
Datafile 2
Datafile 3
Datafile 4
Datafile 5
Datafile 6
Disk 1
Disk 2
Disk 3
310a
310b
310c

FIG. 6

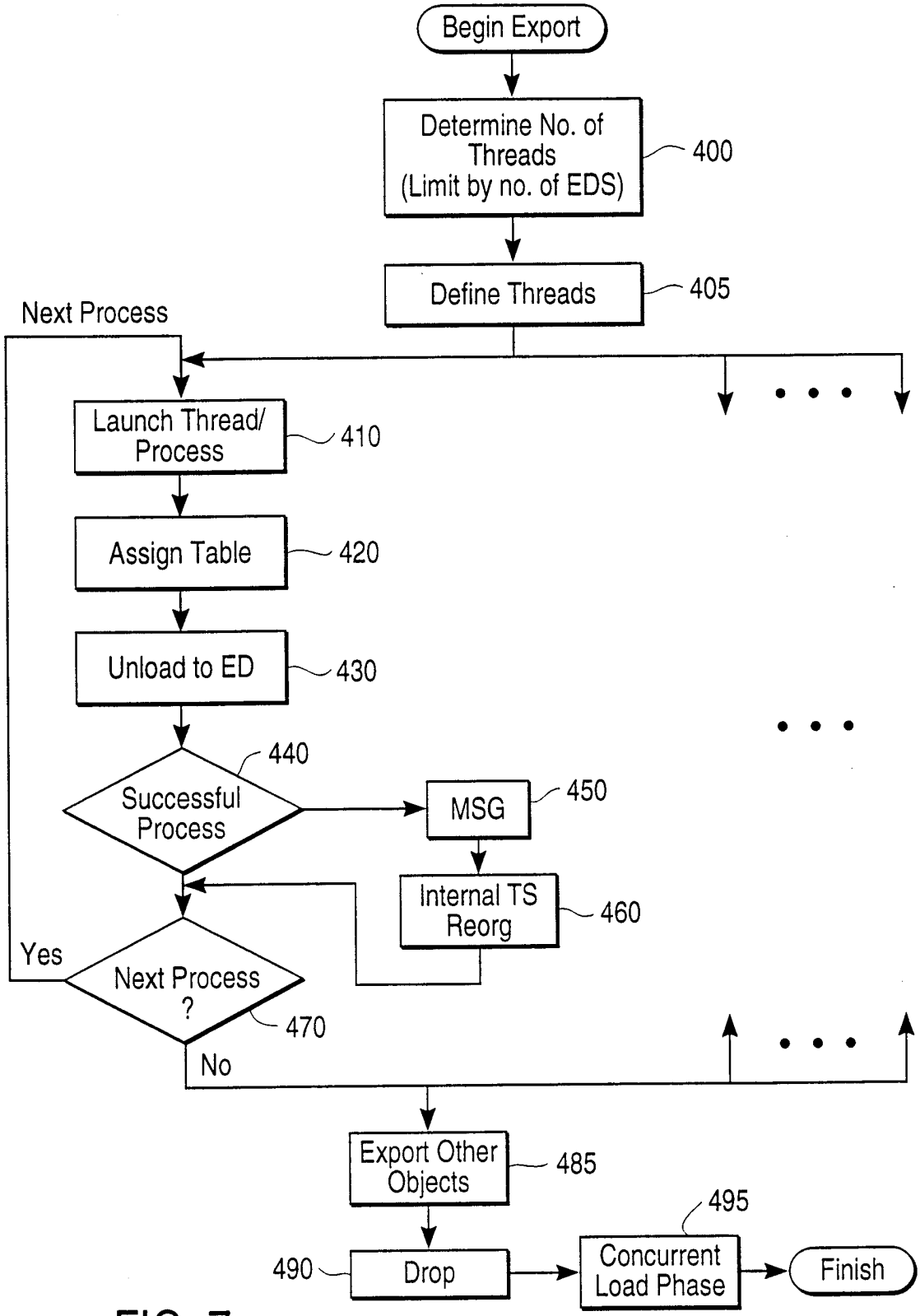

Begin Export
Determine No. of Threads (Limit by no. of EDS)
400
Define Threads
405
Next Process
Launch Thread/ Process
410
Assign Table
420
Unload to ED
430
440
Successful Process
MSG
450
Internal TS Reorg
460
Yes
Next Process ?
470
No
Export Other Objects
485
495
490
Drop
Concurrent Load Phase
Finish

FIG. 7

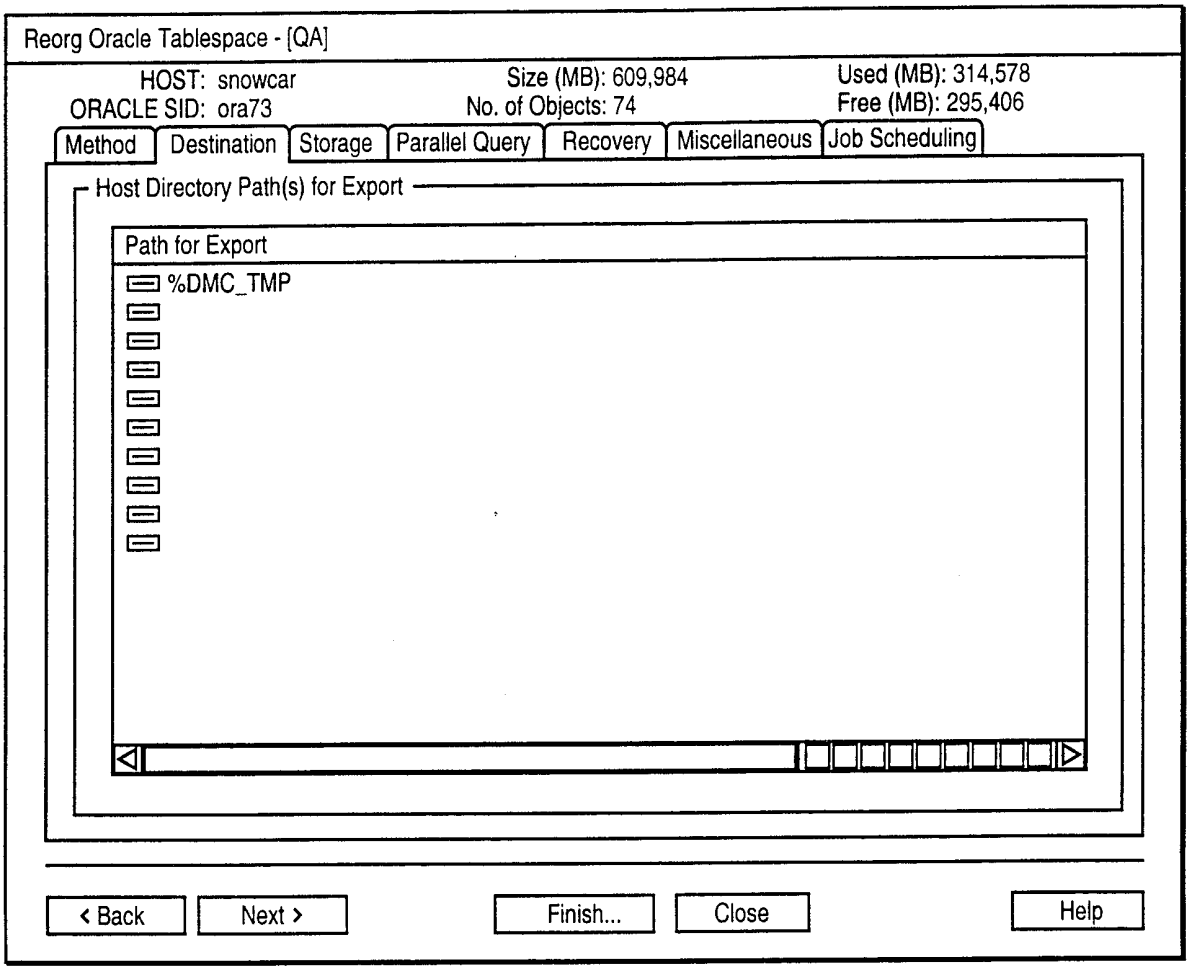

Reorg Oracle Tablespace - [QA]
HOST: snowcar
ORACLE SID: ora73
Size (MB): 609,984
No. of Objects: 74
Used (MB): 314,578
Free (MB): 295,406
Method
Destination
Storage
Parallel Query
Recovery
Miscellaneous
Job Scheduling
Host Directory Path(s) for Export
Path for Export
%DMC_TMP
< Back
Next >
Finish...
Close
Help

FIG. 8

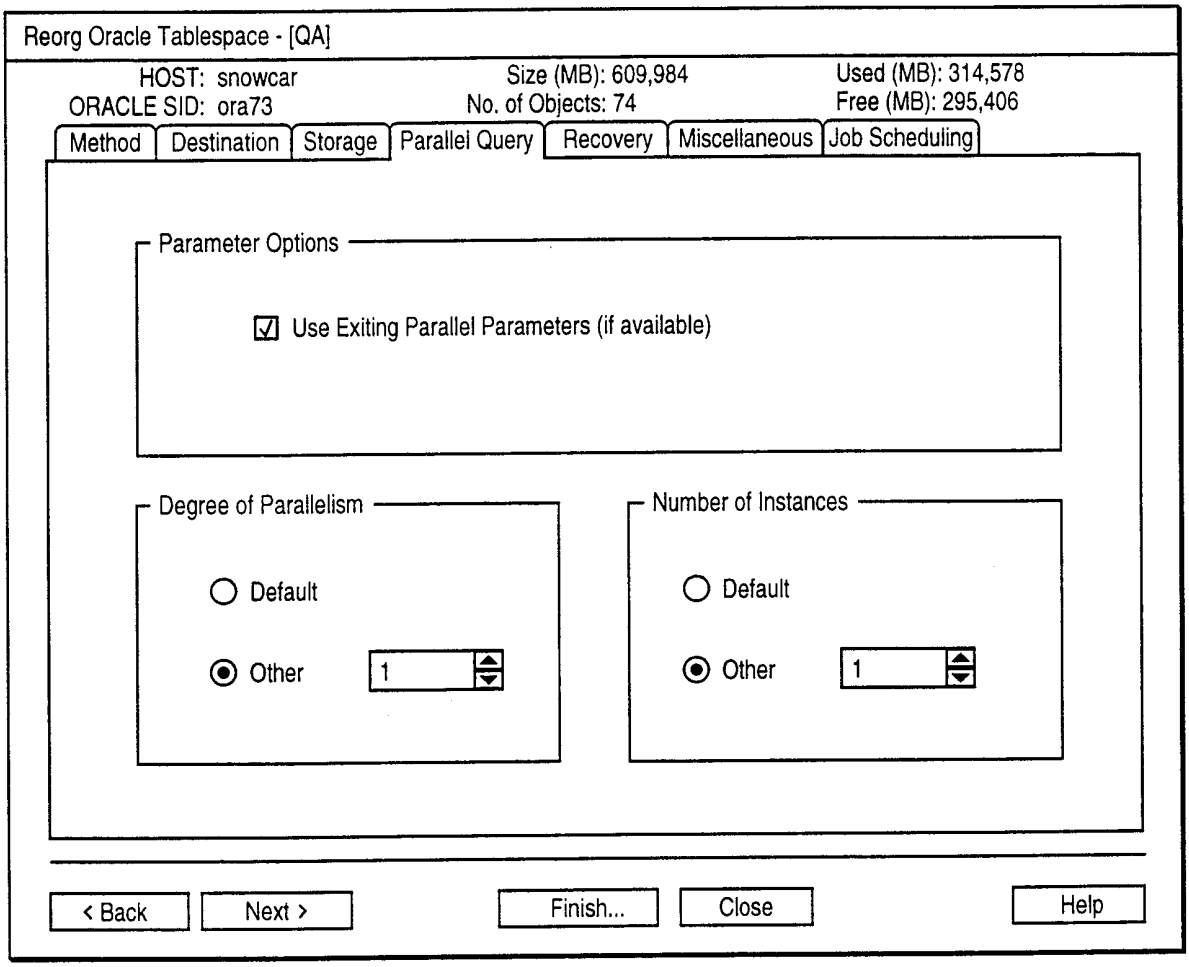

Reorg Oracle Tablespace - [QA]
HOST: snowcar
ORACLE SID: ora73
Size (MB): 609,984
No. of Objects: 74
Used (MB): 314,578
Free (MB): 295,406
Method
Destination
Storage
Parallel Query
Recovery
Miscellaneous
Job Scheduling
Parameter Options
Use Exiting Parallel Parameters (if available)
Degree of Parallelism
Default
Other
1
Number of Instances
Default
Other
1
< Back
Next >
Finish...
Close
Help

FIG. 9

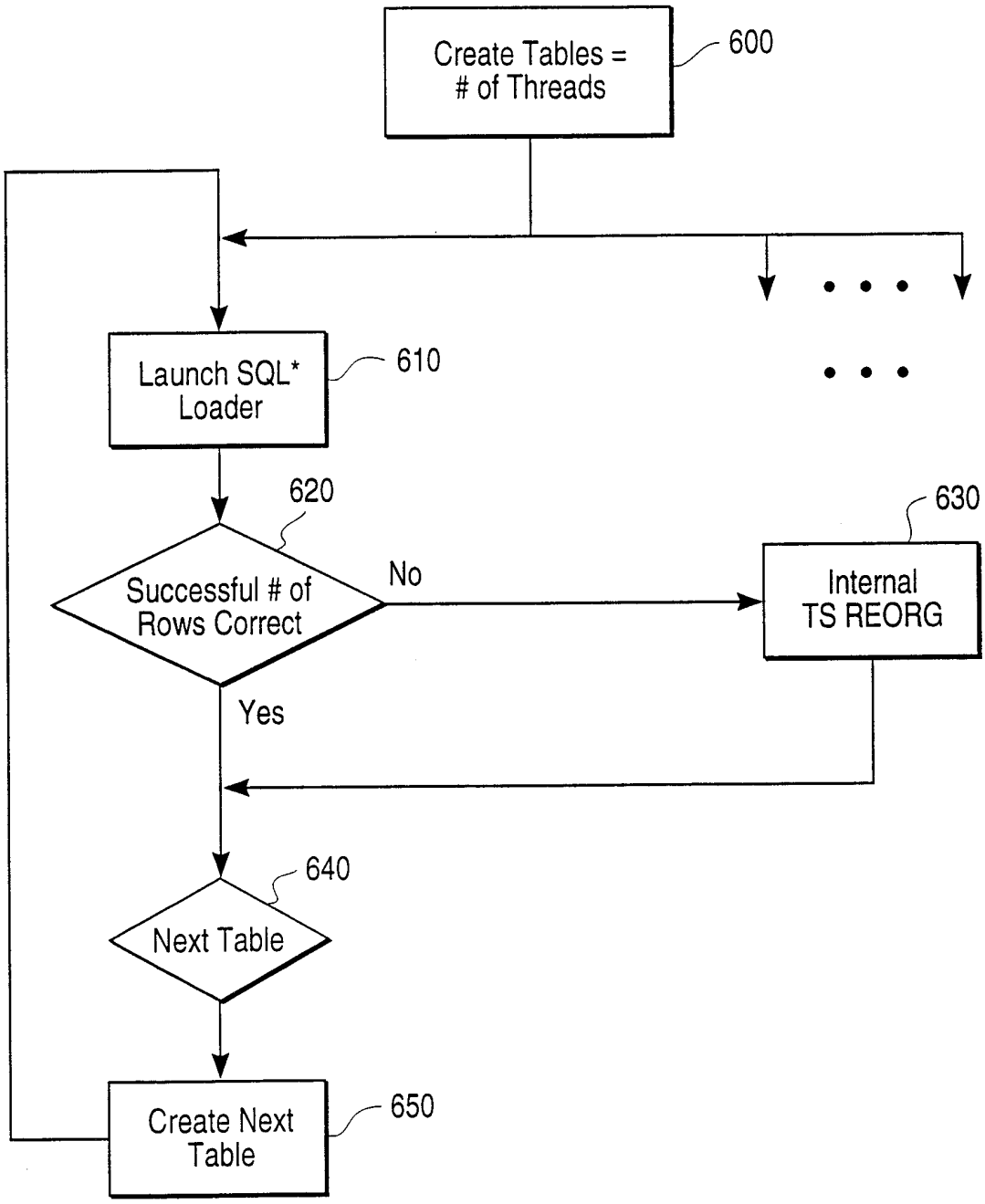

Create Tables = # of Threads
600
Launch SQL* Loader
610
620
Successful # of Rows Correct
No
630
Internal TS REORG
Yes
640
Next Table
Create Next Table
650

FIG. 10

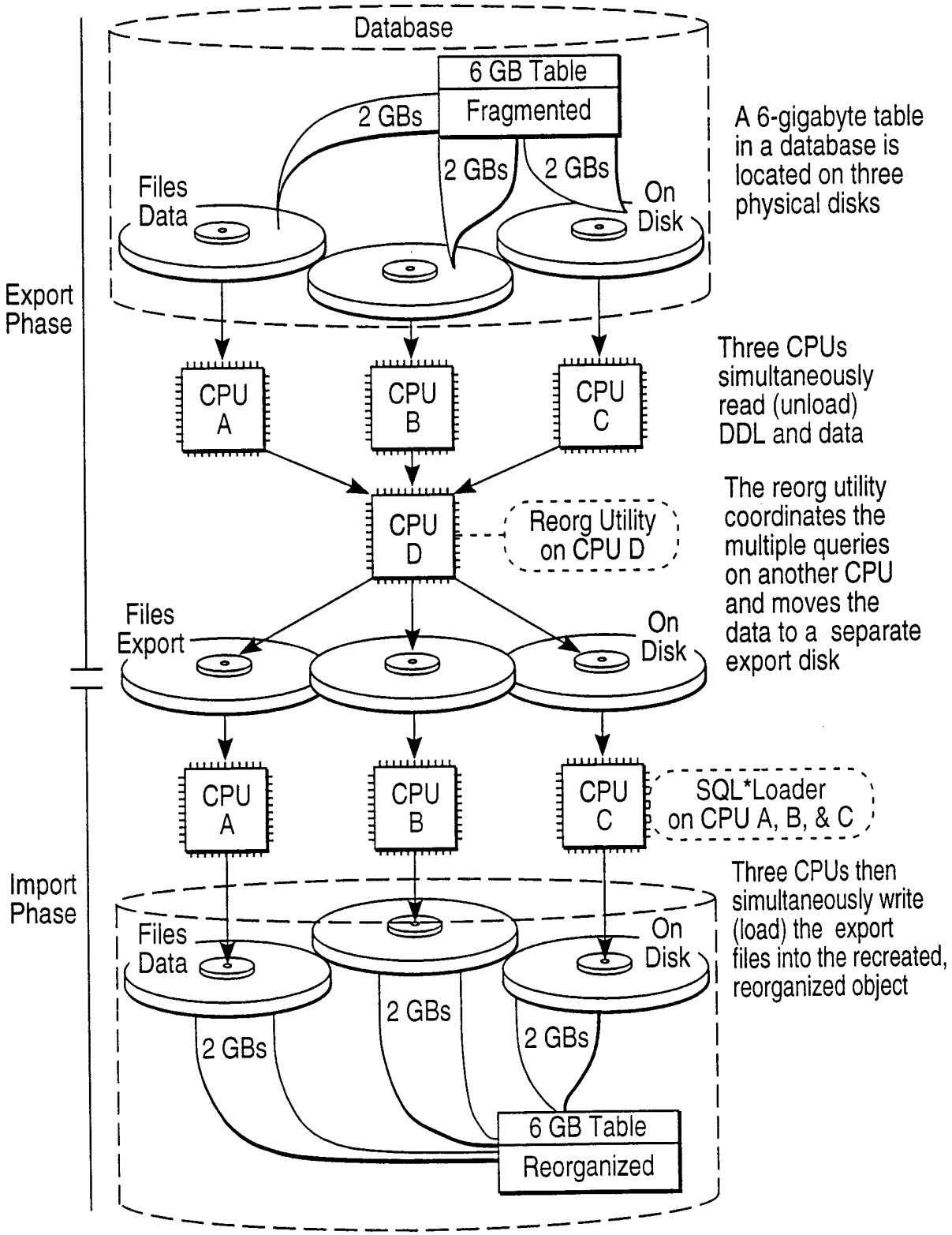

Database
6 GB Table
Fragmented
2 GBs
2 GBs
2 GBs
Files
Data
On
Disk
A 6-gigabyte table in a database is located on three physical disks
Export Phase
CPU A
CPU B
CPU C
Three CPUs simultaneously read (unload) DDL and data
CPU D
Reorg Utility on CPU D
The reorg utility coordinates the multiple queries on another CPU and moves the data to a separate export disk
Files
Export
On
Disk
CPU A
CPU B
CPU C
SQL*Loader on CPU A, B, & C
Import Phase
Three CPUs then simultaneously write (load) the export files into the recreated, reorganized object
Files
Data
On
Disk
2 GBs
2 GBs
2 GBs
6 GB Table
Reorganized

FIG. 11

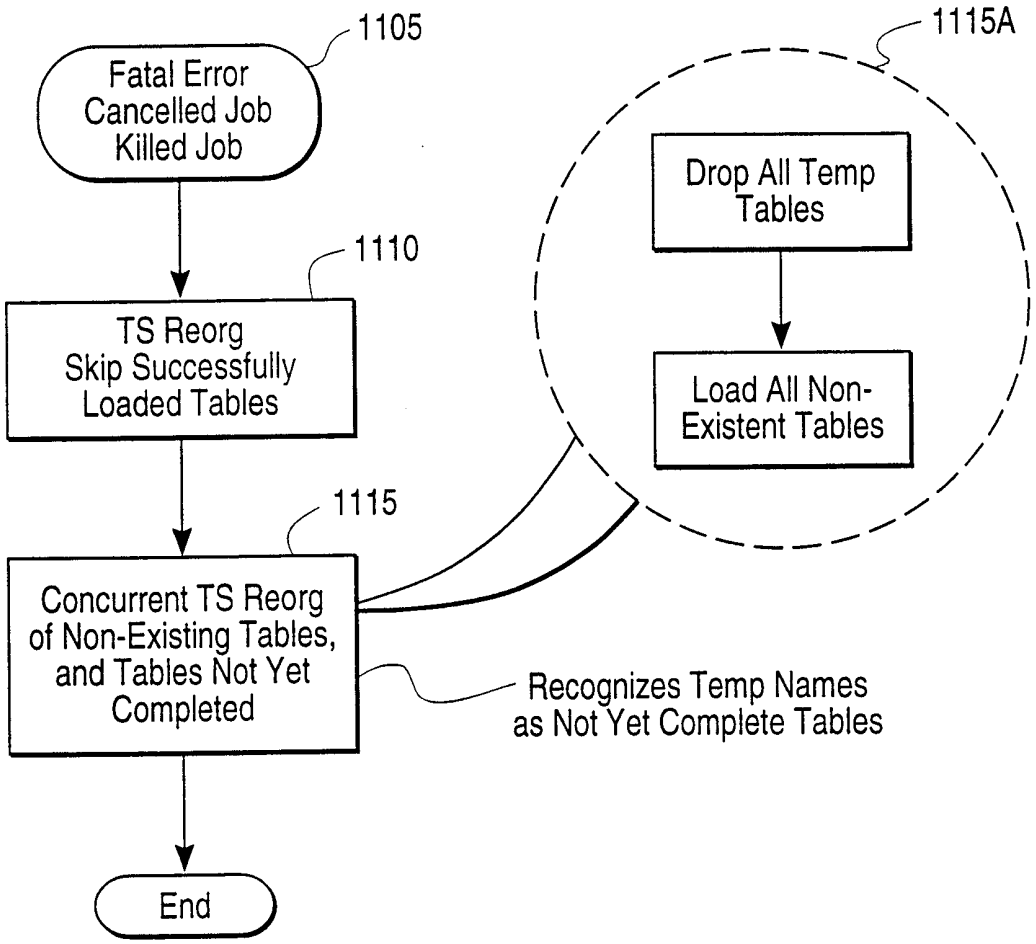

1105
Fatal Error
Cancelled Job
Killed Job
1110
TS Reorg
Skip Successfully
Loaded Tables
1115
Concurrent TS Reorg
of Non-Existing Tables,
and Tables Not Yet
Completed
End
1115A
Drop All Temp
Tables
Load All Non-
Existent Tables
Recognizes Temp Names
as Not Yet Complete Tables

FIG. 12

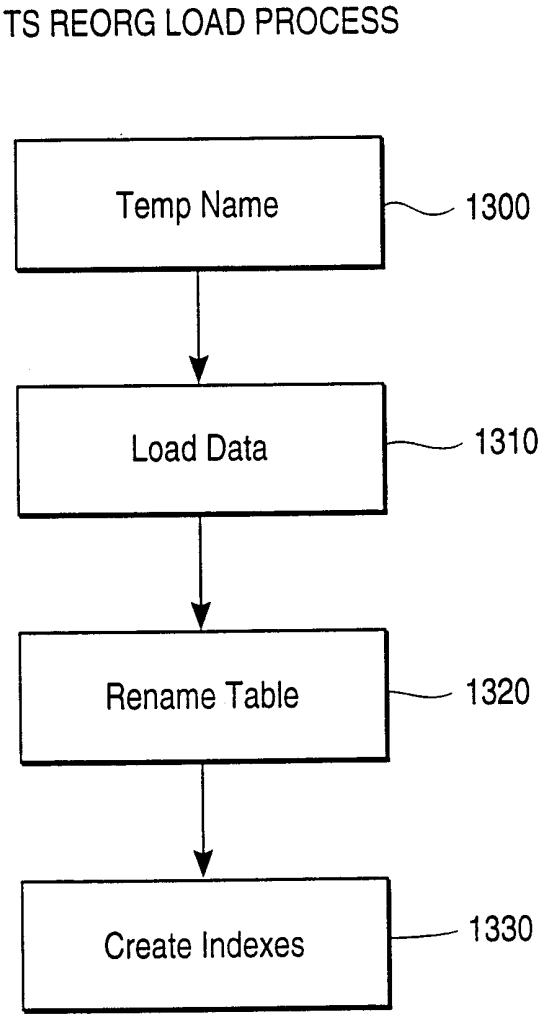

TS REORG LOAD PROCESS
Temp Name
1300
Load Data
1310
Rename Table
1320
Create Indexes
1330

FIG. 13

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US99/27835

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) : G06F 9/46, 12/00, 17/30
US CL : 707/200, 202, 205, 206

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 707/200, 202, 205, 206

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

EAST, WEST

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 5,754,771 A (EPPERSON ET AL) 19 May 1998, Figure 2.	1
Y	US 5,437,032 A (WOLF ET AL) 25 July 1995, column 1, lines 24-33	1-34
A,P	US 5,884,310 A (BRICHTA ET AL) 16 March 1999, Figure 1.	1-34
A,P	US 5,860,070 A (TOW ET AL) 12 January 1999.	1-34

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search
19 FEBRUARY 2000

Date of mailing of the international search report
20 MAR 2000

Name and mailing address of the ISA/US
Commissioner of Patents and Trademarks
Box PCT
Washington, D.C. 20231
Facsimile No. (703) 305-3230

Authorized officer
UYEN LE
Telephone No. (703) 305-4134

Form PCT/ISA/210 (second sheet)(July 1992)★