US 20080222111A1

(19) **United States**

(12) **Patent Application Publication**

**Hoang et al.**

(10) **Pub. No.: US 2008/0222111 A1**

(43) **Pub. Date: Sep. 11, 2008**

(54) **DATABASE SYSTEM WITH DYNAMIC DATABASE CACHING**

(75) Inventors: **Chi Kim Hoang**, Palo Alto, CA (US); **Chih-Ping Wang**, Palo Alto, CA (US); **John Ernest Miller**, Los Altos, CA (US); **Marie-Anne Neimat**, Redwood Shores, CA (US); **Susan Sokeng Cheung**, Redwood Shores, CA (US)

Correspondence Address:
**Stolowitz Ford Cowger LLP**
**621 SW Morrison St, Suite 600**
**Portland, OR 97205 (US)**

(73) Assignee: **Oracle International Corporation**, Redwood Shores, CA (US)

(21) Appl. No.: **12/030,113**

(22) Filed: **Feb. 12, 2008**

**Related U.S. Application Data**

(60) Provisional application No. 61/026,090, filed on Feb. 4, 2008, provisional application No. 60/905,751, filed on Mar. 7, 2007.

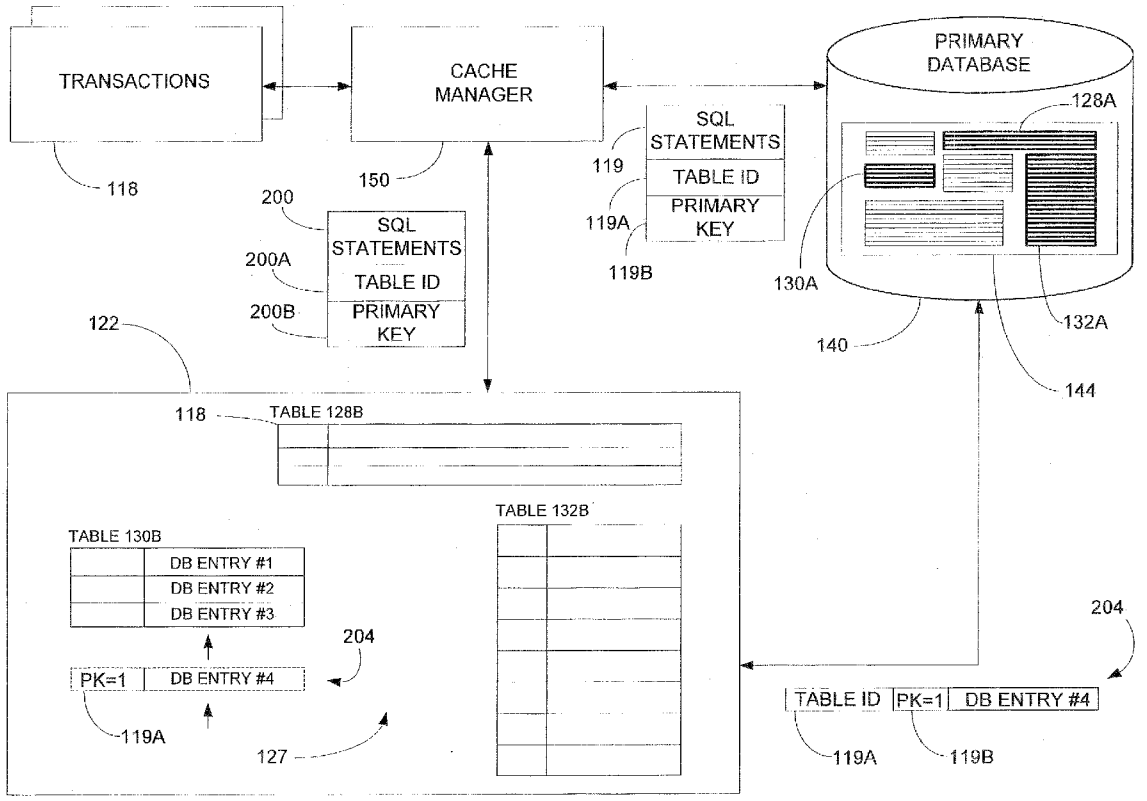
**Publication Classification**

(51) **Int. Cl.**
***G06F 7/06*** (2006.01)

(52) **U.S. Cl.** ..... **707/3**; 707/E17.014

(57) **ABSTRACT**

A fully transactional mid-tier database system services database transactions. A cache manager dynamically loads database entries from a fully transactional backend-tier database system into the mid-tier database system according to the received database transactions. Time based aging or usage based aging can be assigned to selected tables in the mid-tier database system. Database entries contained in the selected tables are then automatically removed according to assigned aging constraints.



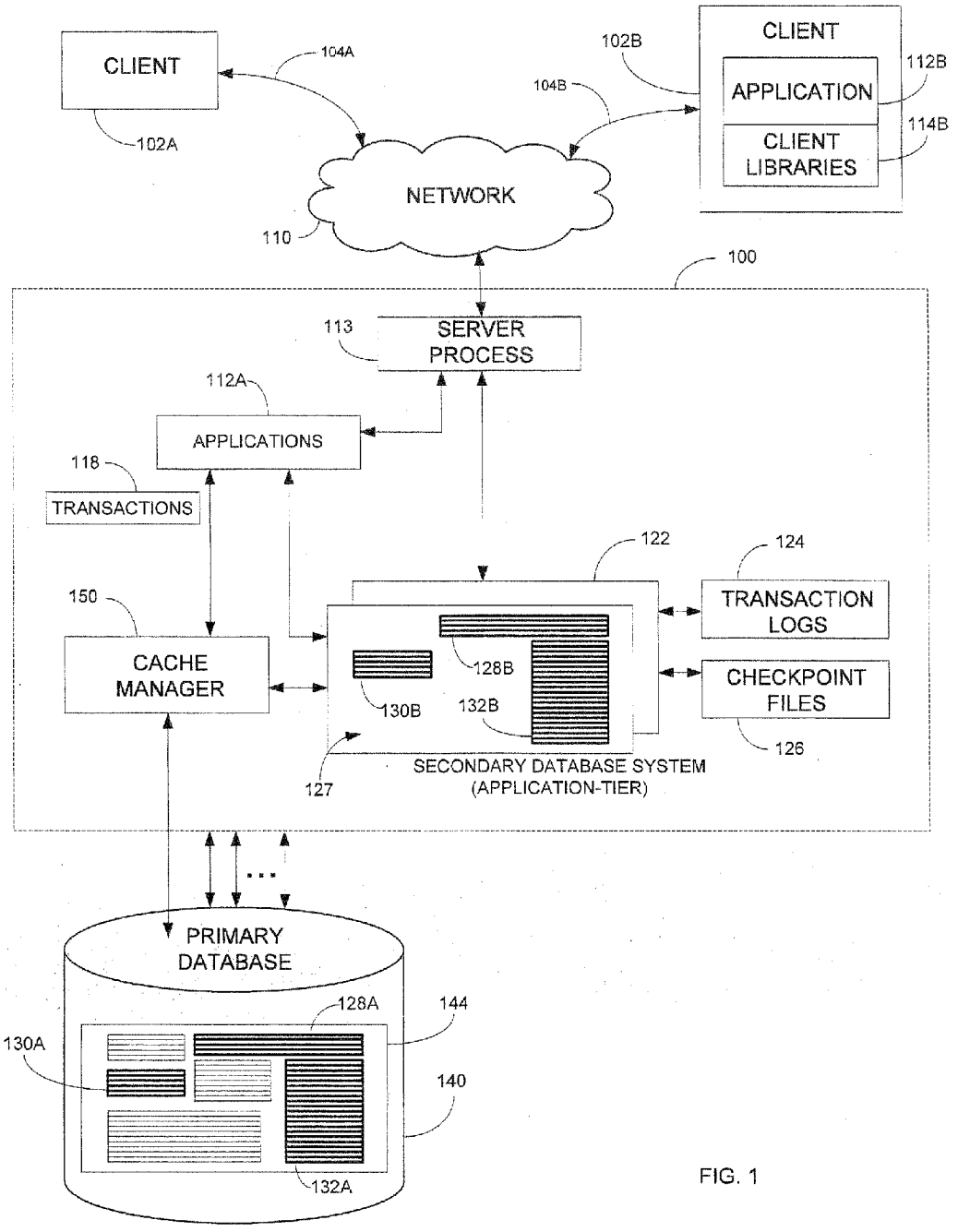

CLIENT
102A
104A
CLIENT
102B
APPLICATION
112B
104B
CLIENT LIBRARIES
114B
NETWORK
110
100
113
SERVER PROCESS
112A
APPLICATIONS
118
TRANSACTIONS
122
124
150
TRANSACTION LOGS
CACHE MANAGER
128B
130B
132B
CHECKPOINT FILES
126
127
SECONDARY DATABASE SYSTEM
(APPLICATION-TIER)
PRIMARY DATABASE
128A
144
130A
140
132A

FIG. 1

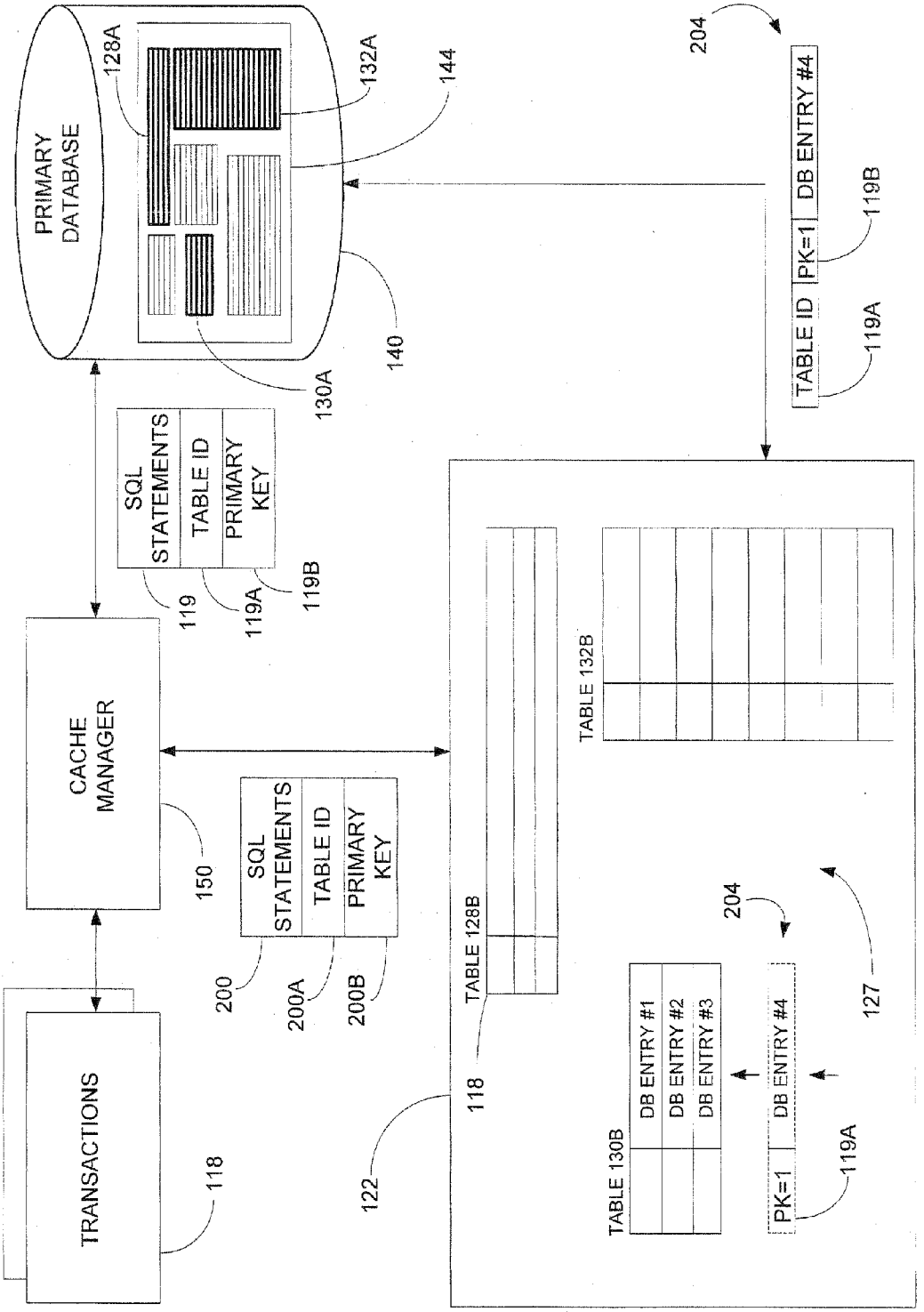

PRIMARY DATABASE
128A
132A
144
140
130A
204
DB ENTRY #4
PK=1
119B
TABLE ID
119A
SQL STATEMENTS
TABLE ID
PRIMARY KEY
119
119A
119B
CACHE MANAGER
150
TRANSACTIONS
118
200
200A
200B
TABLE 128B
TABLE 132B
122
118
204
DB ENTRY #1
DB ENTRY #2
DB ENTRY #3
TABLE 130B
DB ENTRY #4
PK=1
119A
127

FIG. 2

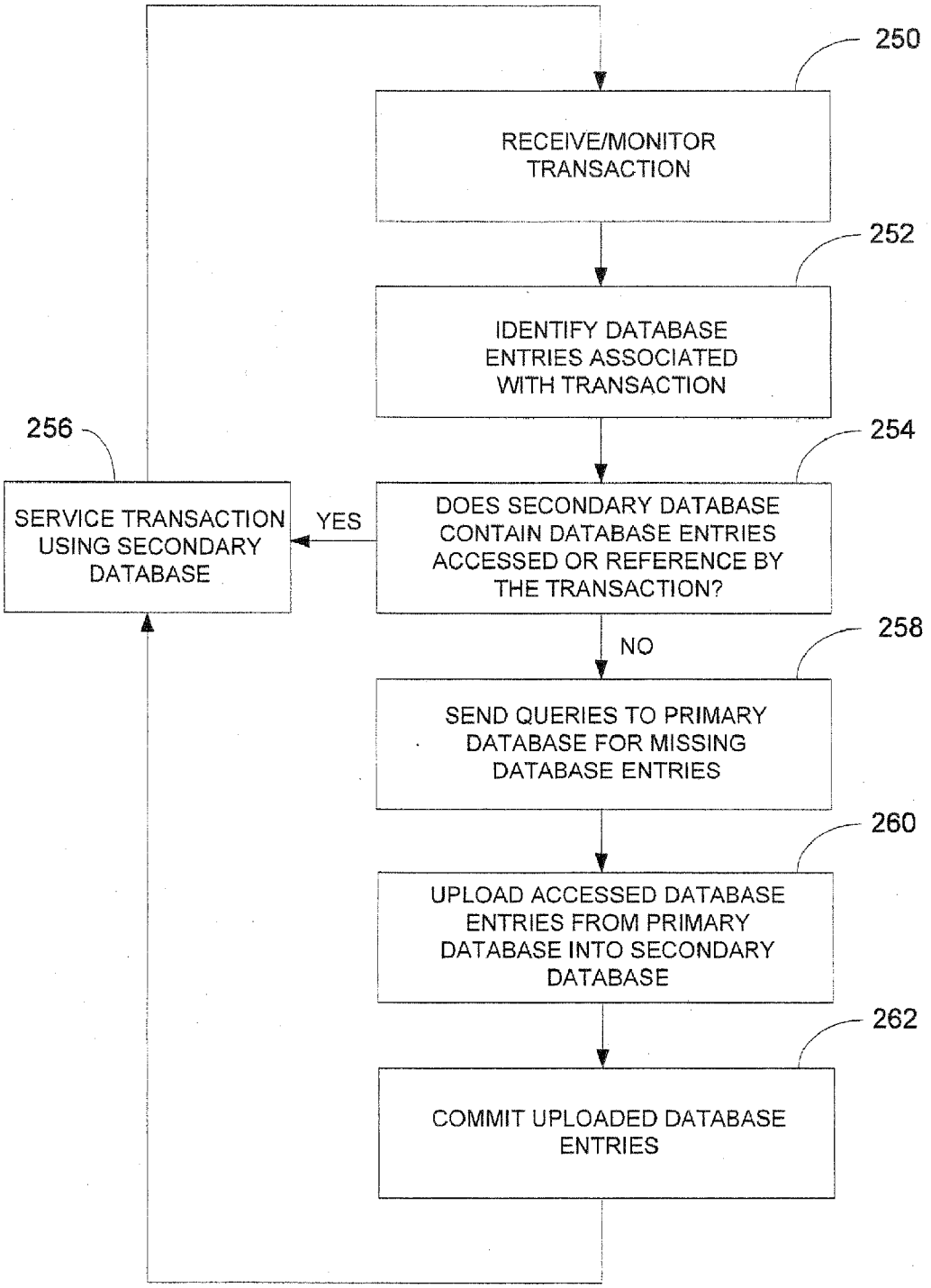

250
RECEIVE/MONITOR TRANSACTION
252
IDENTIFY DATABASE ENTRIES ASSOCIATED WITH TRANSACTION
254
DOES SECONDARY DATABASE CONTAIN DATABASE ENTRIES ACCESSED OR REFERENCE BY THE TRANSACTION?
YES
256
SERVICE TRANSACTION USING SECONDARY DATABASE
NO
258
SEND QUERIES TO PRIMARY DATABASE FOR MISSING DATABASE ENTRIES
260
UPLOAD ACCESSED DATABASE ENTRIES FROM PRIMARY DATABASE INTO SECONDARY DATABASE
262
COMMIT UPLOADED DATABASE ENTRIES

FIG. 3

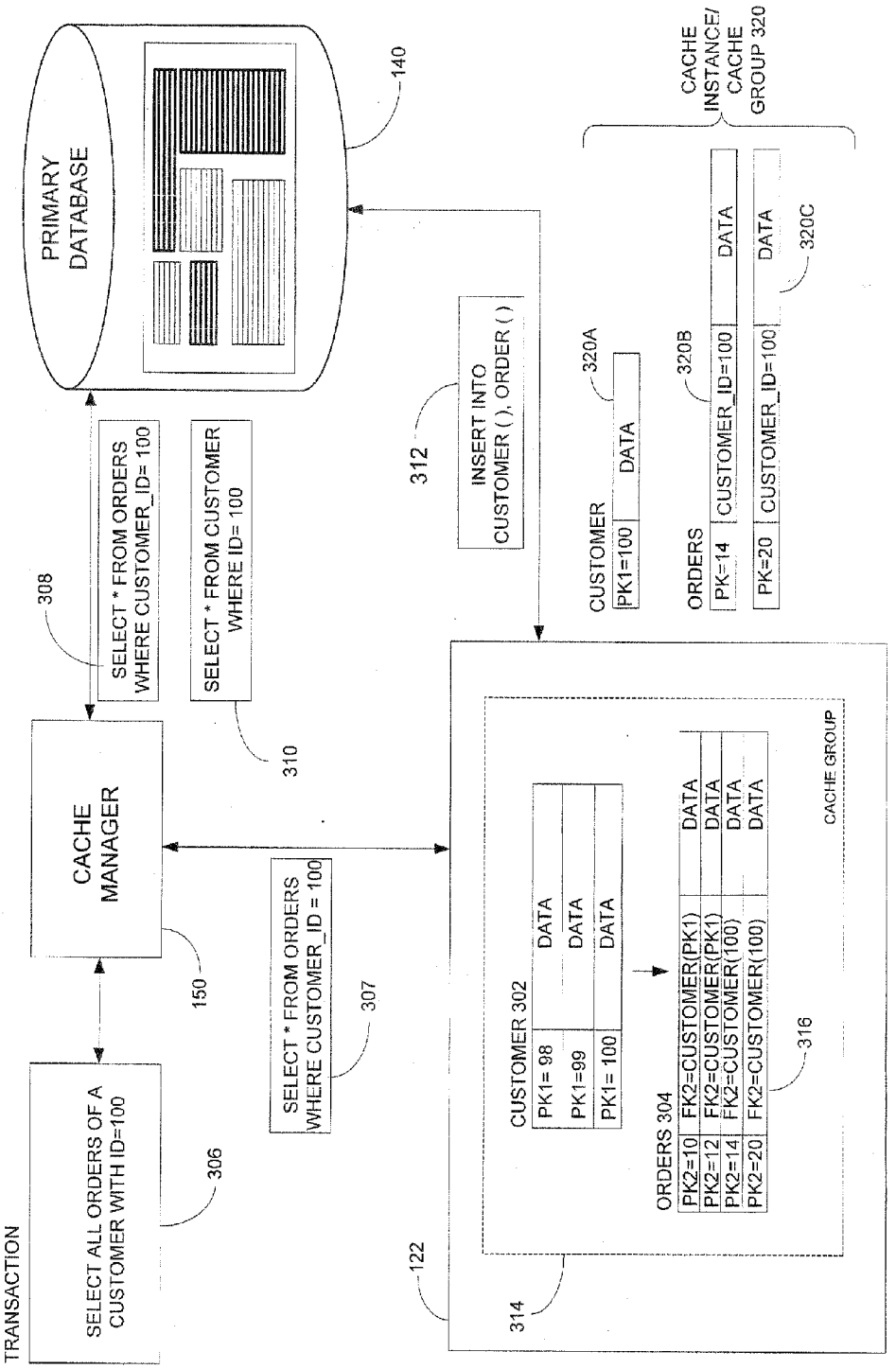

TRANSACTION
SELECT ALL ORDERS OF A CUSTOMER WITH ID=100
306
CACHE MANAGER
150
SELECT * FROM ORDERS WHERE CUSTOMER_ID= 100
308
SELECT * FROM CUSTOMER WHERE ID= 100
310
SELECT * FROM ORDERS WHERE CUSTOMER_ID = 100
307
PRIMARY DATABASE
140
INSERT INTO CUSTOMER ( ), ORDER ( )
312
CACHE INSTANCE/ CACHE GROUP 320
CUSTOMER
320A
PK1=100
DATA
ORDERS
320B
PK=14
CUSTOMER_ID=100
DATA
PK=20
CUSTOMER_ID=100
DATA
320C
CACHE GROUP
CUSTOMER 302
PK1= 98
PK1=99
PK1= 100
DATA
ORDERS 304
PK2=10 FK2=CUSTOMER(PK1) DATA
PK2=12 FK2=CUSTOMER(PK1) DATA
PK2=14 FK2=CUSTOMER(100) DATA
PK2=20 FK2=CUSTOMER(100) DATA
316
122
314

FIG. 4

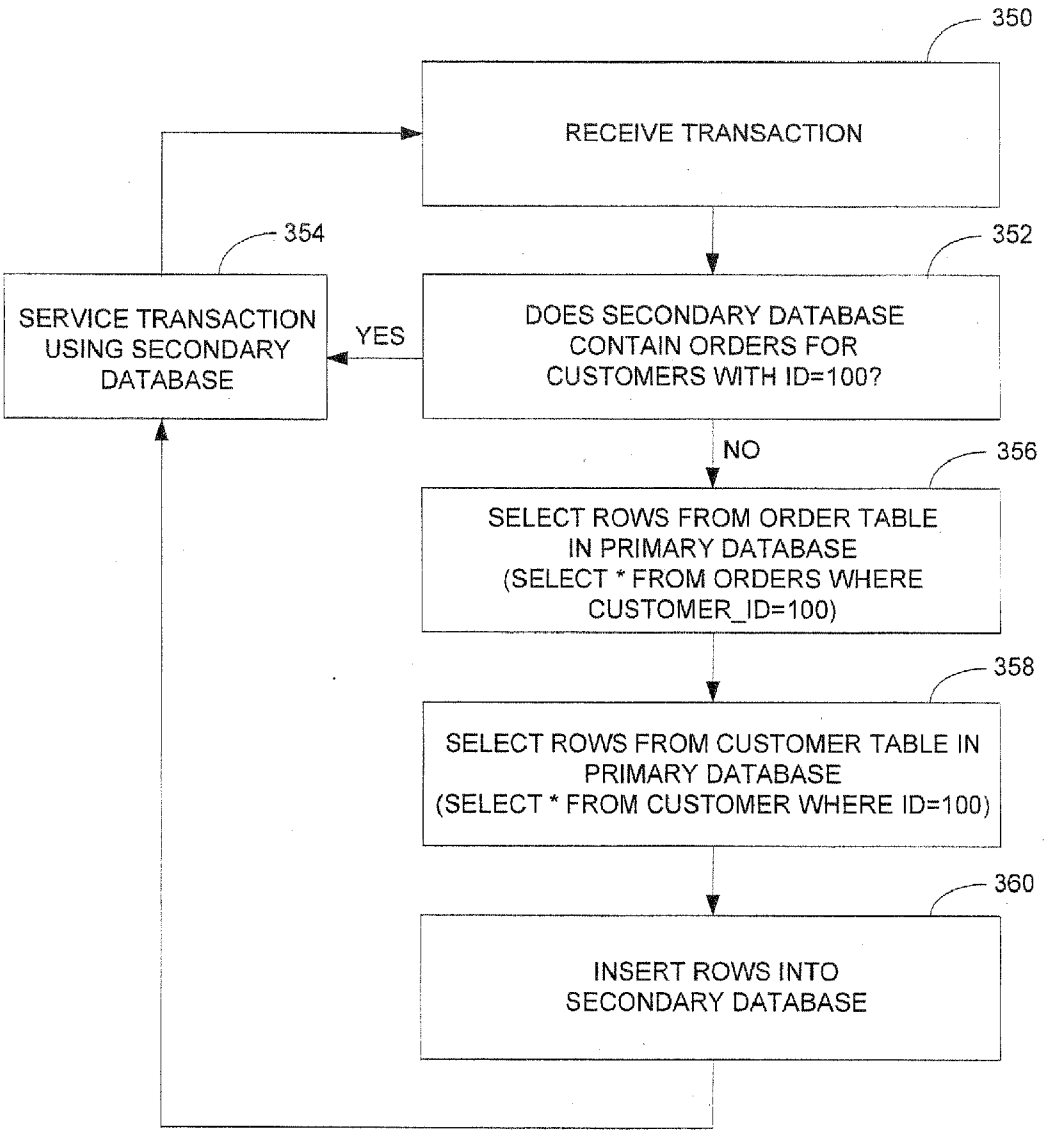

350
RECEIVE TRANSACTION
352
DOES SECONDARY DATABASE CONTAIN ORDERS FOR CUSTOMERS WITH ID=100?
YES
354
SERVICE TRANSACTION USING SECONDARY DATABASE
NO
356
SELECT ROWS FROM ORDER TABLE IN PRIMARY DATABASE (SELECT * FROM ORDERS WHERE CUSTOMER_ID=100)
358
SELECT ROWS FROM CUSTOMER TABLE IN PRIMARY DATABASE (SELECT * FROM CUSTOMER WHERE ID=100)
360
INSERT ROWS INTO SECONDARY DATABASE

FIG. 5

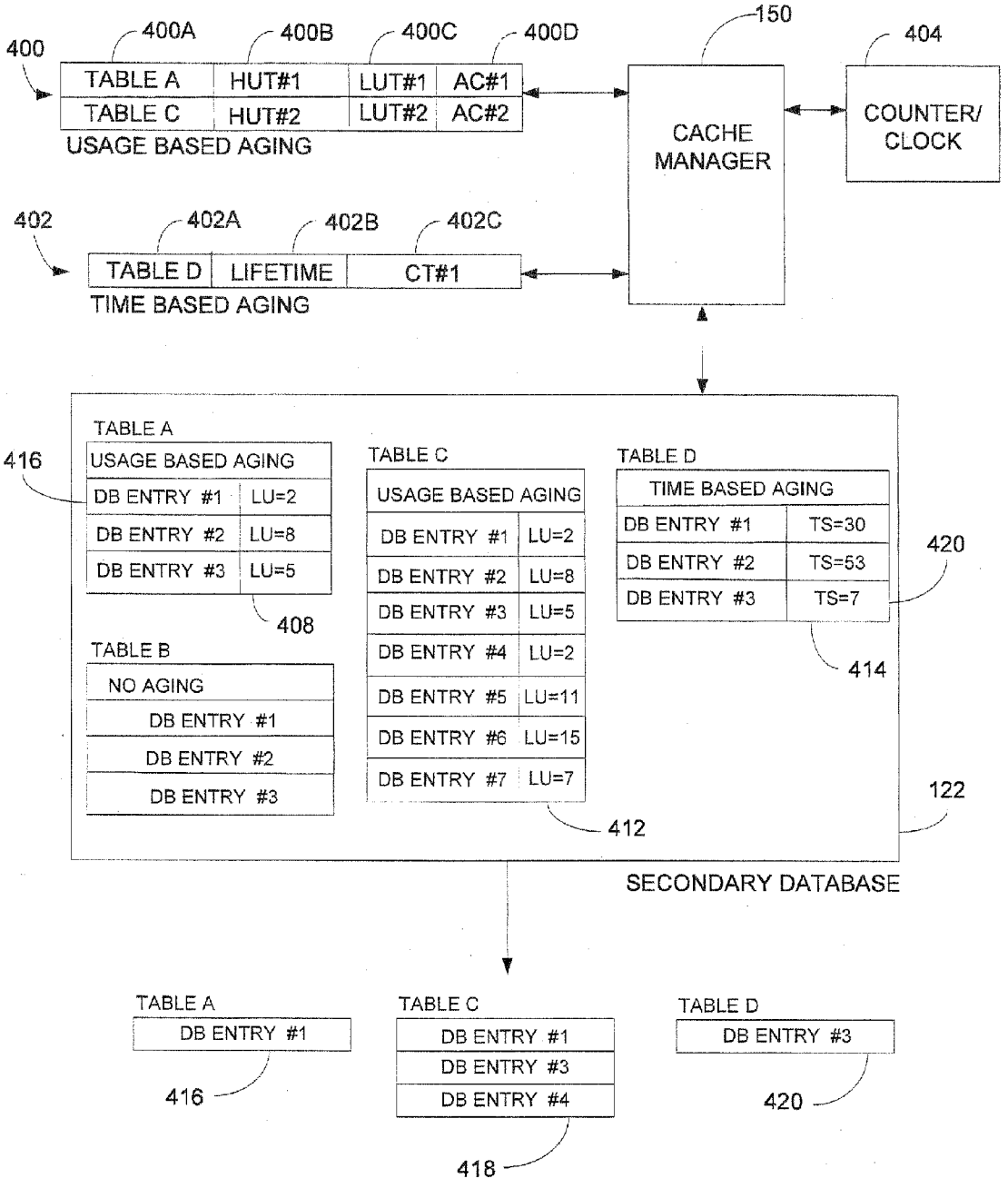

400
400A
400B
400C
400D
TABLE A
HUT#1
LUT#1
AC#1
TABLE C
HUT#2
LUT#2
AC#2
USAGE BASED AGING
150
CACHE MANAGER
404
COUNTER/ CLOCK
402
402A
402B
402C
TABLE D
LIFETIME
CT#1
TIME BASED AGING
416
TABLE A
USAGE BASED AGING
DB ENTRY #1
LU=2
DB ENTRY #2
LU=8
DB ENTRY #3
LU=5
408
TABLE B
NO AGING
DB ENTRY #1
DB ENTRY #2
DB ENTRY #3
TABLE C
USAGE BASED AGING
DB ENTRY #1
LU=2
DB ENTRY #2
LU=8
DB ENTRY #3
LU=5
DB ENTRY #4
LU=2
DB ENTRY #5
LU=11
DB ENTRY #6
LU=15
DB ENTRY #7
LU=7
412
TABLE D
TIME BASED AGING
DB ENTRY #1
TS=30
DB ENTRY #2
TS=53
DB ENTRY #3
TS=7
420
414
122
SECONDARY DATABASE
TABLE A
DB ENTRY #1
416
TABLE C
DB ENTRY #1
DB ENTRY #3
DB ENTRY #4
418
TABLE D
DB ENTRY #3
420

FIG. 6

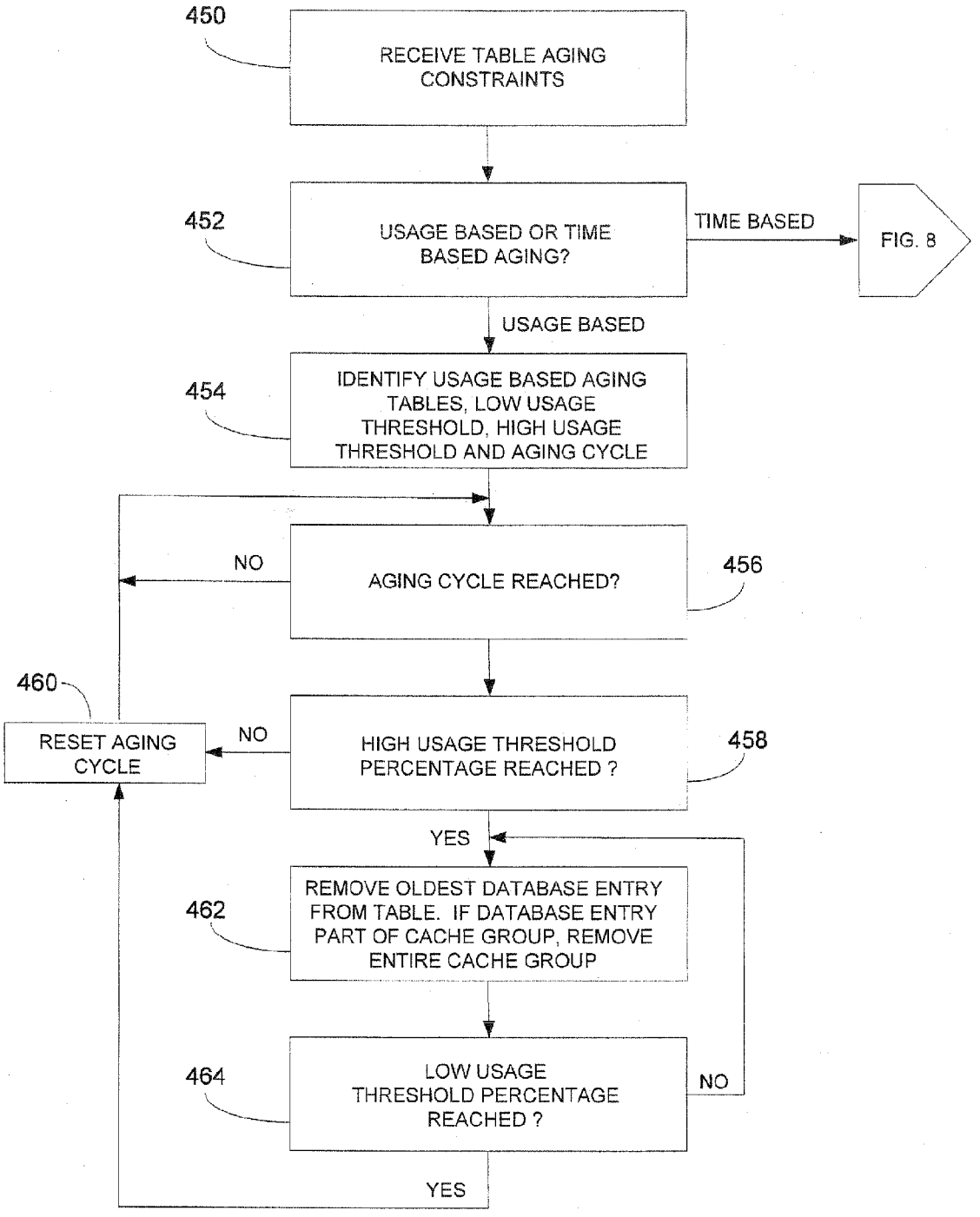

450
RECEIVE TABLE AGING CONSTRAINTS
452
USAGE BASED OR TIME BASED AGING?
TIME BASED
FIG. 8
USAGE BASED
454
IDENTIFY USAGE BASED AGING TABLES, LOW USAGE THRESHOLD, HIGH USAGE THRESHOLD AND AGING CYCLE
NO
AGING CYCLE REACHED?
456
460
RESET AGING CYCLE
NO
HIGH USAGE THRESHOLD PERCENTAGE REACHED ?
458
YES
462
REMOVE OLDEST DATABASE ENTRY FROM TABLE. IF DATABASE ENTRY PART OF CACHE GROUP, REMOVE ENTIRE CACHE GROUP
464
LOW USAGE THRESHOLD PERCENTAGE REACHED ?
NO
YES

FIG. 7

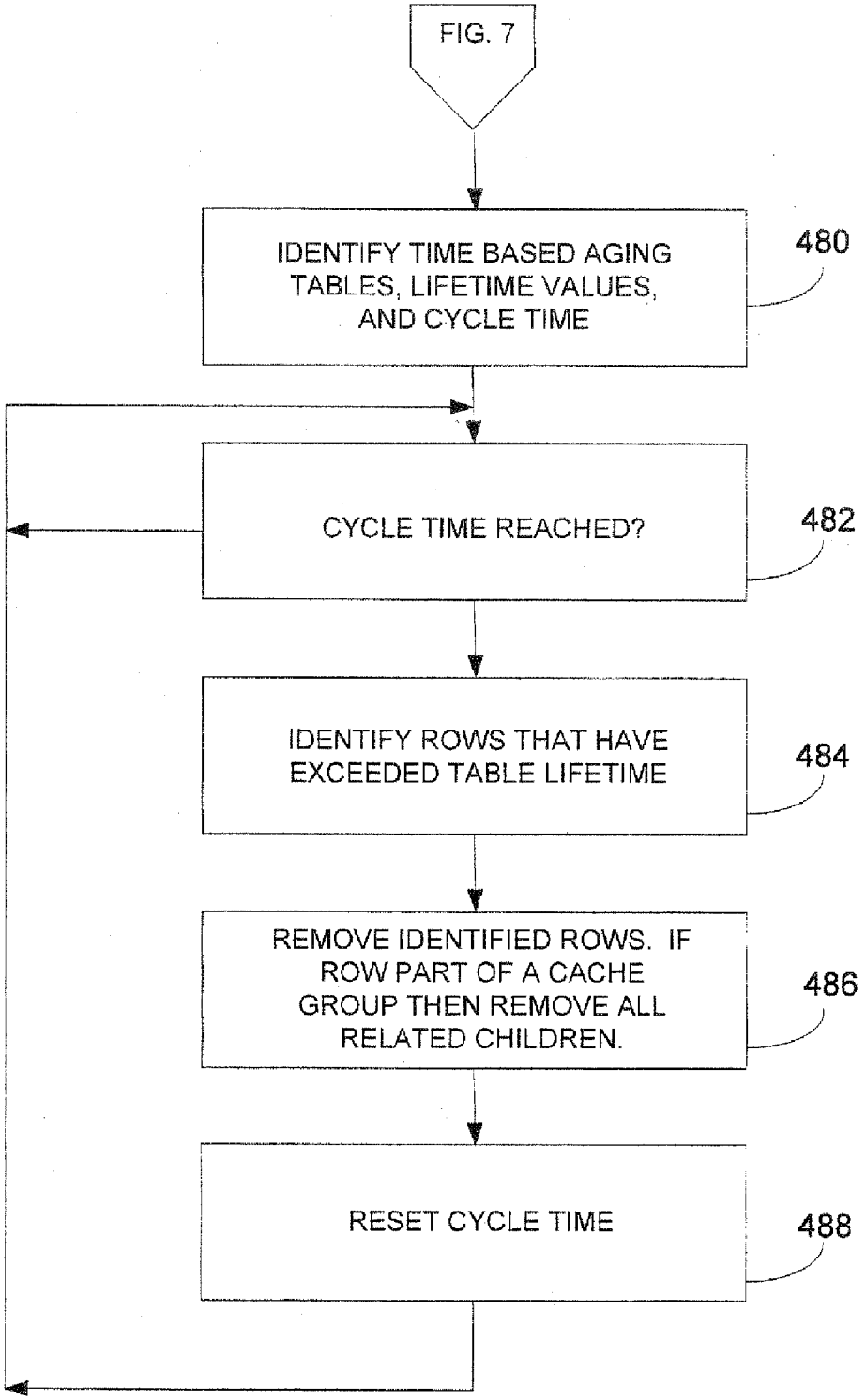


FIG. 7
IDENTIFY TIME BASED AGING TABLES, LIFETIME VALUES, AND CYCLE TIME
480
CYCLE TIME REACHED?
482
IDENTIFY ROWS THAT HAVE EXCEEDED TABLE LIFETIME
484
REMOVE IDENTIFIED ROWS. IF ROW PART OF A CACHE GROUP THEN REMOVE ALL RELATED CHILDREN.
486
RESET CYCLE TIME
488

FIG. 8

# DATABASE SYSTEM WITH DYNAMIC DATABASE CACHING

## CROSS REFERENCE TO RELATED APPLICATIONS

**[0001]** This application claims priority to provisional application Ser. No. 60/905,751 filed on Mar. 7, 2007, entitled MAIN-MEMORY DATABASES and also claims priority to provisional application Ser. No. 61/026,090 filed on Feb. 4, 2008, entitled DATABASE SYSTEM WITH DYNAMIC DATABASE CACHING AND DATABASE SYSTEM WITH ACTIVE AND STANDBY NODES and are both incorporated by reference in their entirety.

**[0002]** This application is also related to the following application filed simultaneously herewith and is incorporated by reference in its entirety.

**[0003]** U.S. patent application Ser. No. 12/030,094 entitled: DATABASE SYSTEM WITH ACTIVE AND STANDBY NODES filed on Feb. 12, 2008.

## TECHNICAL FIELD

**[0004]** The present disclosure relates generally to database systems.

## BACKGROUND

**[0005]** A disk-based Relational Database Management System (RDBMS) uses disk storage to store and access large amounts of data. Much of the work performed by a conventional, disk-optimized RDBMS assumes that data primarily resides on disk. Optimization algorithms, buffer pool management, and indexed retrieval techniques are designed based on this fundamental assumption. One problem with disk storage is that access to the data is relatively slow.

**[0006]** Even when an RDBMS is configured to hold data in main memory, performance is still hobbled by assumptions of disk-based data residency. These assumptions cannot be easily reversed due to hard-coded processing logic, indexing schemes, and data access mechanisms.

**[0007]** In-memory resident relational database systems are deployed in the application-tier and operate in physical memory using standard Sequential Query Language (SQL) interfaces. By managing data in memory and optimizing data structures and access algorithms, in-memory database systems can provide improved responsiveness and throughput compared even to fully cached disk-based RDBMS. For example, the in-memory database can be designed with the knowledge that data resides in main memory and can take more direct routes to data, reducing the length of the code path and simplifying algorithms and structure.

**[0008]** When the assumption of disk-residency is removed, complexity is dramatically reduced. The number of machine instructions drop, buffer pool management disappears, extra data copies are not needed, and index pages shrink. The database design becomes simple and more compact, and data requests are executed faster. However, in-memory database systems currently can only operate on a relatively small static portion of the data contained in a disk-based database system.

## BRIEF DESCRIPTION OF THE DRAWINGS

**[0009]** FIG. **1** is a schematic block diagram showing a database system that provides dynamic database caching.

**[0010]** FIG. **2** is a schematic block diagram showing in more detail how a cache manager caches database entries in a secondary/application-tier database system.

**[0011]** FIG. **3** is a flow diagram explaining operations performed by the cache manager in FIG. **2**.

**[0012]** FIG. **4** is a schematic block diagram showing how the cache manager operates with cache groups.

**[0013]** FIG. **5** is a flow diagram describing in more detail how dynamic database caching operates with cache groups.

**[0014]** FIG. **6** is a schematic block diagram showing how dynamic database caching selectively ages out database entries.

**[0015]** FIG. **7** is a flow diagram describing usage based aging in more detail.

**[0016]** FIG. **8** is a flow diagram describing time based aging in more detail.

## INTRODUCTION

**[0017]** A fully transactional mid-tier database system services database transactions. A cache manager dynamically loads database entries from a fully transactional backend-tier database system into the mid-tier database system according to the received database transactions. Time based aging or usage based aging can be assigned to selected tables in the mid-tier database system. Database entries contained in the selected tables are then automatically removed according to assigned aging constraints.

## DETAILED DESCRIPTION

**[0018]** FIG. **1** is a schematic representation of a multi-tiered database system. A primary database system **140** can be any conventional fully-relational database system, such as a disk-based Relational Database Management System (RDBMS). The primary database system **140** typically uses disk storage to store and access large amounts of data that in one example includes multiple different tables **144**. The primary database system **140** is alternatively referred to as a backend database system or a backend-tier database system.

**[0019]** A secondary database system **122** typically operates on a server **100** that is remote from primary database system **140** and includes a storage manager that stores and manages different tables **127** that contain different database entries. The secondary database **122** in one example is an in-memory fully-relational database that is deployed in an application tier and operates in physical memory of the server **100**. The secondary database system **122** is alternatively referred to as an application-tier database system or an in-memory database system.

**[0020]** Applications **112**A and **112**B are initiated by clients **102**A and **102**B, respectively, via a local or wide area network **110**. The network **110** is alternatively referred to as the Internet. The applications **112** can be any software program that accesses or references database entries in a database. For example, the applications **112** could be software programs used for booking airline reservations, ordering products over the Internet, managing financial transactions for banks or investment institutions, or tracking telephone call usage. Of course these are just a few examples of the essentially limitless number of data management applications that may be used with the database systems shown in FIG. **1**.

**[0021]** Connections from the clients **102** can either be direct connections or client/server connections. Direct connections refer to Sequential Query Language (SQL) libraries

and routines that implement a direct driver. The application **112**A can create a direct driver connection when it runs on the same server **100** that operates the secondary database system **122**. In a direct driver connection, the direct driver directly loads the secondary database **122** into the application's heap space or a shared memory segment. The application **112**A then uses the direct driver to access a memory image of the secondary database **122**. Because no inter-process communication is required, a direct driver connection provides fast performance.

**[0022]** The client/server connection accommodates connections from the remote client **102**B to secondary database **100** over network **110**. Applications **112**B on the client **102**B issue calls to local client driver libraries **114**B that communicate with a server/child process **113** on the server **100** containing secondary database **122**. The server/child process **113**, in turn, issues native requests to the direct driver provided by the server libraries for accessing the secondary database **122**. If a client **102** and server **100** reside on separate nodes in a network, then communication is provided using sockets and Transmission Control Protocol/Internet Protocol (TCP/IP) communications.

**[0023]** The secondary database **122** maintains durability through a combination of transaction logs **124** and periodic refreshes of a disk-resident version of the secondary database **122**. The transaction logs **124** are written to disk asynchronously or synchronous with the completion of transactions **118** and are controlled by the applications **112** at the transaction level. The transaction logs **124** can be used to recover a transaction **118** if the application **112** or database **122** fails, undo transactions **118** that are rolled back, replicate changes to other databases, replicate changes in the secondary database **122** to the primary database **140**, or enable applications **112** to detect changes to database entries.

**[0024]** Checkpoint files **126** are used to keep a snap shot of the secondary database **122**. In the event of a system failure, the checkpoint files **126** are used to restore the secondary database **122** to a last transactionally consistent state. A checkpoint operation scans the secondary database **122** for blocks that have changed since the last checkpoint and updates the checkpoint files **126** with the changes and removes any transaction log files **124** that are no longer needed.

**[0025]** The applications **112** create and manage the tables **127** that may exist only in secondary database **122**. The applications **112**, through cache manager **150**, can also cache frequently used subsets of database entries from the primary database **140**. The tables **127** managed exclusively by the secondary database **122** and the tables **127** cached from primary database **140** may all coexist in the same secondary database **122**, and are all persistent and recoverable.

**[0026]** Queries and updates to the tables **127** are performed by the applications **112** through standard SQL. Applications **112** running on other different mid-tier servers may cache different or overlapping subsets of the data in primary database **140**.

**[0027]** The cache manager **150** can cache entire tables or table fragments from the primary database **140** to the secondary database **122** operating on server **100**. The table fragments are described through an extended SQL syntax and are cached into corresponding tables. For example, tables **128**A, **130**A, and **132**A from primary database **140** are cached into corresponding tables **128**B, **130**B, and **132**B in the secondary database **122**. The cached tables **128**B, **130**B, or **132**B may comprise the entire corresponding tables **128**A, **130**A, or **132**B from primary database **140** of may only include selected database entries from the primary database tables **128**A, **130**A, or **132**B. The database entries can be any record, tuple, column, row or other data item that typically exists in a fully transactional database system.

**[0028]** The secondary database **122** dynamically caches performance-critical subsets of the primary database **140**, enabling both reads and updates, and automatically manages data consistency between the cached secondary database **122** and the primary database **140**. The applications **112** read and update the cached tables **127** using standard SQL, and the cache manager **150** automatically propagates updates from the primary database **140** to the secondary database **122** and vice versa.

**[0029]** Thus, the cached secondary database **122** offers applications **112** the full generality and functionality of a fully-relational database, the transparent maintenance of cache consistency with the primary database **140**, and the real-time performance of an application-tier in-memory database system.

**[0030]** FIG. **2** shows the operations performed by the cache manager **150** in more detail. The cache manager **150** dynamically varies what subset of tables **127** are cached from the primary database system **140** into the secondary database system **122** according to the transactions **118** received from the applications **112** in FIG. **1**.

**[0031]** The cache manager **150** first determines what transactions **118** can be serviced by the secondary database **122**. For example, the cache manager **150** determines if the referenced tables **200**A and referenced primary keys **200**B in SQL statement **200** reside in secondary database **122**. If the referenced database entries reside in the secondary database **122**, the transaction **118** is serviced by the secondary database **122**.

**[0032]** When the database entries referenced by the transaction **118** do not reside in the secondary database **122**, the cache manager **150** may query the primary database **140** for the missing database entries. For example, table identifier **119**A and primary key identifier **119**B reference a database entry **204** in a table **130**B having a primary key value PK=1. Since the database entry **204** is not currently located in the secondary database **122**, the cache manager **150** queries the primary database **140**. The referenced database entry **204** in primary database **140** is then inserted into table **130**B in the secondary database **122**. The transaction **118** may then be serviced by the secondary database **122** using the uploaded database entry **204**.

**[0033]** FIG. **3** explains some of the operations performed by the cache manager **150** in more detail. Referring both to FIGS. **2** and **3**, in operation **250** the cache manager **150** receives or monitors the database transactions **118** directed to secondary database **122**. In operation **252**, the cache manager **150** may identify the database entries associated with the database transaction. For example, the cache manager **150** obtains the table identifiers and keys referenced by the transaction **118**. The cache manager **150** in operation **254** searches the secondary database **122** for the referenced database entries.

**[0034]** If the secondary database **122** contains the referenced database entries in operation **254**, the transaction is serviced by the secondary database in operation **256**. Otherwise, the cache manager **150** sends one or more queries to the primary database **140** that reference the database entries that are not contained in the secondary database **122**.

**[0035]** In some embodiments, the secondary database **122** may contain some, but not all, of the database entries referenced by the transaction **118**. In this situation, the cache manager **150** may send queries referencing only the missing database entries. In other embodiments, when only some of the database entries referenced by the transaction **118** are currently located in the secondary database **122**, the cache manager **150** may query the primary database **140** for all of the database entries referenced by the transaction **118**.

**[0036]** The database entries accessed in the primary database **140** are then uploaded into the secondary database **122** in operation **260**. For example, the cache manager **150** may generate additional SQL statements that cause the primary database entries to be inserted into the secondary database **122**. Any required commitment is performed on the uploaded database entries **204** in operation **262**. The transaction **118** is then serviced by the secondary database in operation **256**.

Cache Groups

**[0037]** FIG. **4** shows how dynamic database caching is used in conjunction with cache groups. A cache instance or cache group is a collection of related records that are uniquely identifiable, and is used to model a complex object. For example, a cache group **314** may be a group of rows that correspond to a set of frequently used tables that are related to each other through foreign key constraints.

**[0038]** Cache instances/cache groups can be used when both loading data from primary database **140** into the secondary database **122** and via versa and when database entries are aged out of the secondary database **122**. The cache group **314** may be configured to contain entire tables or configured to contain only subsets of table rows and/or table columns.

**[0039]** The following SQL syntax is one example of how the cache group **314** is created that includes different database entries from both CUSTOMER table **302** and ORDER table **304**.

```
create cache group cache_customer from
customer(pk1 int not null primary key),
orders(pk2 int not null primary key,
fk2 int, foreign key (fk2) references customer(pk1));
```

**[0040]** In this example, each customer in the CUSTOMER table **302** has a primary key on its ID. One customer may have many orders in the ORDER table **304**, where each order has a foreign key (fk**2**) that references a CUSTOMER(ID). Configuring cache group **314** causes the cache manager **150** to treat all of the order information and associated customer information associated with the transaction as a single cache instance. For example, a transaction may only reference one of the database entries associated with cache group **314**. If the referenced database entry is not contained in secondary database **122**, the cache manager **150** uploads all of the database entries associated with the cache instance from the primary database **140** at the same time.

**[0041]** In this example, CUSTOMER table **302** is considered a root table and ORDERS table **304** is considered a child table. Database entries can be uploaded or flushed based on the root table **302**. For example, all child rows for a root table currently located in the secondary database **122** can also be presumed to be currently located in the secondary database **122**. This prevents the cache manager **150** from having to determine if all of the foreign keys for a cache group exist in the secondary database **122**.

**[0042]** Referring to FIGS. **4** and **5**, the cache manager **150** receives a transaction **306** in operation **350**. The transaction **306** selects all of the orders for a customer having an ID=100. This transaction **306** may be implemented using the following SQL statement **307**.

**[0043]** select * from orders with a customer_ID=100

**[0044]** The cache manager **150** in operation **352** determines if the secondary database **122** contains any database entries in the ORDERS table **304** with the customer_ID=100. If so, the SQL statement **307** is serviced by the secondary database **122** in operation **354** by returning the requested database entries.

**[0045]** When the secondary database **122** does not contain orders with customer_ID=100, the cache manager **150** sends the following query **308** to the primary database **140** in operation **356** selecting the database entries from the ORDER table with customer_ID=100.

**[0046]** select * from orders with a customer_ID=100

**[0047]** However, the cache manager **150** also determines that the referenced database entries are part of the cache group **314**. The cache manager **150** identifies the cache group via the foreign keys assigned to orders in table **304**. Accordingly, the cache manager **150** in operation **358** sends the following second query **310** that selects all of the rows from the CUSTOMER table in the primary database **140** that have an ID=100.

**[0048]** select * from customer where ID=100

**[0049]** The different database entries accessed in the primary database with queries **308** and **310** are referred to as cache instance **320**. It should be understood that the two different queries **308** and **310** select more database entries **320**A-**320**C from the primary database **150** than what was actually referenced by the transaction **306**. Thus, in one embodiment, all of the database entries associated with a same cache instance are loaded into the secondary database at the same time.

**[0050]** The cache manager **150** also sends insert commands **312** to the primary database **140** in operation **360** which cause the rows **320**A-**320**C associated with cache instance **320** to be inserted into the secondary database in operation **360**. The transaction **306** is then serviced by the secondary database in operation **354**.

Aging

**[0051]** FIG. **6** shows how different aging parameters are used to automatically remove database entries from the secondary database **122**. Assigning different aging parameters to different database tables allows the secondary database **122** to dynamically cache more relevant user content. For example, an airlines reservation system may use the secondary database **122** for caching a subset of customer flight reservations and caching a subset of airline flight schedules.

**[0052]** The customer flight reservation tables may be more effectively cached based on usage. For instance, it may be advantageous to maintain customer information in the secondary database **122** for customers who frequently or most recently book airline reservations. This allows faster database response to user reservation queries and further may reduce the amount of traffic between the secondary database **122** and primary database **140**. This is referred to generally as "usage based aging."

**[0053]** Other types of data may be more "time based." For example, airline flight schedules may be highly queried for some period of time. However, after the airline flight arrives at a destination, that flight information is much less likely to be queried again by users. Accordingly, it may be advantageous to remove this type of "time-based" data from the secondary database **122** after a specified time period. The cache manager **150** can be programmed to selectively associate different tables in secondary database **122** with these different usage based and time based aging constraints.

Usage Based Aging

**[0054]** The following SQL, statements may be used for configuring tables A and C in FIG. **6** as usage based aging tables.

```
CREATE TABLE A (C1 INT , C2 INT );
ALTER table A ADD AGING LRU;
TT AgingLRUConfig (LowUsageThreshold,
HighUsageThreshold,AgingCycle)
```

```
CREATE TABLE C (C1 INT , C2 INT );
ALTER table C ADD AGING LRU;
TT AgingLRUConfig (LowUsageThreshold,
HighUsageThreshold,AgingCycle)
```

**[0055]** A listing **400** identifies in column **400**A the tables in secondary database **122** that are configured with usage based aging constraints. In this example, the Least Recently Used (LRU) database entries in tables A and C are periodically removed according the amount of available space in the secondary database **122**.

**[0056]** High Usage Threshold (HUT) values **400**B identify a percentage of used memory space in the secondary database **122** that trigger the cache manager **150** to remove least recently used database entries. Low Usage Threshold (LUT) values **400**C can also be assigned to the tables A and B and identify a second lower percentage of used memory space in the secondary database **122**. When the HUT value **400**B is reached, the cache manager **150** removes least recently used database entries until the storage space in secondary database **122** reaches the LUT value **400**C.

**[0057]** Aging Cycle (AC) values **400**D in listing **400** indicates how often the cache manager **150** evaluates the least recently used database entries in tables A and C. For example, a counter or clock **404** is monitored by cache manager **150**. The cache manager **150** periodically checks the amount of used memory space in the secondary database **122** after counter/clock **404** indicates the expiration of each aging cycle **400**D. If the amount of used memory space reaches HUT **400**B, the cache manager **150** removes the least recently used database entries from the associated tables A and/or C.

**[0058]** Last Used (LU) tags **408** and **412** indicate when the database entries in tables A and C were respectively last used. The LU tags **408** and **412** may use the value provided by counter/clock **404** at the time the associated database item was last accessed or referenced. The LU tags **408** and **412** can then be updated with a current time from counter/clock **404** whenever the associated database entries in tables A or C are accessed or referenced again by another transaction or when the database entries are uploaded again from the primary database **140**.

Time Based Aging

**[0059]** Selected tables in the secondary database **122** can also be assigned time based aging constraints. For example, the following SQL statement configures time based aging constraints for table D.

**[0060]** CREATE TABLE D (Timestamp C**1**, Timestamp c**2**, c**3** INT) AGING USE c**1** LIFETIME {MINUTES|HOURS|DAYS} CYCLE {MINUTES|HOURS|DAYS}

**[0061]** The time based aging SQL statement causes table D to be listed in column **402**A of time based aging listing **402**. A lifetime value **402**B in listing **402** designates how long database entries in table D should reside in the secondary database **122**. A cycle time value **402**C defines a time period for the cache manager **150** to periodically evaluate the database entries in table D. If different cycle times **402**C are defined for different tables, then the cache manager **150** may wake up based on an a single cycle time value for all of the tables, or may wake up according to the cycle times for each individual table.

**[0062]** The time based aging SQL statement above also configures a column in table D with timestamp values **414**. The timestamp values **414** could be a date and time value from counter/clock **404** or could alternatively be a counter value from counter/clock **404** that is continuously incremented until reaching a reset value. In one embodiment, the timestamp values **414** are set to the value of counter/clock **404** when the associated database entries are first loaded into the secondary database **122**.

**[0063]** Referring both to FIGS. **6** and **7**, in operation **450** the tables in the secondary database **122** are configured with different usage based aging constraints and time based aging constraints as described above. In operation **452**, the cache manager **150** identifies the different usage based aging tables and time based aging tables as defined in listings **400** and **402**. In operation **454**, the different aging parameters in listing **400** are identified for the usage based tables. For example, the cache manager **150** in operation **454** identifies the high usage thresholds **400**B, low usage thresholds **400**C and the aging cycles **400**D for tables A and C.

**[0064]** In operation **456** the cache manager **150** waits for one of the aging cycles to be reached for one of the tables A or C. For example, the cache manager **150** determines when the counter/clock **404** reaches the aging cycle **400**D for one of the tables A or C. When an aging cycle is reached in operation **456**, the cache manager **150** determines the amount of memory space currently being used in the secondary database **122**. If the high usage threshold value **400**A is reached for either table A or table C in operation **458**, the least recently used database entries for that table are removed in operation **462**.

**[0065]** For example, the high usage threshold value **400**B for table A may be set to 75% and the high usage threshold value **400**B for table C may be set to 85%. If storage in the secondary database **122** is 80% full when the counter/clock **404** reaches a next aging cycle time **400**D, the least recently used database entry in table A is removed in operation **462**. The LU tag value LU=2 indicates that database entry **416** is the least recently used entry in table A and is accordingly removed from the secondary database **122** in operation **462**.

**[0066]** In operation **464**, the cache manager **150** determines the amount of used storage space after the database entry **416** is removed in operation **462**. If the percentage of used memory space does not drop below the low usage threshold value **400**C for table A in operation **464**, the next least recently used database entry is removed from table A in operation **462**. Database entries are removed from tables A until the amount of utilized space in the secondary database drops below the low usage threshold value in operation **464**. A next aging cycle is started for table A in operation **460** and the cache manager **150** waits for the expiration of the next aging cycle **400**D in operation **456** before conducting the next usage based purge in operation **458**.

**[0067]** If the aging cycles **400**D for tables A and C are different, then a same aging cycle value **400**D may be used. If the HUT values **400**B for both table A and table C are reached at the next common aging cycle, then database entries may be removed from both table A and table C in a round robin fashion. Alternatively, different LRU aging sessions may be separately conducted for tables A and C and LRU database entries for each table removed independently according to their associated HUT values **400**B, LUT values **400**C and aging cycles **400**D.

**[0068]** For example, at the next aging cycle for table C, memory utilization in secondary database **122** may exceed the 85% high usage threshold value **400**B assigned to table C. Accordingly, least recently used database entries are removed from table C in operation **462** until the database storage reaches the low usage threshold value **400**C for table C. In this example, the cache manager **150** removes the least recently used database entries **418** (DB entries #**1**, #**3**, and #**4**) from table C in order to reach the low usage threshold value **400**C associated with table C.

**[0069]** Higher priority data in a particular table may be assigned larger high usage threshold values **400**B and/or larger low usage threshold values **400**C. In addition, the aging cycles **400**D for high priority data may be set to longer time periods. These larger threshold values **400**B, **400**C, and **400**D cause the cache manager **150** to remove the least recently used database items for those tables less frequently. Thus, the usage based aging parameters **400** allow automatic customized removal of different types of selectable data from the secondary database **122**.

**[0070]** FIG. **8** explains in more detail how the cache manager **150** conducts time based aging. Referring to FIGS. **6** and **8**, any tables having time based aging constraints are identified in operation **480**. In this example, table D is assigned a lifetime value **402**B and an associated cycle time value **402**C in listing **402**. The cache manager **150** uses the counter/timer **404** in operation **482** to determine when a next cycle time **402**C is reached. Any rows in table D that have timestamps **414** exceeding the lifetime value **402**B are identified in operation **484** and removed in operation **486**.

**[0071]** For example, the lifetime value **402**B for table D may be set to a particular counter value of say LIFETIME=20. When the cycle time **402**C is reached in operation **482**, the value for counter/clock **404** is compared with the timestamp values **414** in table D. The difference between the value of counter/clock **404** and the timestamp values **414** are determined in operation **484**. In this example, the counter **404** may have a current value of 28. The difference between the current value of counter **404** (28) and the timestamp value for database entry **420** in table D (TS=7) is greater than the lifetime value **402**B (LIFETIME=20). Accordingly, the database entry **420** is removed from table D in operation **486**. Any other database entries in table D with expired lifetimes are also removed in operation **486**.

**[0072]** Similar to usage based aging, different tables can be assigned different lifetime values **402**B and cycle times **402**C. Higher priority data may be assigned larger lifetime values **402**B and/or may be evaluated less frequently by assigning larger cycle time values **402**C.

**[0073]** Tables associated with even higher priority data might not be assigned any aging constraints. For example, table B in FIG. **6** is not assigned any aging constraints. Accordingly, the database entries in table B remain in the secondary database **122** until replaced by updates from the primary database **140**.

Cache Group Aging

**[0074]** Referring back to both operation **462** in FIG. **7** and operation **486** in FIG. **8**, database entries associated with the same cache group may be removed according to usage based or time based aging constraints. Referring also back to FIG. **4**, the CUSTOMER table **302** may both be assigned usage based aging constraints. The high usage threshold may be reached for CUSTOMER table **302** and the database entry PK**1**=100 in the CUSTOMER table **302** may be the least recently used. Accordingly, the entire cache instance **320** may be removed in operation **462** in FIG. **7**. For example, the root database entry PK**1**=100 is removed from the CUSTOMER table **302** and the child database entries PK**2**=14 and PK**2**=20 with referencing foreign keys are removed from the ORDERS table **304**.

**[0075]** In another example, usage based aging may be assigned to the child ORDERS table **304** in FIG. **4**. When storage in secondary database **122** reaches the high usage threshold value for ORDERS table **304**, database entry PK**1**=14 may be the least recently used.

**[0076]** In one embodiment, all of the root and child database entries for the same cache instance **320** are removed from the secondary database in operation **462** in FIG. **7**. In another embodiment, cache groups are only aged based on the database entries in the root table **302**. For example, since database entry PK**2**=14 is located in child table **304** and not located in root table **302**, no usage based aging is performed.

**[0077]** For time based aging, database entries associated with the same cache group may also be controlled by the root table. For example, timestamps may only be applied to the database entries in the root CUSTOMER table **302** in FIG. **4**. Whenever one of the database entries in CUSTOMER table **302** resides in the secondary database **122** beyond a specified lifetime value, all of the database entries associated with that cache instance are removed at the same time.

**[0078]** For example in FIG. **4**, database entry PK**1**=100 may reside in the secondary database **122** beyond a lifetime value assigned to CUSTOMER table **302**. Accordingly, both database entry PK**1**=100 in CUSTOMER table **302** and the other database entries PK**2**=14 and PK**2**=20 in ORDERS table **304** associated with the same cache instance **320** are removed by the cache manager **150**.

**[0079]** The system described above can use dedicated processor systems, micro controllers, programmable logic devices, or microprocessors that perform some or all of the operations. Some of the operations described above may be implemented in software and other operations may be implemented in hardware.

**[0080]** For the sake of convenience, the operations are described as various interconnected functional blocks or distinct software modules. This is not necessary, however, and there may be cases where these functional blocks or modules are equivalently aggregated into a single logic device, program or operation with unclear boundaries. In any event, the functional blocks and software modules or features of the flexible interface can be implemented by themselves, or in combination with other operations in either hardware or software.

**[0081]** Having described and illustrated the principles of the invention in a preferred embodiment thereof, it should be apparent that the invention may be modified in arrangement and detail without departing from such principles. Claim is made to all modifications and variation coming within the spirit and scope of the following claims.

**1**. A method, comprising:
operating a fully transactional mid-tier database;
receiving one or more database transactions at the mid-tier database; and
dynamically loading database entries from a fully transactional backend-tier database into the mid-tier database according to the received database transactions.

**2**. The method according to claim **1** further comprising operating the mid-tier database as an in-memory database.

**3**. The method according to claim **1** further comprising:
identifying database entries associated with the database transactions;
servicing the database transactions with the mid-tier database when the identified database entries are contained in the mid-tier database; and
querying database entries in the backend-tier database when the identified database entries are not contained in the mid-tier database.

**4**. The method according to claim **3** further comprising:
uploading the queried database entries into the mid-tier database; and
servicing the database transactions with the mid-tier database using at least some of the uploaded database entries from the backend-tier database.

**5**. The method according to claim **1** further comprising:
identifying tables and primary keys referenced by the database transactions;
searching the mid-tier database for the identified tables and primary keys;
servicing the database transactions with the mid-tier database when the identified tables and primary keys are located in the mid-tier database; and
accessing the backend-tier database when the identified tables and primary keys are not located in the mid-tier database.

**6**. The method according to claim **1** further comprising:
associating database entries from different tables with a same cache group;
identifying a database transaction that references one or more database entries associated with the cache group;
uploading the entire cache group from the backend-tier database into the mid-tier database when the referenced database entries are not contained in the mid-tier database.

**7**. The method according to claim **6** further comprising:
identifying a first database entry in a first table;
identifying a primary key associated with the first database entry in the first table;
identifying a second database entry in a second table having a foreign key referencing the primary key in the first table; and
associating both the first database entry in the first table and the second database entry in the second table with the same cache group.

**8**. A method, comprising:
operating a database system;
assigning aging parameters to selected tables in the database system; and
removing database items from the selected tables in the database system according to the assigned aging parameters.

**9**. The method according to claim **8** further comprising:
associating time based aging with the selected tables in the database system; and
removing the database items from the selected tables according to how long the database items have resided in the database system.

**10**. The method according to claim **8** further comprising:
associating usage based aging with the selected tables in the database system; and
removing the database items from the selected tables according to how recently the database items have been used in the database system.

**11**. The method according to claim **10** further comprising:
assigning a high usage threshold value, a low usage threshold value, and an aging cycle to the selected tables in the database system;
identifying an amount of available space in the database system for each aging cycle;
removing at least some of the least recently used database items from the selected tables when an amount of storage space in the database system reaches the high usage threshold value; and
removing least recently used database items from the selected tables until the amount of storage space in the database system reaches the low usage threshold value.

**12**. The method according to claim **8** further comprising:
configuring different time based aging tables and usage based aging tables in the database system;
removing at least some of the database items from the time based aging tables that have resided in the database system beyond a configured time period;
removing at some of the least recently used database items from the usage based aging tables when storage in the database system reaches a configured usage threshold; and
skipping removal of database items from non-time based and non-usage based tables in the database system.

**13**. The method according to claim **8** further comprising:
associating some of the database items from different selected tables with a same cache group;
assigning an aging parameter to a root table of the cache group; and
removing the database items from both the root table and one or more child tables associated with the cache group according to the aging parameter assigned to the root table.

**14**. Computer readable media containing instructions that when executed by a computer, comprise:
operating a fully-transactional secondary database;
receiving database transactions directed to the secondary database;

searching the secondary database for data items referenced by the transactions;

servicing the database transactions with the secondary database when the secondary database contains the referenced data items;

querying a fully-transactional primary database when the referenced data items are not contained in the secondary database; and

updating the secondary database with at least some of the data items queried in the primary database.

**15**. The computer readable media according to claim **14** further comprising instructions that when executed result in:

assigning aging parameters to selectable data items in the secondary database; and

automatically removing data items from the secondary database according to the assigned aging parameters.

**16**. The computer readable media according to claim **14** further comprising instructions that when executed result in:

assigning either a time based aging parameter or a least recently used aging parameter to programmably selectable tables in the secondary database; and

removing the data items according to the time based aging parameter or least recently used aging parameter assigned to the tables containing the data items.

**17**. The computer readable media according to claim **14** further comprising instructions that when executed result in:

configuring time based aging tables and usage based aging tables in the secondary database;

removing at least some of the data items from the time based aging tables that have resided in the secondary database beyond a configured time period; and

removing at some of the least recently used data items from the usage based aging tables when storage in the secondary database reaches a configured usage threshold value.

**18**. The computer readable media according to claim **14** further comprising instructions that when executed result in:

associating at least some of the data items from different tables with a same cache group;

identifying transactions referencing one or more of the data items from the cache group;

uploading the entire cache group from the primary database into the secondary database when any of the referenced data items are not contained in the secondary database.

**19**. The computer readable media according to claim **18** further comprising instructions that when executed result in:

assigning an aging parameter to a root table for the cache group; and

removing all of the data items associated with the cache group according to the aging parameter assigned to the root table.

**20**. A database management system, comprising:

an application-tier database system receiving requests from database applications; and

a cache manager configured to maintain at least a sub-set of database entries from a backend-tier database system in the application-tier database system, the cache manager dynamically varying what database entries are loaded from the backend-tier database system into the application-tier database system according to the requests from the database applications.

**21**. The database management system according to claim **20** wherein the cache manager is further configured to:

monitor the different requests received by the application-tier database system;

identify what database entries are referenced by the requests;

service the requests with the application-tier database system when the referenced database entries are contained in the application-tier database system;

temporarily stall the requests when the referenced database entries are not contained in the application-tier database system;

query the backend-tier database system for the referenced database entries;

load the database entries queried from the backend database system into the application-tier database system; and

service the requests using at least some of the database entries loaded into the application-tier database system from the backend-tier database system.

**22**. The database management system according to claim **20** wherein the cache manager is further configured to:

identify cache groups that include both a root table with one or more database entries having primary keys and one or more child tables having one or more database entries having foreign keys referencing the primary keys in the root table; and

dynamically removing the identified cache groups from the application-tier database system or dynamically loading the identified cache groups from the backend-tier database system into the application-tier database system.

**23**. The database management system according to claim **20** wherein usage based aging is assigned to selected tables in the application-tier database system and the cache manager removes database entries contained in the selected tables according to how recently the database entries have been used in the application-tier database system.

**24**. The database management system according to claim **20** wherein time-based aging is assigned to selected tables in the application-tier database system and the cache manager removes database entries contained in the selected tables according to how long the database entries have resided in the application-tier database system.

* * * * *