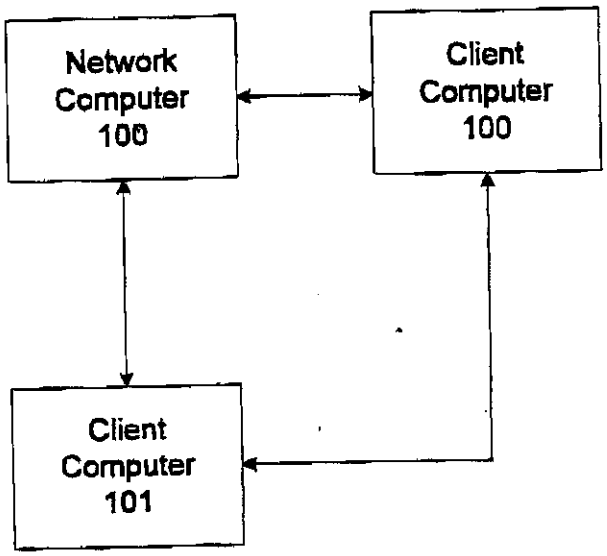

Network Computer 100
Client Computer 100
Client Computer 101

FIG. 1A

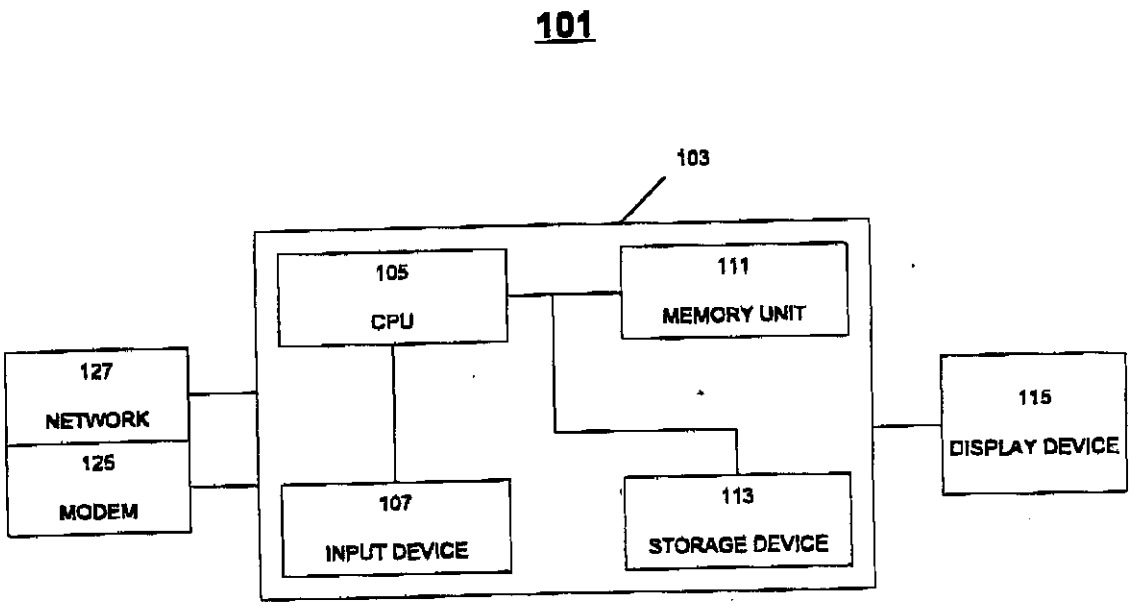

101
103
105
CPU
111
MEMORY UNIT
127
NETWORK
125
MODEM
107
INPUT DEVICE
113
STORAGE DEVICE
115
DISPLAY DEVICE

FIG. 1B

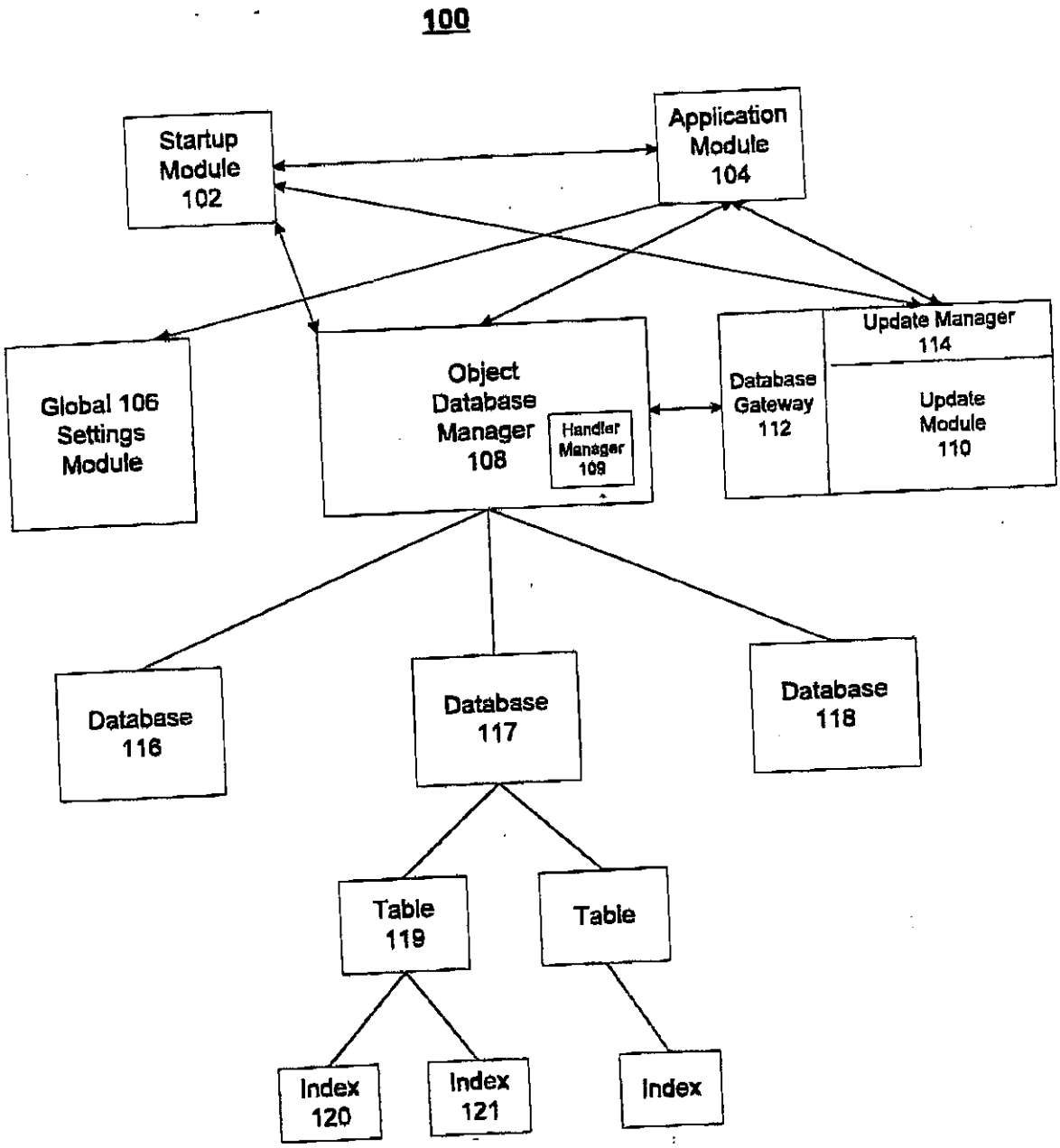

100
Startup Module 102
Application Module 104
Global 106 Settings Module
Object Database Manager 108
Handler Manager 109
Database Gateway 112
Update Manager 114
Update Module 110
Database 116
Database 117
Database 118
Table 119
Table
Index 120
Index 121
Index

FIG. 1C

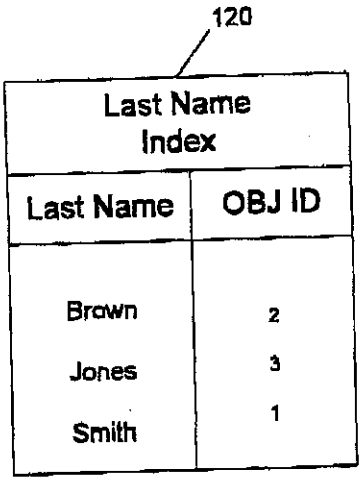


FIG. 1F

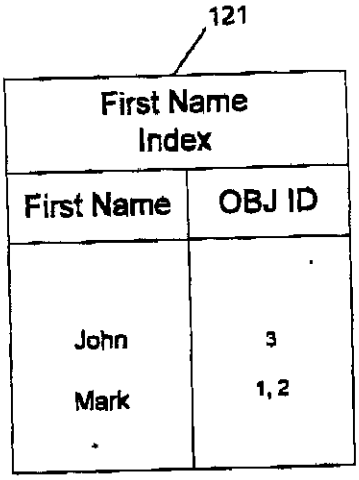


FIG. 1G

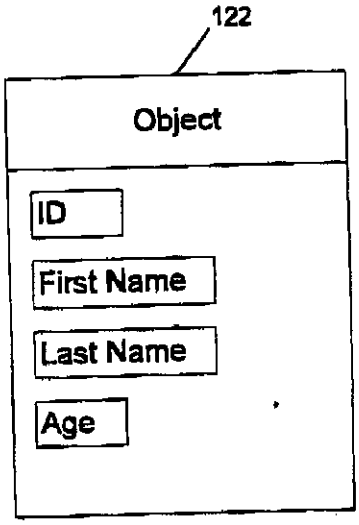


FIG. 1D

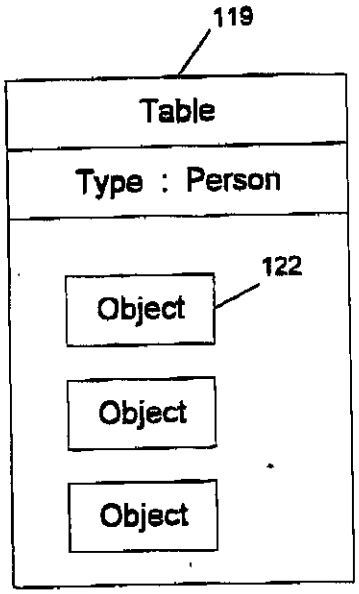


FIG. 1E

File Glossary	
Symbolic Name	**File Name**
Sample database	c:\myapp\database\gzlext2
Sample database—Person Table	c:\myapp\database\gzlext3
Sample database—Person Table—Data	c:\myapp\database\gzlext4
Sample database—Person Table—Data—File map	c:\myapp\database\gzlext5
Sample database—Person Table—Data—RAF	c:\myapp\database\gzlext6
Sample database—Person Table—Index—Last name	c:\myapp\database\gzlext7

FIG. 1H

Table File	
Field	**Value**
Base class name	Person.class
Store handle	Sample database—Person Table—Store
Cache Policy	MEDIUM_CACHING
Number of indices	1
Symbolic Name of index #1	Sample database—Person Table—Index—Last name

FIG. 1I

Data File	Stores info about a particular store
Field	**Value**
RAF Symbolic Name	Sample database—Person Table—Data—RAF
File Map Symbolic Name	Sample database—Person Table—Data—File map

FIG. 1J

File map	
Field	**Value**
File map	<Data structure mapping object ID to file offset>
End of file pointer	<Length of file in bytes>
Total store size	<Total number of bytes in the file dedicated to storing objects>
Total number of live objects in store	<Total number of objects>

FIG. 1K

Index file	
Field	**Value**
Base class	String
Index Structure	<Data structure mapping index values to object IDs>
Is this a unique index?	False

FIG. 1L

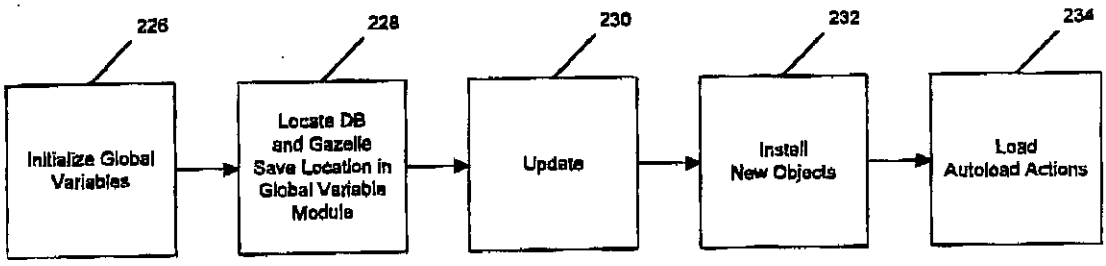

226
228
230
232
234
Initialize Global Variables
Locate DB and Gazelle Save Location in Global Variable Module
Update
Install New Objects
Load Autoload Actions

FIG. 2

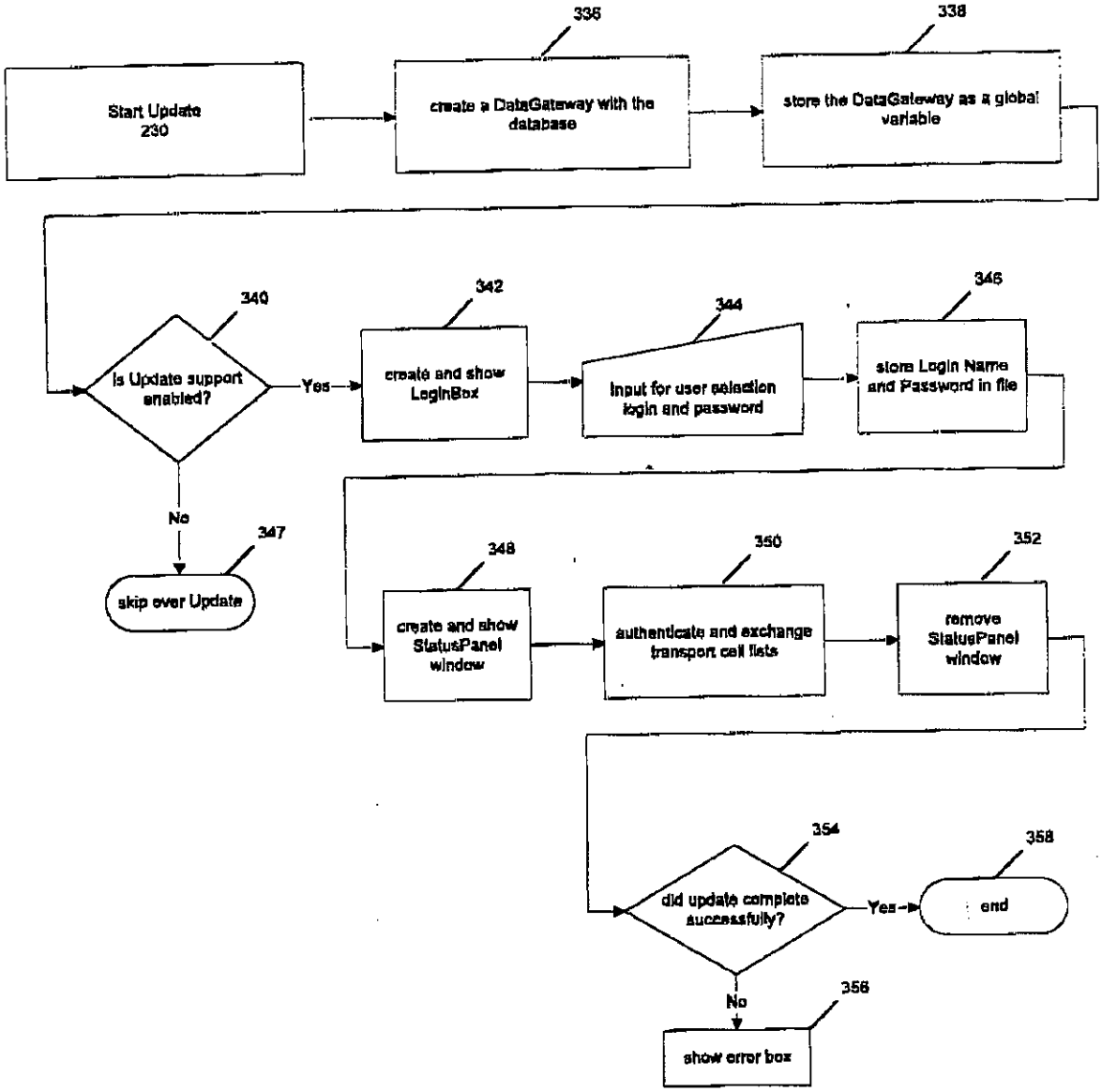

Start Update
230
336
create a DataGateway with the database
338
store the DataGateway as a global variable
340
Is Update support enabled?
Yes
342
create and show LoginBox
344
Input for user selection login and password
346
store Login Name and Password in file
No
347
skip over Update
348
create and show StatusPanel window
350
authenticate and exchange transport cell lists
352
remove StatusPanel window
354
did update complete successfully?
Yes
358
end
No
356
show error box

FIG. 3

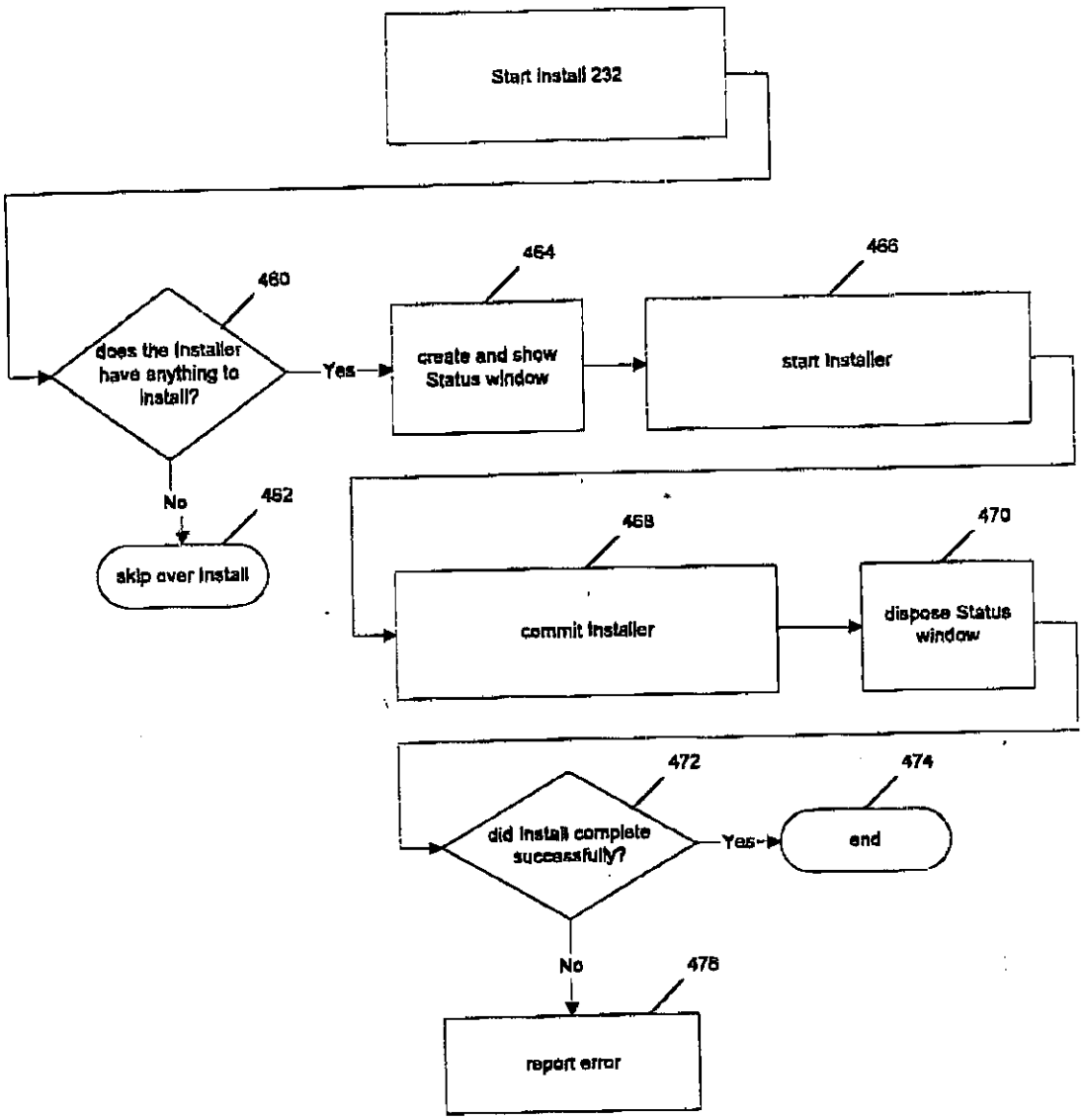

Start Install 232
460
does the Installer have anything to install?
Yes
464
create and show Status window
466
start Installer
No
462
skip over Install
468
commit Installer
470
dispose Status window
472
did Install complete successfully?
Yes
474
end
No
476
report error

FIG. 4

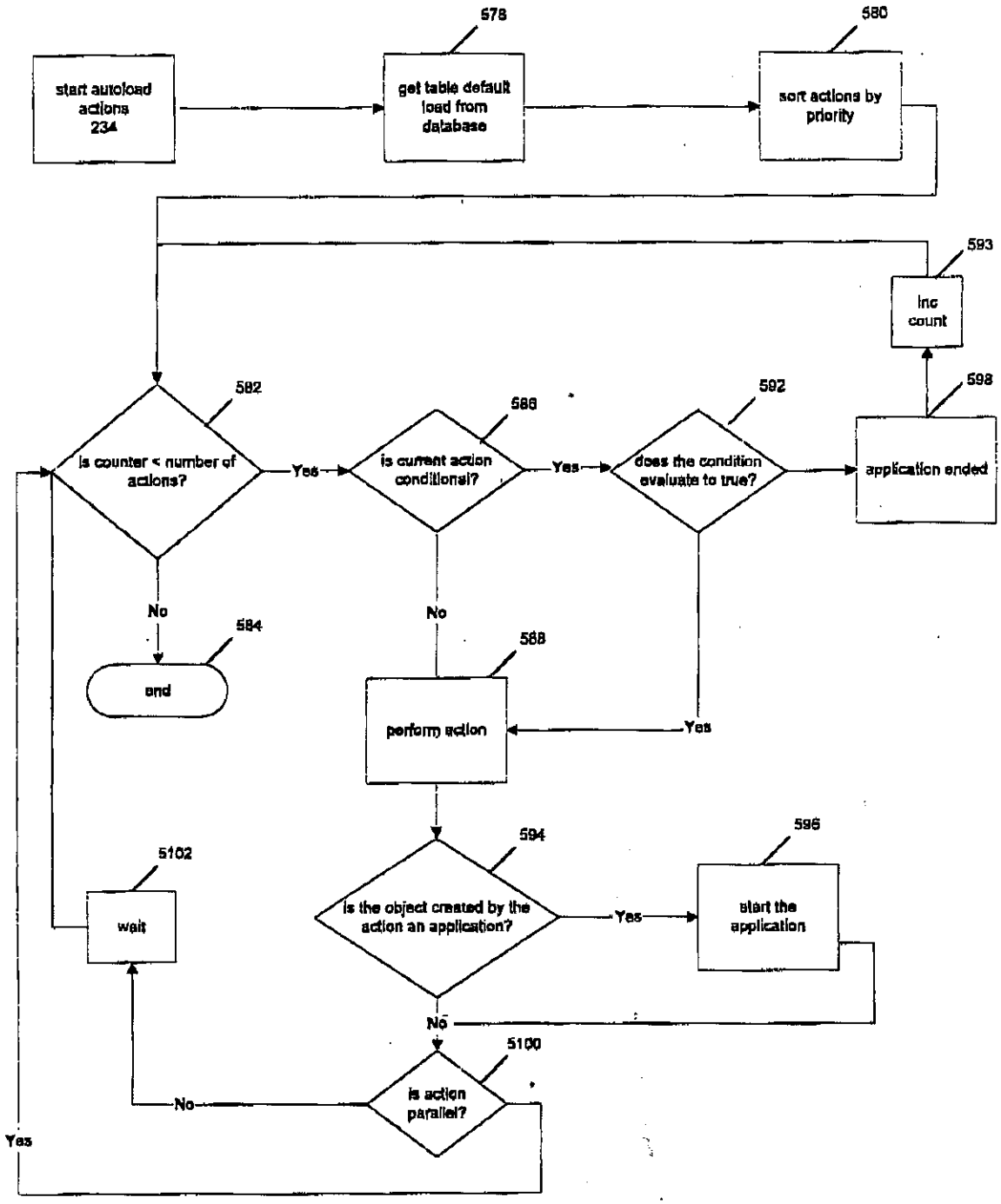

start autoload actions 234
578
get table default load from database
580
sort actions by priority
593
inc count
598
application ended
582
Is counter < number of actions?
Yes
586
is current action conditional?
Yes
592
does the condition evaluate to true?
Yes
No
584
end
No
588
perform action
Yes
5102
wait
594
Is the object created by the action an application?
Yes
596
start the application
No
5100
is action parallel?
No
Yes

FIG. 5

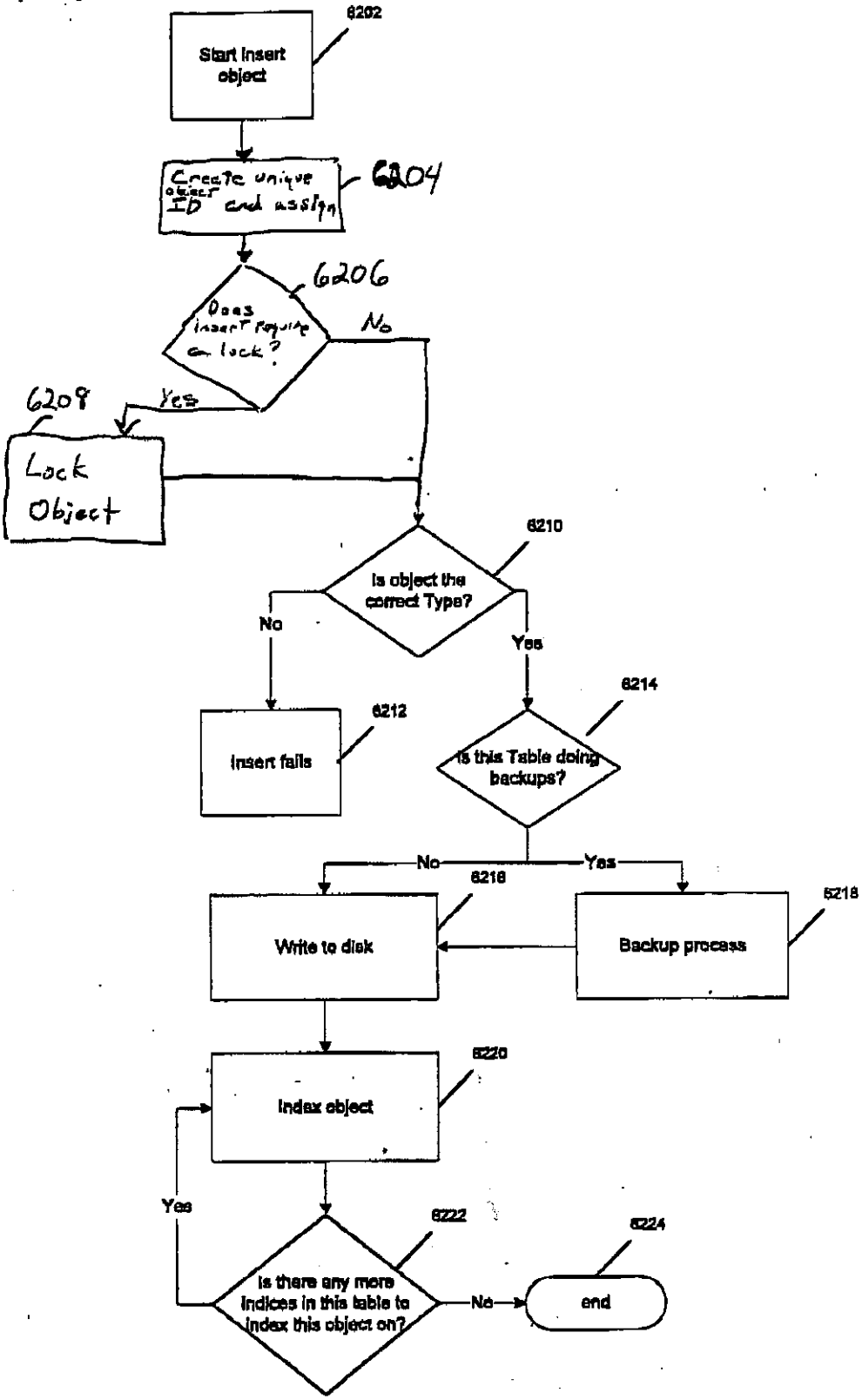

6202
Start Insert object
6204
Create unique object ID and assign
6206
Does insert require a lock?
No
Yes
6208
Lock Object
6210
Is object the correct Type?
No
Yes
6212
Insert fails
6214
Is this Table doing backups?
No
Yes
6216
Write to disk
6218
Backup process
6220
Index object
6222
Is there any more indices in this table to index this object on?
Yes
No
6224
end

FIG. 6A

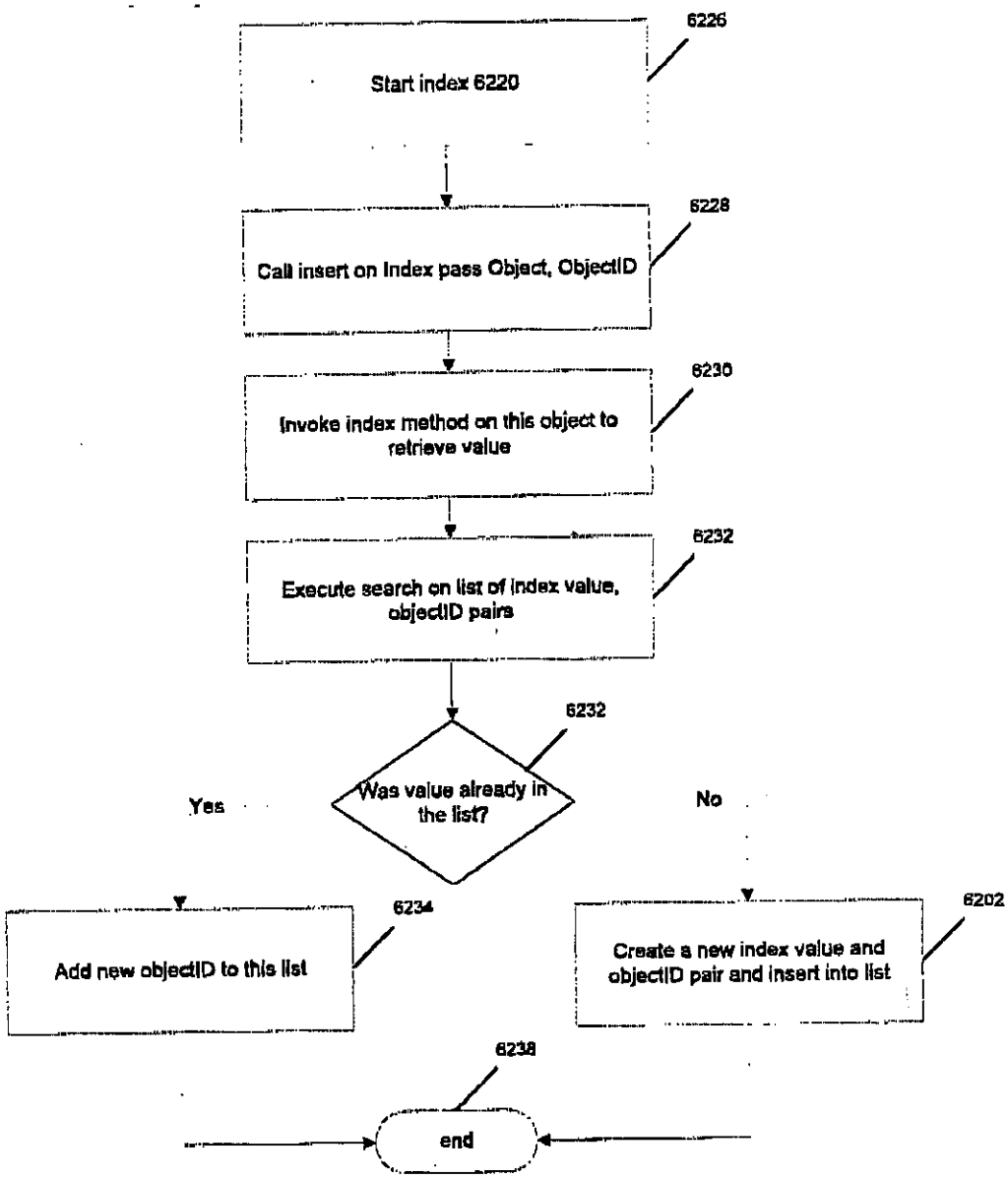

6226
Start index 6220
6228
Call insert on Index pass Object, ObjectID
6230
Invoke index method on this object to retrieve value
6232
Execute search on list of index value, objectID pairs
6232
Was value already in the list?
Yes
No
6234
Add new objectID to this list
6202
Create a new index value and objectID pair and insert into list
6238
end

FIG. 6B

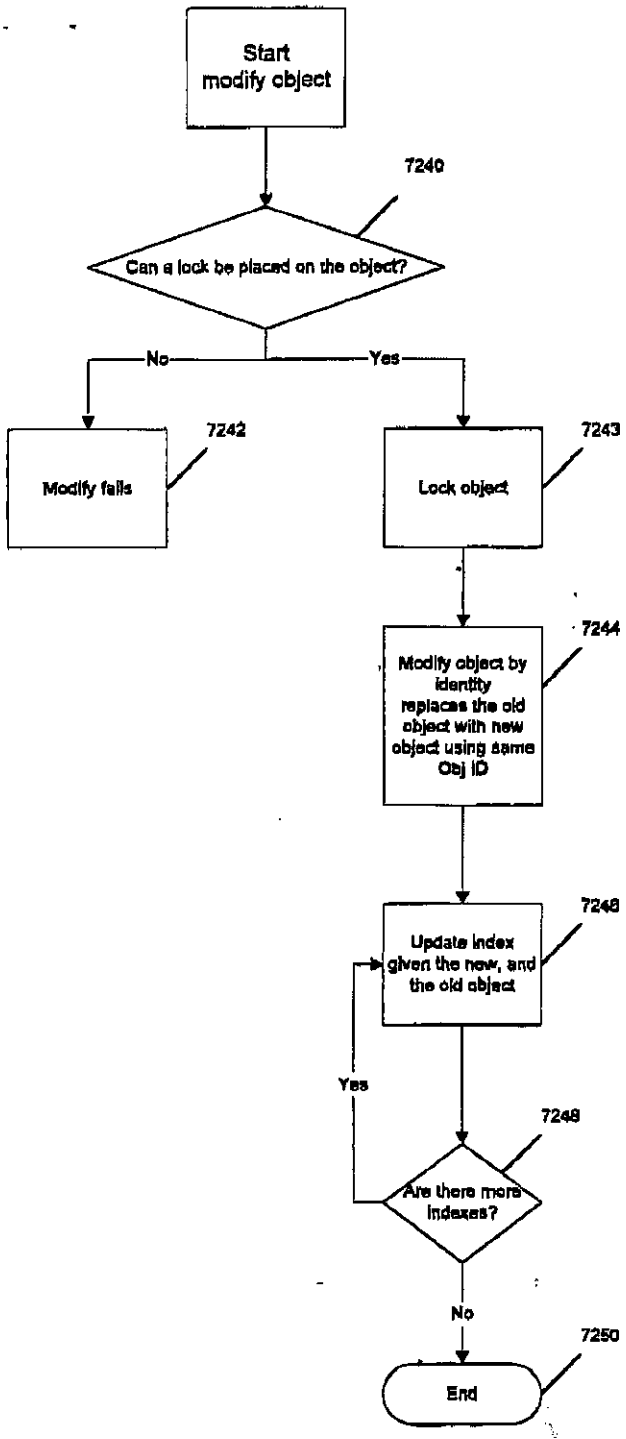

Start modify object
7240
Can a lock be placed on the object?
No
Yes
7242
Modify fails
7243
Lock object
7244
Modify object by identity replaces the old object with new object using same Obj ID
7246
Update index given the new, and the old object
Yes
7248
Are there more indexes?
No
7250
End

FIG. 7A

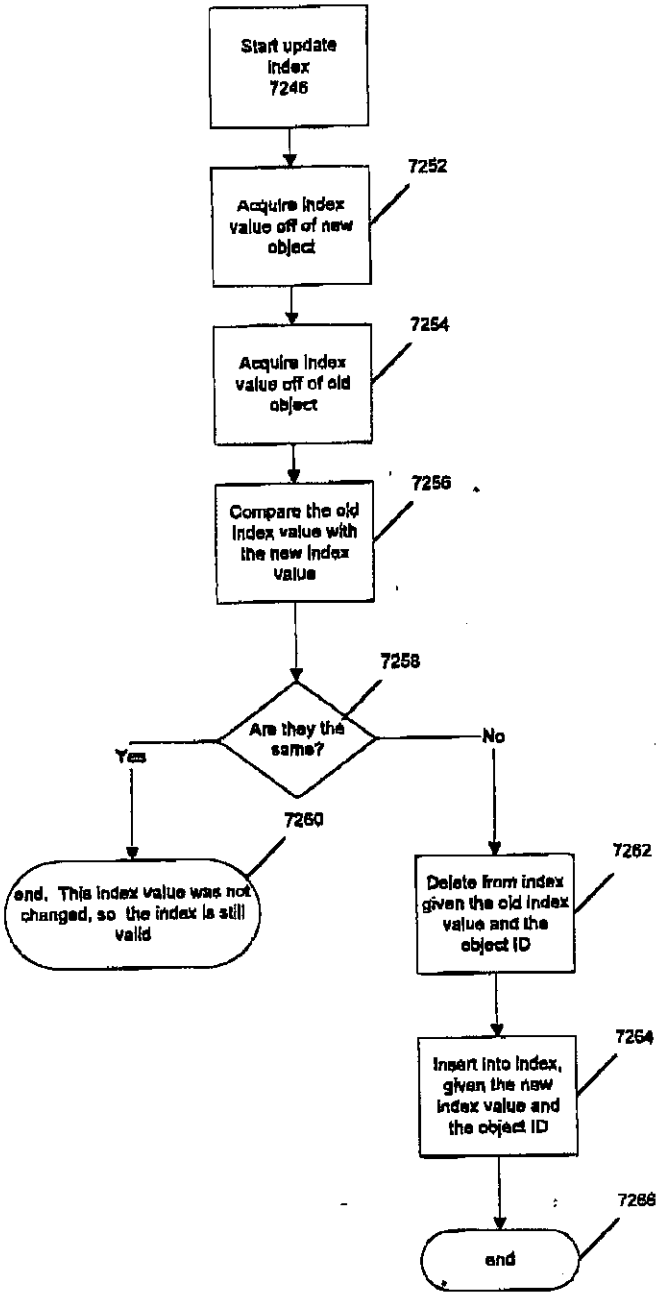

Start update index 7246
7252
Acquire index value off of new object
7254
Acquire index value off of old object
7256
Compare the old index value with the new index value
7258
Are they the same?
Yes
No
7260
end. This index value was not changed, so the index is still valid
7262
Delete from index given the old index value and the object ID
7264
Insert into index, given the new index value and the object ID
7266
end

FIG. 7B

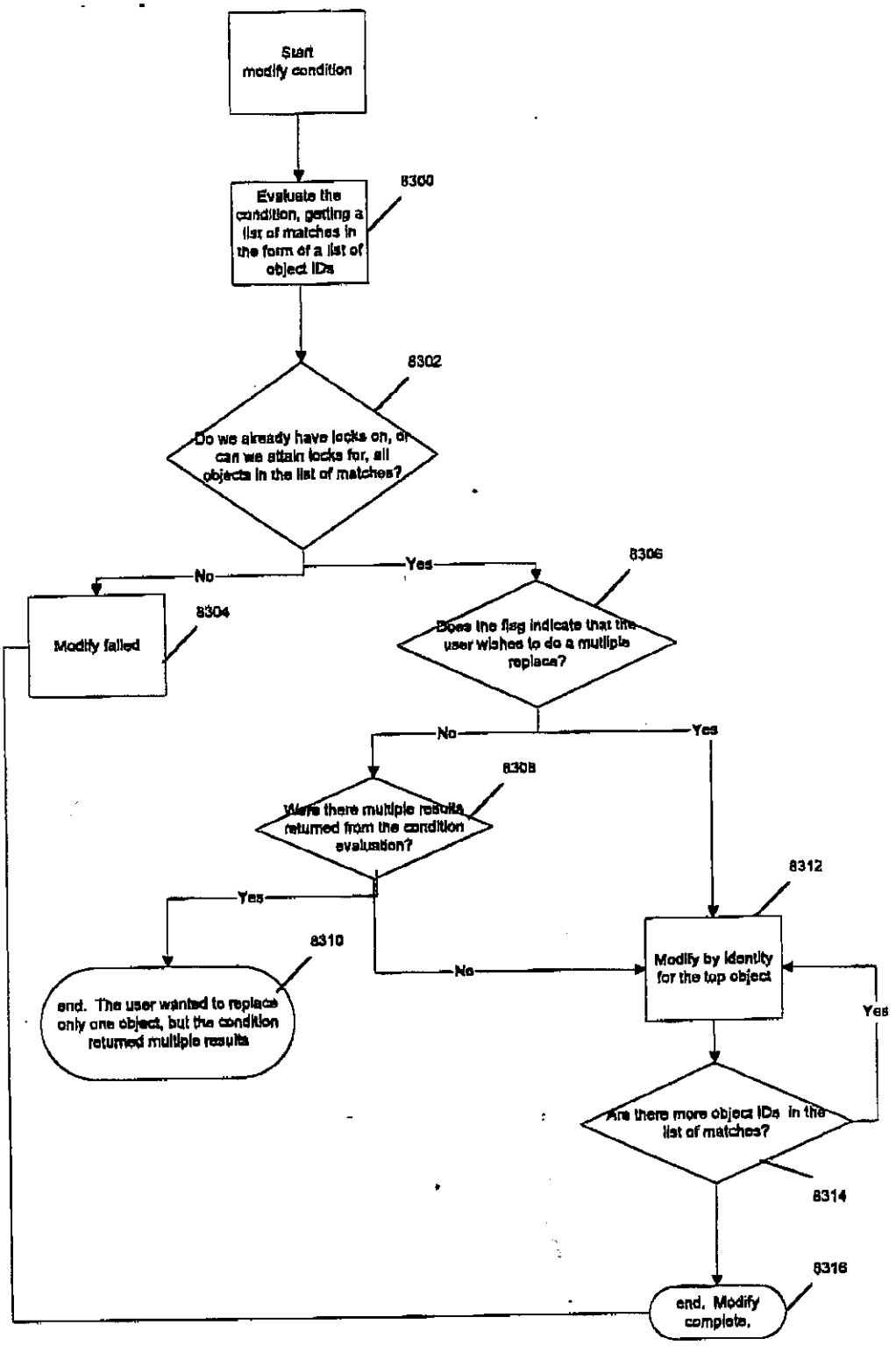

Start modify condition
Evaluate the condition, getting a list of matches in the form of a list of object IDs
8300
8302
Do we already have locks on, or can we attain locks for, all objects in the list of matches?
No
Yes
Modify failed
8304
8306
Does the flag indicate that the user wishes to do a mutliple replace?
No
Yes
8308
Were there multiple results returned from the condition evaluation?
Yes
No
8310
end. The user wanted to replace only one object, but the condition returned multiple results
8312
Modify by identity for the top object
Yes
Are there more object IDs in the list of matches?
8314
8316
end. Modify complete.

FIG. 8

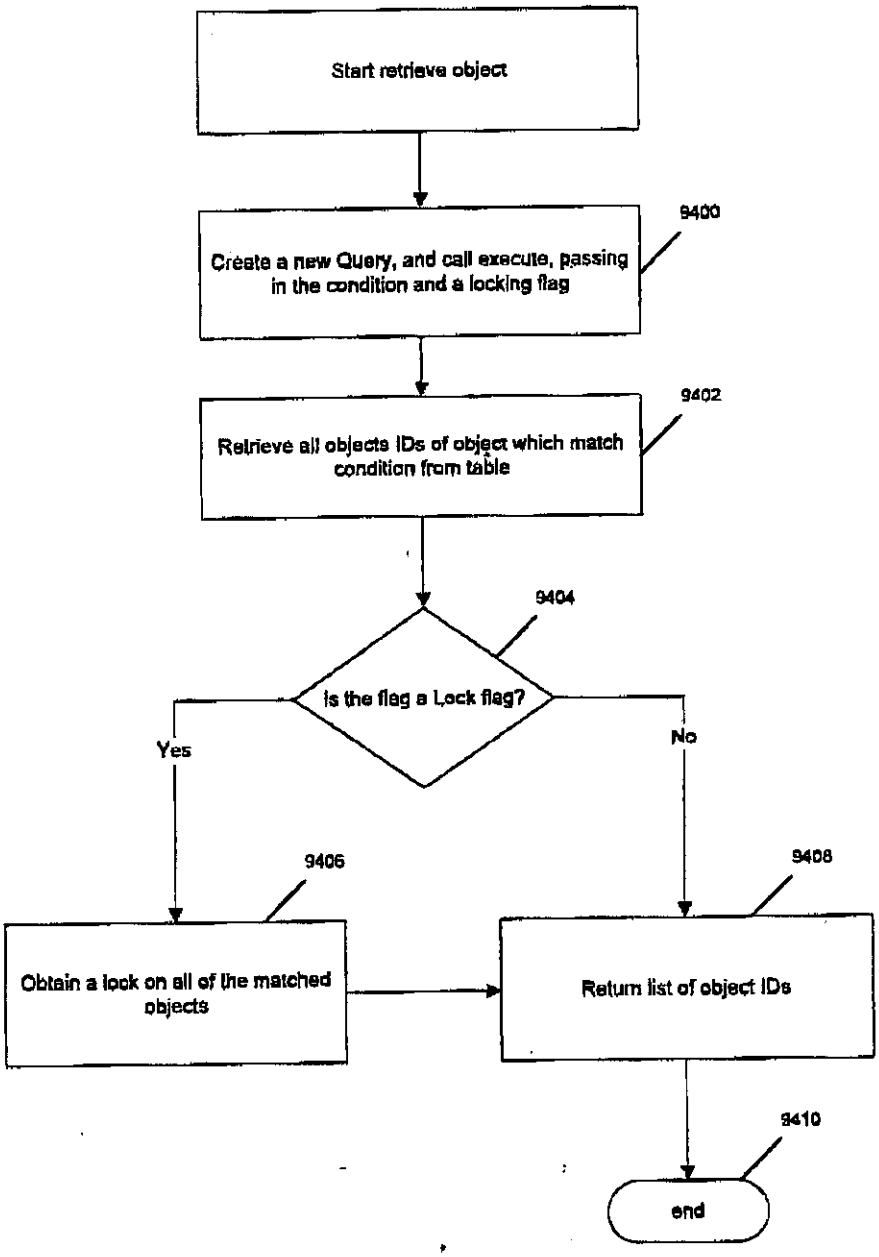

Start retrieve object
Create a new Query, and call execute, passing in the condition and a locking flag
9400
Retrieve all objects IDs of object which match condition from table
9402
Is the flag a Lock flag?
9404
Yes
No
Obtain a lock on all of the matched objects
9406
Return list of object IDs
9408
end
9410

FIG. 9

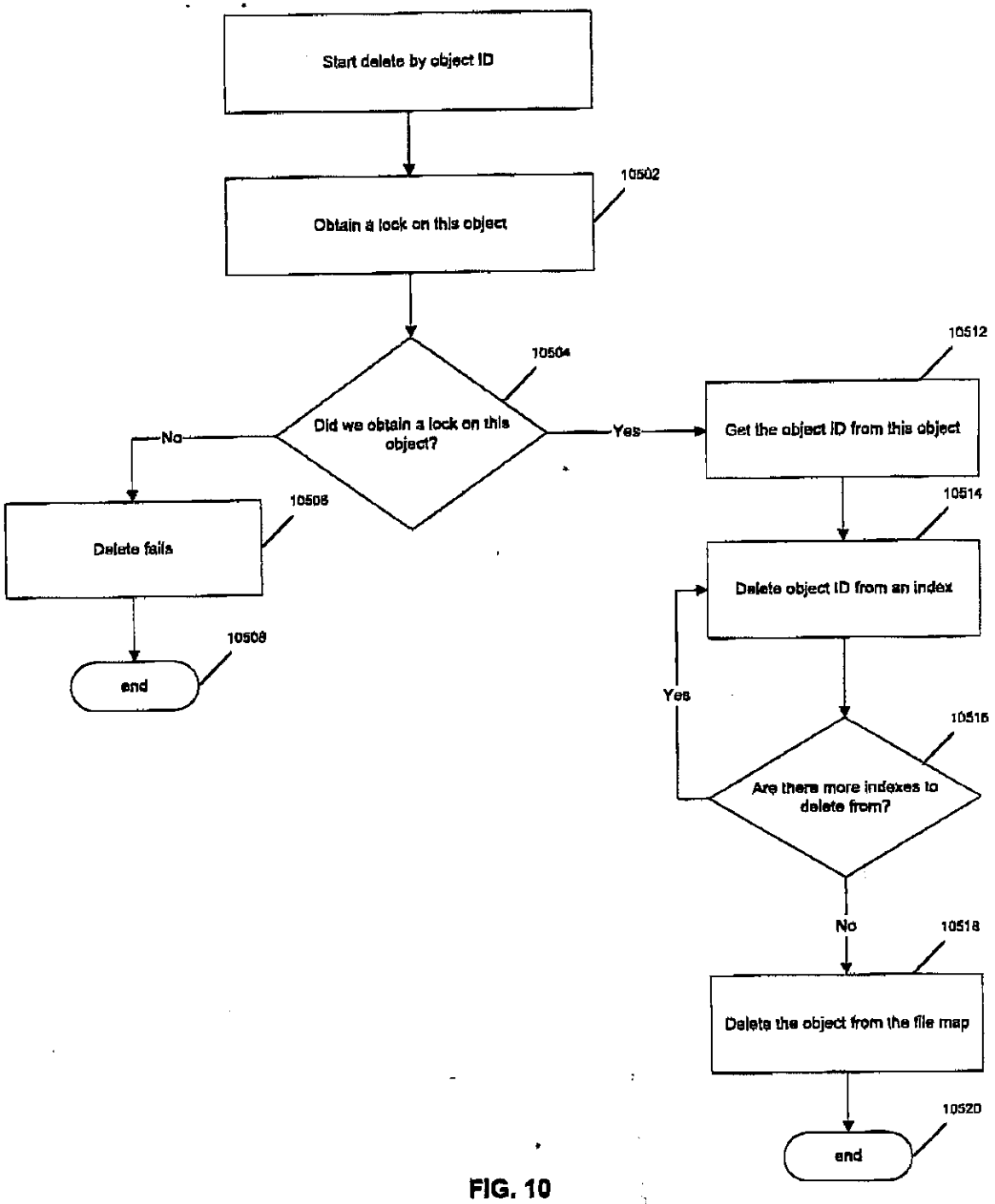

Start delete by object ID
Obtain a lock on this object
10502
Did we obtain a lock on this object?
10504
No
Yes
Delete fails
10506
end
10508
Get the object ID from this object
10512
Delete object ID from an index
10514
Yes
Are there more indexes to delete from?
10516
No
Delete the object from the file map
10518
end
10520

FIG. 10

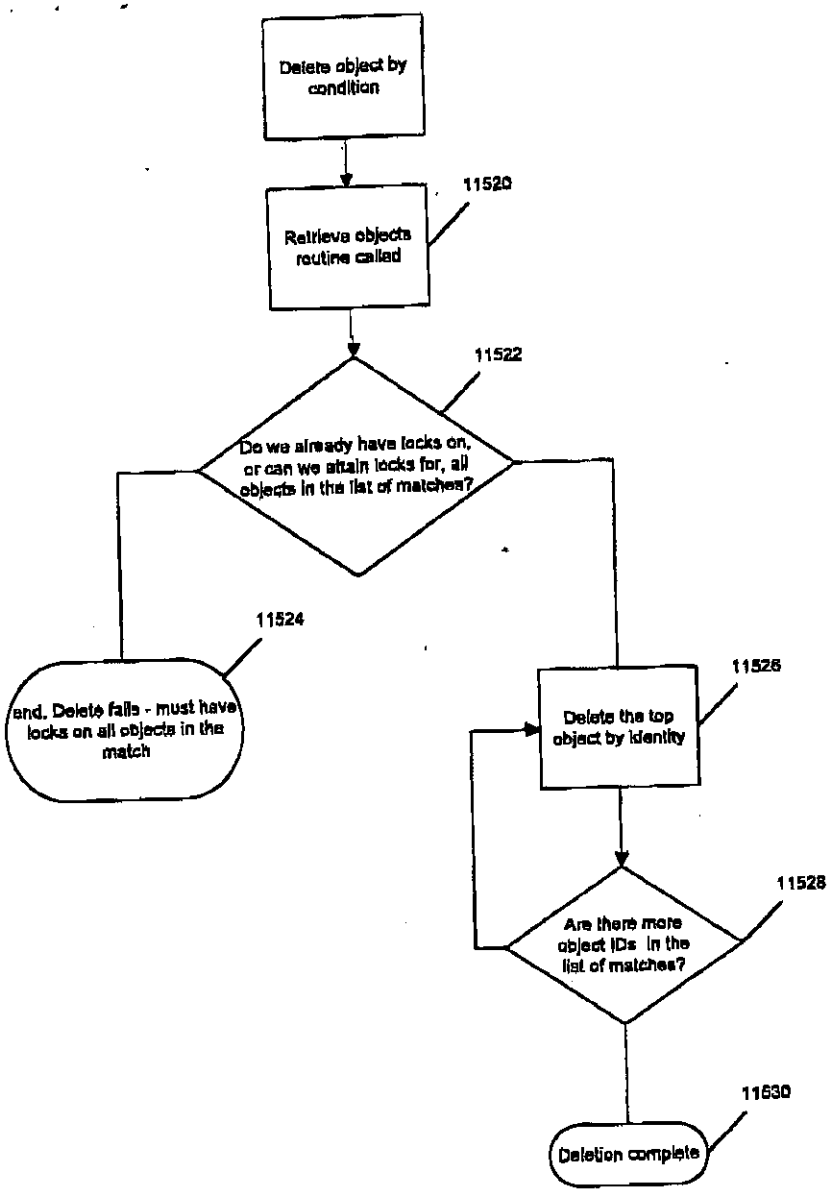

Delete object by condition
11520
Retrieve objects routine called
11522
Do we already have locks on, or can we attain locks for, all objects in the list of matches?
11524
end. Delete fails - must have locks on all objects in the match
11526
Delete the top object by identity
11528
Are there more object IDs in the list of matches?
11530
Deletion complete

FIG. 11

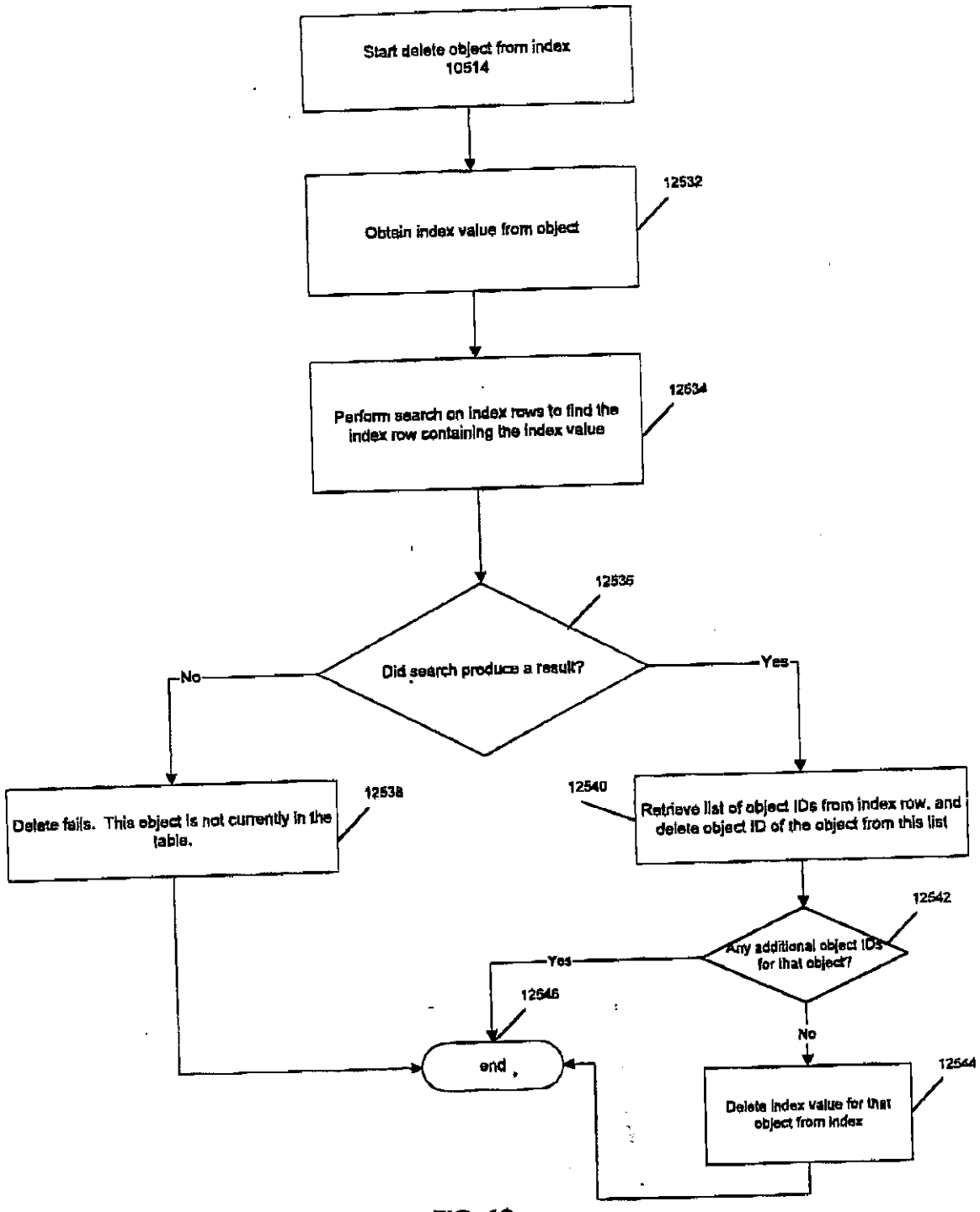

Start delete object from index 10514
Obtain index value from object
12532
Perform search on index rows to find the index row containing the index value
12534
Did search produce a result?
12535
No
Yes
Delete fails. This object is not currently in the table.
12538
12540
Retrieve list of object IDs from index row, and delete object ID of the object from this list
Any additional object IDs for that object?
12542
Yes
No
12546
end
Delete index value for that object from index
12544

FIG. 12

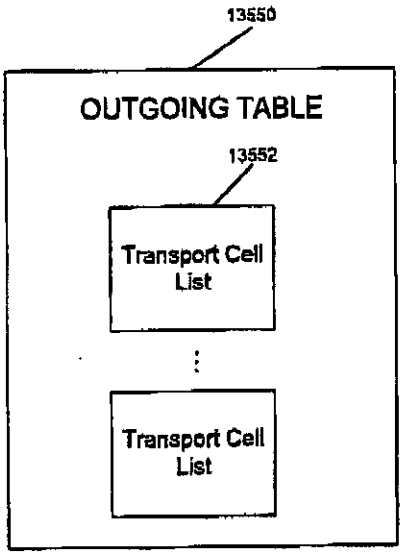


FIG. 13A

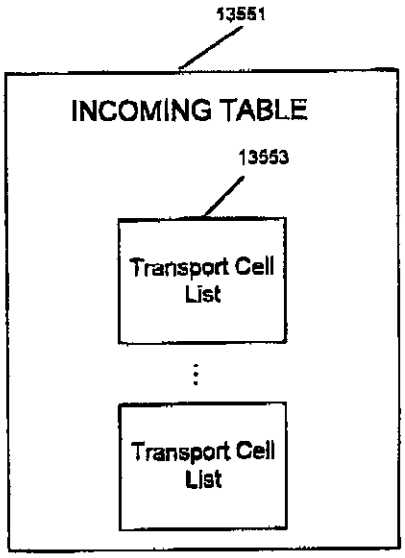


FIG. 13B

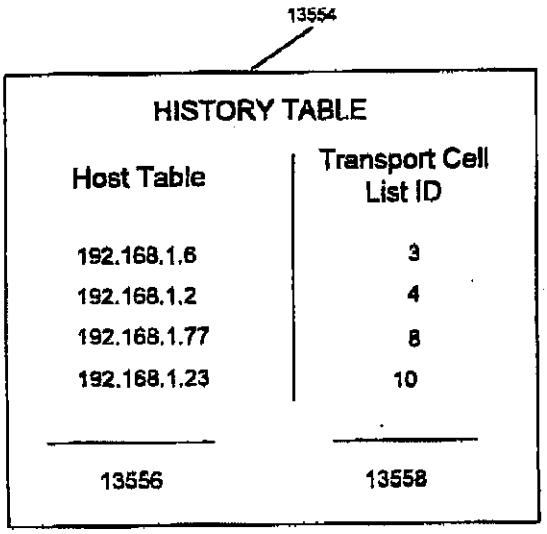


FIG. 13C

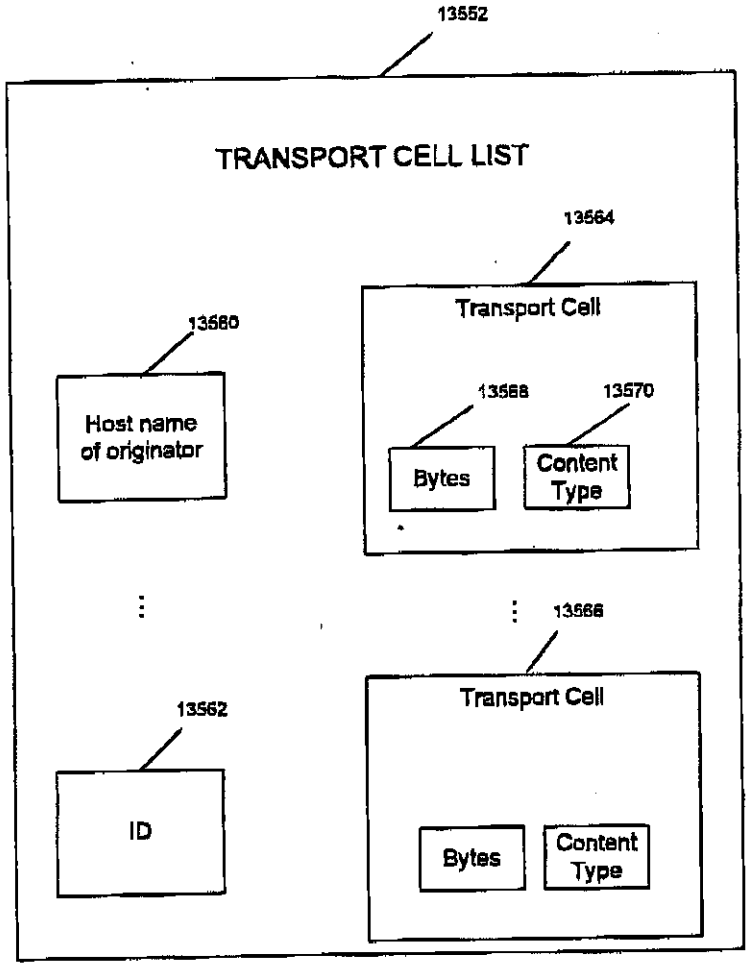

13552
TRANSPORT CELL LIST
13564
Transport Cell
13560
Host name of originator
13568
13570
Bytes
Content Type
13566
Transport Cell
13562
ID
Bytes
Content Type

FIG. 13D

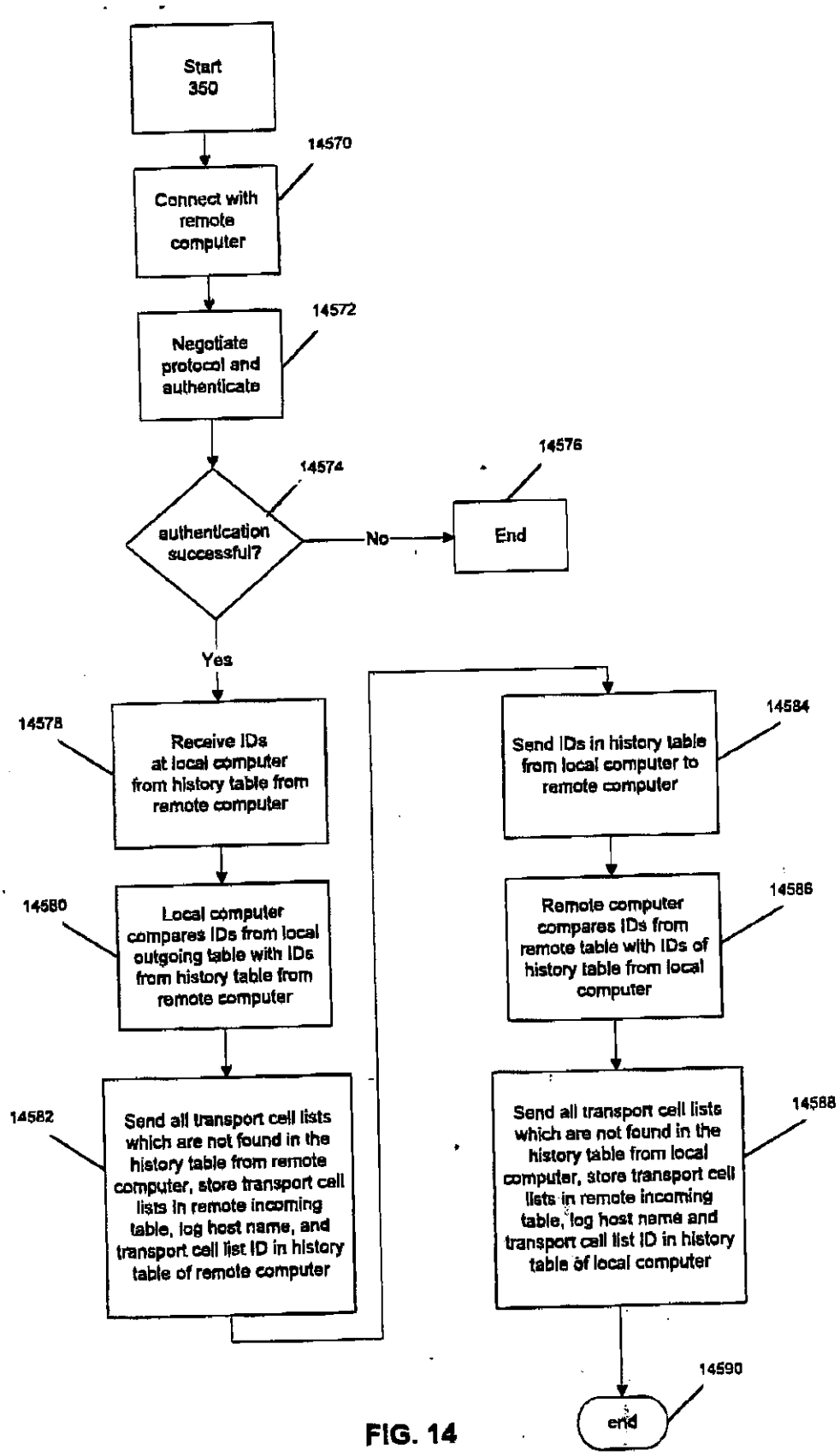

Start
350
14570
Connect with remote computer
14572
Negotiate protocol and authenticate
14574
authentication successful?
No
14576
End
Yes
14578
Receive IDs at local computer from history table from remote computer
14580
Local computer compares IDs from local outgoing table with IDs from history table from remote computer
14582
Send all transport cell lists which are not found in the history table from remote computer, store transport cell lists in remote incoming table, log host name, and transport cell list ID in history table of remote computer
14584
Send IDs in history table from local computer to remote computer
14586
Remote computer compares IDs from remote table with IDs of history table from local computer
14588
Send all transport cell lists which are not found in the history table from local computer, store transport cell lists in remote incoming table, log host name and transport cell list ID in history table of local computer
14590
end

FIG. 14

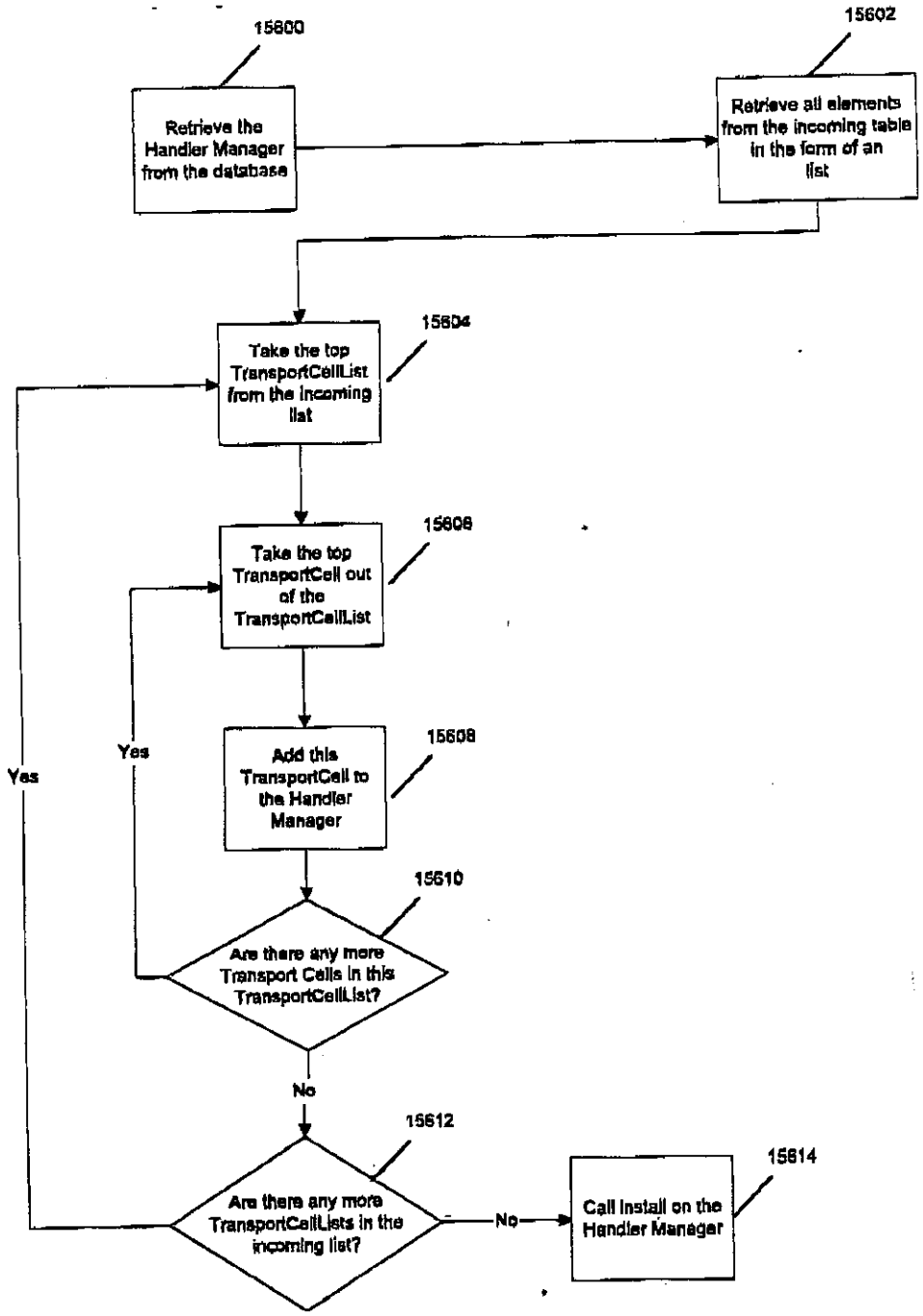

15600
Retrieve the Handler Manager from the database
15602
Retrieve all elements from the incoming table in the form of an list
15604
Take the top TransportCellList from the incoming list
15606
Take the top TransportCell out of the TransportCellList
15608
Add this TransportCell to the Handler Manager
15610
Are there any more Transport Cells in this TransportCellList?
Yes
No
15612
Are there any more TransportCellLists in the incoming list?
Yes
No
15614
Call Install on the Handler Manager

FIG. 15A

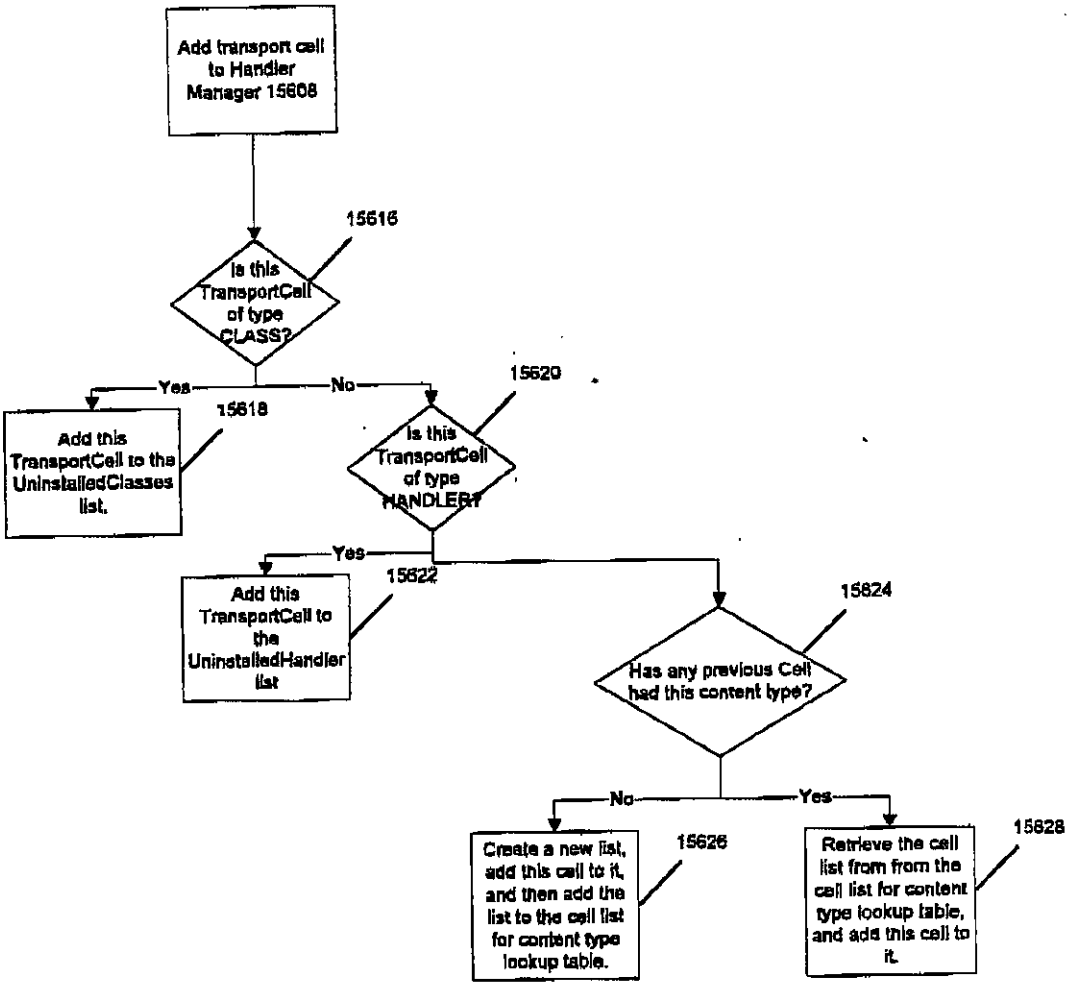

Add transport cell to Handler Manager 15608
15616
Is this TransportCell of type CLASS?
Yes
No
15618
Add this TransportCell to the UninstalledClasses list.
15620
Is this TransportCell of type HANDLER?
Yes
15622
Add this TransportCell to the UninstalledHandler list
15624
Has any previous Cell had this content type?
No
Yes
Create a new list, add this cell to it, and then add the list to the cell list for content type lookup table.
15626
Retrieve the cell list from from the cell list for content type lookup table, and add this cell to it.
15628

FIG. 15B

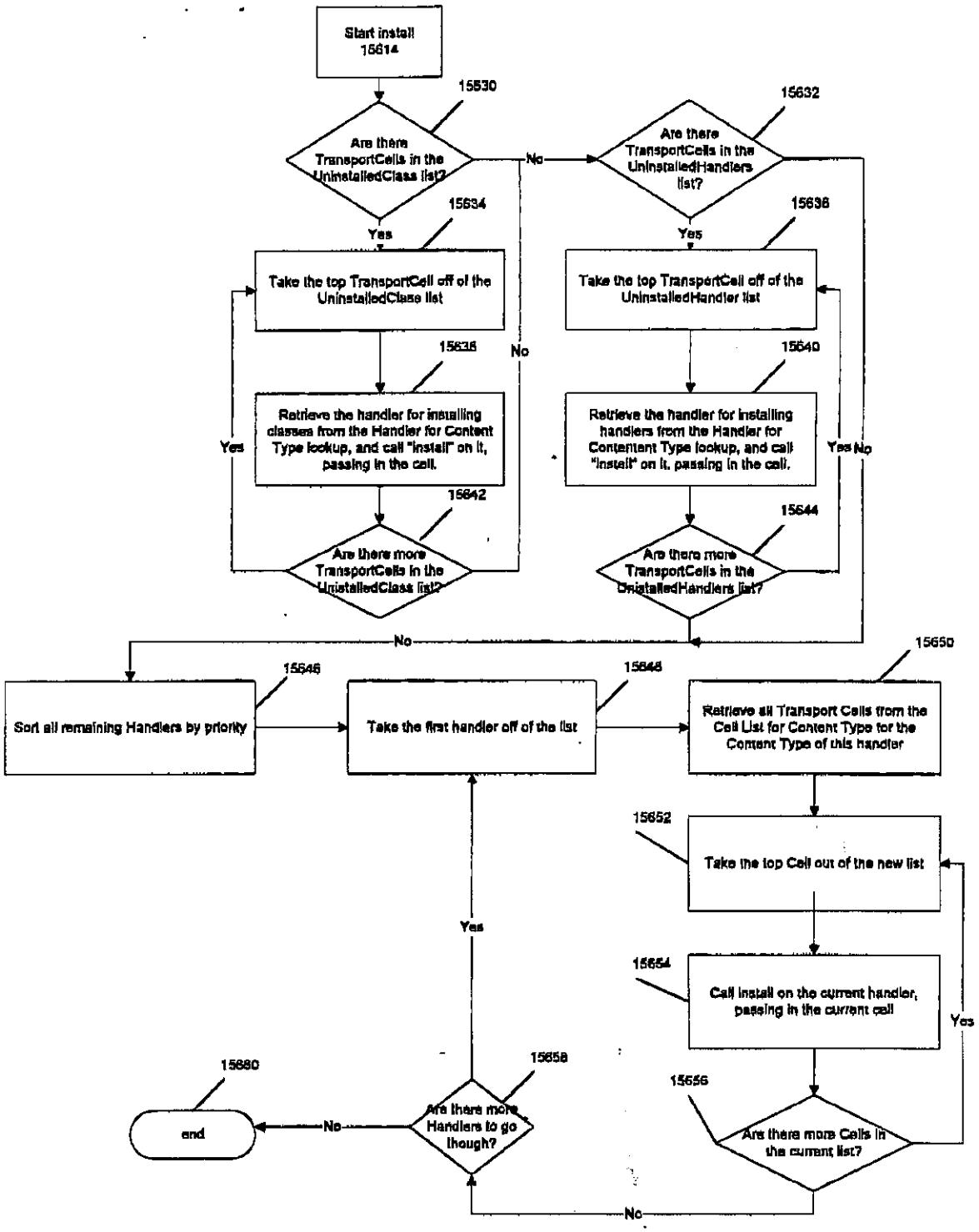

Start install
15614
15630
Are there TransportCells in the UninstalledClass list?
No
15632
Are there TransportCells in the UninstalledHandlers list?
Yes
15634
Take the top TransportCell off of the UninstalledClass list
15636
Retrieve the handler for installing classes from the Handler for Content Type lookup, and call "install" on it, passing in the cell.
15642
Are there more TransportCells in the UninstalledClass list?
Yes
No
Yes
15638
Take the top TransportCell off of the UninstalledHandler list
15640
Retrieve the handler for installing handlers from the Handler for Contentent Type lookup, and call "install" on it, passing in the cell.
15644
Are there more TransportCells in the UninstalledHandlers list?
Yes No
No
15646
Sort all remaining Handlers by priority
15648
Take the first handler off of the list
15650
Retrieve all Transport Cells from the Cell List for Content Type for the Content Type of this handler
15652
Take the top Cell out of the new list
15654
Call install on the current handler, passing in the current cell
15656
Are there more Cells in the current list?
Yes
No
Yes
15658
Are there more Handlers to go though?
No
15660
end

FIG. 15C

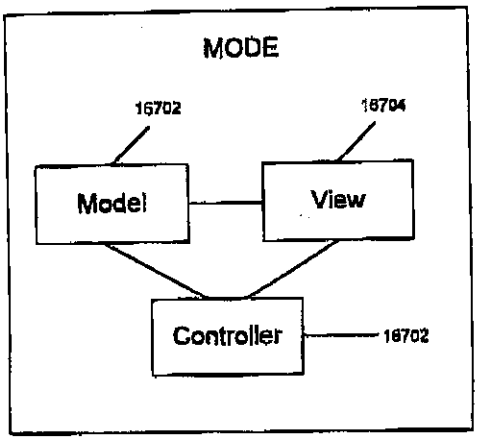


FIG. 16A

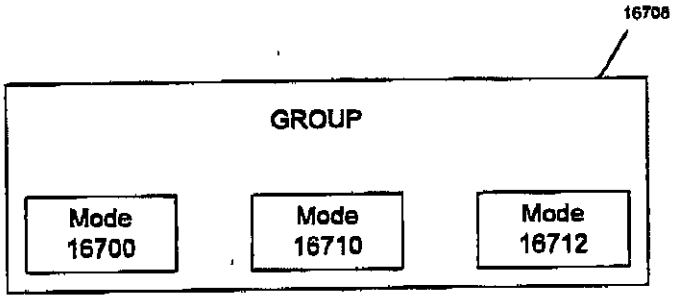


FIG. 16B

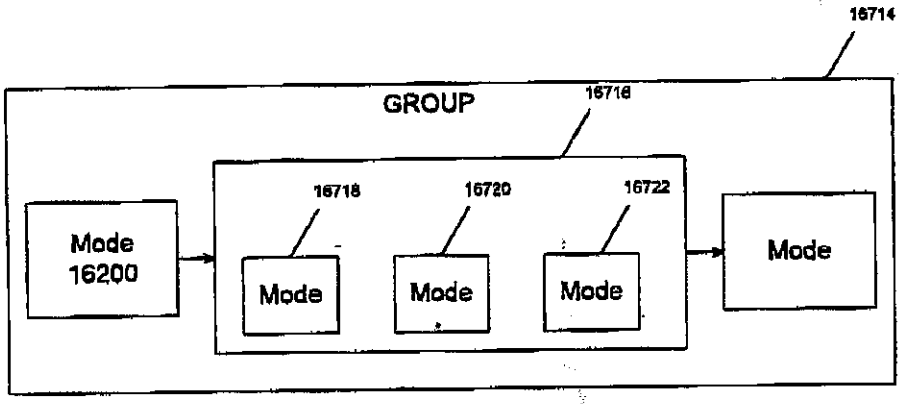


FIG. 16C

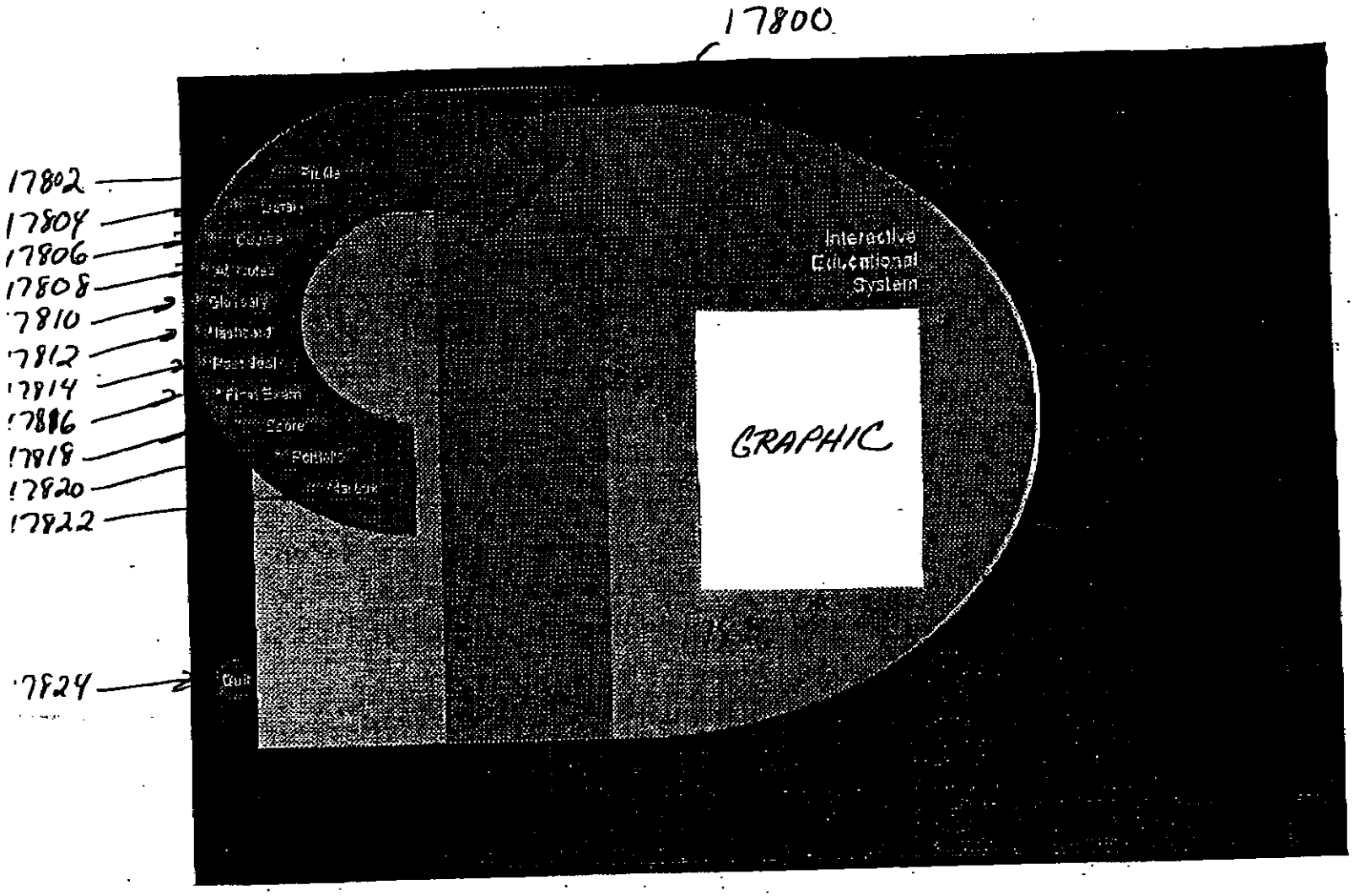


FIG. 17A

07 JUN 2004

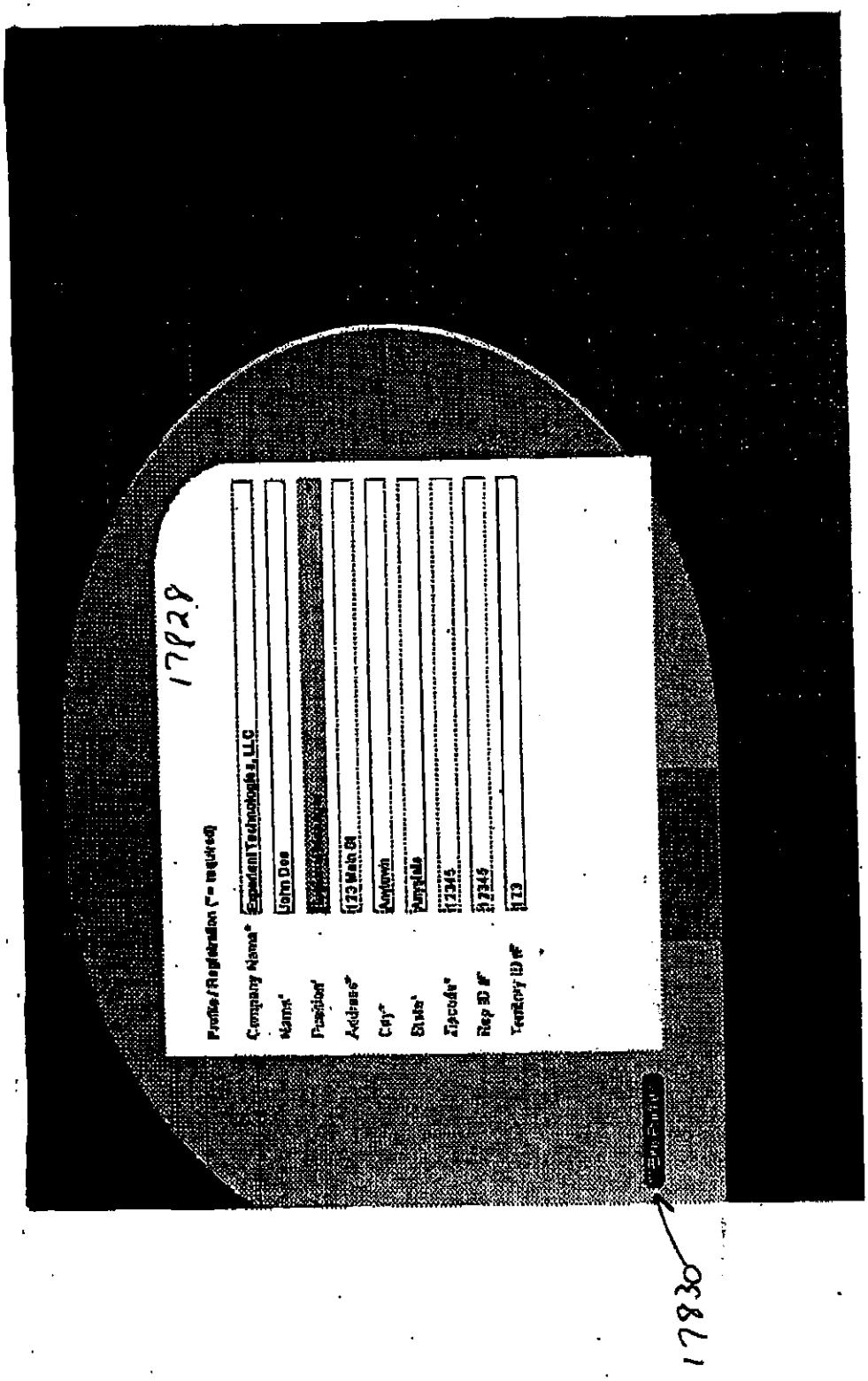

17828
Profile/Registration (* required)
Company Name*
Expedient Technologies, LLC
Name*
John Doe
Position*
Address*
123 Main St
City*
Anytown
State*
Anystate
Zipcode*
12345
Rep ID #*
12345
Territory ID #*
123
17830

FIG. 17B

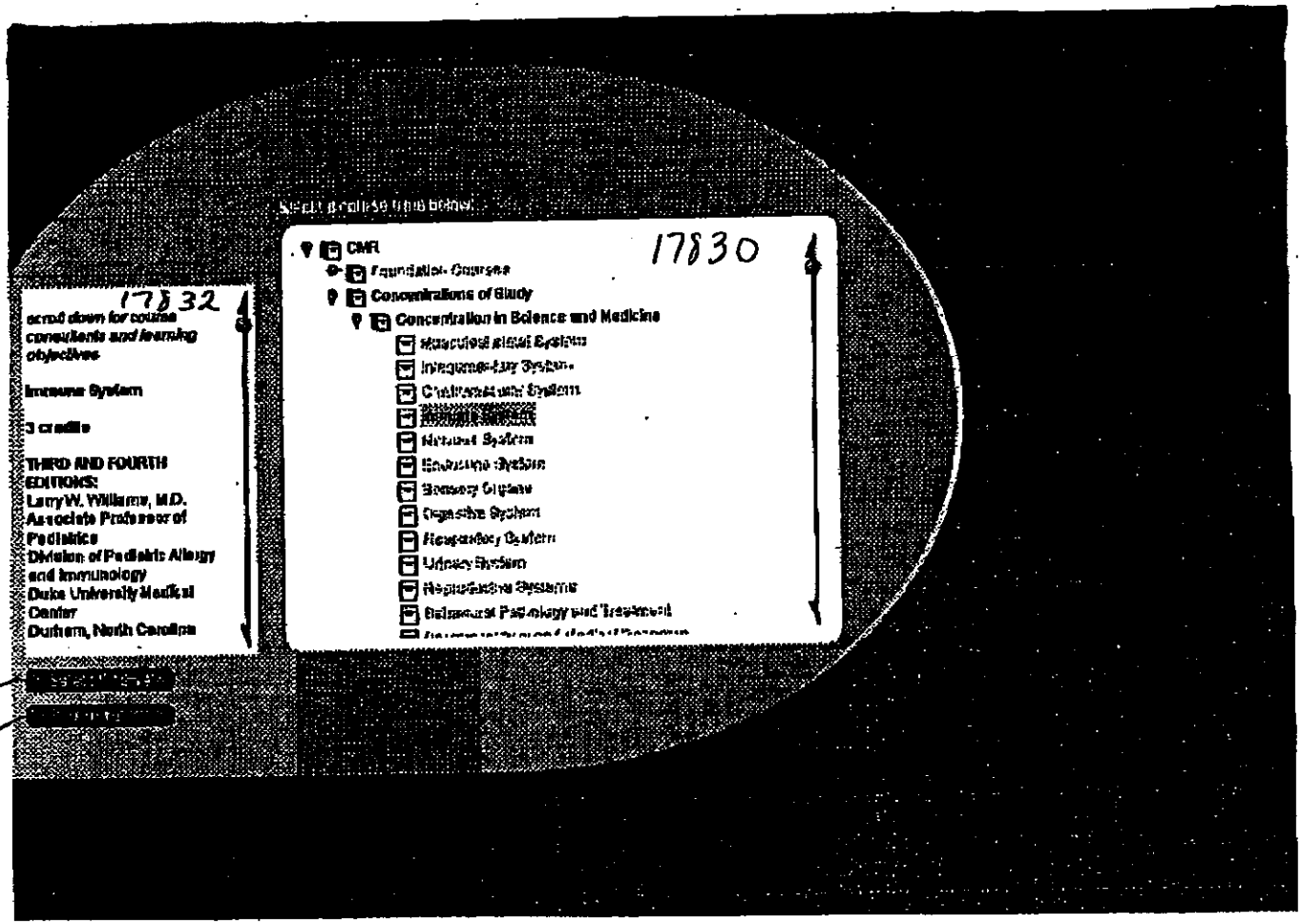


FIG. 17C

07 JUN 2004

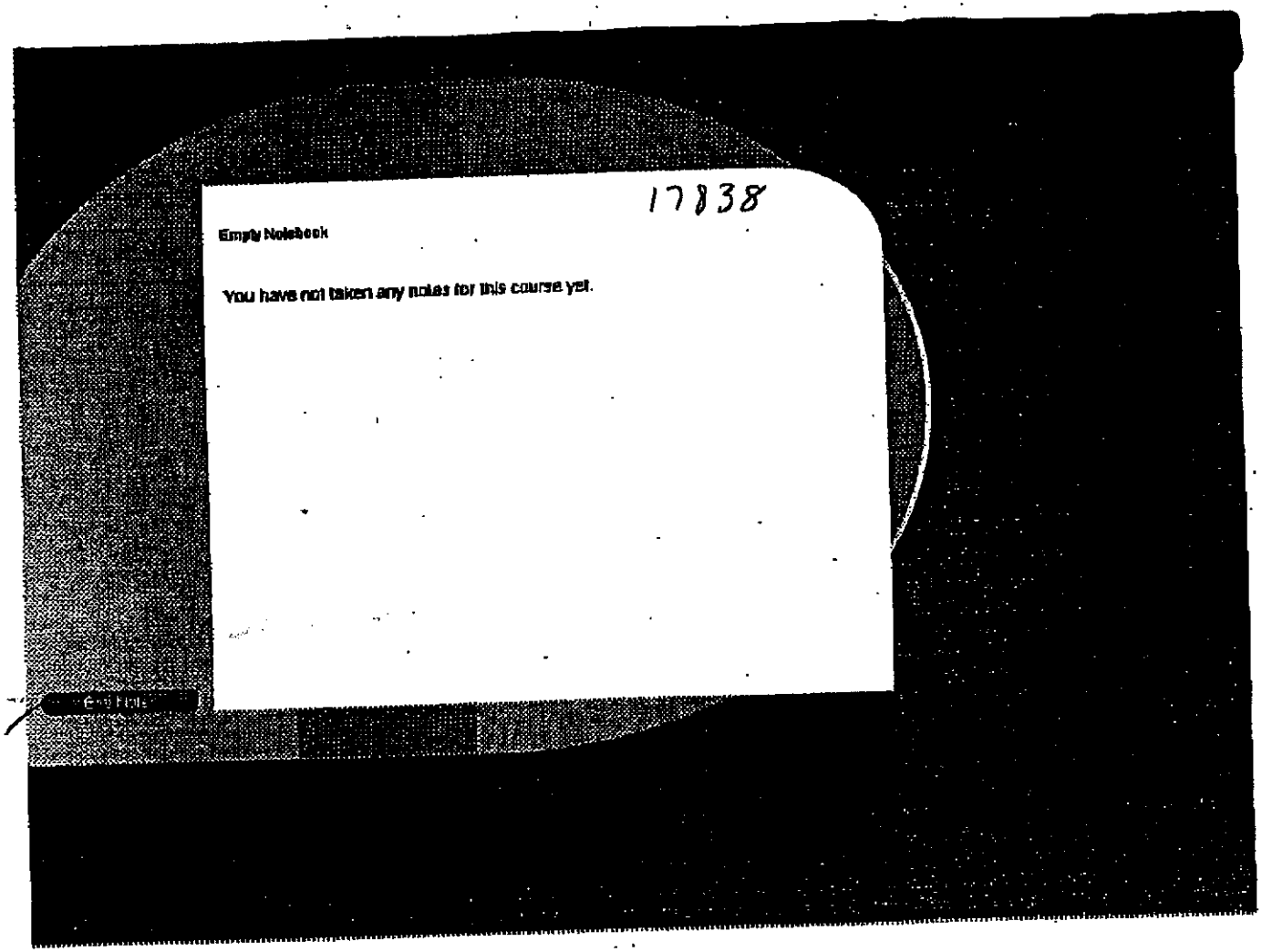


FIG. 17D

07 JUN 2004

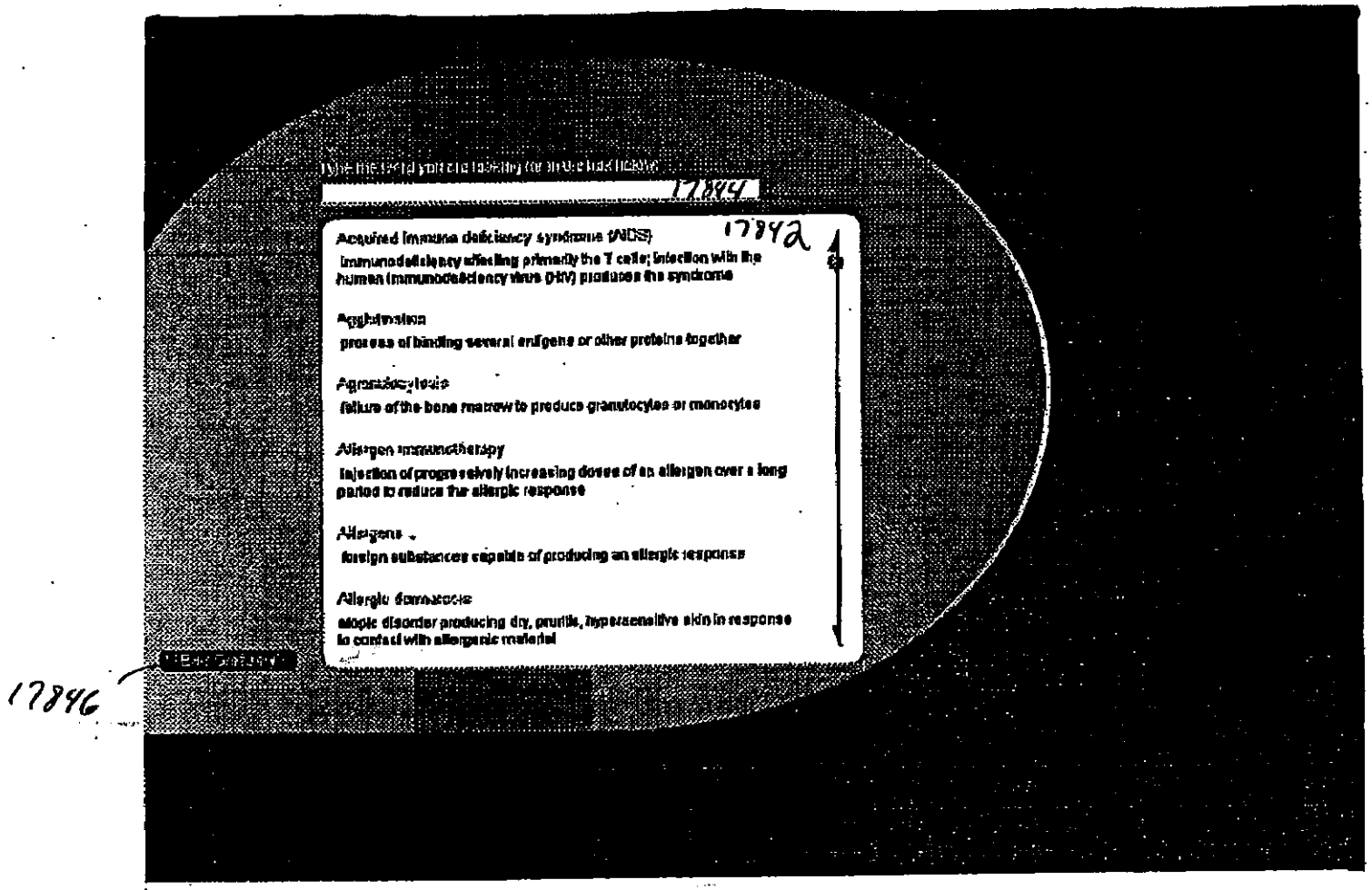


FIG. 17E

07 JUN 2004

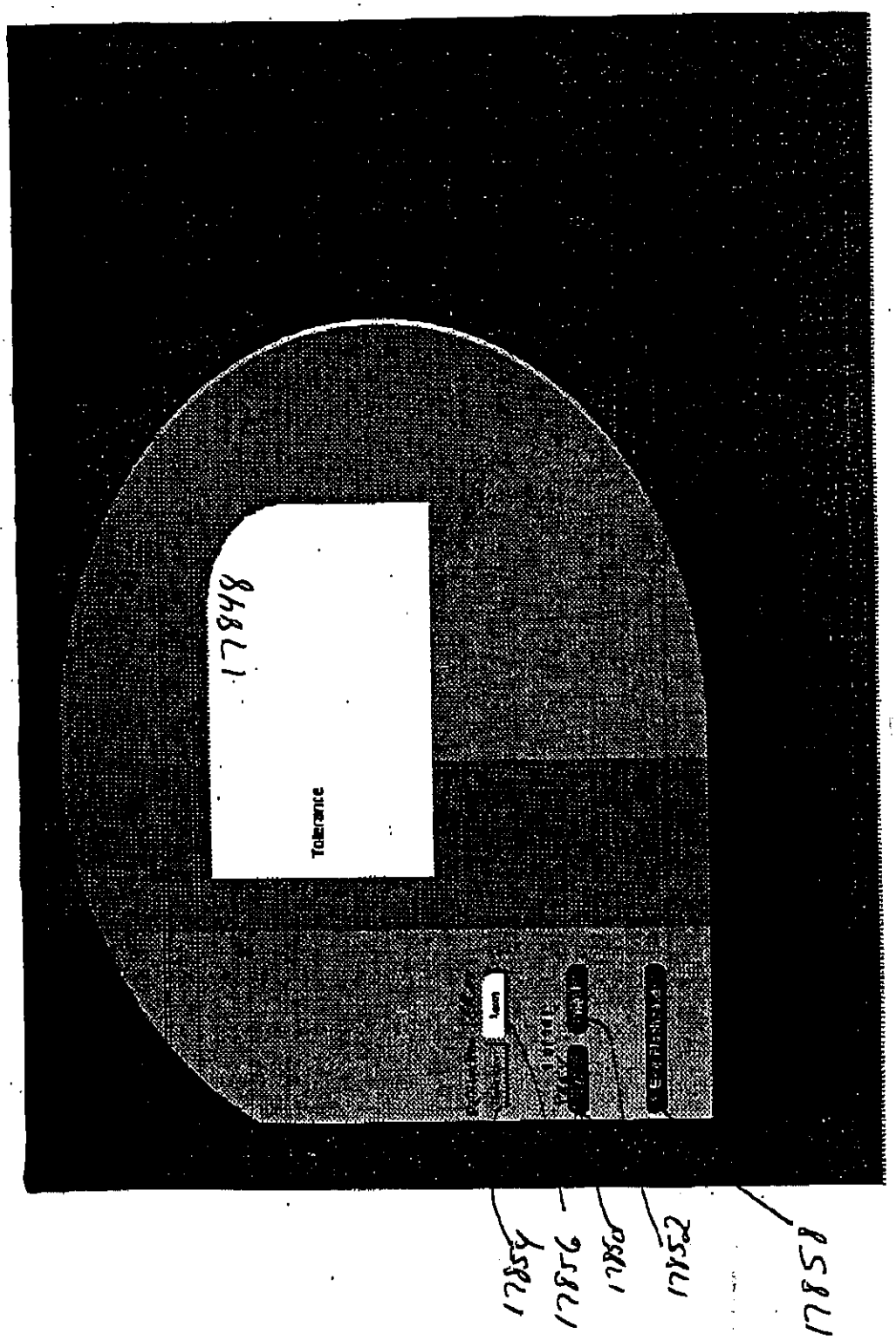


FIG. 17F

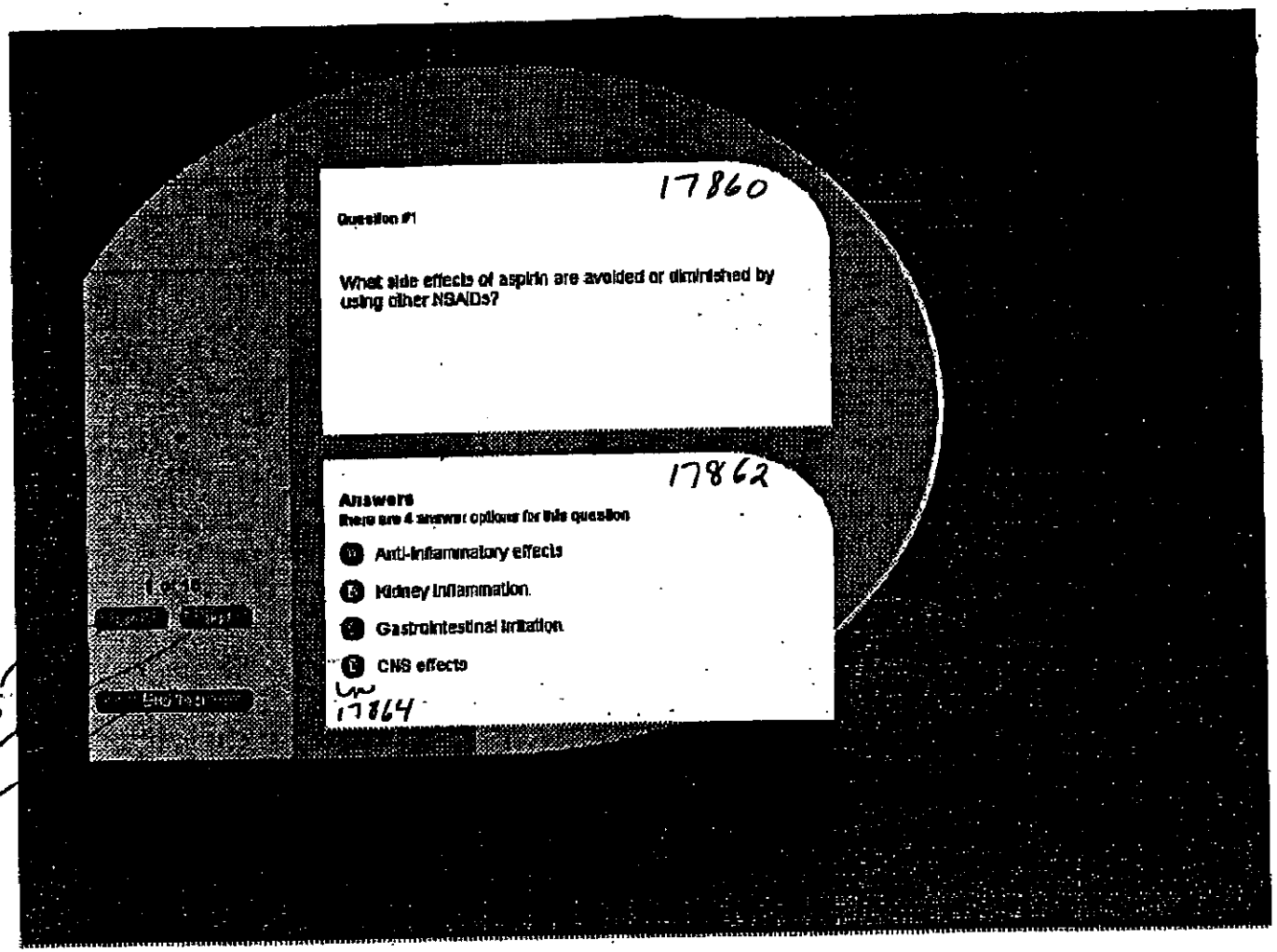


FIG. 17G

07 JUN 2004

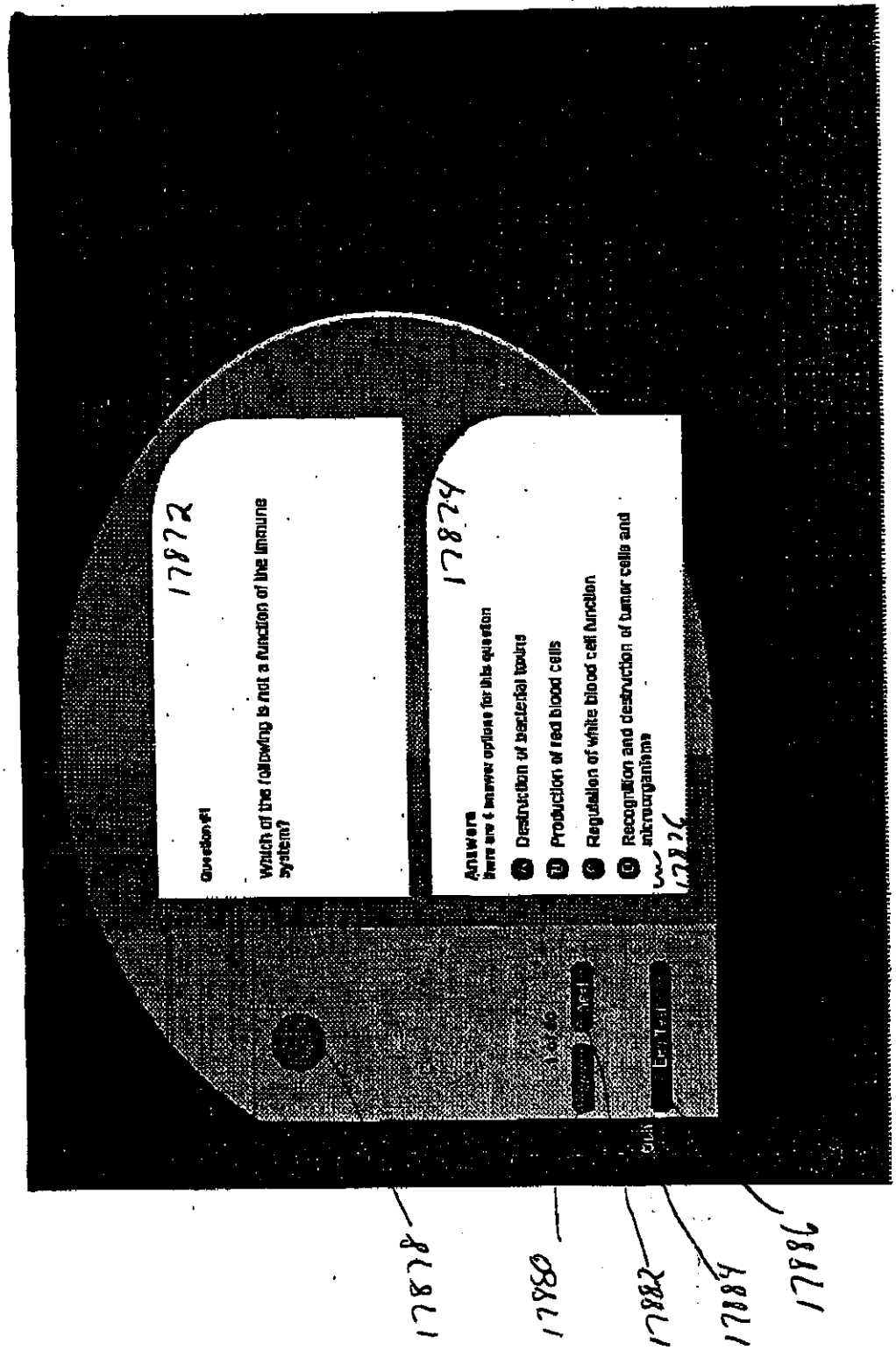

17872
Question #1
Which of the following is not a function of the immune system?
17874
Answers
There are 4 answer options for this question
A Destruction of bacterial toxins
B Production of red blood cells
C Regulation of white blood cell function
D Recognition and destruction of tumor cells and microorganisms
17876
17878
17880
17882
17884
17886

FIG. 17H

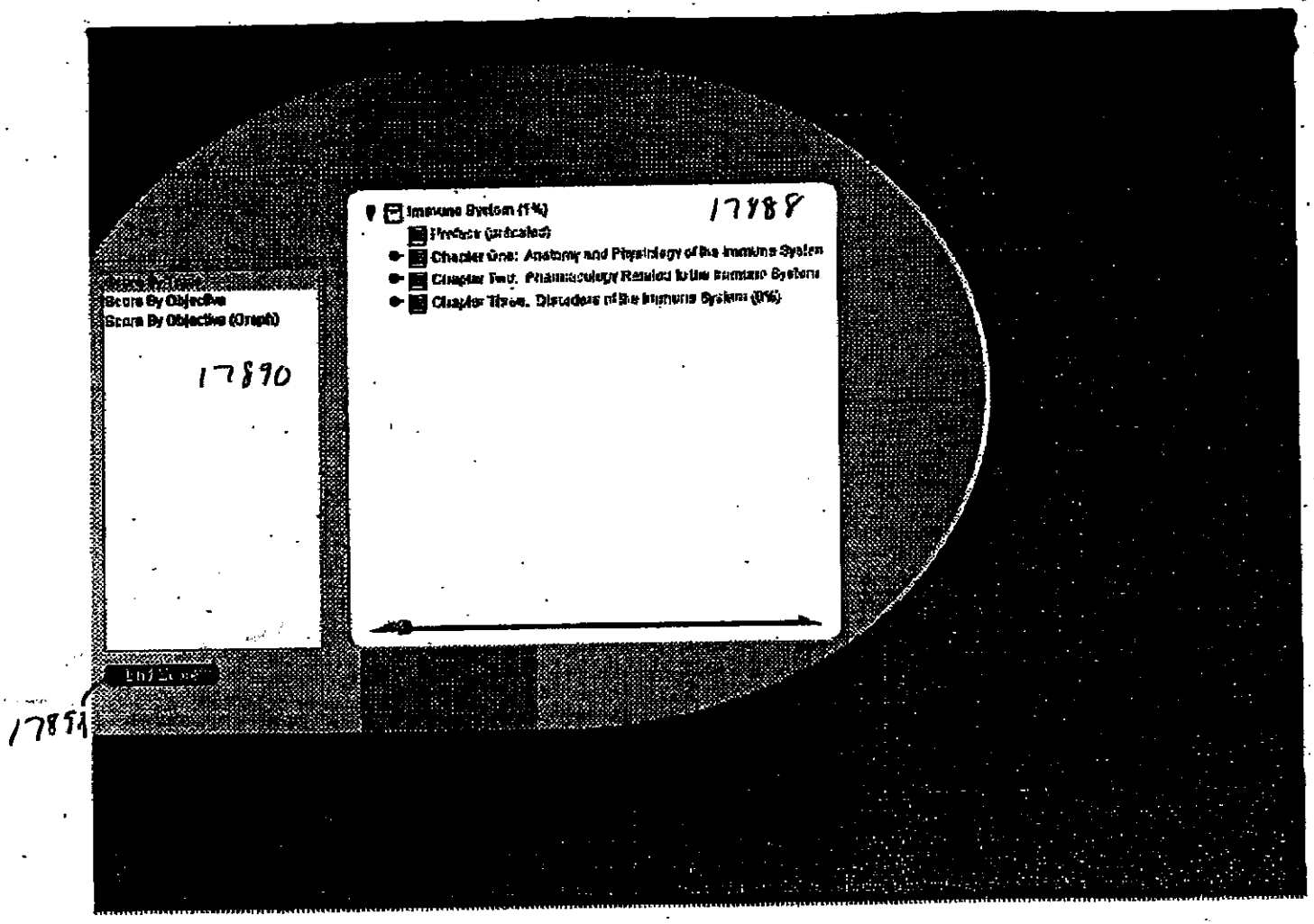


FIG. 17I

07 JUN 2004

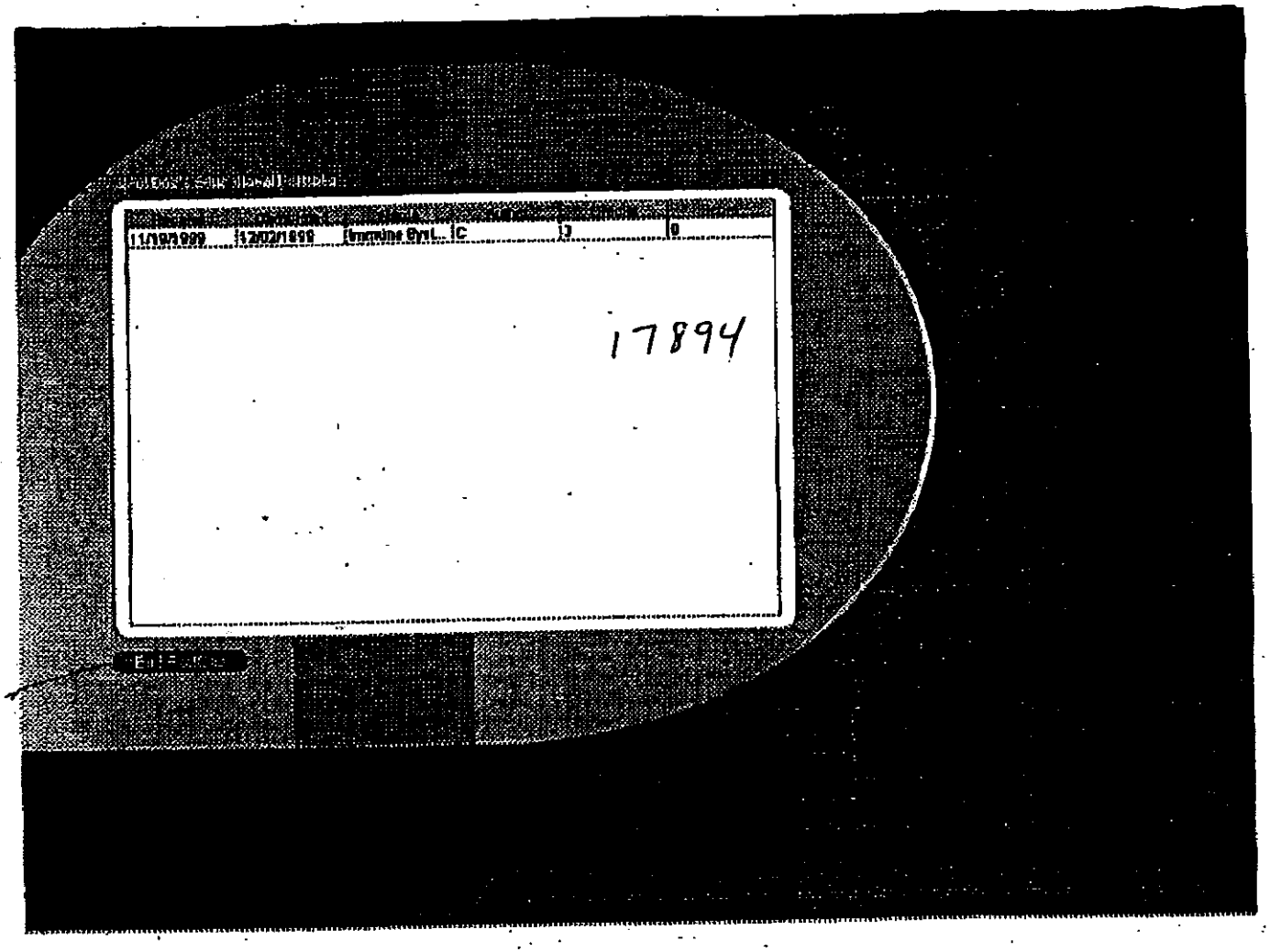


FIG 175

07 JUN 2004

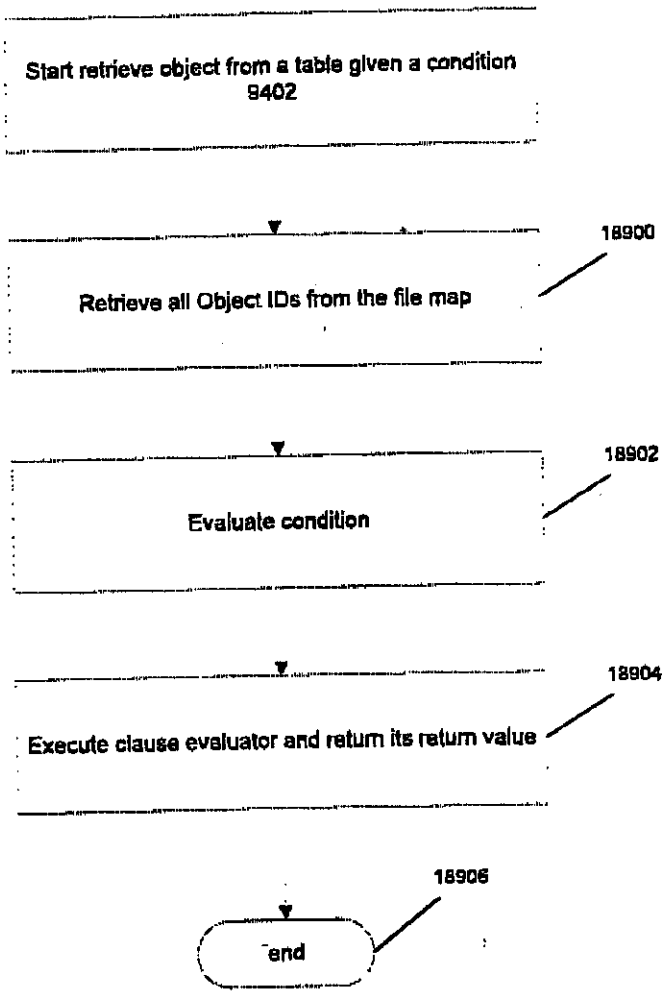

Start retrieve object from a table given a condition
9402
18900
Retrieve all Object IDs from the file map
18902
Evaluate condition
18904
Execute clause evaluator and return its return value
18906
end

FIG. 18A

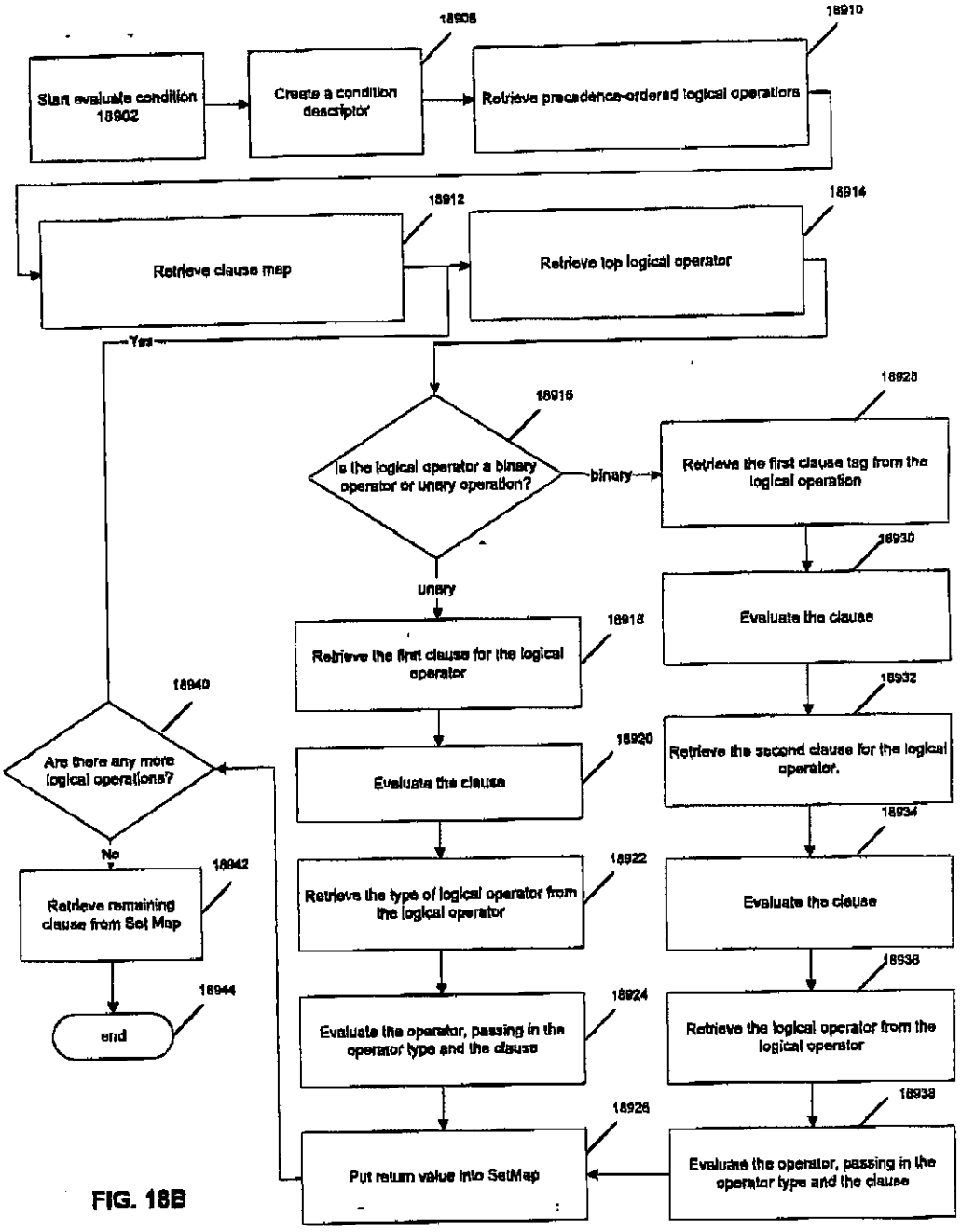

Start evaluate condition 18902
Create a condition descriptor
18908
Retrieve precedence-ordered logical operators
18910
Retrieve clause map
18912
Retrieve top logical operator
18914
Yes
Is the logical operator a binary operator or unary operation?
18916
binary
unary
Retrieve the first clause tag from the logical operation
18928
Evaluate the clause
18930
Retrieve the second clause for the logical operator.
18932
Evaluate the clause
18934
Retrieve the logical operator from the logical operator
18936
Evaluate the operator, passing in the operator type and the clause
18938
Retrieve the first clause for the logical operator
18918
Evaluate the clause
18920
Retrieve the type of logical operator from the logical operator
18922
Evaluate the operator, passing in the operator type and the clause
18924
Put return value into SetMap
18926
Are there any more logical operations?
18940
No
Retrieve remaining clause from Set Map
18942
end
18944

FIG. 18B

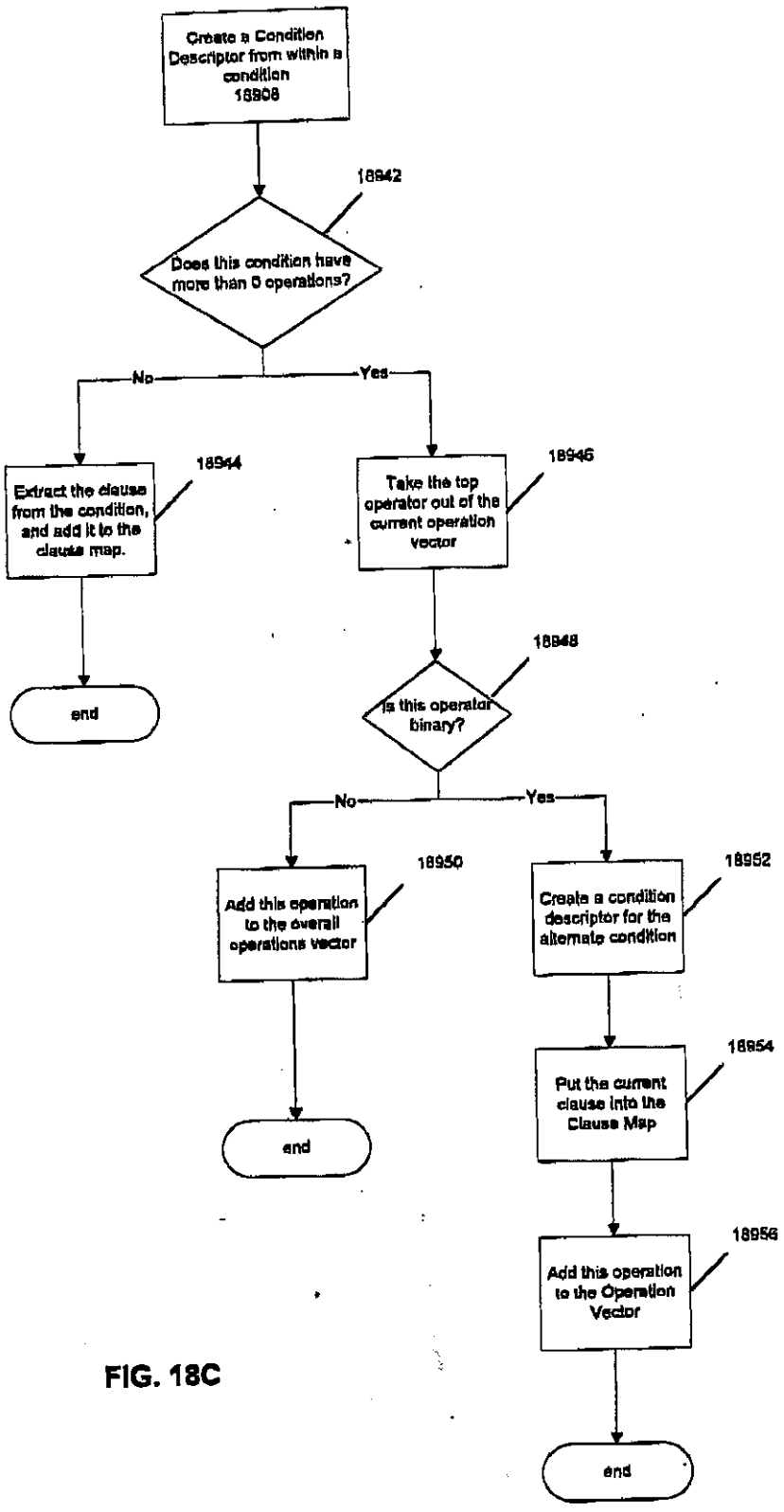

Create a Condition Descriptor from within a condition
18938
18942
Does this condition have more than 0 operations?
No
Yes
Extract the clause from the condition, and add it to the clause map.
18944
Take the top operator out of the current operation vector
18946
end
18948
Is this operator binary?
No
Yes
Add this operation to the overall operations vector
18950
Create a condition descriptor for the alternate condition
18952
end
Put the current clause into the Clause Map
18954
Add this operation to the Operation Vector
18956
end

FIG. 18C

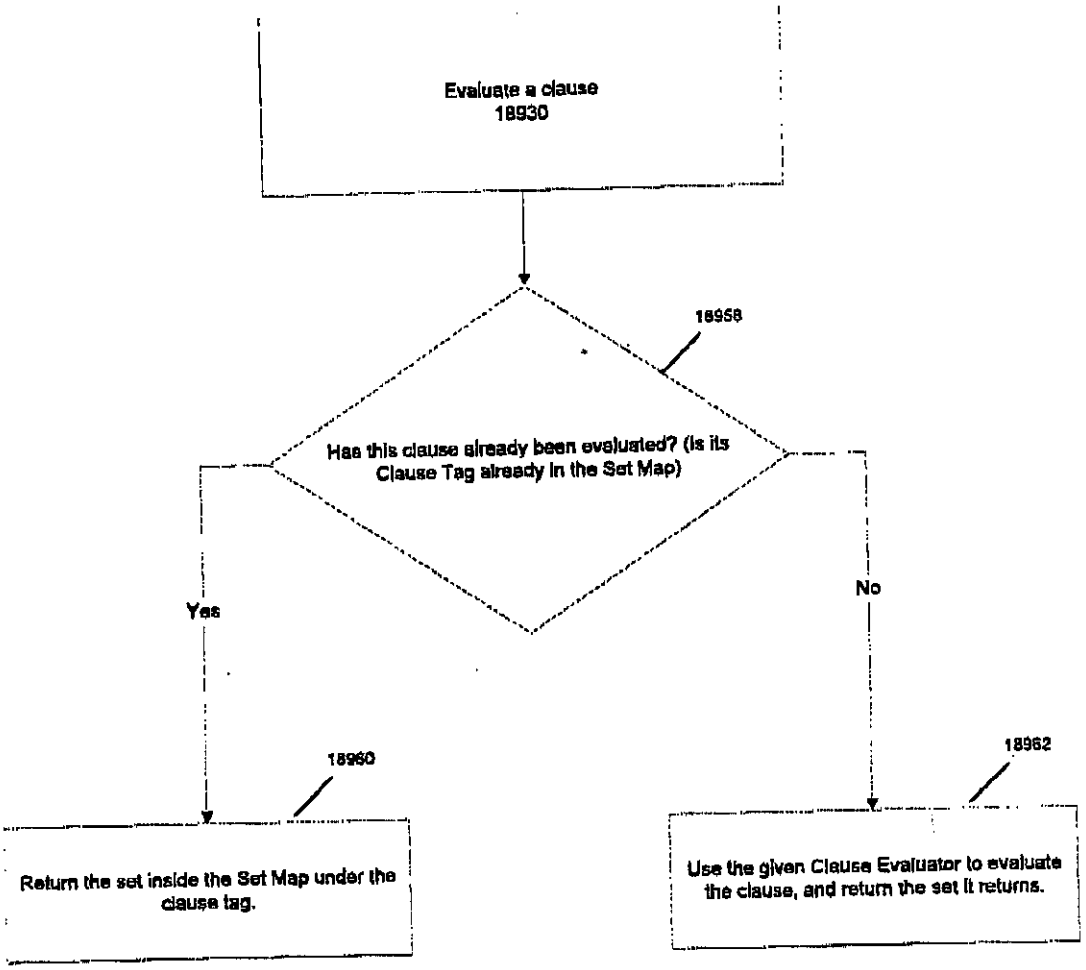

Evaluate a clause
18930
18958
Has this clause already been evaluated? (Is its Clause Tag already in the Set Map)
Yes
No
18960
Return the set inside the Set Map under the clause tag.
18962
Use the given Clause Evaluator to evaluate the clause, and return the set it returns.

FIG. 18D

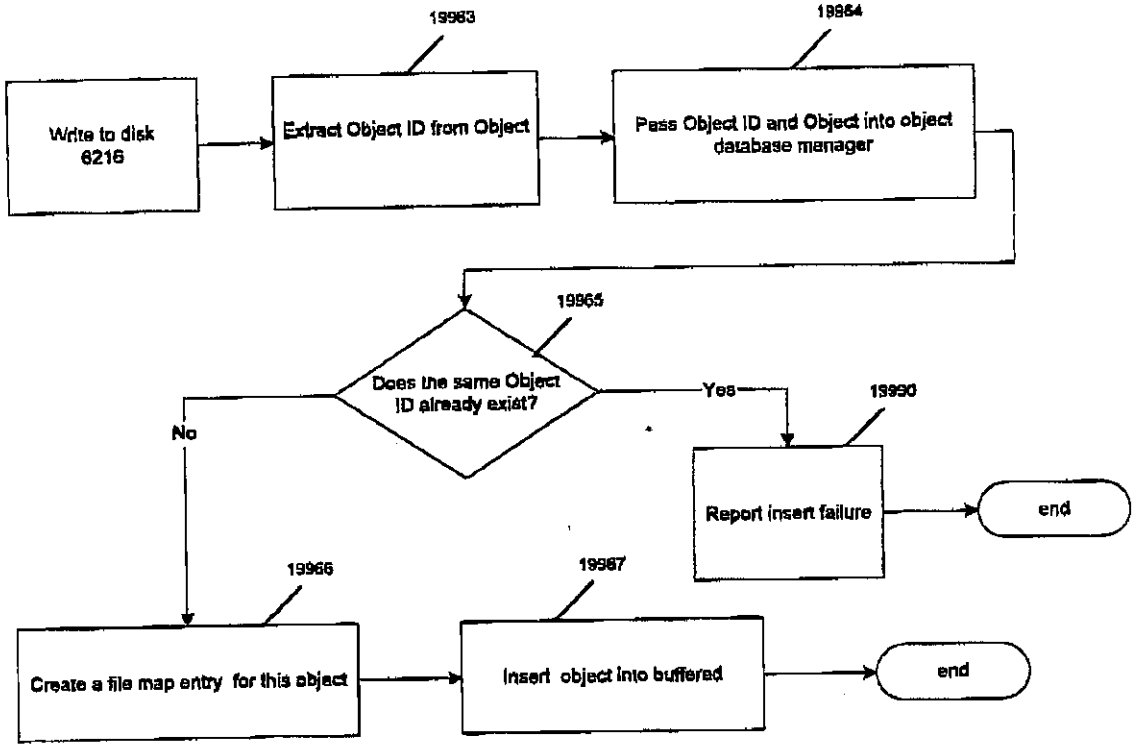

19963
19964
Write to disk 6216
Extract Object ID from Object
Pass Object ID and Object into object database manager
19965
Does the same Object ID already exist?
Yes
No
19990
Report insert failure
end
19966
19967
Create a file map entry for this object
Insert object into buffered
end

FIG. 19A

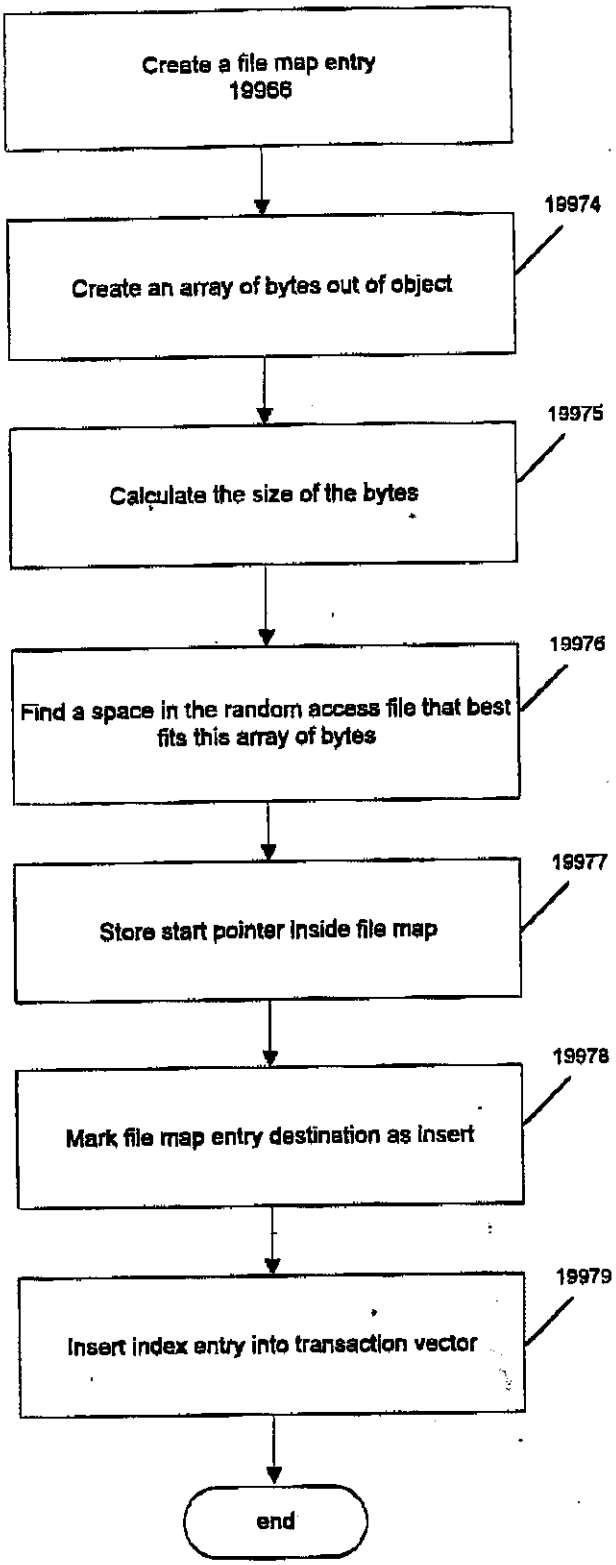

Create a file map entry
19966
19974
Create an array of bytes out of object
19975
Calculate the size of the bytes
19976
Find a space in the random access file that best fits this array of bytes
19977
Store start pointer inside file map
19978
Mark file map entry destination as insert
19979
Insert index entry into transaction vector
end

FIG. 19B

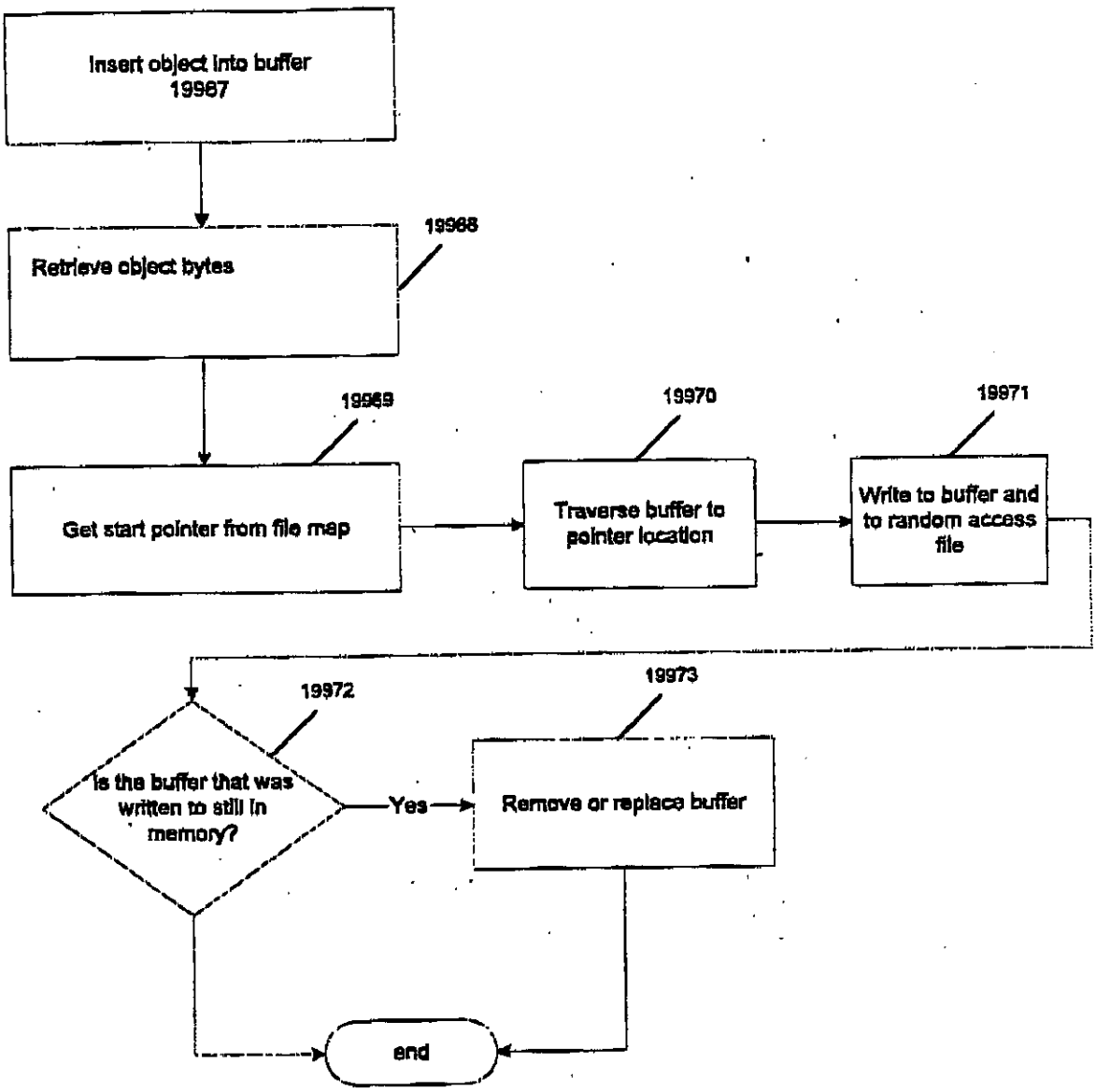

Insert object into buffer
19967
Retrieve object bytes
19968
Get start pointer from file map
19969
Traverse buffer to pointer location
19970
Write to buffer and to random access file
19971
Is the buffer that was written to still in memory?
19972
Yes
Remove or replace buffer
19973
end

FIG. 19C

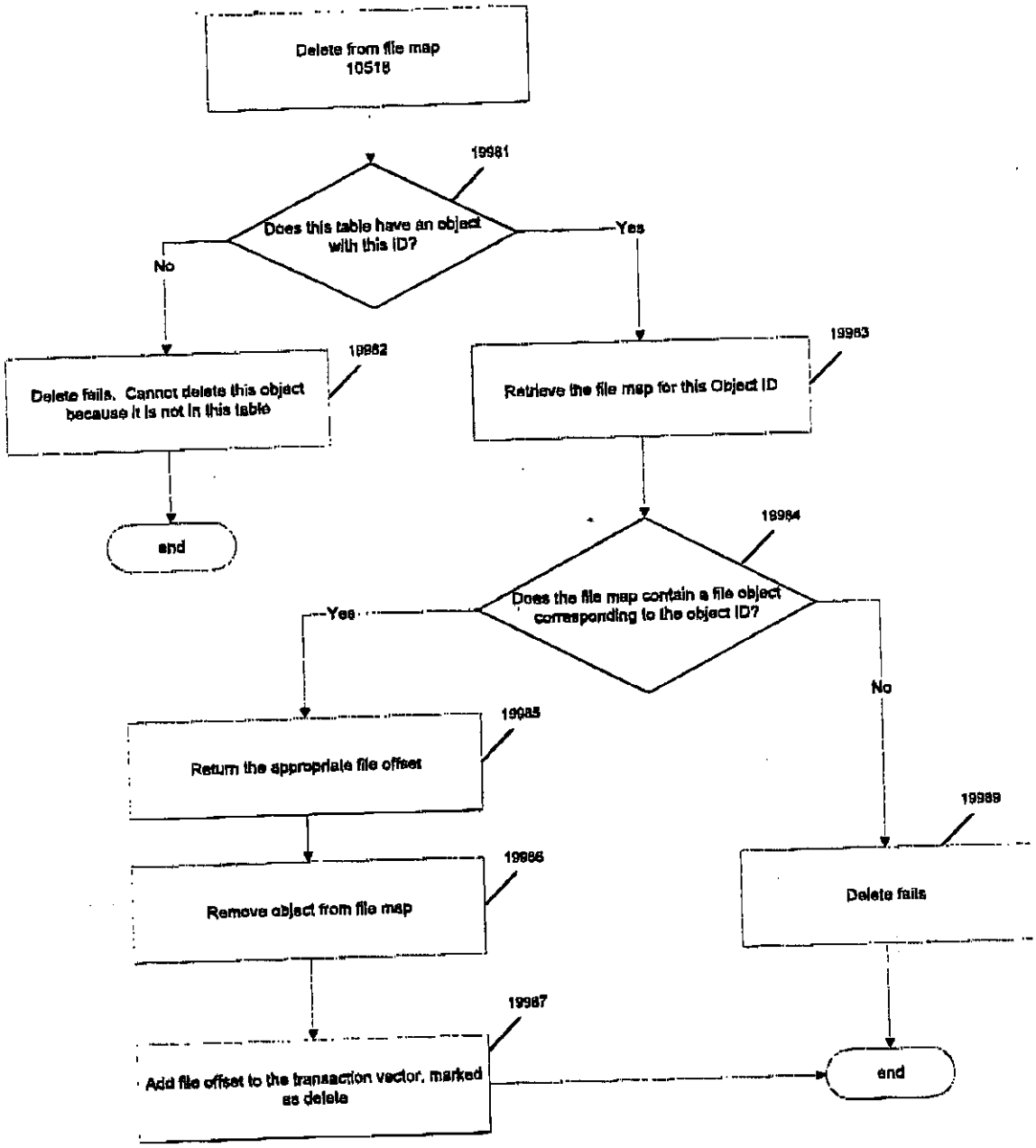

Delete from file map
10518
19981
Does this table have an object with this ID?
No
Yes
19982
Delete fails. Cannot delete this object because it is not in this table
end
19983
Retrieve the file map for this Object ID
19984
Does the file map contain a file object corresponding to the object ID?
Yes
No
19985
Return the appropriate file offset
19986
Remove object from file map
19987
Add file offset to the transaction vector, marked as delete
19989
Delete fails
end

FIG. 19D