Abstract Title: A system for retrieving data from a partially indexed data store

There is disclosed a method, system and computer program for retrieving a data item from a data store in a data processing system. The data processing system comprises an index 705, which at least partially indexes the data store, and also a partiality definition store 720. The partiality definition store 720 indicates a part of the data store which is referenced by the partial index. A request is received for a data item from the data store and responsive to determining that the requested data item is not referenced by the partial index 705, the data store is searched by using the partiality definition store 720 to avoid searching a part of the data store which is known to be indexed.

Figure 6

At least one drawing originally filed was informal and the print reproduced here is taken from a later filed formal copy.

This print takes account of replacement documents submitted after the date of filing to enable the application to comply with the formal requirements of the Patents Rules 1995
Figure 1
Receive a request for a data record

Examine list 535 & the partiality definition store 520

Is the next page in list 535 indexed

Use index for next group of indexed pages

Does required record exist in page

Scan page itself

Return requested data record

Figure 5b
FIELD OF THE INVENTION

The present invention relates to data retrieval and more particularly to data retrieval using an index.

BACKGROUND OF THE INVENTION

Data retrieval may be facilitated via the use of an index which is typically based on one or more keys defining search criteria which can be used to access stored data. Such an index may, for example, chart the location of data stored within records in a database system or messages held by a messaging system.

Some systems build a full index upfront of all the data stored and add to this as and when changes are made to the data. The problem with this is that if there is a large amount of data stored, then the index is also likely to be costly in terms of storage. If non-volatile storage is used this can get expensive. On the other hand, if the index is maintained in volatile memory any system crash results in a slowed restart time due to the index having to be rebuilt at system startup.

The building of partial indexes dynamically is also known - particularly in the database environment. See for example "Partial Indices" at http://kuva.vsml.org/docs/pgsql/postgres/partial-index.html; "The Case for Partial Indexes" by Michael Stonebraker of the Department of Electrical Engineering and Computer Sciences University of California, Berkley, CA 94720; http://www.zaval.org/products/file-search/user_guide/#usage; "Operational Data Storage Unification" by James Hamilton of the Department of Computer Science University of Waterloo, Waterloo, Ontario, Canada NZL 3G1 (http://www.cs.uwaterloo.ca/cs-archive/CS-1997/16/intsto-1.pdf); "Designing the B-1. A universal system for information" by MJ Carey et al of the University of Berkeley (http://db.cs.berkeley.edu/~jmh/b1-vldb-final.pdf); and US 6345,266.

In systems which build an index dynamically, the index is typically built based on the data that a user retrieves. In this way storage space is not wasted and the index should be tailored towards the most common user requests.

The disadvantage of having a partial index is that because the partial index does not contain a reference to all of the data stored, it is possible that a requested data item will not be found in the index. In
such a situation, the partial index is of no use in finding the requested
data and it is therefore necessary to search the actual data. This can be
time consuming.

SUMMARY OF THE INVENTION

Accordingly the invention provides a method for retrieving a data
item from a data store in a data processing system comprising an index, which at least partially indexes the data
store, and a partiality definition store, the partiality definition store
indicating a part of the data store which is referenced by the partial
index, the method comprising the steps of: receiving a request for a data
item from the data store; and responsive to determining that the requested
data item is not referenced by the partial index, searching the data store
by using the partiality definition store to avoid searching a part of the
data store which is known to be indexed.

In this way it is not necessary to search from the beginning of the
data store for a non-indexed data item and much time and effort is saved.

By way of example, the data processing system may be a messaging
system having one or more queues of messages. Alternatively the data
processing system may be a database system.

In, for example, a database embodiment the partiality definition
store may hold a list of indexed pages. Alternatively the partiality
definition store may hold a list of non-indexed pages. In either case the
partiality definition store provides an indication of the part of the data
store which is referenced by partial index.

Note, the term "referenced" is intended to encompass the situation in
which the partial index actually holds the indexed data items, but also the
situation where the partial index only points to the location of indexed
data items in the data store.

Preferably whilst searching for the requested data item, searched
data items are added to the partial index. In this way the index is
dynamically built and thus is tailored to user queries.

In one preferred embodiment a first indication (for example a
pointer) of the last indexed data item is maintained in the partiality
definition store.
In one preferred embodiment data items are searched which are subsequent to the last data item indicated by the first indication.

In one preferred embodiment the data store has an end part and data items are added to the partial index as they are inserted into the end part. Preferably a second indication (for example a pointer) is maintained in the partiality definition store of the first inserted data item which is referenced in the partial index.

Preferably in this embodiment only data items present between the first and second indications are searched.

In one preferred embodiment the partial index is segmented: a first segment comprising indexed data items prior to the first indication; and the second segment including data items post the second indication. Preferably in this embodiment, responsive to determining that the next data item of a particular type is requested and that the data item is not referenced by the first segment, the data items in the data store between the first and second indications are searched before the second index segment is searched. For example, the user may want to return the first, second and third data item of type "purple" and in that specific order.

In one preferred embodiment, responsive to determining that the data store has fewer than a predetermined number of data items therein, deleting both the partiality definition store and the partial index. In certain environments the data store may grow and shrink relatively rapidly (e.g. a messaging environment). If the data store is small, there may be little point in maintaining an index.

In one preferred embodiment data items are records held on pages in a database system and wherein the partiality definition store comprises a list of indexed pages. In this embodiment a list of all pages stored is used to search pages in the data store, whilst the partiality definition store is used to avoid searching those pages which are known to be indexed.

Alternatively the partiality definition store may comprise a list of non-indexed pages and the partiality definition store is used to deduce those pages that are indexed in order to avoid searching such pages.

In one preferred embodiment the partial index is segmented, each segment comprising an index to at least one sequential page. In this embodiment, responsive to determining that the next data item (according to a specific order) of a particular type is requested, the searching of index segments is interleaved with searching of one or more pages. For example,
the user may require the first record (in a database system) referring to a user with "blue eyes".

According to one preferred embodiment a page is retrieved based on a first indexed attribute and records held by the retrieved data page are indexed according to another attribute.

According to another aspect, the invention provides a system for retrieving a data item from a data store, the system comprising: a data store; an index, which at least partially indexes the data store; a partiality definition store, the partiality definition store indicating a part of the data store which is referenced by the partial index; means for receiving a request for a data item from the data store; and means, responsive to determining that the requested data item is not referenced by the partial index, for searching the data store by using the partiality definition store to avoid searching a part of the data store which is known to be indexed.

It will be appreciated that the invention may be implemented in computer software.

BRIEF DESCRIPTION OF THE DRAWINGS

Preferred embodiments of the present invention will now be described, by way of example only, and with reference to the following drawings in which:

Figure 1 illustrates a typical messaging system in accordance with the prior art;

Figure 2 shows the processing of the present invention in accordance with a first embodiment;

Figure 3 illustrates the components involved in the first embodiment; and

Figure 4 illustrates an optimisation of the first embodiment;

Figures 5a and 5b illustrate an optimisation of a second embodiment when ordering is a requirement; and

Figure 6 illustrates an optimisation of the second embodiment when ordering is not a requirement.
DETAILED DESCRIPTION

First Preferred Embodiment

Messaging systems such as IBM’s WebSphere(R) MQ family of products allow keyed retrieval of messages. Such systems will typically allow an administrator to specify index requirements such as which queue(s) to index and which attribute (e.g. message identifier or correlation identifier) to index on.

If no index requirements have been specified then keyed retrieval is achieved by sequentially searching the queue for a message with matching attribute value(s). Use of an index however greatly improves performance by converting the key to the location(s) of message(s) with matching key. In this way the need for a sequential search is dispensed with and instead the message can be found directly via the index.

According to a preferred embodiment, such an index is built dynamically. Figure 1 illustrates a typical messaging system in accordance with the prior art and should be read in conjunction with figure 2 which shows the processing steps by which such an index is created in accordance with a preferred embodiment of the present invention.

With reference to figure 1, messaging system 10 executes programs 20 and 30. Programs 20 and 30 communicate with a queue 40 running on queue manager 50 via a message queuing interface (MQI) 60. For example, program 20 may wish to put a message 70 to queue 40 for retrieval by program 30. Program 30 may use an index (not shown) to aid retrieval of such a message from the queue 40.

With reference to figure 2, program 30 makes a request for keyed retrieval of a message from queue 40 (step 100). For example, program 20 may issue an MQGET on message identifier x. At step 110, it is determined whether an index exists for the specified attribute (e.g. message identifier). Note, in the initial state (including the state immediately following restart after queue manager failure) there will be no index for the queue.

If an appropriate index does not exist, then an empty index is created at step 130. If the index does exist, then it is searched at step 200 for the specified attribute value. If the attribute is found in the index, then this is used to locate the corresponding message and to process the message as required (steps 210, 190).
On the other hand, if the message is not found in the index, then the queue itself is searched sequentially (step 140). This is also the case, if no index existed and one had to be created at step 130.

In searching the queue, each message is examined (browsed) in turn in order to locate the requested message. If the particular message being examined is the requested message, then it is processed as required at step 190. If the particular message being examined is not the requested message, then the message is added to the index (step 160). Note, optionally even if the message examined is the requested message it too may be added to the index (this is useful in situations where the requested message is not removed from the queue - e.g. the request was a browse request).

This process loops round until either all messages on the queue have been examined without success (steps 170, 180) or the requested message has been found and processed (steps 150, 190, 180).

Note, every time a message is added to the index, this entry in the index is remembered as the last indexed position. In this way if a message is not found in the index, it is not necessary to start searching the queue from the beginning. Instead the queue can be searched from the last indexed position. This is because the index is built from the head of the queue, whilst messages are added to the tail of the queue. If a requested message is not in the index, then it does not exist prior to the last indexed position.

Figure 3 exemplifies this process. A queue 300 has messages A, B, D, F and C on it. The queue has previously been scanned and messages A, B and D added to an index 310. The index preferably includes each message's identifier and a pointer to the actual location of the particular message. There is also a pointer 330 to the last indexed position 320. Suppose now that a process wishes to retrieve message C. The index 310 is first examined to determine whether the message exists therein. Assuming the message is not found in the index, without the invention the queue would have to be searched sequentially from the beginning. This is time consuming and wasteful (especially if there are a large number of messages on the queue). Now the pointer 330 which is associated with the index 310 can be used to quickly retrieve messages not identified by the index. All messages prior to the last indexed position 320 will exist in the index. It has however already been determined that the required message does not exist in the index. Thus rather than searching the queue from the very beginning, the queue now need only be searched from last indexed position
320. Thus in this example, it is only necessary to search 2 messages (F and C) rather than 5 messages (A, B, D, F and C).

Note, it will be appreciated that in the messaging environment it is often inefficient to maintain an index for a queue since the queue will typically not contain enough messages. It is therefore preferable to verify the depth of the queue before creating an index and only creating one if the queue is deeper than a predetermined threshold. Also, the small overhead of index maintenance may become not worthwhile as a queue becomes small. For example if the message consumer has been offline for a time, the queue may build up and become worth indexing; but when the consumer returns the queue fairly quickly becomes small. At this point, the partial dynamic index should be dispensed with.

It will be appreciated that the main purpose of the pointer 330 is to distinguish between the first part of the queue (which is indexed) and the latter part of the queue (which is not). For consistency of terminology herein this pointer will be known as the partiality definition store - even though in this embodiment the 'store' holds only a single pointer.

Note, there is no discussion herein regarding the structure that should be used to implement the index. Any standard indexing technique may be used as appropriate. For example, the index may be held as a sequential summary of the records indexed; this requires index scanning, but such scanning is much cheaper than scan of the full message data. Alternatively, structures such as hashtables or b-trees may be used.

**First Embodiment Optimisation**

For some patterns of queue usage, additional benefit can be obtained if, in addition to building indexes from the front as just described, insert requests also build the index from the back as new messages arrive.

The two parts of the index can be kept separate (i.e. the part built as a result of scanning the index from the head of the queue; and the part expanded as and when messages are added to the tail of the queue). With this optimisation, a second pointer is maintained. This second pointer indicates the first message added to the second part of the index.

Figure 4 provides an example of this optimisation. In this figure, there is shown a queue 400 with a number of messages thereon which are to be indexed by colour. An index 405 charts the location of messages on the queue. As a message is searched for from the head of the queue, non-matching messages are added to the first part of the index 410. Index
position 1 denotes the location of the last message added to the first part of the index 410. As messages are added to the tail of the queue, these are entered into the second part of the index 420. Index position 2 indicates the position of the first message entered into index part 420.

Commonly a user requesting a message will desire the first message of that type - e.g. the first purple message. In this case, if the first part of the index 410 does not give an appropriate message, the system must start to scan unindexed section N. This will correctly yield the first purple message from section N.

However, where neither the first index 410 nor the unindexed section N yields a result, the second index 420 may be used to avoid a scan of the second indexed section. Thus, a request for the first Orange message will need to scan the unindexed section N, but will not need to scan either indexed section (instead the index itself may be used to locate the orange message).

On the other hand, there are occasions when it does not matter as long as a message of type x is retrieved (this will be indicated by the requestor). In this case the whole of the index 410, 420 can be used to locate messages before it becomes necessary to scan the unindexed section N (i.e. if the requested message is not found in the index).

Assuming the whole of the index is useable, then pointers 1 and 2 can both be used to aid message retrieval. If a message does not exist in either parts of the index, then only the part of the queue between the two pointers need be searched (i.e. part N). For example, a green message may be requested. The index 405 does not hold any entry regarding such a message and thus this indicates that no messages to the left of index 1 and to the right of index 2 are green. It is therefore only necessary to scan part N. In this way, only two messages instead of five messages need be scanned. Note, as part N is scanned messages are added to the index. Thus index position 1 is moveable, but index position 2 remains fixed. When the two index positions meet, the whole of the queue is indexed and the two index sections are merged.

In this example, it is the combination of the two pointers index position a and index position 2 that constitutes the partiality definition store.
Second Preferred Embodiment

The present invention is also applicable to a database environment. A database includes a plurality of records stored in memory pages. New records are typically inserted at the end of the database.

Thus as with a messaging system, an index can be built dynamically and tailored to particular user requirements.

If an index does not exist, then the database is scanned from the very beginning and each non-matching record is entered into an index. A pointer is maintained to indicate the last indexed record. Once again this removes the need for database to be searched from the very beginning if an appropriate index entry does not exist. Instead the database can be searched from the last indexed point.

It is also possible to expand the index as records are added to the end of the database. Once again an index of two parts and also two pointers are maintained. The first pointer indicates the last indexed position, and the second indicating the first record added to the second part of the index.

Suppose a user is searching for a record referring to a person with "blue eyes". If the user requires the first record referring to blue eyes, then as with the messaging embodiment, the first part of the index should be scanned first, then the part of the data store existing between the pointers and then the second part of the index. If, on the other hand, the user does not have such a requirement, then the whole of the index can be used before it is necessary to resort to scanning the data store itself.

Where the database implements the insertion or update of records in place, an update into a segment of the database already indexed may require an associated update of the partial index. This partial index update will behave in the same manner as an update to a full index in the prior art.

This embodiment is almost identical to that described with reference to the messaging embodiments, and as in these cases the partiality definition store will hold two pointers to indicate the indexed areas of data at the start and end of the entire data store.

Second Embodiment Optimisation

It is common in databases to have several indices on the same data. This means that data held in data pages is accessed much more randomly than
is typical in a messaging system. For example, a majority of accesses to a user table in a database may be on name or personnel number, so these fields will be permanently indexed in the conventional manner. However, it may sometimes become worthwhile to create a partial indices on other attributes; for example if a study is being made on eye colour a temporary partial index may be built on eye colour. As standard queries are made on name or number, data pages containing user records will be retrieved. As each page is available, the opportunity is preferably taken to enhance the eye colour partial index with information for all records on a retrieved page.

With reference to figures 5a and 5b, a database system (not shown) includes a number of pages 530, each including a number of records (also not shown). A complete list 535 of all the pages stored is held in memory and each page entry contains a pointer 545 to the relevant stored page.

In order to aid retrieval of records held by the pages, an index 505 is maintained and the relevant location of the indexed record in the data store is pointed to. As discussed in the messaging embodiment, there may be cases where queries are expected to return results in an order corresponding to the underlying order of the stored data (i.e. a user may requires the first record of a certain type to be returned). In this case, the index is segmented. Accordingly a segment may be for a single page (e.g. page 4) or a sequential set of pages (e.g. pages 1 and 2).

In embodiments described previously, the partial index 505 referred to either one or two sequential sections of data. In this embodiment the partial index represents the indexation of a fragmented set of pages. Thus, it is no longer adequate to hold just one or two pointers to remember which data is indexed. Instead partiality definition store 520 consists of a list of index data page identifiers.

A search is conducted using the index 505 for pages which are indexed, and by page scan for pages that are not indexed. The overall search 540 may include interleaved processing of indexed and non-indexed data, with the partiality definition store 520 and the list 535 used to guide this interleaving.

Figure 5b exemplifies the processing that takes place on receipt of a request. A request for the first record referring to a person with blue eyes is received (step 600). In this example that record exists within page 5.
List 535 and partiality definition store 520 are examined (step 610). From this it is determined whether the next page stored by the database system is indexed (step 620) (Note, an indexed page may reference and point to multiple records). In this example, page 1 is the first page in list 535 and this page also exists within partiality definition store 520 - in other words, this page is indexed. The index is thus used to determine whether the group of pages held by the particular index segment containing for example page 1 and 2 has a record matching the search criterion (625, 630). In this instance the answer is no and so the process repeats itself (step 610). Again the partiality definition store 520 and list 535 are examined (step 610) and it is determined at step 620 that next page (page 3) is not indexed. In this case, the page itself is scanned (browsed) for a relevant record (step 640). Once again this page does not contain a relevant record (step 630) and the process loops round. List 535 and partiality definition store 520 are again examined (step 610) and it is determined that the next page (page 4) is indexed (step 620) and consequently the index is used to determine whether a relevant record exists therein (steps 625 and 630). In this way the search path 540, denoted by the thick arrow in figure 5a is followed, until a relevant record is found in page 5 and the record is returned to the requestor (step 650).

Whilst figures 5a and 5b illustrate the processing when ordering is a requirement, it is much more common in database for the query result to have no special order. This is illustrated in Figure 6. In this case, a single index 705 may be kept representing all indexed pages. The search 740 first checks this index and returns records accordingly. Where no record (or insufficient records) are found to satisfy the query, the non-indexed pages (3,5) are scanned, with the partiality definition store 720 and list 735 used to identify the pages that do not need to be included in the scan.

Note, it is possible to determine whether or not ordering is a requirement by the command issued on the request. For example an ordered SQL request would have the ‘by order’ command associated therewith.

Further note, a hybrid of the systems disclosed with respect to figures 5a and b and figure 6 may be used. This is useful in a system that typically does not require order but may require such on occasions. In this instance, a segmented index is maintained - see figure 5a. If ordering is specified then all segments are scanned before progressing to the actual stored data items themselves if the index does not reveal an appropriate record.
It will be appreciated that although the partiality definition store has been disclosed as holding a list of pages in the index. It may be convenient for some implementations to hold instead the list of pages NOT held in the index. This will require resulting minor differences to the processing described above. These differences will be readily appreciated by the skilled person and thus will not be discussed in detail herein.

All disclosed embodiments

It will be appreciated that the index may contain a reference to data items (e.g. messages, records) stored, or may contain the complete data item therein. Alternatively, the index may hold complete information for only some of the data items (e.g. short messages).

The base store (i.e. the queue of messages; the data records) may be held in non-volatile or volatile memory.

Likewise the index and partiality definition store may be held in either volatile or non-volatile memory.
CLAIMS

1. A method for retrieving a data item from a data store in a data processing system, the data processing system comprising an index, which at least partially indexes the data store, and a partiality definition store, the partiality definition store indicating a part of the data store which is referenced by the partial index, the method comprising the steps of:

   receiving a request for a data item from the data store; and

   responsive to determining that the requested data item is not referenced by the partial index, searching the data store by using the partiality definition store to avoid searching a part of the data store which is known to be indexed.

2. The method of claim 1 comprising the step of:

   during the step of searching for the requested data item, adding searched data items to the partial index.

3. The method of claim 2 comprising the step of:

   maintaining a first indication in the partiality definition store of the last indexed data item.

4. The method of claim 3, wherein the search step comprises:

   searching data items subsequent to the last data item indicated by the first indication.

5. The method of claim 3 or 4, wherein the data store has an end part and wherein data items are added to the partial index as they are inserted into the end part, the method comprising the step of:

   maintaining a second indication in the partiality definition store of the first inserted data item which is referenced in the partial index.

6. The method of claim 5, wherein the search step comprises:

   searching only data items present between the first and second indications.

7. The method of claim 5 or 6, wherein the partial index is segmented, a first segment comprising indexed data items prior to the first indication
and the second segment including data items post the second indication, the method comprising the step of:

responsive to determining that the next data item of a particular type is requested and that the data item is not referenced by the first segment, searching the data items in the data store between the first and second indications before searching the second index segment.

8. The method of any preceding claim comprising:

responsive to determining that the data store has fewer than a predetermined number of data items therein, deleting both the partiality definition store and the partial index.

9. The method of claim 1, wherein data items are records held on pages in a database system and wherein the partiality definition store comprises a list of indexed pages, wherein the search step comprises:

using a list of all pages stored to search pages in the data store, whilst using the partiality definition store to avoid searching those pages in the data store which are known to be indexed.

10. The method of claim 1, wherein data items are records held on pages in a database system and wherein the partiality definition store comprises a list of non-indexed pages, wherein the search step comprises:

using the partiality definition store to deduce those pages that are indexed in order to avoid searching such pages in the data store.

11. The method of claim 9 or 10, wherein the partial index is segmented, each segment comprising an index to at least one sequential page, the method further comprising:

responsive to determining that the next data item according to a specific order of a particular type is requested, interleaving the searching of index segments with searching of one or more pages in the data store.

12. The method of any of claims 9 to 11 comprising the step of:

retrieving a page based on a first indexed attribute; and

indexing the records held by the retrieved data page according to another attribute.
13. A system for retrieving a data item from a data store, the system comprising:

a data store;

an index, which at least partially indexes the data store;

a partiality definition store, the partiality definition store indicating a part of the data store which is referenced by the partial index;

means for receiving a request for a data item from the data store; and

means, responsive to determining that the requested data item is not referenced by the partial index, for searching the data store by using the partiality definition store to avoid searching a part of the data store which is known to be indexed.

14. The system of claim 13 comprising:

means for, whilst searching the data store, adding searched data items to the partial index.

15. The system of claim 14 comprising:

means for maintaining a first indication in the partiality definition store of the last indexed data item.

16. The system of claim 15, wherein the means for searching comprises:

means for searching data items subsequent to the last data item indicated by the first indication.

17. The system of claim 15 or 16, wherein the data store has an end part and further comprises:

means for adding data items to the partial index as they are inserted into the end part; and

means for maintaining a second indication in the partiality definition store of the first inserted data item which is referenced in the partial index.
18. The system of claim 17, wherein the means for searching comprises:

means for searching only data items present between the first and second indications.

19. The system of claim 17 or 18, wherein the partial index is segmented, a first segment comprising indexed data items prior to the first indication and the second segment including data items post the second indication, the system further comprising:

means, responsive to determining that the next data item of a particular type is requested and that the data item is not referenced by the first segment, for searching the data items in the data store between the first and second indications before searching the second index segment.

20. The system of any of claims 13 to 18 comprising:

means, responsive to determining that the data store has fewer than a predetermined number of data items therein, for deleting both the partiality definition store and the partial index.

21. The system of claim 13, wherein data items are records held on pages in a database system and wherein the partiality definition store comprises a list of indexed pages, wherein the means for searching comprises:

means for using a list of all pages stored to search pages in the data store, whilst using the partiality definition store to avoid searching those pages which are known to be indexed.

22. The system of claim 13, wherein data items are records held on pages in a database system and wherein the partiality definition store comprises a list of non-indexed pages, wherein the means for searching comprises:

means for using the partiality definition store to deduce those pages that are indexed in order to avoid searching such pages in the data store.

23. The system of claim 21 or 22, wherein the partial index is segmented, each segment comprising an index to at least one sequential page, the system further comprising:

means, responsive to determining that the next data item according to a specific order of a particular type is requested, for interleaving the searching of index segments with searching of one or more pages in the data store.
24. The system of any of claims 21 to 23 comprising:

   means for retrieving a page based on a first indexed attribute; and

   means for indexing the records held by the retrieved data page
   according to another attribute.

25. A computer program for retrieving a data item from a data store in a
    data processing system, the data processing system comprising an index,
    which at least partially indexes the data store, and a partiality
    definition store, the partiality definition store indicating a part of the
    data store which is referenced by the partial index, the computer program
    comprising program code means adapted to perform the method of any of
    claims 1 to 12 when said program is run on a computer.
**Application No:** GB0322844.2  
**Examiner:** Mr Ben Widdows  
**Claims searched:** 1-25  
**Date of search:** 21 April 2004

### Patents Act 1977: Search Report under Section 17

**Documents considered to be relevant:**

<table>
<thead>
<tr>
<th>Category</th>
<th>Relevant to claims</th>
<th>Identity of document and passage or figure of particular reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>1-5,9,11-17,21&amp;23-25</td>
<td>US 6058392 A1 (SAMPSON) see whole document</td>
</tr>
<tr>
<td>X</td>
<td>1-5,9,11-17,21&amp;23-25</td>
<td>&quot;Operational Data Storage Unification&quot; (HAMILTON) 06/07/97. Retrieved on the 16/04/04 via <a href="http://www.cs.uwaterloo.ca/cs-archive/CS-1997/16/intsto-1.pdf">http://www.cs.uwaterloo.ca/cs-archive/CS-1997/16/intsto-1.pdf</a> see esp. pages 24-26</td>
</tr>
<tr>
<td>X</td>
<td>1,2,9,11-14,21&amp;23-25</td>
<td>US 6547829 B1 (MICROSOFT CORP) see esp. abstract</td>
</tr>
<tr>
<td>X</td>
<td>1,2,9,11-14,21&amp;23-25</td>
<td>US 2002/0078087 A1 (STONE) see paragraph 19</td>
</tr>
<tr>
<td>A</td>
<td>-</td>
<td>US 2003/0005233 A1 (STEWART ET AL) see abstract and figs</td>
</tr>
</tbody>
</table>

**Categories:**

<table>
<thead>
<tr>
<th>X</th>
<th>Document indicating lack of novelty or inventive step</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>Document indicating lack of inventive step if combined with one or more other documents of same category.</td>
</tr>
<tr>
<td>&amp;</td>
<td>Member of the same patent family</td>
</tr>
<tr>
<td>A</td>
<td>Document indicating technological background and/or state of the art.</td>
</tr>
<tr>
<td>P</td>
<td>Document published on or after the declared priority date but before the filing date of this invention.</td>
</tr>
<tr>
<td>E</td>
<td>Patent document published on or after, but with priority date earlier than, the filing date of this application.</td>
</tr>
</tbody>
</table>

**Field of Search:**

Search of GB, EP, WO & US patent documents classified in the following areas of the UKC:

- **G4A**
- **W**

Worldwide search of patent documents classified in the following areas of the IPC:

- **G06F**
- **H**

The following online and other databases have been used in the preparation of this search report:

- **WPI, EPODOC, JAPIO, THE INTERNET**