(19) DANMARK

(10) DK/EP 2146292 T3

(12) Oversættelse af europæisk patentskrift

Patent- og Varemærkestyrelsen

(51) Int.Cl.: *G 06 F 17/30 (2006.01)* *G 06 F 12/08 (2016.01)*

(45) Oversættelsen bekendtgjort den: **2019-05-06**

(80) Dato for Den Europæiske Patentmyndigheds bekendtgørelse om meddelelse af patentet: **2019-01-23**

(86) Europæisk ansøgning nr.: **09164490.6**

(86) Europæisk indleveringsdag: **2009-07-03**

(87) Den europæiske ansøgnings publiceringsdag: **2010-01-20**

(30) Prioritet: **2008-07-18 US 81761** **2008-07-18 SE 0801708**

(84) Designerede stater: **AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HR HU IE IS IT LI LT LU LV MC MK MT NL NO PL PT RO SE SI SK SM TR**

(73) Patenthaver: **QlikTech International AB, Scheelevägen 24-26, 223 63 Lund, Sverige**

(72) Opfinder: **Wolgé, Håkan, Terminsvägen 14, 224 67, Lund, Sverige**

(74) Fuldmægtig i Danmark: **Patrade A/S, Ceresbyen 75, 8000 Århus C, Danmark**

(54) Benævnelse: **Fremgangsmåde og apparat til udtræk af information fra en database**

(56) Fremdragne publikationer:
WO-A-03/081464
US-A1- 2004 059 719
US-A1- 2006 230 024
US-A1- 2006 294 088
US-A1- 2008 091 646
US-B1- 6 275 819
US-B1- 6 347 312
JONSSON B D ET AL: "PERFORMANCE AND OVERHEAD OF SEMANTIC CACHE MANAGEMENT" ACM TRANSACTIONS ON INTERNET TECHNOLOGY, ACM, NEW YORK, NY, US, vol. 6, no. 3, 1 August 2006 (2006-08-01), pages 302-331, XP001542908 ISSN: 1533-5399

DESCRIPTION

Technical Field

[0001] The present invention relates to techniques for extracting information from a database, and in particular to techniques that involve a sequential chain of main calculations comprising a first main calculation which operates a first selection item on a data set representing the database to produce a first result, and a second main calculation which operates a second selection item on the first result to produce a second result.

Background art

[0002] It is often desired to extract specific information from a database, and specifically to summarize a large amount of data in the database and present the summarized data to a user in a lucid way. Such data processing is normally carried out by a computer, and may require significant memory capability and processing power of the computer. The data processing may aim at creating a large data structure commonly known as a multidimensional cube, which in turn may be accessed by the user to explore the data of the database, for example by visualizing selected data in pivot tables or graphically in 2D and 3D charts. An example of an efficient algorithm for creating such a multidimensional cube is known from US705862.

[0003] This prior art algorithm, like many other algorithms that operate on data in a database, involves a sequential chain of main calculations, in which the result of one main calculation is used as input data by a subsequent main calculation. For example, in the context of US7058621, the data records in the database is read into primary memory, whereupon a user may select one or more variables, and optionally a value or range of values for each such variable, thereby causing the algorithm to extract a corresponding subset of the data records in the database. The extracted subset forms an intermediate result. The multidimensional cube is then calculated by evaluating a selected mathematical function on the extracted subset, wherein the evaluation of the mathematical function is made based on a selected set of calculation variables, and wherein the dimensions of the cube are given by a selected set of classification variables.

[0004] Although the prior art algorithm is efficient, it may still need to carry out a large number of operations to create the multidimensional cube, especially if large amounts of data are to be analyzed. In such situations, the algorithm may set undesirably high requirements on the processing hardware and/or present a calculation time that is undesirably long.

[0005] US2006/0230024 relates to a method for a context-based cache infrastructure to enable subset query over a cached object. Responsive to detecting a query to a root context of a context tree, the tree is traversed for a parent context of a subcontext corresponding to the

name and value pair, which is identified by a user in the query. If the parent context caches all query results, the query results are iterated and the remaining name and value pairs are filtered out. However, if the parent context does not cache all query results, the traversing step is repeated for next parent context of the subcontext until a root context is encountered. If a root context is encountered, a query is issued to the database for the name and value pair and the result of the database query is cached in a new context.

Summary

[0006] It is an object of the invention to at least partly overcome one or more of the above-identified limitations of the prior art.

[0007] This and other objects, which will appear from the description below, are at least partly achieved by means of a method, a computer readable medium and an apparatus according to the independent claims, embodiments thereof being defined by the dependent claims.

[0008] A first aspect of the invention is a computer-implemented method for extracting information from a database, said method comprising a sequential chain of main calculations comprising a first main calculation which operates a first selection item on a data set representing the database to produce an intermediate result, and a second main calculation which operates a second selection item on the intermediate result to produce a final result, said method further comprising retrieving the final result by:

1. (a) calculating a first selection identifier value (ID1) as a statistically unique digital fingerprint generated by a hash function of at least the first selection item (S1);
2. (b) searching the objects of the data structure for the first selection identifier value (ID1), and if the first selection identifier value (ID1) is found, locating and retrieving a first result identifier (ID2) stored together with the first selection identifier value (ID1) as associated objects in a previous iteration;
3. (c) if the first result identifier (ID2) is found in sub-step (b),

   calculating a second selection identifier value (ID3) as a statistically unique digital fingerprint generated by a hash function of at least the second selection item (S2) and the retrieved first result identifier (ID2), and

   searching the objects of the data structure for the second selection identifier value (ID3), and if the second selection identifier value (ID3) is found locating and retrieving a final result (R2) stored together with the second selection identifier value (ID3) as associated objects in a previous iteration;

4. (d) if the first result identifier (ID2) is not found in sub-step (b),

   executing the first main calculation (P1) to produce the intermediate result (R1) and the first result identifier value (ID2) as a digital fingerprint generated by a hash function of

the intermediate result (R1),

storing the first selection identifier value (ID1) and the first result identifier value (ID2) as associated objects in the data structure; and

storing the first result identifier value (ID2) and the intermediate result (R1) as associated objects in the data structure,

calculating a second selection identifier value (ID3) as a statistically unique digital fingerprint generated by a hash function of the first result identifier value (ID2) and the second selection item (S2), and

searching the objects of the data structure based on the second selection identifier value (ID3), and if the second selection identifier value (ID3) is found locating and retrieving a final result (R2) stored together with the second selection identifier value (ID3) as associated objects in a previous iteration;

5. (e) if the final result (R2) is not found in sub-step (c) or (d),
   searching the objects of the data structure based on the first result identifier value (ID2);
6. (f) if the first result identifier value (ID2) is not found in sub-step (e)

   executing the first main calculation (P1) to produce the intermediate result (R1) and the first result identifier value (ID2) as a digital fingerprint generated by a hash function of the intermediate result (R1),

   storing the first result identifier value (ID2) and the intermediate result (R1) as associated objects in the data structure, and

   executing the second main calculation (P2) to produce the final result (R2) and storing the second selection identifier value (ID3) and the final result (R2) as associated objects in the data structure; and

7. (g) if the first result identifier value (ID2) is found in sub-step (e)
   retrieving the intermediate result (R1) stored together with the first result identifier value (ID2) as associated objects in a previous iteration, and

executing the second main calculation (P2) to produce the final result (R2) and storing the second selection identifier value (ID3) and the final result (R2) as associated objects in the data structure.

[0009] Thus, in the method according to the first aspect, the first and second results are cached in computer memory and made available for re-use in subsequent iterations of the method, thereby reducing the need to execute the first and/or second main calculations for extracting the information. The re-use may involve calculating the first and/or second selection identifier values during a subsequent iteration and accessing the data structure to potentially retrieve the first and/or second result.

[0010] A second aspect of the invention is a computer readable medium having stored thereon a computer program which, when executed by a computer, is adapted to carry out the method according to the first aspect.

[0011] A third aspect of the invention is an apparatus for extracting information from a database, said apparatus comprising means for executing a sequential chain of main calculations comprising a first main calculation which operates a first selection item on a data set representing the database to produce a first result, and a second main calculation which operates a second selection item on the first result to produce a second result, said apparatus further comprising means for retrieving the final result by:

1. (a) calculating a first selection identifier value (ID1) as a statistically unique digital fingerprint generated by a hash function of at least the first selection item (S1);
2. (b) searching the objects of the data structure for the first selection identifier value (ID1), and if the first selection identifier value (ID1) is found, locating and retrieving a first result identifier (ID2) stored together with the first selection identifier value (ID1) as associated objects in a previous iteration;
3. (c) if the first result identifier (ID2) is found in sub-step (b),

   calculating a second selection identifier value (ID3) as a statistically unique digital fingerprint generated by a hash function of at least the second selection item (S2) and the retrieved first result identifier (ID2), and

   searching the objects of the data structure for the second selection identifier value (ID3), and if the second selection identifier value (ID3) is found locating and retrieving a final result (R2) stored together with the second selection identifier value (ID3) as associated objects in a previous iteration;

4. (d) if the first result identifier (ID2) is not found in sub-step (b),

   executing the first main calculation (P1) to produce the intermediate result (R1) and the first result identifier value (ID2) as a digital fingerprint generated by a hash function of the intermediate result (R1),

   storing the first selection identifier value (ID1) and the first result identifier value (ID2) as associated objects in the data structure; and

   storing the first result identifier value (ID2) and the intermediate result (R1) as associated objects in the data structure,

   calculating a second selection identifier value (ID3) as a statistically unique digital fingerprint generated by a hash function of the first result identifier value (ID2) and the second selection item (S2), and

   searching the objects of the data structure based on the second selection identifier value (ID3), and if the second selection identifier value (ID3) is found locating and retrieving a final result (R2) stored together with the second selection identifier value (ID3) as

associated objects in a previous iteration;

5. (e) if the final result (R2) is not found in sub-step (c) or (d),
   searching the objects of the data structure based on the first result identifier value (ID2);
6. (f) if the first result identifier value (ID2) is not found in sub-step (e)

   executing the first main calculation (P1) to produce the intermediate result (R1) and the first result identifier value (ID2) as a digital fingerprint generated by a hash function of the intermediate result (R1),

   storing the first result identifier value (ID2) and the intermediate result (R1) as associated objects in the data structure, and

   executing the second main calculation (P2) to produce the final result (R2) and storing the second selection identifier value (ID3) and the final result (R2) as associated objects in the data structure; and

7. (g) if the first result identifier value (ID2) is found in sub-step (e)
   retrieving the intermediate result (R1) stored together with the first result identifier value (ID2) as associated objects in a previous iteration, and

executing the second main calculation (P2) to produce the final result (R2) and storing the second selection identifier value (ID3) and the final result (R2) as associated objects in the data structure.

[0012] The apparatus of the third aspect shares the advantages of the method of the first aspect, and may comprise further features corresponding to any of the embodiments described above in relation to the first aspect.

[0013] Still other objectives, features, aspects and advantages of the present invention will appear from the following detailed description, from the attached claims as well as from the drawings.

Brief Description of the Drawings

[0014] Embodiments of the invention will now be described in more detail with reference to the accompanying schematic drawings, in which the same reference numerals are used to identify corresponding elements.

Fig. 1 illustrates a process involving a chain of calculations for extracting information from a database, wherein identifiers and results are selectively stored in and retrieved from a computer memory.

Fig. 2 illustrates one example of a process as shown in Fig. 1.

Fig. 3 illustrates another example of a process as shown in Fig. 1.

Fig. 4 illustrates an embodiment of the process in Fig. 1.

Fig. 5 illustrates yet another example of a process as shown in Fig. 1.

Fig. 6 is an exemplifying flow chart for the process in Fig. 5.

Fig. 7 is an overview of the process in Fig. 5 as implemented in a specific context

Fig. 8 is a block diagram of a computer-based environment for implementing embodiments of the invention.

Detailed Description of Exemplary Embodiments

[0015] The present invention relates to techniques for extracting information from a database. For ease of understanding, some underlying principles will first be discussed in relation to a generalized example. Then, different aspects, features and advantages will be discussed in relation to a specific implementation.

GENERAL

[0016] Fig. 1 illustrates an example of a computer-implemented process for extracting information from a database DB, which may or maynot be stored externally of the computer that implements the process. The extraction process invalves extracting an initial data set or scope RO from the database DB, e.g. by reading the initial data set RO into the primary memory (e.g. RAM) of the computer. The initial data set RO may include the entire contents of the database DB, or a subset thereof.

[0017] The process of Fig. 1 invalves a sequence of main calculation procedures PI, P2 which operate to generate a final result R2 based on the initial data set RO. Specifically, a first procedure PI operates on the initial data set RO to produce an intermediate result R1, and the second procedure P2 operates on the intermediate result to produce the final result R2.

[0018] The first procedure Plis controlied by a first selection item S1, which mayor may not originate from user input. Similarly, the second procedure P2 is controlied by a second selection item S2, which may or may not originate from user input. Each selection item S1, S2 may include any combination of variables and for mathematical functions that define are finement of the input data to the respective procedure, i.e. the data set RO and the intermediate result R1, respectively.

[0019] Fig. 1 also indicates that the extraction process interacts with a computer memory 10

(typically RAM or cache memory), by the first and second procedures PI, P2 operating to store data items in the memory 10 and retrieve data items from the memory 10. In the illustrated example, the first procedure PI operates to store and retrieve identifiers, generally denoted by ID, and intermediate results R1, and the second procedure P2 operates to store and retrieve identifiers, generally denoted by ID, intermediate results R1 and final results R2. In the following, the procedure of storing identifiers and results in computer memory 10 is also referred to as "caching".

[0020] Different identifiers are typically generated by the procedures PI, P2 as a function of one or more process parameters, such as another identifier and/or a selection item S1, S2 and/or a result R1, R2. Different functions may or may not be used for generating different identifiers. The function(s) for generating an identifier may be a hashing algorithm that generates a digital fingerprint of the relevant process parameter(s). The function/functions is/are suitably configured such that each unique combination of parameter values results in an identifier value which is unique among all identifier values that are generated for all different identifiers within the process. In this context, "unique" not only includes theoretically unique identifier values, but also statistically unique identifier values. One non-limiting example of such a function is a hashing algorithm that generates a digital fingerprint of at least 256 bits.

[0021] In a process, further illustrated in Fig. 2, the first procedure P1 is configured to calculate a first selection identifier value ID 1 as a function of the first selection item S1, i.e. ID1=f(S1), and the second procedure P2 is configured to calculate a second selection identifier value ID3 as a function of the second selection item S2 and the intermediate result R1, i.e. ID3=f(S2, RI). The first procedure P1 is also configured to store ID1 and the intermediate result R1 as associated objects in a data structure 12 in the computer memory, and the second procedure P2 is configured to store ID3 and R2 as associated objects in the data structure 12. Thus, the data structure 12 in the computer memory 10 is configured to store a heterogeneous set of objects, i.e. objects of different types.

[0022] This process enables a reduction in the response time for the extraction process and/or a reduction in the processing requirements of the computer that implements the extraction process, by reducing the necessity to execute the main calculation procedures PI, P2 for calculating the intermediate result R1 and the final result R2, respectively. For example, the extraction process may be configured to use the data structure 12, whenever possible, to find the final result R2 based on the first selection item S1 and the second selection item S2. Thus, when the process discovers a need to calculate the final result R2, based on S1 and S2, it may generate ID1=f(S1) and access the data structure 12 based on ID1. If an identical first selection item S1 has been used with the first procedure P1 before, it is likely that the generated value of ID1 is found in the data structure 12 and associated with the corresponding intermediate result R1. Thus, the intermediate result R1 may be retrieved from the data structure 12 instead of being calculated by the procedure P1. If the intermediate result R1 is not found in the data structure 12, the process may cause the first procedure P1 to calculate the intermediate result R1. Furthermore, after obtaining the intermediate result R1, the process may generate ID3=f(R1, S2) and access the data structure 12 based on ID3. Again, if the

same operation has been executed by procedure P2 before, it is likely that the generated value of ID3 is found in the data structure 12 and associated with the corresponding final result R2. Thereby, the final result R2 may be retrieved from the data structure 12 instead of being calculated by the procedure P2.

[0023] In process, further illustrated in Fig. 3, the first procedure P1 is further configured to calculate a first result identifier value ID2 as a function of the intermediate result R1. The first procedure P1 is also configured to store ID 1 and ID2 as associated objects in the data structure 12, and to store ID2 and the intermediate result R1 as associated objects in the data structure 12.

[0024] This embodiment process enables a reduction in the size of the computer memory required by the process, since each intermediate result R1 is only stored once in the data structure 12, even if two or more first selection items S1 yield identical intermediate results R1. This embodiment is particularly relevant when the intermediate results R1 are large, which is often the case when processing information from databases.

[0025] The calculation of the first result identifier value ID2 also enables an embodiment, illustrated in Fig. 4, in which the intermediate result R1 is represented by the first result identifier value ID2 in the calculation of the second selection identifier value ID3, i.e. ID3=f(ID2, S2).

[0026] This embodiment results in a reduced need to store the intermediate result R1 in the data structure 12, since the final result R2 can be retrieved from the data structure 12 based on ID3, which is generated based on ID2, not the intermediate result R1. This enables efficient calculation of the final result R2, even if the intermediate result R1 has been purged from the data structure 12. For example, the process may be configured to use the data structure 12, whenever possible, to find the final result R2 based on the first selection item S1 and the second selection item S2. Thus, when the process discovers a need to calculate the final result R2, based on S1 and S2, it may generate ID1=f(S1) and access the data structure 12 based on ID1 to retrieve ID2 associated therewith, if an identical first selection item S1 has been used with the first procedure P1 before. Then, the process may generate ID3=f(ID2, S2) and access the data structure 12 based on ID3 to retrieve the final result R2 associated therewith, if the second procedure P2 has operated on an identical intermediate result R1 and an identical second selection item S2 before. In this example, the final result R2 can thus be retrieved from the data structure 12 even if the intermediate result R1 has been deleted.

[0027] In a process, illustrated in Fig. 5, the first procedure P1 is further configured to calculate a second result identifier value ID4 as a function of the final result R2. The second procedure P2 is also configured to store ID3 and ID4 as associated objects in the data structure 12, and to store ID4 and the final result R2 as associated objects in the data structure 12.

[0028] This process enables a reduction in the size of the computer memory required by the

process, since each final result R2 is only stored once in the data structure 12, even if two or more second selection items S2 yield identical final results R2. This process is particularly relevant when the final results R2 are large.

[0029] Hitherto, the database DB, and thus the data set R0, has been presumed to be static. If the database is dynamic, it may be suitable to generate the first selection identifier ID1 as a function of the first selection item S1 and the data set R0, i.e. ID1=f(S1, R0). With such a modification, all of the processes and embodiments described in relation to Fig. 1-5 are equally applicable to a dynamic database, i.e. a database that may change at any time.

[0030] Fig. 6 is a flow chart illustrating one exemplifying implementation of the process shown in Fig. 5, adapted to operate on a dynamic database. The process starts by inputting the data set R0 (step 600), the first selection item S1 (step 602) and the second selection item S2 (604). Then, a value of the first selection identifier ID1 is generated as a function of S1 and R0 (step 606). A lookup is made in the data structure based on ID1 (step 608). If the value of ID1 is found in the data structure, i.e. has been cached in a previous iteration, the process retrieves the value of the first result identifier ID2 associated therewith (step 610) and proceeds to step 612.

[0031] If the value of ID1 is not found in the data structure in step 608, the process causes the first procedure P1 to calculate R1, by operating S1 on R0 (step 614). Then, the value of ID2 is generated as a function of R1 (step 616), and the values of ID1, ID2 and R1 are stored in the data structure in associated pairs ID1:ID2 and ID2:R1 (step 618). The process then proceeds to step 612.

[0032] In step 612, the value of the second selection identifier ID3 is generated as a function of S2 and ID2. Then, a lookup is made in the data structure based on ID3 (step 620). If the value of ID3 is found in the data structure, i.e. has been cached in a previous iteration, the process retrieves the value of the second result identifier ID4 associated therewith (step 622). A further lookup is made in the data structure based on ID4 (step 624). If the value of ID4 is found in the data structure, i.e. has been cached in a previous iteration, the process retrieves the final result R2 associated therewith (step 626).

[0033] If the value of ID3 is not found in the data structure in step 620, a further lookup is made in the data structure based on the value of ID2 determined in step 610 or step 616 (step 628). If the value of ID2 is found in the data structure, i.e. has been cached in a previous iteration, the process retrieves the first result R1 associated therewith (step 630). The process then causes the second procedure P2 to calculate R2, by operating S2 on R1 (step 632). In order to update the data structure, the process also generates the value of ID4 as a function of R2 (step 634) and stores the values of ID3, ID4 and R2 in the data structure in associated pairs ID3:ID4 and ID4:R2 (step 636).

[0034] If the value of ID2 is not found in the data structure in step 628, the process causes the first procedure P1 to calculate R1, by operating S1 on R0 (step 638), and stores the values of

ID2 and R1 in the data structure in an associated pair ID2:R1 (step 640). The process then proceeds to step 632. However, it should be realized that if the intermediate result R1 was already calculated in step 614, it is not necessary to perform steps 628, 630, 638 and 640. In such a case, if ID3 is not found in step 620, the process may proceed directly to step 632, in which the second procedure P2 is caused to calculate R2, by operating S2 on R1.

[0035] If the value of ID4 is not found in the data structure in step 622, the process causes the second procedure P2 to calculate R2, by operating S2 on R1 (step 642). In order to update the data structure, the process also generates the value of ID4 as a function of R2 (step 644) and stores the values of ID4 and R2 in the data structure in an associated pair ID4:R2 (step 646).

[0036] The skilled person readily understands that the processes and embodiments in Figs 2-4 result in corresponding storage and retrieval processes, albeit using different combinations of identifiers. For brevity of presentation, these processes are not illustrated in flow charts, but merely given as exemplifying processes and embodiments in the foregoing Summary section.

[0037] It is to be understood that any data structure 12, linear or non-linear, may be used for storing the identifiers and results. However, for reasons of processing speed it may be preferable to use a data structure 12 with an efficient index system, such as a sorted list, a hash table, or a binary tree, such as an AVL tree.

SPECIFIC EMBODIMENTS, IMPLEMENTATIONS AND EXAMPLES

[0038] In the following, embodiments of the invention, implementations and examples are discussed and exemplified in further detail.

[0039] In embodiments of the invention, previous calculations and results are used in the processing of successive requests for new data and new calculations. To this end, the extraction process is designed to cache results during the processing of the data requests. When a subsequent request is processed, the extraction process determines if an appropriate previous result has already been generated and cached. If so, the previous result is used in the processing of the subsequent request. Since the prior calculations need not be regenerated, the processing time for the subsequent request may be reduced considerably.

[0040] In embodiments of the invention, digital identifiers (digital fingerprints) are used to identify the cached information, and in this way a cached result can be reused also when reached in a different way than in the previous calculation.

[0041] In embodiments of the invention, the digital identifiers themselves are stored in the cache. Specifically, the identifier of the input for a calculation procedure is stored together with the digital identifier of the output of the calculation procedure. Hence, the final result of a many-step operation can be reached also when the needed complex intermediate result(s) has been purged from the cache. Only the digital identifier of the intermediate result(s) is needed.

[0042] In embodiments of the invention, the cache is implemented by a data structure that can store heterogeneous objects, such as tables, data sub-sets, arrays and digital identifiers.

[0043] Embodiments of the invention may thus serve to minimize, or at least reduce, the response times for a user who queries a data storage using a query that has been executed recently by the same or another user.

[0044] Embodiments of the invention may also serve to minimize, or at least reduce, the memory usage by the cache by re-using the same cache entry for several different queries or calculations, in the case that two queries or calculations happen to yield the same result.

[0045] Embodiments of the invention are applicable for extracting any type of information from any type of known database, such as relational databases, post-relational databases, object-oriented databases, hierarchical databases, etc. The Internet may also be regarded as a database in the context of the present invention.

[0046] Fig. 7 discloses a specific embodiment of the invention, which is an extraction process or information search that involves a database query with a subsequent chart calculation based on the query result. The result of the chart calculation, denoted *Chart Result,* is typically data which is aggregated, sorted or grouped in one, two or multiple dimensions, e.g. in the form of a multidimensional cube as discussed in the Background section.

[0047] In a first step, the *Scope* for the information search is defined. In the case of a database query, the scope is defined by the tables included in a SELECT statement (or equivalent) and how these are joined. For an Internet search, the scope may be an index of found web pages, usually also organized as one or more tables. The output of the first step is thus a data set (cf. R0 in Figs 1-6).

[0048] In a second step, a user makes a *Selection* in the data set, causing an *Inference Engine* to evaluate a number of filters on the data set. The inference engine could be e.g. a database engine, a query tool or a business intelligence tool. For example, in a query on a database that holds data of placed orders, this could be demanding that the order year be '2007' and the product group be 'Dairy products'. The selection may thus be uniquely defined by a list of included fields and, for each field, a list of selected values or, more generally, a condition.

[0049] Based on the selection (cf. S1 in Figs 1-6), the inference engine executes a calculation procedure (cf. P1 in Figs 1-6) to generate a *Data subset* (cf. R1 in Figs 1-6) that represents a part of the scope (cf. R0 in Figs 1-6). The data subset may thus contain a set of relevant data records from the scope, or a list of references (e.g. indices, pointers or binary numbers) to these relevant data records. In the above example, the relevant data records would be only the data records that pertain to the year '2007' and to the product group 'Dairy products'.

[0050] If the selection has never been made before, the inference engine in Fig. 7 is operated to calculate the data subset. However, if the calculation has been made before, the inference engine is instead operated to re-use the previous result by accessing a specific data structure: a "cache".

[0051] The next step is often to make some further calculations, e.g. aggregation(s) and/or sorting(s) and/or grouping(s), based on the data subset. In the example of Fig. 7, these subsequent calculations are made by a *Chart Engine* that calculates the *Chart Result* based on the data subset and a selected set of *Chart Properties* (cf. S2 in Figs 1-6). The chart engine thus executes a chart calculation procedure (cf. P2 in Figs 1-6) to generate the chart result (cf. R2 in Figs 1-6). If these calculations have never been made before, the chart engine in Fig. 7 is operated to generate the chart result. However, if these calculations have been made before, the chart engine is instead operated to re-use the previous result by accessing the aforesaid cache. The chart result may then be visualized to a user in pivot tables or graphically in 2D and 3D charts.

[0052] Fig. 7 also illustrates the process of using the cache, with f representing the hashing algorithm that is operated to generate digital identifiers, ID1-ID4 representing the thus-generated digital identifiers, and solid line arrows representing the flow of data for generation of the identifiers ID1-ID4. Further in Fig. 7, dashed arrows represent cache look-ups.

[0053] In Fig. 7, when a user makes a new selection, the inference engine calculates the data subset. Also, the identifier ID1 for the selection together with the scope is generated based on the filters in the selection and the scope. Subsequently, the identifier ID2 for the data subset is generated based on the data subset definition, typically a bit sequence that defines the content of the data subset. Finally, ID2 is put in the cache using ID1 as lookup identifier. Likewise, the data subset definition is put in the cache using ID2 as lookup identifier.

[0054] In Fig. 7, the chart calculation takes place in a similar way. Here, there are two information sets: the data subset and the relevant chart properties. The latter is typically, but not restricted to, a mathematical function together with calculation variables and classification variables (dimensions). Both of these information sets are used to calculate the chart result, and both of these information sets are also used to generate the identifier ID3 for the input to the chart calculation. ID2 was generated already in the previous step, and ID3 is generated as the first step in the chart calculation procedure.

[0055] The identifier ID3 is formed from ID2 and the relevant chart properties. ID3 can be seen as an identifier for a specific chart generation instance, which includes all information needed to calculate a specific chart result. In addition, a chart result identifier ID4 is created from the chart result definition, typically a bit sequence that defines the chart result. Finally, ID4 is put in the cache using ID3 as lookup identifier. Likewise, the chart result definition is put in the cache using ID4 as lookup identifier.

[0056] In this specific example, a two-step caching of the result is performed in both the

inference procedure and the chart calculation procedure. In the inference procedure, ID1 and ID2 represent different things: the selection and the data subset definition, respectively. If two different selections yield the same data subset, which is quite possible, the two-step caching (ID1:ID2; ID2: data subset) causes the data subset to be cached only once. This is denoted Object Folding in the following, i.e. several data objects in the cache share the same cache entry. Similarly, in the chart calculation procedure, ID3 and ID4 represent different things: the chart generation instance and the chart result definition, respectively. If two different chart generation instances yield the same chart result, which is quite possible, the two-step caching (ID3:ID4; ID4: chart result) causes the chart result to be cached only once.

[0057] Further, by caching ID3, the chart result can be recreated also if the data subset definition has been purged from the cache. This is a relevant advantage since the data subset definition can be very large and hence prone to get purged from the cache if a cache purging mechanism is implemented. A non-limiting example of such a mechanism will be described further below.

[0058] During the extraction process, identifiers are calculated from the selection, the relevant chart properties, etc. and used to lookup possibly cached calculation results, as indicated by the dashed arrows in Fig. 7. If the identifier is found, the corresponding cached result will be re-used. If not found, the extraction process will generate new identifiers and cache them with the respective result.

[0059] To further exemplify the extraction process, consider the above-mentioned selection of order year '2007' and product group 'Dairy products'. The first step is to generate a digital identifier ID1 as a function of this selection, e.g. (written in hexadecimal notation): *'31dca7ad013964891df428095ad9b78ad7a69eaaa1ca3886bcf05d8f8184e84a'.*

[0060] For the sake of brevity, each identifier is represented by its initial 4 characters in the following example. So, ID1 instead becomes '31dc'. Furthermore, for reasons of clarity the illustrating tables below include identifier labels, e.g. 'ID1:' in front of the digital identifiers. This is not necessary in the real solution.

[0061] The subsequent extraction process is the following: When ID1 has been generated, it is looked for in the cache. The first time the selection is made, this identifier will not be found in the cache, so the resulting data subset must be calculated the normal way. Once this is done, ID2 can be generated from the data subset to be e.g. 'd2b8'. Then ID1 is cached, pointing at ID2; and ID2 is cached, pointing at the bit sequence that defines the resulting data subset. This bit sequence can be considerable in size. The content of the cache is shown in Table 1 below.

Table 1:

ID	Cached value
ID1:31dc	ID2:d2b8
ID2:d2b8	<data records in resulting data subset>

[0062] The next time the same selection is made, the process will be different: Now ID1 is found in the cache, pointing at 'ID2:d2b8', which in turn is used for a second look-up, whereupon the bit sequence of the resulting data subset is found, retrieved and used instead of a time-consuming calculation.

[0063] Now consider the case where a different selection is made, but yielding the same resulting data subset. For example, it may happen that a user selects exactly those customers that have bought 'Dairy products' without explicitly demanding 'Dairy products' and these have bought nothing but dairy products. ID1 is now generated as e.g. 'f142', and will not be found in the cache. So, the resulting data subset must be calculated the normal way. Once this is done, ID2 can be generated from the data subset, and is found to be 'd2b8', which already is stored in the cache. So, the algorithm need only add one entry to the cache, the one where 'ID1:f142' points to 'ID2:d2b8'. The content of the cache is shown in Table 2 below.

Table 2:

ID	Cached value
ID1:f142	ID2:d2b8
ID1:31dc	ID2:d2b8
ID2:d2b8	<data records in resulting data subset>

[0064] No calculation time was saved, this time, but cache entries are re-used to prevent the cache from growing unnecessarily. And now both 'ID1:f142' and 'ID1:31dc' point to the cache entry containing the same resulting data subset: 'ID2:d2b8', and both can be used in later look-ups. This is thus an example of the aforesaid "object folding".

[0065] A further advantage of caching digital identifiers will become clear when the subsequent chart calculation is performed. So, assume that the above selections have been made and the subsequent chart calculation has been performed. ID3 and ID4 have been generated as 'e40A' and '7505', respectively, and stored in the cache. The content of the cache is shown in Table 3 below.

Table 3:

ID	Cached value
ID1:f142	ID2:d2b8
ID1:31dc	ID2:d2b8
ID2:d2b8	<data records in resulting data subset>
ID3:e40A	ID4:7505
ID4:7505	<matrix of numbers representing chart result>

[0066] Of the five entries in Table 3, one is most likely to be considerably larger than all other: 'ID2:d2b8' containing the entire bit sequence that defines the potentially large data subset. Its

size makes it a candidate to be purged when/if the cache is maintained, as described further below. So, after a while the content of the cache may be as shown in Table 4 below.

Table 4:

ID	Cached value
ID1:f142	ID2:d2b8
ID1:31dc	ID2:d2b8
ID3:e40A	ID4:7505
ID4:7505	<matrix of numbers representing chart result>

[0067] However, since the digital identifiers are cached, it is still possible to obtain the chart result without having to recalculate the intermediate data subset. Instead, when the selection is made, ID1 is calculated. Next, a look-up of ID1 is made in the cache, resulting in ID2 being retrieved. ID3 is subsequently generated from the combination of the relevant chart properties and ID2. A look-up of ID3 in the cache is made, and ID4 is retrieved. Finally, a look-up of ID4 in the cache is made and the chart result is recuperated. Hence, the chart result is found without any heavy calculations, but only based on digital identifiers, which may be generated by fast and processing-efficient operations.

[0068] From the above, it is understood that the digital identifiers should be unique so that the meaning of each identifier in the cache is unambiguous. In one embodiment, the digital identifiers are generated using a hashing algorithm or function. Hashing algorithms are transformations that take an input of arbitrary size (the message) and return a fixed-size string, which is called the hash value (message digest). The algorithm typically chops and mixes, e.g. substitutes or transposes, the input to create a digital fingerprint thereof. The simplest and oldest hashing algorithms are simple modulo by prime operations. Hashing algorithms are used for a variety of computational purposes, including cryptography. Generally speaking, a hashing algorithm should behave as much as possible like a random function, by generating any possible fixed-size string with equal "probability", while still really being deterministic.

[0069] There are several well-known and frequently used hashing algorithms that may be used for generating the above-mentioned digital identifiers. Different hashing algorithms are optimised for different purposes, some being optimised for efficient and fast computation of the hash value, whereas others are designed for high cryptographic safety. An algorithm with high cryptographic safety is designed to make it difficult to calculate a message that matches a given hash value within reasonable time, and to find a second message that generates the same hash value as a first given message. Such hashing algorithms include SHA (Secure Hash Algorithm) and MD5 (Message-Digest algorithm 5). Processing-efficient hashing algorithms typically exhibit lower cryptographic safety. Such hashing algorithms include FNV algorithms (Fowler/Noll/Vo), which are designed to be fast while generally maintaining a very low collision rate. An FNV algorithm typically starts with an offset base, which in principle could be any random string of values, but typically by tradition always is the signature of the inventor in hexadecimal code run through the original FNV-0 algorithm. For generating a 256-bit FNV

hash value, the following offset base is usually used:
'0xdd268dbcaac550362d98c384c4e576ccc8b1536847b6bbb31023b4c8caee0535'.

[0070] For each byte in the input to the hashing algorithm, the offset is first multiplied by a large prime number, then subsequently compared with the byte from the input and finally the bitwise symmetric difference (XOR) is calculated to form the hash value for the next loop. Appropriate prime numbers are found in open literature. Any large prime numbers will work, but some are more collision-resistant than others.

[0071] The digital identifiers may be generated using any hashing algorithm, which is reasonably collision-resistant. In one embodiment, the identifiers are generated using a fast hashing algorithm with high collision resistance and low cryptographic safety.

[0072] In one specific embodiment, a 256-bit identifier may be created by concatenating four 64-bit FNV hashes, each generated using a different prime multiplier. By using four shorter hashes and concatenating them, the identifier can be generated faster. To further speed up the generation of the identifier, the algorithm may be modified to use not only one byte of the input per loop, but instead four bytes. This may result in a loss of cryptographic safety, while the collision resistance remains roughly the same.

[0073] Identifiers with a length of at least 256 bits may yield a beneficial collision-resistance. A 256-bit hash value means that there are approximately 1E+77 possible identifier values. This number can be compared to the number of atoms in the universe which has been estimated to 1E+80. This means that the risk of collisions, i.e. the risk that two different selections/data subsets/chart properties/chart results yield the same identifier, is not only extremely small, but negligible. So we can safely say that the risk of collisions is acceptably small. This means that although the hashing algorithm does not generate theoretically unique identifiers, it does however generate statistically unique identifiers. However, it to be understood that identifiers of shorter bit length, such as 64 or 128 bits, may be sufficiently statistically unique for a specific application.

[0074] As mentioned above, a purging mechanism may be implemented to purge the cache of old or unused entries. One strategy may be to eliminate the lowest-usage entry/entries in the cache. However, a more advanced purging mechanism may be implemented to support optimisation of both processor usage and memory usage. One embodiment of such an advanced purging mechanism operates on three parameters: Usage, Calculation time and Memory need.

[0075] The Usage parameter is a numeric value that may consider both if an entry has been accessed "recently, but not often" and if the entry has been accessed "often, but not recently". This may be accomplished by associating each entry with a usage parameter U which is increased by e.g. one unit every time the entry is accessed, but decreases its value exponentially, or by any other function, over time. In one implementation, all values of U in the cache are periodically reduced by a fixed amount. Thus, the usage parameter has a half-life,

similar to a radioactive decay. The value of U will now reflect how much and how recently the entry has been accessed.

[0076] If the processor time needed to calculate an entry is considerable, then the entry should be kept longer in the cache. Conversely, if the processor time needed for the calculation is small, then the cost of re-calculating is small and the benefit of keeping the entry in the cache is also small. Thus, each entry is associated with a time parameter T that represents the estimated calculation time.

[0077] If the memory space needed to store an entry is considerable, then it costs a lot of the cache resources to keep it and it should be purged from the cache sooner than an entry that requires less memory space. Conversely, an entry requiring little memory space can be kept longer in the cache. Thus, each entry is associated with a memory parameter M that represents the estimated memory need.

[0078] For each entry in the cache, the values of the U, T and M parameters are evaluated by a weight function W given by: W = U * T / M.

[0079] A large value of W for an entry indicates that there are good reasons to keep this entry in the cache. Thus, the entries with large W values should be kept in the cache and the ones with small W values should be purged.

[0080] An efficient purging mechanism may involve sorting the cache according to the W values and purging the sorted cache from one end, i.e. the entries with the smallest W values. One possible, but not necessary, way to keep a sorted cache would be to store the identifiers, results and U, T, M and W values as an AVL (Adelson-Velsky and Landis) tree, i.e. a self-balancing binary search tree.

[0081] The purging mechanism may intermittently purge all entries with a W value that falls below a predetermined threshold value.

[0082] Alternatively, the purging mechanism may be controlled by the amount of available memory on the computer, or the ratio of available memory to total memory. Thus, whenever the size of the cache memory reaches a memory threshold value, the purging mechanism removes entries from the cache entries based on their respective W value. By setting the memory threshold, it is possible to adapt the cache size to the local hardware conditions, e.g. to trade processing power for memory. For example, it is possible to compensate for a slower processor in a computer by adding more primary memory to the computer and increasing the memory threshold. Thereby, more results will retained in the cache and the need for processing will be reduced.

[0083] Embodiments of the invention also relate to an apparatus for performing any one of the algorithms, methods, processes and procedures described in the foregoing. This apparatus may be specially constructed for the required purpose or it may comprise a general-purpose

computer which is selectively activated or reconfigured by a computer program stored in the computer.

[0084] Fig. 8 is a block diagram of a computer-based environment for implementing any of the embodiments of the invention. A user 1 interacts with a data processing system 2, which includes a processor 3 that executes operating system software as well as one or more application programs that implement an embodiment of the invention. The user enters information into the data processing system 2 by using one or more well-known input devices 4, such as a mouse, a keyboard, a touch pad, etc. Alternatively, the information may be entered with or without user intervention by another type of input device, such as a card reader, an optical reader, or another computer system. Visual feedback may be given to the user by showing characters, graphical symbols, windows, buttons, etc, on a display 5. The data processing system further includes the aforesaid memory 10. The software executed by the processor 3 stores information relating to the operation thereof in the memory 10, and retrieves appropriate information from the memory 10. The memory 10 typically includes a primary memory (such as RAM, cache memory, etc) and a non-volatile secondary memory (hard disk, flash memory, removable medium). The database may be stored in the memory 10 of the data processing system, or it may be accessed on an external storage device via a communications interface 6 in the data processing system 2.

[0085] The invention has mainly been described above with reference to a few embodiments. However, as is readily appreciated by a person skilled in the art, other embodiments than the ones disclosed above are equally possible.

[0086] For example, the present invention is not only applicable for calculating multidimensional cubes, but may be useful in any situation where information is extracted from a database using a chain of calculations.

[0087] Further, the inventive extraction process may be applied to a chain of calculations that involve more than two consecutive calculations. For example, each of two or more intermediate results in a chain of calculations may be cached and subsequently retrieved similarly to the intermediate result as described in the foregoing.

[0088] Further, the inventive extraction process need not cache and subsequently retrieve the final result, but may instead operate only to cache and retrieve one or more intermediate results in a chain of calculations.

[0089] Still further, it should be realized that the initial step of extracting an initial data set or scope from the database may be omitted, and the extraction process may instead operate directly on the database.

REFERENCES CITED IN THE DESCRIPTION

This list of references cited by the applicant is for the reader's convenience only. It does not form part of the European patent document. Even though great care has been taken in compiling the references, errors or omissions cannot be excluded and the EPO disclaims all liability in this regard.

Patent documents cited in the description

- US705862A [0002]
- US7058621B [0003]
- US20060230024A [0005]

PATENTKRAV

1. Computer-implementeret fremgangsmåde til udtræk af information fra en database, hvilken fremgangsmåde omfatter en sekventiel kæde af hovedberegninger omfattende en første hovedberegning (P1), som styrer et første selektionselement (S1) på et datasæt (R0), som repræsenterer databasen for at frembringe et mellemresultat (R1) og en anden hovedberegning (P2), som styrer et andet selektionselement (S2) på mellemresultatet (R1) for at frembringe et slutresultat (R2), hvilken fremgangsmåde endvidere omfatter at hente det endelige resultat ved:

(a) at beregne en første selektionsidentifikationsværdi (ID1) som et statistisk entydigt digitalt fingeraftryk genereret af en hashfunktion af i det mindste det første selektionselement (S1);

(b) at søge i datastrukturens objekter for den første selektionsidentifikationsværdi (ID1), og hvis den første selektionsidentifikationsværdi (ID1) er fundet, at lokalisere og hente en første resultatidentifikator (ID2), der er lagret sammen med den første selektionsidentifikationsværdi (ID1) som forbundne objekter i en tidligere iteration;

(c) hvis den første resultatidentifikator (ID2) findes i undertrin (b),

at beregne en anden valgidentifikationsværdi (ID3) som et statistisk entydigt digitalt fingeraftryk genereret af en hashfunktion af i det mindste det andet selektionselement (S2) og den hentede første resultatidentifikator (ID2) og at søge i datastrukturens objekter for anden selektionsidentifikationsværdi (ID3), og hvis den anden selektionsidentifikationsværdi (ID3) er fundet, at lokalisere og hente et slutresultat (R2) gemt sammen med den anden selektionsidentifikationsværdi (ID3) som forbundne objekter i en tidligere iteration;

d) hvis den første resultatidentifikator (ID2) ikke findes i undertrin (b),

at udføre den første hovedberegning (P1) for at frembringe mellemresultatet (R1) og den første resultatidentifikationsværdi (ID2) som et digitalt fingeraftryk genereret af en hashfunktion af mellemresultatet (R1), at lagre den første selektionsidentifikationsværdi (ID1) og den første resultatidentifikationsværdi (ID2) som forbundne objekter i datastrukturen; og

at lagre den første resultatidentifikationsværdi (ID2) og mellemresultatet (R1) som forbundne objekter i datastrukturen,

at beregne en anden selektionsidentifikationsværdi (ID3) som et statistisk unikt digitalt fingeraftryk genereret af en hashfunktion af den første resultatidentifikationsværdi (ID2) og det andet selektionselement (S2), og

at søge i datastrukturernes objekter baseret på den anden selektionsidentifikationsværdi (ID3), og hvis den anden selektionsidentifikationsværdi (ID3) findes, at lokalisere og hente et slutresultat (R2) lagret sammen med den anden valgidentifikationsværdi (ID3) som forbundne objekter i en tidligere iteration;

e) hvis slutresultatet (R2) ikke findes i undertrin (c) eller (d),

at søge i datastrukturernes objekter baseret på den første resultatidentifikationsværdi (ID2);

f) hvis den første resultatidentifikationsværdi (ID2) ikke findes i undertrin (e)

at udføre den første hovedberegning (P1) for at frembringe mellemresultatet (R1) og den første resultatidentifikationsværdi (ID2) som et digitalt fingeraftryk genereret af en hashfunktion af mellemresultatet (R1),

at lagre den første selektionsidentifikationsværdi (ID2) og mellemresultatet (R1) som forbundne objekter i datastrukturen; og

at udføre den anden hovedberegning (P2) for at frembringe slutresultatet (R2) og lagre den anden selektionsidentifikationsværdi (ID3) og slutresultatet (R2) som forbundne objekter i datastrukturen; og

g) hvis den første resultatidentifikationsværdi (ID2) findes i undertrin (e)

at hente mellemresultatet (R1) lagret sammen med den første resultatidentifikationsværdi (ID2) som forbundne objekter i datastrukturen, og

at udføre den anden hovedberegning (P2) for at frembringe slutresultatet (R2) og lagre den anden selektionsidentifikationsværdi (ID3) og slutresultatet (R2) som forbundne objekter i datastrukturen.

2. Fremgangsmåden ifølge krav 1 hvori det digitale fingeraftryk omfatter mindst 256 bits.

3. Fremgangsmåden ifølge krav 1 eller 2, der yderligere omfatter trinnet at selektivt slette dataposter indeholdende førnævnte tilknyttede objekter i datastrukturen baseret i det mindste på størrelsen af dataposterne.

4. Fremgangsmåden ifølge krav 3, hvori trinnet af at selektivt slette er konfigureret til at fremme sletning af dataposter indeholdende førnævnte første resultat (R1).

5. Fremgangsmåden ifølge krav 3 eller 4, der yderligere omfatter trinnet af at forbinde hver datapost med en vægtningsværdi, som beregnes som en funktion af en brugsparameter for hver datapost, en beregningstidsparameter for hver datapost og en størrelsesparameter for hver datapost, hvor brugsparameteren er en numerisk værdi, der repræsenterer, hvor ofte og hvor nyligt dataposten er blevet tilgået, beregningstidsparameteren repræsenterer den estimerede beregningstid for dataposten, og størrelsesparameteren repræsenterer størrelsen af datapost.

6. Fremgangsmåden ifølge krav 5, hvor vægtningsværdien beregnes ved at evaluere en vægtningsfunktion, som er givet af W = U*T/M, hvor U er brugsparameter, T er beregningstidsparameteren, og M er størrelsesparameteren.

7. Fremgangsmåden ifølge krav 5 eller 6, hvor værdien af brugsparameteren forøges, når dataposten tilgås, samtidig med at den reduceres eksponentielt som en funktion af tiden.

8. Fremgangsmåden ifølge et hvilket som helst af kravene 4-7, hvor trinnet med selektiv sletning er baseret på vægtningsværdien af dataposterne i datastrukturen.

9. Fremgangsmåden ifølge et hvilket som helst af kravene 4-8, hvor trinnet med selektiv sletning udløses baseret på en sammenligning mellem en aktuel størrelse af datastrukturen og en grænseværdi.

10. Fremgangsmåden ifølge et hvilket som helst af de foregående krav, hvor databasen er en dynamisk database, og således en database, som kan ændre sig på et hvilket som helst tidspunkt, og den første selektionsidentifikationsværdi (ID1) beregnes som en funktion af i det mindste det første selektionselement (S1) og datasættet (R0).

11. Fremgangsmåden ifølge et hvilket som helst af de foregående krav, hvori det første selektionselement (S1) definerer et sæt felter i datasættet (R0) og en betingelse for hvert felt, hvori mellemresultatet (R1) er repræsentativt for et udsnit af datasættet

(R0), hvor det andet selektionselement (S2) definerer en matematisk funktion, hvor en eller flere beregningsvariabler er inkluderet i mellemresultatet (R1) og en eller flere klassifikationsvariabler er inkluderet i mellemresultatet (R1), og hvori slutresultatet (R2) er en flerdimensionel terningedatastruktur, der indeholder resultatet af at styre den matematiske funktion på førnævnte en eller flere beregningsvariabler for hver entydig værdi af hver klassifikationsvariabel.

12. Et computertilgængeligt medie der har lagret et computerprogram der, når dette udføres af en computer, er indrettet til at udføre fremgangsmåden ifølge et hvilket som helst af kravene 1-11.

13. Apparat til udtræk af information fra en database, hvilket apparat omfatter indretninger til at udføre en sekventiel kæde af hovedberegninger omfattende en første hovedberegning (P1), som styrer et første selektionselement (S1) på et datasæt (R0), som repræsenterer databasen for at frembringe et mellemresultat (R1) og en anden hovedberegning (P2), som styrer et andet selektionselement (S2) på mellemresultatet (R1) for at frembringe et slutresultat (R2), hvilket apparat endvidere omfatter at hente det endelige resultat ved:

(a) at beregne en første selektionsidentifikationsværdi (ID1) som et statistisk entydigt digitalt fingeraftryk genereret af en hashfunktion af i det mindste det første selektionselement (S1);

(b) at søge i datastrukturens objekter for den første selektionsidentifikationsværdi (ID1), og hvis den første selektionsidentifikationsværdi (ID1) er fundet, at lokalisere og hente en første resultatidentifikator (ID2), der er lagret sammen med den første selektionsidentifikationsværdi (ID1) som forbundne objekter i en tidligere iteration;

(c) hvis den første resultatidentifikator (ID2) findes i undertrin (b),

at beregne en anden valgidentifikationsværdi (ID3) som et statistisk entydigt digitalt fingeraftryk genereret af en hashfunktion af i det mindste det andet selektionselement (S2) og den hentede første resultatidentifikator (ID2) og at søge i datastrukturens objekter for anden selektionsidentifikationsværdi (ID3), og hvis den anden selektionsidentifikationsværdi (ID3) er fundet, at lokalisere og hente et slutresultat (R2) gemt sammen med den anden selektionsidentifikationsværdi (ID3) som forbundne objekter i en tidligere iteration;

d) hvis den første resultatidentifikator (ID2) ikke findes i undertrin (b),

at udføre den første hovedberegning (P1) for at frembringe mellemresultatet (R1) og den første resultatidentifikationsværdi (ID2) som et digitalt fingeraftryk genereret af en hashfunktion af mellemresultatet (R1), at lagre den første selektionsidentifikationsværdi (ID1) og den første resultatidentifikationsværdi (ID2) som forbundne objekter i datastrukturen; og

at lagre den første resultatidentifikationsværdi (ID2) og mellemresultatet (R1) som forbundne objekter i datastrukturen,

at beregne en anden selektionsidentifikationsværdi (ID3) som et statistisk unikt digitalt fingeraftryk genereret af en hashfunktion af den første resultatidentifikationsværdi (ID2) og det andet selektionselement (S2), og

at søge i datastrukturernes objekter baseret på den anden selektionsidentifikationsværdi (ID3), og hvis den anden selektionsidentifikationsværdi (ID3) findes, at lokalisere og hente et slutresultat (R2) lagret sammen med den anden valgidentifikationsværdi (ID3) som forbundne objekter i en tidligere iteration;

e) hvis slutresultatet (R2) ikke findes i undertrin (c) eller (d),

at søge i datastrukturernes objekter baseret på den første resultatidentifikationsværdi (ID2);

f) hvis den første resultatidentifikationsværdi (ID2) ikke findes i undertrin (e)

at udføre den første hovedberegning (P1) for at frembringe mellemresultatet (R1) og den første resultatidentifikationsværdi (ID2) som et digitalt fingeraftryk genereret af en hashfunktion af mellemresultatet (R1),

at lagre den første selektionsidentifikationsværdi (ID2) og mellemresultatet (R1) som forbundne objekter i datastrukturen; og

at udføre den anden hovedberegning (P2) for at frembringe slutresultatet (R2) og lagre den anden selektionsidentifikationsværdi (ID3) og slutresultatet (R2) som forbundne objekter i datastrukturen; og

g) hvis den første resultatidentifikationsværdi (ID2) findes i undertrin (e)

at hente mellemresultatet (R1) lagret sammen med den første resultatidentifikationsværdi (ID2) som forbundne objekter i datastrukturen, og

at udføre den anden hovedberegning (P2) for at frembringe slutresultatet (R2) og lagre den anden selektionsidentifikationsværdi (ID3) og slutresultatet (R2) som forbundne objekter i datastrukturen.

DRAWINGS

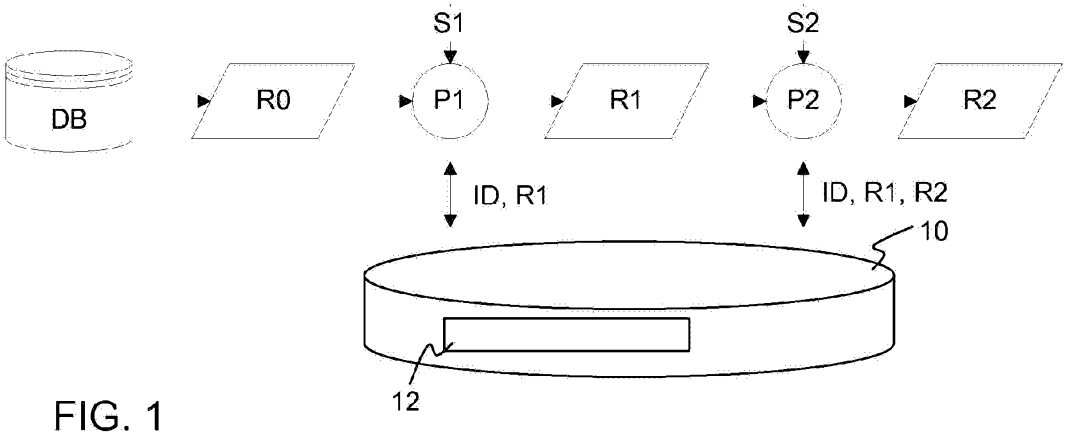

FIG. 1

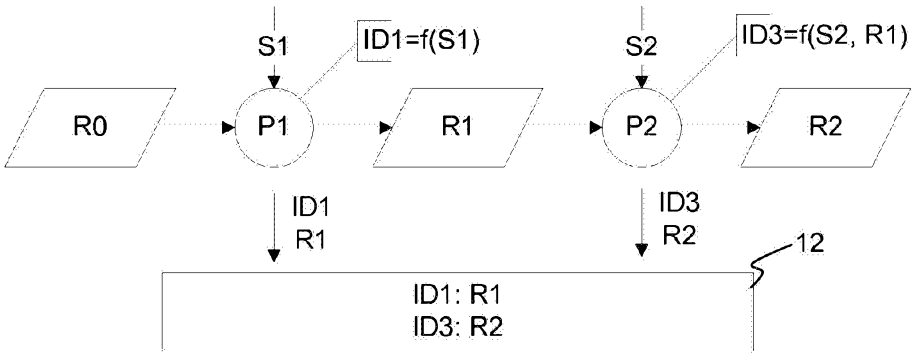

FIG. 2

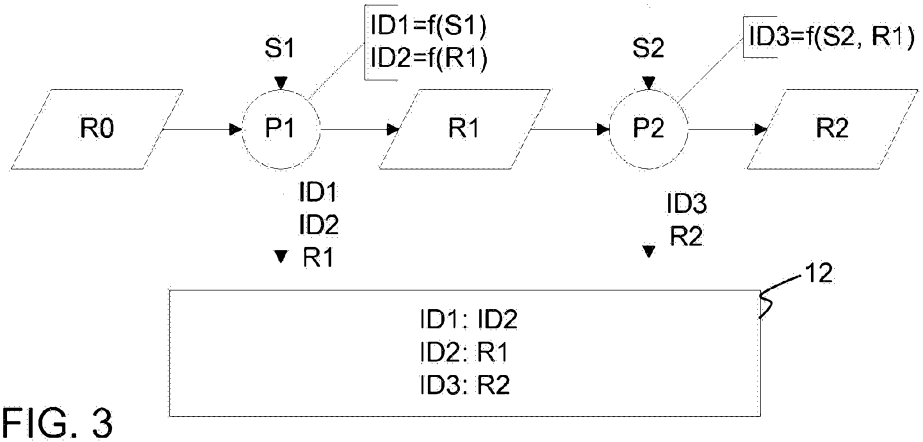

FIG. 3

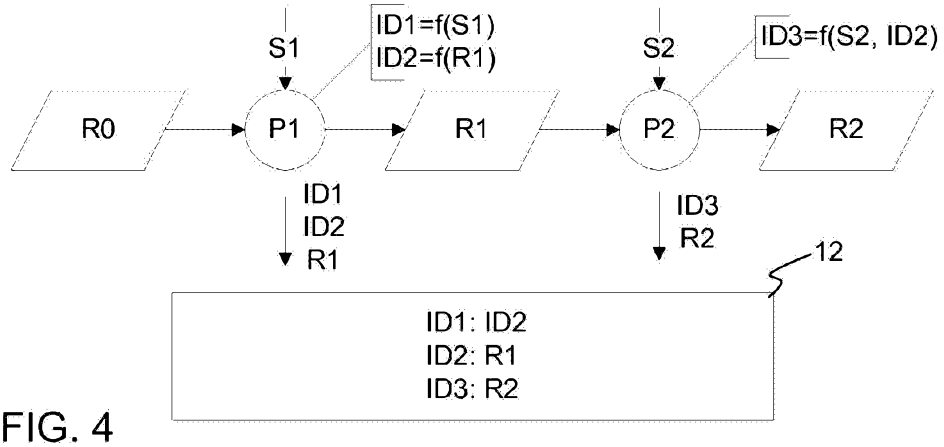
S1
ID1=f(S1)
ID2=f(R1)
S2
ID3=f(S2, ID2)
R0
P1
R1
P2
R2
ID1
ID2
R1
ID3
R2
12
ID1: ID2
ID2: R1
ID3: R2

FIG. 4

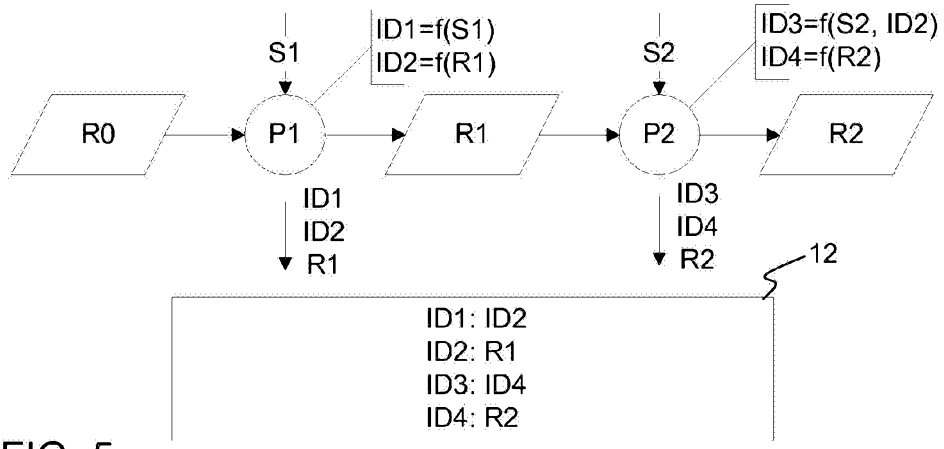
S1
ID1=f(S1)
ID2=f(R1)
S2
ID3=f(S2, ID2)
ID4=f(R2)
R0
P1
R1
P2
R2
ID1
ID2
R1
ID3
ID4
R2
12
ID1: ID2
ID2: R1
ID3: ID4
ID4: R2

FIG. 5

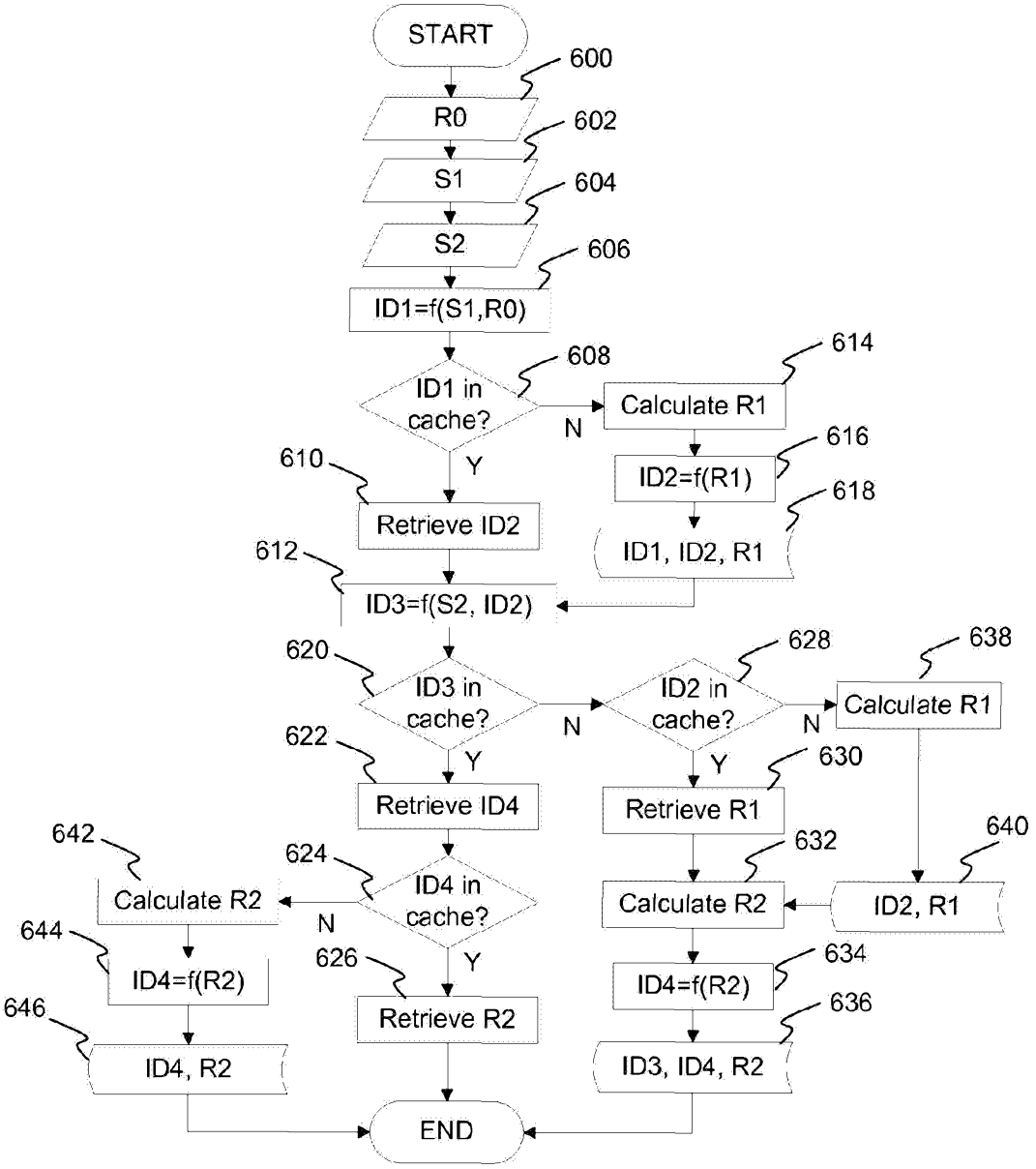
START
600
R0
602
S1
604
S2
606
ID1=f(S1,R0)
608
ID1 in cache?
N
614
Calculate R1
616
ID2=f(R1)
618
ID1, ID2, R1
Y
610
Retrieve ID2
612
ID3=f(S2, ID2)
620
ID3 in cache?
N
628
ID2 in cache?
N
638
Calculate R1
Y
622
Retrieve ID4
Y
630
Retrieve R1
640
ID2, R1
632
Calculate R2
642
Calculate R2
624
ID4 in cache?
N
644
ID4=f(R2)
626
Y
Retrieve R2
634
ID4=f(R2)
646
ID4, R2
636
ID3, ID4, R2
END

FIG. 6

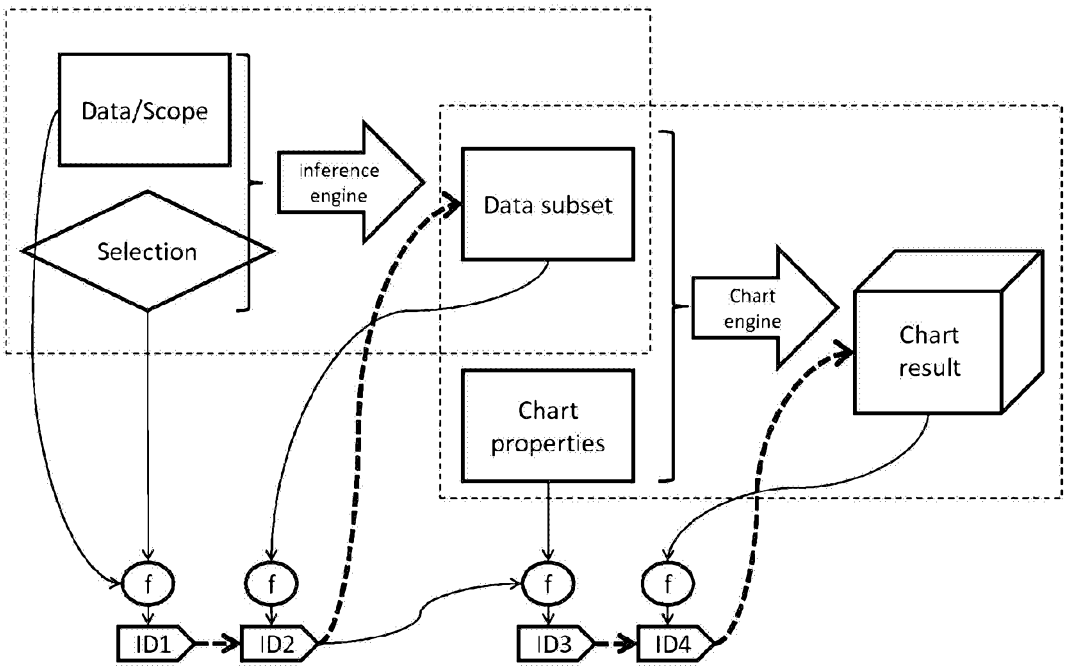
Data/Scope
Selection
inference engine
Data subset
Chart properties
Chart engine
Chart result
f
f
f
f
ID1
ID2
ID3
ID4

FIG. 7

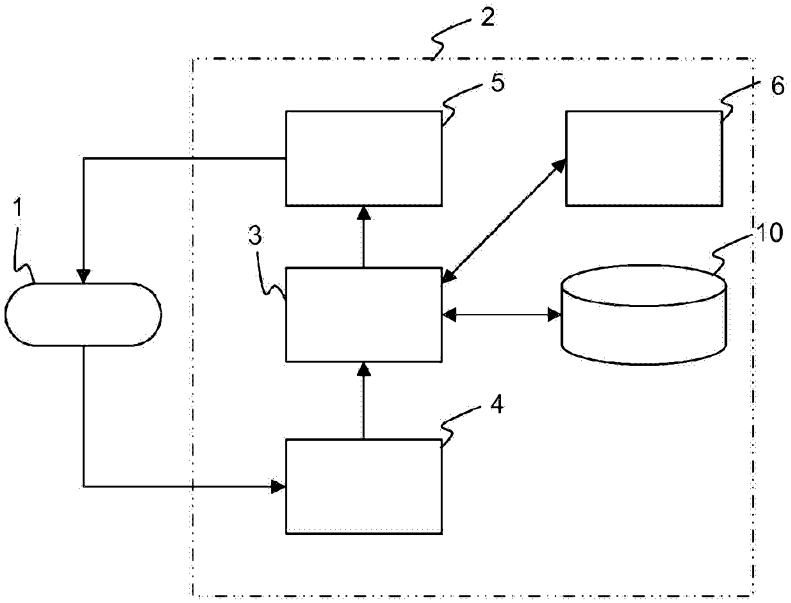
2
5
6
1
3
10
4

FIG. 8