InfoCharger Technology Briefing

InfoCharger version 2.4

Part No AS_IC_WP

InfoCharger Technology Briefing

Introduction - - - 5

Web Traffic Analysis - - - 5
A New Approach - - - 6
Requirements - - - 6
Storing Information - - - 8

Architecture - - - 8

Column Representations – CORE’s - - - 8
Data Encoding - - - 9
Importing data - - - 10
Aggregate Functions - - - 11

Analytics - - - 11

Histograms - - - 11
Cross-tables - - - 12
Results as COREs - - - 12
Zoom-in Functions - - - 13
Hierarchies - - - 14
Associations - - - 15
Ordered Associations (Sequence Analysis) - - - 18
Transformations - - - 18
Partitioning COREs - - - 20

Performance - - - 20

Memory Usage - - - 21
SMP or MPP Platforms - - - 21
Importing Data - - - 21
Exporting Data - - - 23
Paged COREs - - - 23
Clustering - - - 24

Supported Environments - - - 25

Introduction

Over the last 20 years there has been tremendous progress in computer hardware. Processor speed, memory size, communication speed, disk size and access times have all improved. With improved system performance has come more demand. For traditional Decision Support applications, the volume of data and the speed which users are demanding information is still outstripping system performance.

In addition, the Internet has caused an explosion in the volume of data businesses must handle. The digital profiles and transaction records created from connected customers and business partners enable the capture of valuable information not available before the Internet's widespread use. It is obvious that today's data analysis is highly demanding of the underlying database management systems.

"It is obvious that today's data analysis is highly demanding of the underlying database management systems."

Examples of such data analysis applications are OLAP, rOLAP and Business Information applications that may scan millions of records, or Data Mining applications that scan complete databases many times.

Interestingly, database technology has not undergone major changes in the way data is stored and accessed. The model that is still used is disk-centric and row-based. Even speciality OLAP products and database management systems which have implemented OLAP features can be seen as simply workaround solutions for analysing large volumes of data.

Web Traffic Analysis

The use of the Internet as a medium for doing business is growing exponentially. Most major organizations have a web presence and many of these sell their products on-line through their websites. As these web-based activities become more important, the host organization's investment in its websites increases. Every organization has to make sure its website is reliable, continually available and that it provides the right level of information and services to its customers. How does an organization know that its website is achieving its business goals? Who are the customers coming to the site? How do they use the site? What do they like and dislike? The answers to these questions lie in web traffic analysis.

"Every organization has to make sure its website is reliable, continually available and that it provides the right level of information and services to its customers."

Analysis of customer behaviour is key to the success of any business. Analysis gives the business manager the means to make informed decisions and understand the results of those decisions. By combining the analysis of web traffic data with other corporate data, managers can get a more complete picture of the website and its role in the success of the business. For example, by integrating customer satisfaction data with the website traffic data, business managers can see if calls to customer support are decreasing due to improved web-based customer support.

A few years ago the web traffic analysis market barely existed, and today it is a multi-million dollar segment of the e-business industry. A recent report by International Data Corporation estimates that the web traffic analysis market will break $100 million by 2002. Already the volume of web traffic data in some corporations exceeds their analytical capacity. InfoCharger's innovative approach to analysis on this scale means that these problems can be attacked successfully giving real business answers in the time required.

"Already the volume of web traffic data in some corporations exceeds their analytical capacity."

Copyright © 2000 by Accrue Software Inc.

A New Approach

"InfoCharger is very well positioned as the database engine of choice for decision support solutions."

With new software technology of an innovative nature, and designed to store and access data extremely quickly, InfoCharger is very well positioned as the database engine of choice for decision support solutions.

This paper introduces InfoCharger and describes the design principles and pioneering opportunities when used for Data Mining or On-line Analytical processing tasks.

The technology is based on a few observations:

- Many applications access few columns per row, but potentially access many rows, typically to obtain aggregate information like 'counts' or 'sums'. This is a very clear case for data analysis applications, but many OLTP applications access few rows after an initial insert of the data, and updates affect even fewer columns.
- Data that is accessed frequently can be stored in memory; potentially in an encoded format, as this allows for efficient data storage and increased performance
- Performance is critical. There is a need to improve performance for data analysis applications by orders of magnitude; for OLTP there is a need for a more modest and steady increase in performance.

InfoCharger focuses on data analysis applications with a need for very high performance, so complex analytical tasks can be done interactively.

Requirements

Current data mining tools can find patterns in data that is stored within a single table. To store relevant data in this format, pre-processing is required. This typically includes the following components:

- Storing additional columns that identify higher hierarchical levels of data, e.g. one may add a column that groups individual products to a product-group or that groups cities to a region.
- Storing additional columns that summarize dynamic behaviour. One can think of columns that specify the number of transactions that a customer has done, the total amount for the transactions, etc. Note that in most cases the dynamic behaviour of systems (like customers or networks) predicts future behaviour better than static properties.
- Transforming original data, for example to add the contents of several revenue columns into a single one.

"InfoCharger greatly reduces the need for extensive pre-processing"

Pre-processing greatly complicates the life cycle of data mining as one has to re-do the pre-processing phase if new insights lead to the need to include additional data. InfoCharger greatly reduces the need for extensive pre-processing for data mining while adding functionality for analysis of dynamic data, such as financial transactions.

OLAP and business analysis tools are not limited to single table access, but expect 'star' or 'snowflake' schemas. Other requirements are very similar to data mining requirements and include functions like aggregation at various levels in the hierarchy to support 'slice-and-dice' and 'drill-down' functions.

Copyright © 2000 by Accrue Software Inc.

The following main features are available in InfoCharger to enable efficient data mining and other types of analytical processing.

- **Aggregations**

InfoCharger is able to build aggregates in one or more dimensions at very high speeds, thus eliminating the tedious and expensive task of maintaining many different aggregation tables. Maintaining aggregate tables is one of the major costs in many DSS deployments in terms of personnel and disk space

"InfoCharger is able to build aggregates in one or more dimensions at very high speeds"

- **Zoom-in Functions**

InfoCharger allows for defining subsets of the data based on simple or complex predicates. Aggregation functions can be executed using these defined subsets. One can also refine the subsets, simply by adding further predicates.

- **Hierarchies**

InfoCharger supports star and snowflake-model schemas, so there will no longer be a need to de-normalize data models. In addition, new dimensions can be added without the need to reload the original data.

- **Associations**

InfoCharger supports the notion of 'clusters', multiple records that form one group, within which associations can be detected.

- **Transformations**

InfoCharger will be able to create new columns that contain data that is constructed using the original columns.

- **Continuous Variables**

InfoCharger will be able to deal with continuous variables, for example by 'binning' values into a limited number of value ranges. This eliminates the need to reduce the number of different values in a numeric column during pre-processing.

- **Security**

Access to InfoCharger will only be granted after supplying managed user identification and password details.

Copyright © 2000 by Accrue Software Inc.

Architecture

Storing Information

"In classic database management systems tables are implemented as individual files."

In classic database management systems tables are implemented as individual files. A file consists of rows (a.k.a.records), and each row consists of a number of columns. The structure of each row is identical. Typically, the physical organization of the file consists of blocks, where each block may contain a number of records. In some implementations, records may span multiple blocks. If this complexity is ignored, all data for an individual row is co-located. This is a great advantage for many applications, as these need access to many columns for single records. It is a disadvantage in some other cases: if few columns are selected from wide rows, a lot of information is read into memory in vain.

Also, the structure of the files gets complex, as information about the physical structure of the files needs to be maintained at various levels:

- At the block-level one needs to maintain index structures and pointers between blocks (e.g. a B-tree);
- Within a block one needs to maintain pointers to the beginning of individual and potentially variable-length records;
- Within a record one needs to be able to identify individual and potentially variable-length columns.

This complexity causes a significant overhead, especially for operations that need to access few columns and many or even all rows in the table.

"Data analysis tools have a need to access individual columns very efficiently"

Data analysis tools have a need to access individual columns very efficiently, orders of magnitude faster than the speeds that traditional database management systems can provide currently. This requires a new data structure based on Column Representations (**CO**lumn **RE**presentations – **CORE**s).

Column Representations – CORE's

Let us assume the following table with 4 columns (Figure 1). InfoCharger stores each of the columns in a separate list. We call this list a **s**imple **C**Olumn **RE**presentation or **sCORE**. An sCORE is implemented as a simple file on disk.

Figure 2 shows the sCOREs for columns 1 and 4. The table will consist of four separate COREs that are stored as four separate files on disk.

"A random memory-lookup of the nth record is a very efficient operation"

Each record can still be constructed from the COREs: in order to re-construct the nth record we simply join the nth entries from each of the COREs. The width of each entry in a CORE is fixed, so a random memory-lookup of the nth record is a very efficient operation. The width of an entry is dependent on the maximum size of the values in the column. For example, in column 2, the only two values are 0 and 1 so the width of each entry in a CORE can be as low as a single bit. Sometimes, such a CORE is referred to as a bitmap.

Copyright © 2000 by Accrue Software Inc.

Col-1	Col-2	Col-3	Col-4
3	0	East	12213
4	0	West	74545
6	1	East	12365
1	0	East	67823
9	1	East	986
3	1	East	65231
4	0	North	89621
6	1	West	764213
2	0	East	87
7	0	East	674
4	0	South	76546
5	0	East	873409
2	1	West	57556
2	1	East	56513
1	0	East	13254

Figure 1 - Simple Table

Col-1
3
4
6
1
9
3
4
6
2
7
4
5
2
2
1

Col-4
12213
74545
12365
67823
986
65231
89621
764213
87
674
76546
873409
57556
56513
13254

Figure 2 - Simple Column Representation of Figure 1

Data Encoding

Working with large amounts of data presents unique challenges in the actual physical storage and placement of the data. Reading the data to place it into a CORE gives us the opportunity to reduce the physical storage requirement of the data. This is accomplished through data encoding.

"Working with large amounts of data presents unique challenges in the actual physical storage and placement of the data."

Copyright © 2000 by Accrue Software Inc.

Simple COREs

“Simple COREs use no encoding, simply storing the actual values”

The number of different values in a column or a CORE is the Unique Entry Count (UEC). If the UEC is not much smaller than the row-count of the table (e.g. Col-4 in Figure 1), a simple CORE (COREs) may be used to store the column data. These COREs use no encoding, simply storing the actual values found in the column directly in a CORE.

Numerically Encoded COREs

In many cases the UEC is small, even though the column may be wide, such as a large integer or a text-array like that in Col-3 in Figure 1. In such cases one may add a value-table (VAT) to a CORE, and store encoded values in the CORE, which we now call a numerically encoded Column Representation (nCORE). Let’s consider Col-3 in Figure 1. We can code the Char(5) column for the region-name in just 2 bits as there are only four different values. It reduces the column-width by a factor of 20.

nCORE Region (Col-3)
1
2
1
1
1
1
3
2
1
1
4
1
2
1
1

VAT Region
1,East
2,West
3,North
4,South

Figure 3 - nCORE for Col-3 with Value Table (VAT)

Importing data

“The complexity of building VATs or encoding the data is completely hidden from the user”

One can load or add rows into InfoCharger tables. The complexity of building VATs or encoding the data is completely hidden from the user: one must only specify if COREs are simple or encoded. For encoded columns one may optionally specify the width of the nCORE. If the width is not specified, InfoCharger will choose the optimal width. A more detailed discussion on importing data is provided in “Importing Data” on page 21.

Copyright © 2000 by Accrue Software Inc.

Analytics

Aggregate Functions

Data mining algorithms depend largely on frequency-distributions, or COUNT functions in SQL terminology. However, some data mining tools use other aggregate functions as well. Applications such as rOLAP tools will always be interested in aggregate functions such as Sum, Average or Standard Deviation.

"Applications such as rOLAP tools will always be interested in aggregate functions such as Sum, Average or Standard Deviation."

In OLAP, the logical view of the data is an n-dimensional cube. One would aggregate data for several dimensions to reduce the number of dimensions that are presented to the users. Also one can zoom into areas of interest in the database, e.g. one can zoom in on a product group and on a particular month. The main difference with the requirements for data mining is the need to examine a variety of performance indicators such as amounts, profits and quantities. Data mining operations typically deal with counts only. In SQL terminology there would be a need to support operations like SUM, AVERAGE, Standard DEViation, MIN, etc. beyond COUNT operation.

Histograms

Histograms are used to obtain information that is aggregated in one dimension. Examples in SQL-terminology are:

- "SELECT COUNT(*) from .. GROUP BY Product"
- "SELECT SUM(PRICE) from .. GROUP BY Product"

The data structures used in InfoCharger allow for very fast calculation of histograms: on a single 300Mhz Pentium II processor one can build histograms at a speed of around 50 million rows per second. This means pre-aggregation for performance reasons is no longer required. Instead, a flexible and inexpensive solution is available.

"The data structures used in InfoCharger allow for very fast calculation of histograms"

Before starting any aggregation, all required COREs are read in from disk. A memory management function keeps COREs in memory based on a least-recently-used algorithm.

The number of entries for a histogram will equal the UEC. Continuous variables may have a very high UEC that complicates further analysis or visualization. For such columns, typically stored in a simple CORE, one can reduce the UEC by Binning (see "Transformations" on page 18).

Figure 4 illustrates how histograms (and n-dimensional crosstables) can contain functions such as SUM and COUNT on numeric attributes stored in COREs.

Copyright © 2000 by Accrue Software Inc.

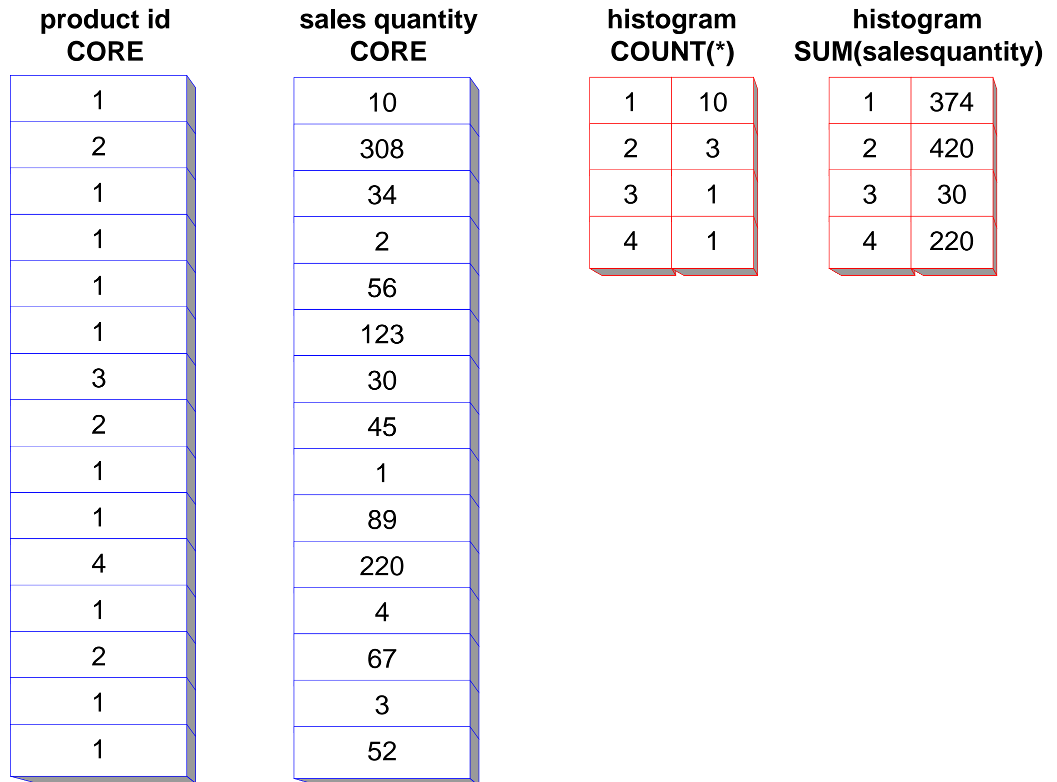

Figure 4 - Histograms of functions on numeric attributes stored in COREs.

Cross-tables

"On a single 300Mhz Pentium II processor, InfoCharger builds cross-tables at around 10 million rows per second."

A cross-table is an n-dimensional histogram. In the following example, the cross-table for a COUNT-function between Col-2 and Col-3 of Figure 1 is shown. On a single 300Mhz Pentium II processor, InfoCharger builds cross-tables at around 10 million rows per second.

	0	**1**
East	6	4
West	1	2
North	1	0
South	1	0

Figure 5 - 2D Cross-table for Col-2 and Col-3 of Fig 1

Results as COREs

Histograms and cross-tables can be stored as COREs in the same way as for original source data.

Copyright © 2000 by Accrue Software Inc.

Zoom-in Functions

So far, we have considered analysing ‘full’ COREs, i.e. all values in a CORE are included in aggregates. In many cases one may want to zoom into specific areas of the COREs. For example, one may only be interested in certain values or certain value ranges. If one needs to access such areas in the COREs, one builds a so-called **pCORE** (**p**redicate **CORE**) that contains pointers to the relevant entries in the full COREs. The pointers are the sequence numbers of the records, which we call the *RowIds*.

Figure 6 shows the COREs for Col-1 and Col-3 of Figure 1. The predicate CORE pCORE1 contains pointers to all COREs for the table that fulfil a given predicate: in this case “where col-1 is between 4 and 7”. This type of pCORE is implemented as a simple list of pointers each of 32-bit width. As an alternative, pCORE2 represents the same result in a bitmap format, a binary 1 indicating that the row at that position has been selected. Both types of pCORE are supported in InfoCharger.

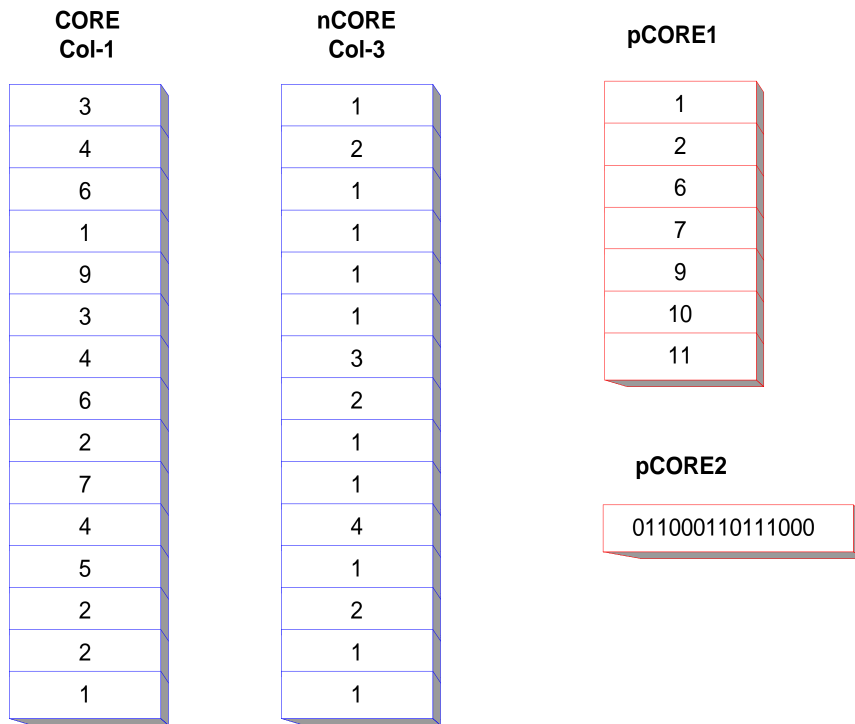

Figure 6 - pCOREs derived from simple COREs

Predicate CORE performance

pCOREs can be built at a very high speed. For simple predicates, one can build a predicate CORE at a speed of around 10 million rows per second on a single 300Mhz Pentium II processor.

Predicate COREs use 32-bit RowIds to point to the original COREs. If the selectivity for predicates is low, the pCOREs will get quite big. The lifetime of a pCORE is limited to the user-session in which it was created. During the creation of histograms or cross-tables the indirection causes some overhead, but the number of entries in the COREs that are processed is reduced so overall performance will usually improve.

“The lifetime of a pCORE is limited to the user-session in which it was created.”

Copyright © 2000 by Accrue Software Inc.

Hierarchies

"It is now possible to implement grouping at any level in a hierarchy"

VATs are used to decode the values stored in nCOREs. One can add hierarchies to this structure by including grouping-COREs. The grouping codes point to VATs that are used to describe higher levels in the hierarchy. In the example (Figure 7), the original table contains city names. In an nCORE structure these were encoded as integers. The original city names can be found in the City-VAT. The cities are grouped in regions, in the City-Region-VAT. This is reflected by a pointer to the region-VAT. It is easy to see that such structures can be implemented at low memory costs. In addition, it is now possible to implement grouping at any level in a hierarchy.

Note that in database terminology there needs to be a 1:n relation between the tables to define a hierarchical relationship.

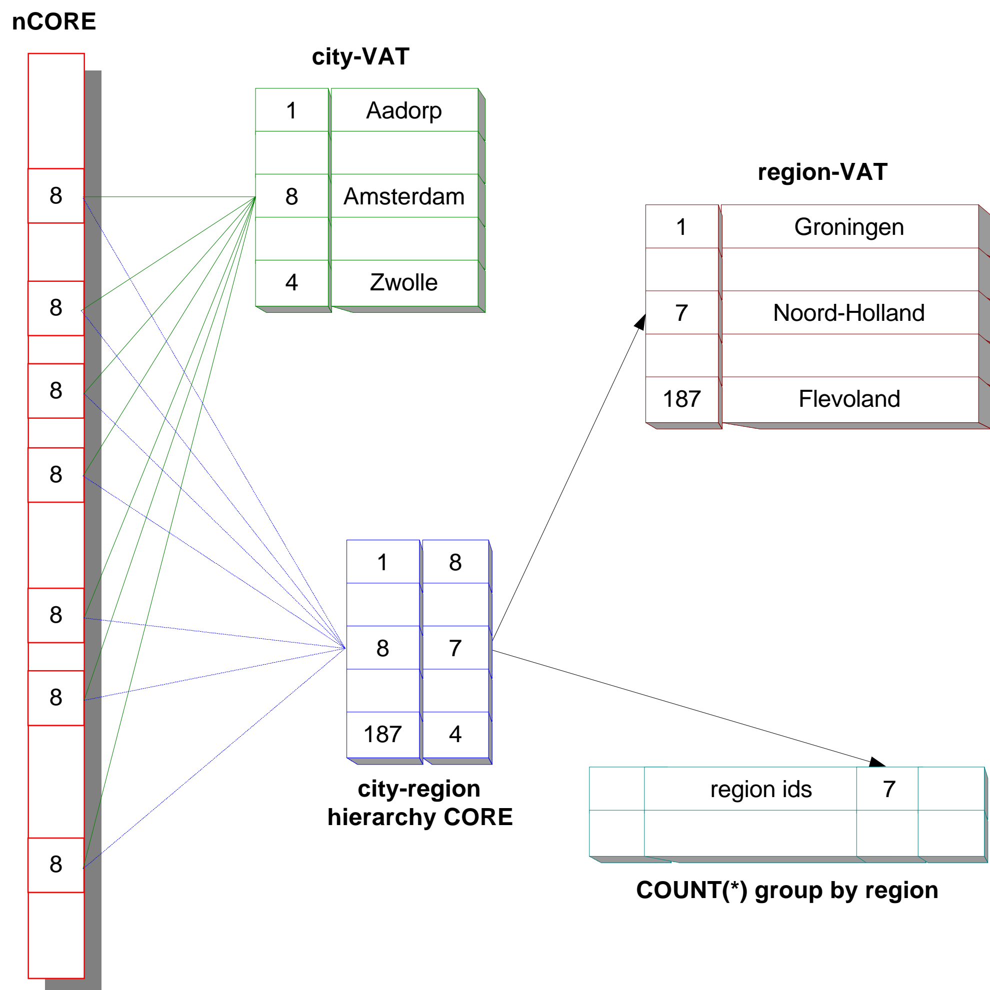

Figure 7 - Hierarchical Group of Cities into Region

The example shows a single, simple hierarchy. This can be expanded to multiple and more complex hierarchies.

As a further example, one could also group cities by size. The structure can be used to implement star models or snowflake models as they are used in OLAP.

Copyright © 2000 by Accrue Software Inc.

Hierarchies can be accessed transparently from the base-table. In the given example, the Region will appear as a separate, virtual column with its own CORE-id and its own name.

Histograms and cross-tables can still be built very efficiently. One simply creates an array with an entry for every code in the CORE at the appropriate aggregation level. Next one scans the CORE, and for every entry adds one to the appropriate entry in the histogram array.

Associations

Association may give users an insight into some of the most interesting aspects of business, and may help answer questions like:

- What products are often bought together?
- Which products were bought the most with each Banner Ad?
- When buying a particular product what is the most likely next purchase?
- What is the most purchased product for customers spending over $100?

Traditionally in data mining one can only make associations between columns that are stored in a single row. If a retailer sells 30,000 different articles and wishes to analyse the contents of baskets, he could build one row per basket with 30,000 columns that indicate whether an item was present or not. Per item, he could either store a binary switch or maintain a count (and potentially a price). He could also count how often individual products and combinations of different products occur in a basket. Such analysis is typically referred to as *basket analysis*.

"Traditionally in data mining one can only make associations between columns that are stored in a single row."

Needless to say, such a mechanism either takes a lot of memory, or if the data is stored in a compressed format, takes a lot of CPU cycles.

As an example let's look at the following table comprising of 7 baskets and 9 different products. Note that this matrix will become very large and very sparse if the number of products is 30,000 instead of 9:

	1	2	3	4	5	6	7	8	9
1	-	X	X	-	-	X	X	-	-
2	-	-	-	-	X	X	-	-	X
3	X	X	-	-	-	-	X	X	-
4	-	-	-	-	X	-	-	-	X
5	-	-	-	X	-	-	X	-	X
6	-	X	-	-	-	-	X	-	-
7	-	-	X	-	-	X	-	-	-

Figure 8 - Example basket table

Simple Associations

InfoCharger will store data in a relational manner for associations. There may be a basket table with general information per basket, but in addition there will be an item table that contains a record for every product that is present in the basket. Each basket is identified by a basket-id, and the basket-id is propagated to the item table.

"InfoCharger will store data in a relational manner for associations."

Copyright © 2000 by Accrue Software Inc.

For basket analysis it will now be necessary to group the records in the item table by basket-id.

Once this has been done, one can again count how often individual products and combinations of different product occur in a basket. Figure 9 shows how the same data from Figure 8 is now stored in two nCOREs where the first one contains the coded basket-id and the second one contains the coded product-id (in fact the data will be stored in this way in most relational databases anyway). Note however that it is now possible to enter multiple rows for the same products in a basket, indicating that one has purchased more than one copy of the product.

basket-id nCORE	product-id nCORE
1	2
1	3
1	6
1	7
2	5
2	6
2	9
3	1
3	2
3	2
3	7
3	8
4	5
4	9
5	4
5	7
5	9
6	2
6	7
7	3
7	6

Figure 9 - Example basket table in COREs

The association between the products from Figure 9 is shown in Figure 10. Note that the diagonal is not empty in this case.

For a two-dimensional association table the cells (m,n) and (n,m) have the same meaning. One can thus return only those cells with m>=n. The cells (n,m) with m=n is the diagonal of the association table; it will only contain non-zero values if a group contains records of the same type, e.g. if a basket may contain multiple records for the same product, for example product 2 occurs twice in basket 3. An associated problem is the combination of product 1 and product 2 in basket 3. There are 2 such combinations. One can choose to either count this as:

"For a two-dimensional association table the cells (m,n) and (n,m) have the same meaning."

Copyright © 2000 by Accrue Software Inc.

- 1 occurrence of the combination - how many baskets contain combinations of product 1 and 2, or
- 2 occurrences of the combination – how many combinations of product 1 and 2 can be found in the baskets.

	1	2	3	4	5	6	7	8	9
1	0	-	-	-	-	-	-	-	-
2	1	1	-	-	-	-	-	-	-
3	0	1	0	-	-	-	-	-	-
4	0	0	0	0	-	-	-	-	-
5	0	0	0	0	0	-	-	-	-
6	0	1	2	0	1	0	-	-	-
7	1	3	1	1	0	1	0	-	-
8	1	1	0	0	0	0	1	0	-
9	0	0	0	1	2	1	1	0	0

Figure 10 - Association table for example basket data

By comparing the actual counts of combinations of products with calculated *a priori* chances, one can find interesting patterns.

The result matrix may get very large, e.g. for 30,000 products it would have 900 million entries. If each entry was 32 bits wide, this would require 3.6 GB. This is a lot even for current memory sizes. One can however calculate the first n% of the table, where one would take all columns for the first n% rows of the matrix. All other entries are ignored. When finished with the first n% of the rows and when the results are given to the front-end, one can process the next n% etc. Alternatively, one may wish to initially zoom in on some items of particular interest, by first building a histogram of the data to see which items are most frequent, or using some other analysis, such as item value. In this way one greatly reduces the size of the table that is formed.

For an efficient implementation one would have to guarantee that:

- All rows for a given basket are stored in a single partition;
- The rows within a partitions are physically grouped by the grouping column, in the example the basket-id. If data is not physically grouped this way, the data can be re-loaded using the clustering facility (see “Clustering” on page 24).

Another consideration is the amount of data that will be returned to the front end. As a first optimisation, one may only return the cells that have a non-zero value. The second optimisation would be to return only those values that exceed certain limits. This ‘support’ limit is implemented: it excludes all entries in the matrix that are rare (i.e.count < n).

In all cases, one can still use hierarchies to group the data, either on both axes, or a single axis. If an initial two-dimensional table is built, one can then zoom into the data. For example one may have found that buying product X increases the chance that product Y is bought. One may next want to check if combinations of product X with other products will increase this chance even further:

	1	2	3	4	5	6	7	8	9
2,7	1	-	1	0	0	1	-	1	0

Figure 11 - Zooming into a product combination

In this example we would examine the combination of products 2 and 7 and see if there are any combinations of 3 products that occur more frequently than can be expected *a priori*. This does not seem to be the case here. The zooming can be done for individual product combinations, or for a list of product combinations. For each product combination the number of counts that will be calculated equals the number of products minus 2.

Adding hierarchies to the basket database would allow for analysis based on customers, day-of-week or shops rather than on single baskets. An efficient implementation of such features is non-trivial. It will require all baskets for a single customer (or shop or…) to be grouped together. It can be achieved by building index structures on top of the COREs.

Ordered Associations (Sequence Analysis)

“If one wants to examine if a particular event or behaviour can be predicted by a combination of other events or behaviours, one needs to take the sequence of events into account.”

In many cases associations may be more complex. Suppose a table holds ‘clickstream’ data for a website. If one wants to examine if a particular event or behaviour can be predicted by a combination of other events or behaviours, one needs to take the sequence of events into account.

For a two-dimensional association table, the cells (m,n) and (n,m) now have a different meaning. The cell (m,n) contains the number of times that an event of type m occurred before an event of type n.

One can limit the distance between events: e.g. one could count associations only if event n immediately follows event m, or one could consider association only if the distance between m and n is between 2 and 6 events. Note that this distance is not a direct indication of elapsed time.

Another example where distance between events can be important is in text searches: if words are events that are grouped in a document, one may maximize the distance between words that are used as a search criterion. To take this a step further, one can build histograms that reflect the distance distribution between two types of events.

Transformations

“In some cases, source data is not ideal for immediate analysis”.

In some cases, source data is not ideal for immediate analysis. One could for example have to clean the data, or to build new columns that have a clearer business meaning than the original data.

Transformations build new COREs based on existing ones. One can for example build a column called ‘profit’ from the columns ‘purchase price’ and ‘sell price’, by specifying “Profit = Sell price - Purchase price”.

Transformations may include virtual columns (see “Hierarchies” on page 14.). Transformations work on all individual rows within a table.

Copyright © 2000 by Accrue Software Inc.

COREs

Purchase Price			Sell Price				Profit
90			100				10
100			125				25
27			33				6
30			40				10
42			51				9
63			70				7

Figure 12 - Transformation (Profit = Sell Price - Purchase Price)

Transformations can contain arithmetic and mathematical functions like ‘sum’, ‘exponent’, ‘log’, ‘absolute’,‘round’, ‘ceiling’ and ‘random’ but also text manipulation functions like ‘mid’ and logical functions like ‘if’. Many functions are similar to Microsoft Excel™ functions. Functions work on simple COREs and nCOREs. Depending on the CORE type, the InfoCharger engine will retrieve the data either from the CORE directly or from a VAT. Virtual columns as defined in hierarchies can also be used.

“Transformations can contain arithmetic, mathematical and text manipulation functions”

On creating a transformation, one has to specify the name of the new CORE and the CORE type (simple CORE vs. nCORE). The definition of the transformation is stored for future reference.

Sampling

One can use the Random function to generate a random value between 0 and 100 for each row in a table. Next, one can use the create pCORE function to create a random sample of n% of the tables by specifying “where Random () > n”.

Binning

Binning is a special transformation used for automatic grouping of high-UEC COREs. In some cases, COREs with numeric values have a high UEC, and one does not want to reduce the UEC before or while loading the data into the COREs (such columns would be good candidates for simple COREs rather than for nCOREs). However, one may still want to present data as manageable histograms, i.e. histograms where the number of entries is limited to something like 100.

“Binning is a special transformation used for automatic grouping of high-UEC COREs.”

Special transformation functions - binning - generate a new CORE with a pre-defined UEC. The values of the original CORE are mapped into the new CORE using equal-width histograms.

In order to create the binning CORE one would have to specify the desired number of entries, the method (equal-width or equal-height) and the axis (linear or logarithmic).

Copyright © 2000 by Accrue Software Inc.

Performance

Partitioning COREs

"COREs can be split up over n partitions to allow parallel processing."

COREs can be split up over n partitions to allow parallel processing. One simply stores the first x rows in the first partition, the second n rows in the second one, etc. The only important rule is that all columns are partitioned in exactly the same way, so combining data for a single row will always be a local operation.

All partitions of a given CORE will share the same VAT, this guarantees that the encoding in all partitions is identical.

The process of loading partitioned COREs has some interesting aspects that are discussed in "Importing Data" on page 21.

Building aggregates is now done in two steps:

- First, a separate engine for every partition will build the partial aggregates. These engines will run in parallel;
- Second, the partial aggregates per partition will be consolidated to one aggregate result that will be returned to the client. For Count or Sum operations the consolidation process is as simple as adding the results of corresponding cells.

Usually result sets are small so the time to build the partial aggregates far exceeds the time to consolidate the results. This guarantees good scalability.

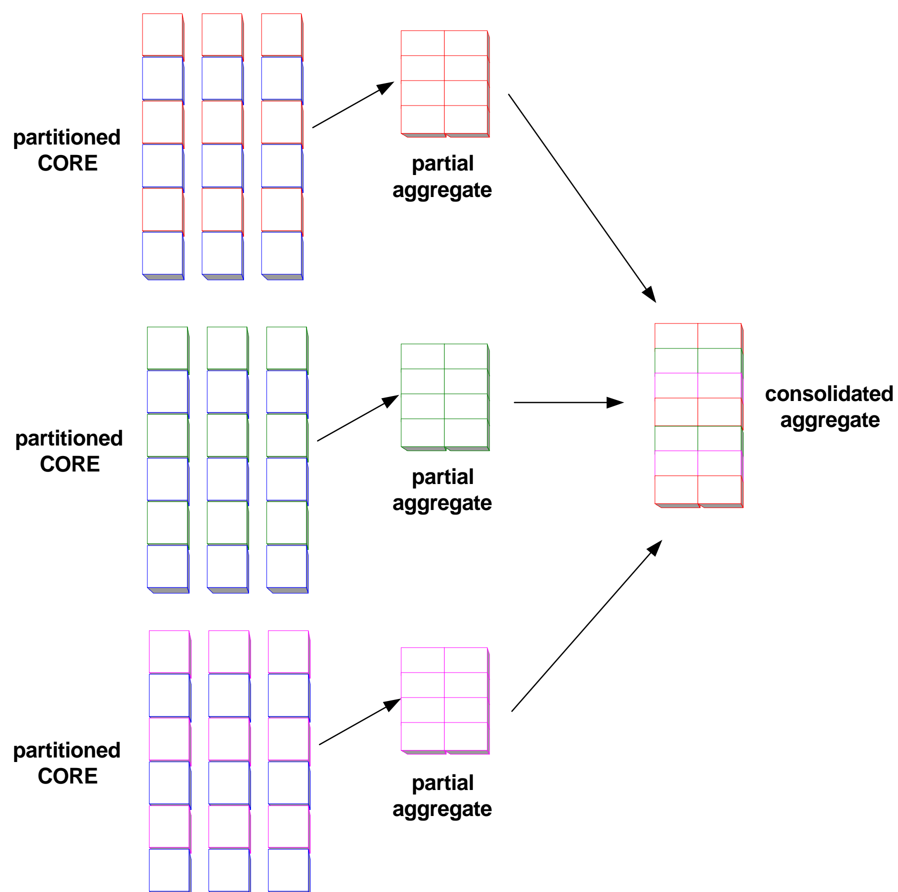

Figure 13 - Parallel Aggregation & Consolidation

Copyright © 2000 by Accrue Software Inc.

Memory Usage

The efficient memory usage of COREs and the possibility to partition COREs over CPUs allows InfoCharger to store full COREs in memory. Assume a table with 10 million rows and 100 columns that can each be re-presented by 8-bit wide nCOREs. The total memory consumption for the nCOREs would only be 1GB. One would of course need to store the value-tables as well, but these would usually take only a small percentage of the space required for the nCOREs.

"The efficient memory usage of COREs and the possibility to partition COREs over CPUs allows InfoCharger to store full COREs in memory."

Dynamic objects such as pCOREs and result tables will also take significant space, especially pCOREs with low-selectivity predicates (4 bytes per entry). As a rough guide, one should allow all static objects (COREs and VATs) that are accessed regularly to occupy no more than half of the memory available. InfoCharger will use the rest of the memory for pCOREs, result tables etc. For very large tables, one should calculate the memory requirements in more detail.

InfoCharger reads complete COREs into memory. A large buffer, referred to as cache, is used to store multiple COREs. A least-recently-used algorithm is used to remove the oldest COREs from the cache if there is not enough space to load new ones. The minimum size of the cache should allow for storing all data that is needed for one (aggregation) operation.

As a side remark, all COREs for a given table with a given width all have the same size. 1-Bit wide COREs will have the smallest size, other COREs will have a size that is n times as big, where n = 2,4, 8, 16 or 32. This greatly helps avoiding memory and disk fragmentation.

SMP or MPP Platforms

InfoCharger is designed to work on single CPU systems, SMP configurations, MPP configurations or clusters of SMP configurations. The most important characteristic to consider is the availability of main memory. With enough memory, InfoCharger will be able to saturate CPU's. In SMP configurations, one should make sure that the amount of memory that is available is enough to load the relevant data. If one cannot install enough memory per node, it will be better to add nodes to a cluster than to increase the number of CPU's per node.

"InfoCharger is designed to work on single CPU systems, SMP configurations, MPP configurations or clusters of SMP configurations."

Importing Data

InfoCharger COREs are typically populated from systems where data is stored in a traditional row-oriented way. Given the amount of data that may need to be analysed, load processing must be highly efficient. At present, InfoCharger imports data via a full load. Incremental load will be delivered in the future.

"InfoCharger COREs are typically populated from systems where data is stored in a traditional row-oriented way."

In order to store data in InfoCharger several steps must be taken:

- Rows must be read from some external source.
- Distribution: rows must be distributed to the correct partition. Note that the original data may not be partitioned, or may be partitioned differently.
- Encoding: rows must be split up in individual columns that will be stored in separate COREs, and for encoded COREs the individual entries per column must be encoded.

Copyright © 2000 by Accrue Software Inc.

- Store: the encoded values (or non-encoded values for simple COREs) will be written to disk and potentially be stored in memory as well.

"Standard import modules are available to load data from flat files, NonStop SQL and Oracle."

InfoCharger provides general facilities to load data that cover the three steps of the above list. One can develop import modules that use these facilities. Standard import modules are available to load data from flat files, NonStop SQL and Oracle.

The semantics for loading data are quite simple: all COREs will be made available for processing when all COREs have been loaded completely. During the load process, COREs may be in an inconsistent state, and will not be available for processing.

Multiple input streams may generate data concurrently; these streams can be processed in parallel.

The Distribution Function

Distribution of data over partitions can be done in various ways:

- Based on key-ranges, the actual value of a given column determines in which partition the data is stored. E.g. all rows with a customer number less than 1 million end up in the first partitions, with a value between 1 and 2 million end up in the second partition etc.
- Based on a round-robin scheme, the first record to be inserted ends up in the first partition, the second in the second etc. If a record is stored in the last partitions, the next one will be stored in the first one again.
- Based on a clustered round-robin scheme. This is very similar to the previous option, but records are now clustered. For example, data is delivered for sold items per basket. All rows for the first basket will now be stored in the first partition, all rows for the second basket in the second partition etc. The basket is the clustering criterion.
- Based on a fixed destination partition. All rows are stored in the same partition. This mechanism may work well if different load threads work in parallel, each loading its own private partitions.

The Encoding Function

The encoders split up the row into individual columns and encode these if the data is to be stored as a nCORE. Encoding is a central function. There is one master copy of the VAT that is managed by a single manager process.To optimise performance, the encoders may request a copy of the VAT, and may request encoded values from the VAT manager if they encounter new values. Such request may be done for individual values or for groups of-values. The encoders use hashing functions for efficient table look-up.

The Store Function

After data has been distributed and encoded, it needs to be written to disk. This can be done efficiently, as all write-operations will be sequential at the end of the existing COREs. During a LOAD operation the various COREs on disk may contain different numbers of entries. As long as the various analysis functions are guaranteed to see the same number of entries for all COREs, this is not a problem.

Performance Characteristics

The performance of load operations is important. InfoCharger's load function can load 1 billion rows with 50 columns each on a 25 CPU system in a 4-hour window. This means

Copyright © 2000 by Accrue Software Inc.

3000 rows or 150,000 fields per second per CPU (300Mhz Pentium II). Note that such figures apply to the load function itself. Some sources may not be able to deliver data at this speed.

Exporting Data

Some applications require large numbers of records to be exported, for example to feed customer data to a marketing campaign. Typically one has used business-intelligence or data mining techniques to make appropriate selections, but the next stage would be to export many records based on selection criteria that have been implemented as a pCORE.

"Some applications require large numbers of records to be exported, for example to feed customer data to a marketing campaign."

The export facility allows users to select a number of attributes, including virtual ones, to be downloaded to a disk-file. In order to use parallel processing as much as possible, separate output files are created for each partition of the InfoCharger table.

The export facility injects field and record separators. This way one can for example build comma separated files that can be loaded easily into spreadsheets.

Optionally, the export facility creates a metadata file that describes the exported data. This file has the same format as the metadata file that is used to describe text input for the load facility.

Fetching Individual Rows

One can retrieve individual rows from InfoCharger. Similar to SQL, one specifies a set of columns. This type of data retrieval is used to visualize data at the most detailed level.

One can also retrieve data from InfoCharger by supplying a RowId. This is a mechanism that basically returns random rows, as the user has no influence over the allocation of RowIds. One is guaranteed, however, that using a given RowId will always return the same row. One can also request a random row, in which case the RowId will be returned as well.

One can also provide InfoCharger with some values from certain columns, such as customer names, customer number, phone numbers etc. For columns with a low selectivity, one could then create a pCORE and next access all qualifying rows.

Paged COREs

InfoCharger is optimised for analytical processing and hence for column-oriented access. Some auxiliary functions like exporting or fetching data may have a different pattern: the output of these operations is a list of records, with potentially many columns. Using the general mechanism of retrieving complete COREs before operating on them might cause the export (and fetch) operations to fail on large tables because of lack of memory. For these situations, paged COREs are introduced.

"InfoCharger is optimized for analytical processing and hence for column-oriented access."

The export and fetch operations access COREs sequentially, though (many) rows are skipped if a pCORE is used to apply predicates. For these operations a single buffer per CORE is used to provide a 'window' to the data.

Before accessing a new row, the export and fetch functions test if the row is accessible through the window; if this is not the case, the next set of entries from the COREs that are selected is retrieved from disk. Export and fetch functions will use COREs that have al-

Copyright © 2000 by Accrue Software Inc.

ready been loaded in memory, but for large tables, they will not load new COREs in memory; instead they will use paged COREs.

Clustering

For some operations, groups of records are examined: association analysis is a good example of this. It is then important to navigate quickly between the various members of the group. This can be achieved most efficiently by re-writing the COREs so all group members are stored next to each other.

The Cluster command implements the operation in a very efficient way:

- The clustering CORE is read in from disk.
- For this clustering-CORE, build an in-memory table with two columns, one to hold the count per value, the other to hold the number of entries (per value) that have been loaded in the target CORE. In fact, one can also store the count and current offset in the target CORE.
- For each relevant CORE:
  - Scan the CORE (using bulk-IO)
  - Look up the target position in the in-memory table using the value from the clustering CORE
  - Write entry in target CORE (using bulk-IO).

Clustering requires two full COREs to reside in memory: the target CORE and the clustering CORE. The source CORE can be windowed (see “Paged COREs” on page 23). In addition, the in-memory tables need to be maintained, their size depending on the UEC of the clustering CORE.

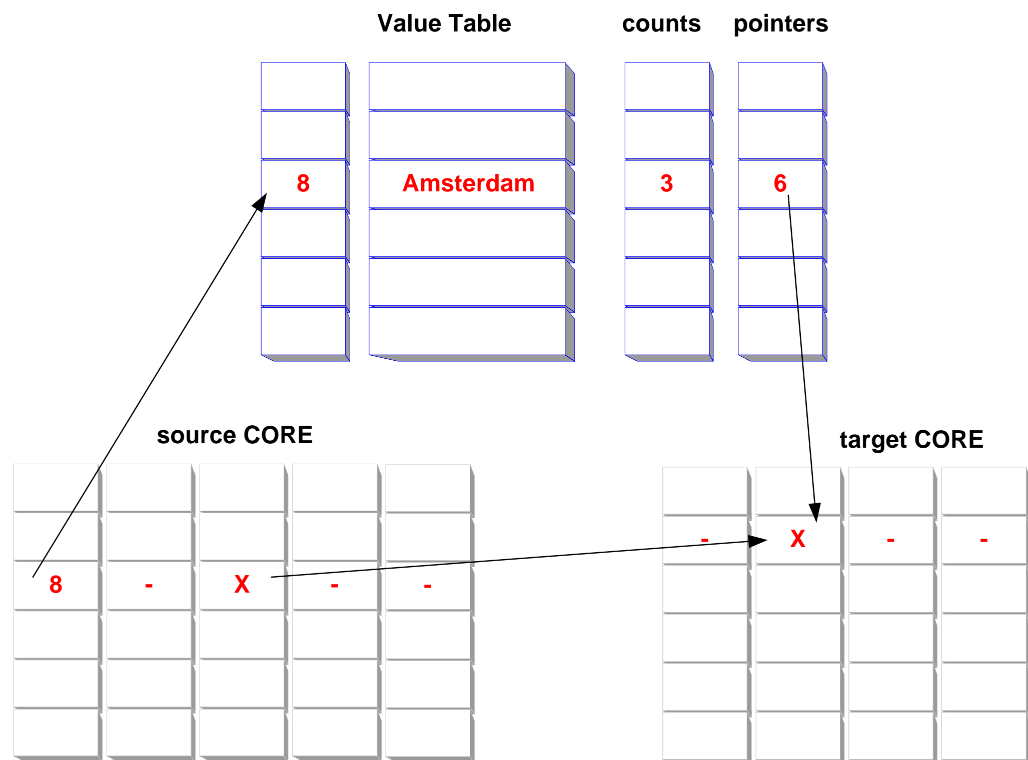

Figure 14 - Cluster Function

Copyright © 2000 by Accrue Software Inc.

Supported Environments

InfoCharger runs on several different platforms:

- Windows NT available today
- Various UNIX flavours will be delivered in the near term. Solaris being the first.

InfoCharger can be configured to run on a single node, single CPU system, like desktop PCs or even laptops.

However, InfoCharger also takes advantage of all available CPU power in SMP and MPP environments.

InfoCharger obtains its high levels of performance by combining highly efficient mechanisms and parallel execution.

Standard import modules are available to load data from NonStop SQL, Oracle and flat files. User written import programs can load data from any other source.

A sample program that provides a simple text-based interface to InfoCharger is available for educational purposes.

A Software Development Kit is available allowing Integration of InfoCharger into Customer applications.

InfoCharger is managed using either a scripting tool or a Java™ based management console.

The current release is version 2.4

Copyright © 2000 by Accrue Software Inc.

Copyright © 2000 by Accrue Software Inc.

Part No AS_IC_WP Copyright © 2000 by Accrue Software, Inc.