illuminate

Data warehouses that don't compromise.

FLEXIBILITY SPEED CORRELATIONS

Comparison of Data Warehousing DBMS Platforms

An analysis of the advantages and disadvantages of relational, columnar and correlation databases for complex and demanding analytics environments.

Abstract

Although relational databases (RDBMS) are the most common choice for data warehouse implementations, their record-based structure is far from ideal. As data volumes grow and users demand more sophisticated analytical capabilities, the deficiencies of the RDBMS to data storage become more conspicuous. RDBMS data warehouse systems are difficult to design; extremely inefficient in their use of disk space and I/O; challenging to maintain; and, worst of all, require designers to compromise between optimizing query performance and maximizing query flexibility.

In response to these shortcomings, alternative database architectures—columnar and correlation—have been developed. Columnar databases use less disk space and are more efficient in their I/O demands than records-based data warehouses but force their own compromise between optimizing for new record insertion versus data selection and retrieval.

A radically new data warehouse platform, the correlation DBMS (CDBMS), eliminates all such design tradeoffs. There is virtually no upfront design effort required. The CDBMS builds a data-driven schema and indexes 100% of data values automatically during the data loading process, which optimizes for both performance and query flexibility. A CDBMS minimizes disk storage and I/O requirements, makes new data ready for use as soon as the load process is complete, and enables creation and execution of unique types of queries not supported by either RDBMSs or columnar databases.

This white paper compares the data storage models, hardware demands, performance levels, design considerations and analytical flexibility of all three data structures in an analytics environment.

Introduction

The term “data structure” has two meanings in the realm of information management systems: the database management structure that manages the physical storage of data and the structure employed by the database designer to organize the various sets of data. Both levels must be considered when creating a data warehouse since both have a significant effect on performance.

Over the years, many different structures and methods have been used by DBMSs for storing data. Each has had its own advantages and drawbacks. Currently, several different internal data structures are used in commercial products. The structures considered by organizations as best suited for data warehouse applications are the record-oriented structure of the RDBMS, the column structure in columnar databases and the value-based structure of the correlation DBMS (CDBMS). As the names indicate, these structures differ primarily in how they organize the data stored.

RDBMS record-oriented structure

The record-based structure is the most common choice for data warehouse applications. In this structure, data is stored in physical records using the common physical location of data values as the logical connection across all data points of the individual record. For example, to find a customer address, the system must first locate the customer using an indexed value such as customer number, then scan across the record to the position for address. As a result, the smallest addressable unit of storage is *an entire record*, and all physical I/O functions will always move complete records or sets of complete records.

Since entire records are physically written and read as single units, selecting any field requires reading the entire record, and changing a single value requires reading and rewriting the entire record. If the new value requires more space than the original, extensive additional I/O is needed. The modified record may be written into an overflow area, forcing either multiple read operations at query time or reorganizing and rewriting the previous area.

This constant movement of full records causes performance issues that have no hardware-based solution. While RAM, processor speed and disk memory have improved dramatically in performance (and dropped in cost), internal bandwidth performance—the speed of moving data from disk to memory and back again—has improved only marginally over the years. Unfortunately, constantly moving complete records imposes a major load on this limited resource and significantly impacts response times.

Columnar database column-based structure

The newer column structure has gained interest as the indexing and data transfer problems associated with record structures have proved problematic for analytics applications. The column-based DBMS stores all of the values from one column of a table in a contiguous data set. This allows the reading and/or writing of parts of records. It conserves I/O bandwidth by transferring only the values that may be used in the query. Since most data warehouse applications use only a few columns from a table during a typical single access, the resulting bandwidth savings can be substantial.

Writing new records requires updating each of the column sets that belong to the record. Reading a complete record requires locating the correct value from each column set and assembling those into the original record format. Therefore, full record operations show little improvement over record-based systems and, in some cases, even can perform more slowly.

Correlation DBMS value-based structure

The most recent innovation, value-based storage (VBS)—used in the correlation database management system (CDBMS)—also avoids the indexing problems of record structures, but in addition, it has exploratory and analytical features not available with either record-oriented or column-based structures. The value-based model stores each unique value once, then uses extended metadata to maintain the information needed to manage record formatting. VBS minimizes bandwidth consumption, as the actual data value storage is so small that the entire set of all data values can frequently stay in memory, affecting I/O bandwidth only when the system is started.

Detailed Comparison of Three DBMS Storage Structures

RDBMS record-oriented structure

The sample records in the table below illustrate the physical differences among these three storage structures by showing what is actually stored in each type of structure; the first table represents the physical storage of the record-based structure in RDBMS, which is the simplest. The data is stored exactly as in the table. Variations like clustered indexing may change the sequence of the rows, but all rows, columns, and values will be stored as in the table.

Cust ID	Name	City	State	Region
12222	ABC Corp	Minneapolis	MN	Central
19434	A1 Mfg	Duluth	MN	North
20523	J&J Inc	St Paul	MN	
28495	Acme	Minneapolis	MN	Central
30023	XYZ Corp	Rochester	MN	South

Columnar database column-based structure

The second table represents the physical storage of a column-based structure in columnar databases, which varies depending upon the intended use of the system and the skill of the designer. With no special design, the storage is structured as shown. Each of the separate blocks in this diagram represents separate storage areas, either as separate files or separate areas of a large block of storage managed by the DBMS. Indexing features vary from one column-based product to another, but the actual data value storage will be as shown below.

Cust ID

Record	Value
1	12222
2	19434
3	20523
4	28495
5	30023

Name

Record	Value
1	ABC Corp
2	A1 Mfg
3	J&J Inc
4	Acme
5	XYZ Corp

City

Record	Value
1	Minneapolis
2	Duluth
3	St Paul
4	Minneapolis
5	Rochester

State

Record	Value
1-5	MN

Region

Record	Value
1	Central
2	North
4	Central
5	South

In the column-based structure, consecutive duplicates within a single column would then be automatically removed, and null values would not be recorded since the missing record ID implies a null value. The data physically stored can vary significantly based on the logical design. This is based on decisions made by a DBA when designing the schema to meet specific business requirements. Columns can be sorted rather than stored in entry

order. For example, the City column in the diagram on the previous page could be optimized for searching by ordering the values as seen in the diagram to the right.

Although this optimizes record location for query functions, it greatly increases load time since the column needs to be resorted every time data is added or changed. Special techniques can be used to optimize columns where there are very few values, such as the State column in the example. However, in all cases, the data storage will remain essentially as shown.

City

Record	Value
2	Duluth
1, 4	Minneapolis
3	St Paul
5	Rochester

Correlation DBMS value-based structure

ID	Value
1	12222
2	19434
3	20523
4	28495
5	30023
6	A1 Mfg
7	ABC Corp
8	Acme
9	Central
10	Duluth
11	J&J Inc
12	Minneapolis
13	North
14	Rochester
15	St Paul
16	South
17	XYZ corp

In a value-based storage (VBS) structure, the physical storage is completely different. Each unique value is stored only once, regardless of the number of occurrences or locations in the original data. As with the column approach, separate indexing and linking information are stored, but the data storage is structured as shown to the left.

With VBS structure used in a correlation DBMS, there are no design decisions to be made, and evolving business requirements never force changes to the physical structure. All values are always stored in ordered sets. The sets are not related to any order or usage in records and never need to be reorganized. In addition, since there is only one location to search when selecting records, all searches always gain the speed advantage of indexing.

While other data structures—such as object databases—are sometimes employed to meet highly specialized needs, these are not viable structures for a data warehouse.

Continued on next page

Advantages and disadvantages of data warehouse DBMS platforms

A brief description of the three primary data warehouse platform structures follows, along with a discussion of their key advantages and disadvantages in the context of data warehouses.

Relational Database (RDBMS)

There are many commercial variations of record-based databases available. All share the common characteristic of storing related values in a physically contiguous record. The most commonly used form used for data warehousing and analytics is the RDBMS, which has dominated the entire database market for years due to its strengths in processing and storing transactions and in generating reports.

Despite its nearly universal acceptance as a standard data structure, the RDBMS has significant limitations that detract from its usefulness. Among the most serious is the problem of indexing data properly for use in an analytical environment. In addition, the RDBMS structure suffers from performance problems due to the need to move entire records in response to any read or write activity.

The speed issues become obvious when attempting to perform unrestricted business information discovery and analysis that requires all data values to be available for use in qualifying queries. When data is stored in a physical record structure, finding data values requires reading every qualified record from the disk and examining the contents of the column being used for selection. Each time a qualifying value is found, the record is then included in the response to the query. If a table is large, as in a data warehouse, this process is very slow and consumes excessive computing resources.

Indexing provides a partial solution by allowing the computer to read only the desired records. Each index added to a table provides another path for rapid access using the values in just the column that was used to create the index. However, each time a different column is needed to perform a search, another index must be built. This quickly becomes an onerous process as a customer record can easily contain hundreds of columns and, in an analytical environment, many or all of those columns should be indexed to provide maximum business value. To build a data warehouse that supports ad hoc data discovery and exploration functions, *all* columns would need to be indexed.

RDBMS vendors address this problem by supporting the ability to create many indices on all tables. However, since each index increases processing time and disk storage requirements, there is a practical limit on the number of indices that can be built. So, not all columns can be indexed and used effectively.

In fact, for large records, most columns cannot be indexed. For example, it's relatively easy to provide four or five indices for a small record (such as a sales transaction) that contains only 10-12 columns. However, a large record (like a customer record) can have a 100 or more columns. The problem is that limits on index creation in an RDBMS make it feasible to provide only 10-12 indices for this data.

Unfortunately, most exploratory and analytical uses of data warehouse information are related to customer information to some degree. Therefore, the limited number of indices supported significantly restricts the range of analyses possible. Moreover, every index requires additional work from the system when records are added, modified or deleted. This slows the data loading process and can bring analysis to a halt. While one or two indices will not significantly impact processing time, 10-12 indices on every table will definitely multiply the update time. In a large data warehouse environment, wherein many tables are updated in large batch processes, the time required for indexing during batch operations severely limits the availability of databases to business analysts.

A second problem is that the RDBMS can't resolve a query and optimize the process to use only indexed fields if there are too many indices involved in the query. While "too many" is a relative term with no precise value, every RDBMS has a practical limit. Therefore, none can reasonably contain enough indices to fully support complex discovery and analytical processes.

Indices pose other problems as well. Database management systems include sophisticated software that analyzes the content of a query and optimizes processing. This requires analysis of the qualifying fields used in the queries, the links between tables needed to find all of the data, and the estimated volume of data that needs to be scanned. When the query is too complex, the optimizer is no longer able to determine an optimal method and resorts to full table scans to find the desired records, ignoring the indexing. A full table scan on a moderate-sized table may take only seconds, but on the very large tables found in data warehouses, it can take several minutes. And when performing a complex query that involves many tables and many indices on those tables, full table scans can force a single query to take hours or even days. Clearly, this is not acceptable for a system that is supposed to support rapid query and response, ad hoc query, reporting and unplanned exploration and discovery.

A third problem with RDBMSs for data warehouses is the limitations imposed on changing a record format. A single table may contain millions of records; for a data manager to make sense of this content, every record must have exactly the same fields in the same sequence. If a new column is added, either the entire table must be rewritten onto disk with the new column in place, or the new column must be written elsewhere. This requires the database to retain information about the physical storage of the column and to logically attach the column in the right place for input and output.

To make matters worse, not only does the column need to be inserted in the right place in the record, the values in that column must be in exactly the same sequence as the rest of the records. That is, the value in the 10^{th} record must occupy the 10^{th} position in the disconnected column store. This structural requirement significantly impacts data analysis performance and flexibility.

A final problem with RDBMSs is the need to reorganize the database at regular intervals. This wouldn't be an issue if data were never added, updated, or deleted, but in the real world, data is added, deleted, and changed regularly. This activity creates empty spots on the blocks stored on the disk and requires long strings of pointers to maintain the correct record structure. The problem is compounded by the fact that changing business requirements often require index modifications and the addition of new data sources. In

very large systems, this requires paying close attention to data partitioning and a great deal of alteration when systems become unbalanced.

Constant add/delete/change activities make the record-oriented RDBMS less and less efficient, until eventually it has to be reorganized or become essentially useless. The reorganization process takes the database offline, requires substantial resources to perform and further reduces system availability for analysts.

Collectively, these issues create substantial and often unacceptable costs. Unsurprisingly, many organizations are discovering that a data warehouse built on an RDBMS will never provide acceptable performance, availability or functionality in an open, mixed-workload environment.

Columnar Database

A columnar database stores data much differently than does a RDBMS. In this structure, data is stored in sets by column—that is, all of the values stored in column one are stored in one set, all the values in column two in another set, and so on. In addition to the values, the information needed to reinsert them into the proper position in the original record format is stored with each set.

This provides some significant advantages for analytical processes and ad hoc queries and provides the additional benefit of using less disk space than does an RDBMS.

The most obvious advantage of this structure is that when selecting records in a query process, it is only necessary to read the columns that are included as qualifiers, or results, in the query. This can make response up to 10 times faster than with an RDBMS. When columns are stored in a sorted sequence, the response speed improvement can be even greater.

The chief disadvantage of columnar databases is that they perform less satisfactorily in terms of import, export, bulk reporting and the efficient use of computer resources than do RDBMSs when required to carry out transactional processes. As a result, record-oriented processing is usually required in a data warehouse environment. The drawbacks of the columnar databases are readily apparent in the new record insertion process, which requires reading a column store set for every column in the record, determining where the new value belongs in that set, inserting it in the right place and then building the required linking information to be able to retrieve it and insert it correctly into the original record format.

This isn't a serious problem if the column values are stored in the order received and all that is needed is appending the new value to the end of the set. However, using this method, record selection and retrieval processes will be slower since the column will always have to be read from beginning to end.

However, if the columns are stored in a sorted order, adding a new record is time-consuming due to the required sorting and reorganizing of the column values. And while this sort method optimizes the selection and retrieval processes, it requires continual tuning to provide optimum performance and meet changing business needs. While the performance-tuning issues are different, they are often just as extensive as in RDBMSs.

Commercial vendors have acknowledged this drawback. Vertica, for example, claims a unique approach to solving the problem. Its system maintains data in a sequence that is optimized for retrieval while also maintaining a small temporary set of recent updates in entry sequence. This allows the system to maintain a reasonable update speed while still providing optimized query performance.

The effectiveness of this approach, however, assumes there is enough free processing time available on the hardware system to enable the updating of the column sets from the temporary entry format. This can be problematic, as essentially all queries must access both storage areas. In other words, the query must perform two different types of searches, which degrades overall query performance. The use of this temporary storage also increases the overall workload involved in loading data since there will be two distinct steps to be completed before the load is complete. The system works well if there are sufficient free processor cycles and available I/O bandwidth. However, when the system becomes bogged down by user demand, Vertica's structure can exacerbate performance problems.

Correlation Database (CDBMS)

The newest entry in the data warehouse platform arena is the correlation database (CDBMS) with its value-based storage (VBS) model. This structure stores data values independently from the logical schema and uses extended metadata to maintain the information needed to reconstruct the original records. VBS offers performance equal to or better than the columnar structures, plus additional features not provided by either RDBMSs or columnar databases. In VBS, each unique data value is stored only once, making the database extremely compact and fast.

With VBS, query performance is outstanding since the qualification steps of the query never read a record or column. Instead, the search process is conducted through a set of values that is always ordered and an indexing algorithm that identifies the qualifying records. Both steps involve very small I/O loads and, consequently, execute very quickly. No records or columns are pushed through the most restricted computer resource, the I/O bandwidth.

In a CDBMS, load performance is also significantly better than with columnar DBMSs. Once the majority of unique values have been loaded, new record inserts require only updating the indexing set. This saturation of values usually occurs very quickly and databases of only 20 gigabytes have very stable sets of stored values. Deletions do not affect the stored value sets, requiring only the removal of entries in the indexing set.

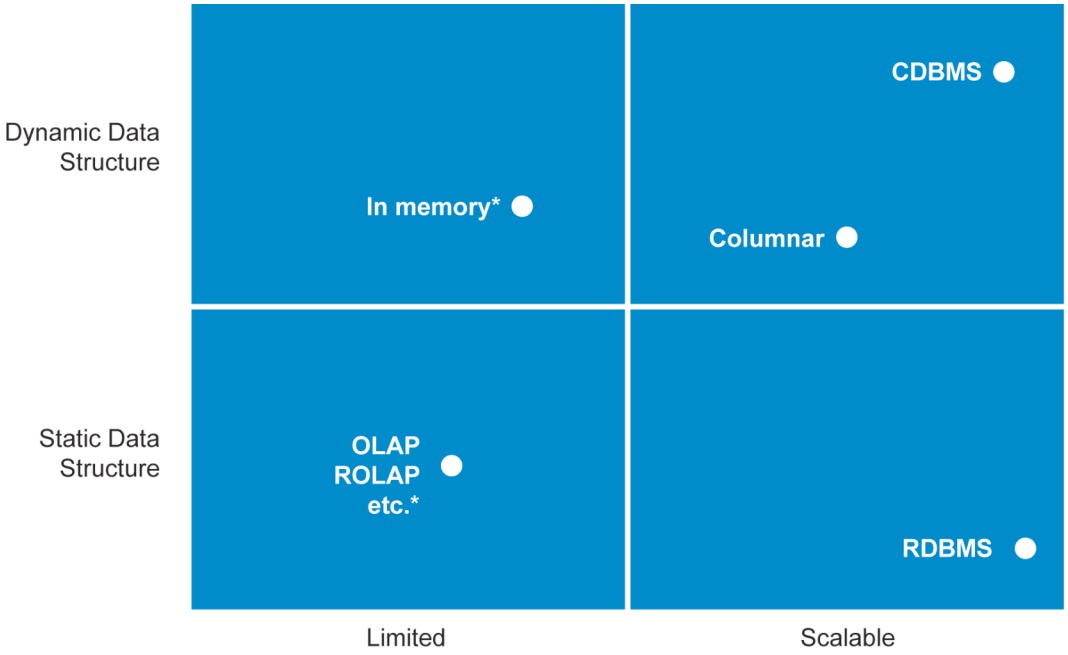
Performance is also enhanced by the inherent compression of VBS since each unique value is stored only once. The volume of data is reduced to a small fraction of equivalent record or column structures without the need for additional compression/decompression algorithms.

As the database grows and the value set becomes saturated, the incremental compression ratio of the data sets approaches infinity. As new data is added, the indexing set continues to grow at a constant rate, increasing overall database size, but at a small fraction of the raw data set size. A very small database may require two times as much space as the original raw data. With just 100 million rows stored, a CDBMS is about equal

in size to the raw data file. As the database grows, the size of the CDBMS relative to the raw data declines.

There is no indexing of tables in a CDBMS since all values are indexed in the VBS approach. Therefore, the overall database size is never increased by building indices. This has the dual benefit of saving disk space and speeding load processes.

The quadrant below indicates the performance of these three DBMSs as analytical data warehouse platforms.



*These structures are included but not discussed due to their limited ability to scale for data warehousing purposes.

Comparison of Data Warehouse Design

Data warehouse design is challenging. Many theories and methods have been proposed for optimizing data warehouse design, but even the definition of “optimize” is inconsistent. Some designs focus on minimizing query response time, others on maximizing the flexibility of the logical structure for meeting changing needs. Still others propose very academic approaches to providing optimized semantic layers and metadata content. And the TPC-H benchmark provides a standard definition of a data warehouse that will never match any real enterprise and yet defines the performance issues to be optimized.

Regardless of how “optimized” is defined, two dimensions of optimization must be considered: hardware performance and human interaction. This forces designers to accommodate two mutually exclusive design parameters, speed and flexibility. Although no RDBMS solves this dilemma, the RDBMS is usually viewed an acceptable compromise for the mixed-use data warehouse and has become, by far, the dominant platform. But a closer look at the characteristics of a data warehouse and how the various structural options described earlier in the paper address data warehouse design issues challenges this prevailing compromise.

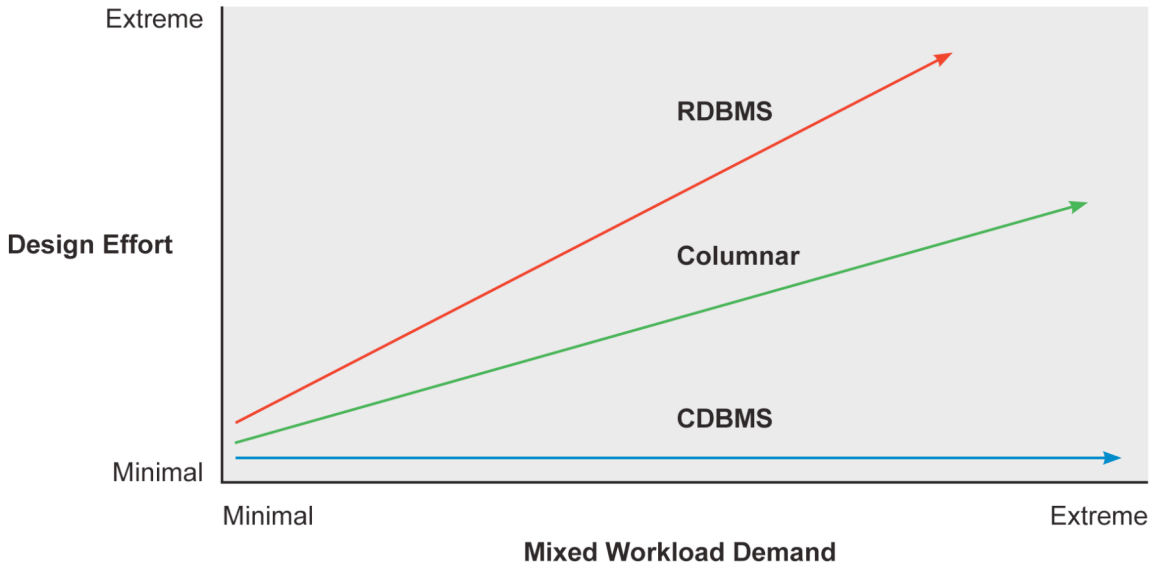
Physical Data Structure Design

Physical design is the primary consideration when building an RDBMS data warehouse. Physical design is also important in columnar database systems; the design process parameters are different from relational environments, but the process is just as critical to performance.

With the CDBMS, the physical design process is nearly eliminated. For small data warehouses (less than 50 gigabytes), there simply *is no* physical design process. All that is needed is sufficient disk space.

For moderate-sized data warehouses (up to five terabytes), the importance of physical design is joined by the need to optimize the use of bandwidth between processors and disk, especially when the system has many users. But a CDBMS needs little design work even in this environment. Optimization consists of locating the data store on one disk system, the temporary space on another, and the indexing and metadata store on a third.

Even in the largest systems, physical design requires only the best disk arrangement to provide the widest possible I/O bandwidth. There is never a requirement to separate and physically locate parts of the data on separate I/O systems. The diagram below illustrates the design effort associated with each type of data warehouse platform for mixed-workload demand.



Large Data Volumes

Large data volumes create two issues for a data management system: 1) loading new data; and 2) finding, selecting and retrieving information.

Data Loading—All data structures support data loading well. RDBMS systems have the largest installed based, and their vendors have created dozens of proven, proprietary tools to enhance load speed, so there is little need to review these approaches here. Columnar and correlation DBMSs are much newer, however, so an understanding of their theoretical performance challenges and possible resolutions is helpful.

First, it is generally assumed that due to the additional processing required to parse each incoming record into fields and deal with each field separately, columnar and correlation DBMSs will load data more slowly than an RDBMS.

However, columnar and correlation DBMSs are capable of loading data at very high speeds. Load speed is rarely a significant issue except in the very largest data warehouse implementations. In some cases, the total load times can even be faster with a columnar or correlation database than with a record structure.

Time-to-Analytics—The key question is not raw data load time, but rather *time-to-analytics*—the total elapsed time between receiving new raw data to be added to a data warehouse and being able to use that data. If the sole consideration is how long it takes to move a gigabyte of new data to disk storage, then an RDBMS is faster at loading data than is a column or correlation DBMS. However, an RDBMS is *not* always fastest at making new data available for user access. The additional processing needed to create structures like indices, aggregations, materialized views and sorts when adding records always takes additional time. In a mixed-use data warehouse, the additional time needed for these operations can actually make the complete load process slower than it would be for the correlation DBMS. Columnar databases frequently require values to be stored in a sequence other than the order received, which makes loading data take longer and require more disk space than with the correlation DBMS. Limiting the number of columns to be sorted improves load speed but imposes restrictions on flexibility.

In the final analysis, while each database may be the fastest or slowest depending on factors such as database size, record length and variability in data values, all structures provide adequate load speed for most analytical applications. However, since the data warehouse is a read-mostly environment, load speed is an important but generally secondary issue. More important is the selection and retrieval speed from large data sets using the various structures.

Selection and Retrieval Speed—If full records are being selected from the data warehouse then an RDBMS will be very fast. If *all* records and all columns are being selected, then an RDBMS will provide the fastest retrieval speed of any structure. But the data warehouse is very rarely accessed in this manner. Typical access involves selection of a relatively small set of records from a subset of all columns.

In a typical data warehouse selection and retrieval process, a single query will rarely access more than one-tenth of the total stored data or retrieve more than one-tenth of the accessed data. In a one terabyte data warehouse, for example, a single query will rarely, if ever, access 10 gigabytes of data. Typical information access processes are much smaller than this, making the best physical structure the one that optimizes access processes to small subsets of the entire system.

RDBMSs excel at selection and retrieval of full records when the selection criteria fit the designed indexing strategy. In these cases, they provide excellent response time. But when there is no index to use or the indexing becomes too complex for the query optimizer, the database manager resorts to full table scans, reading every record in the required tables.

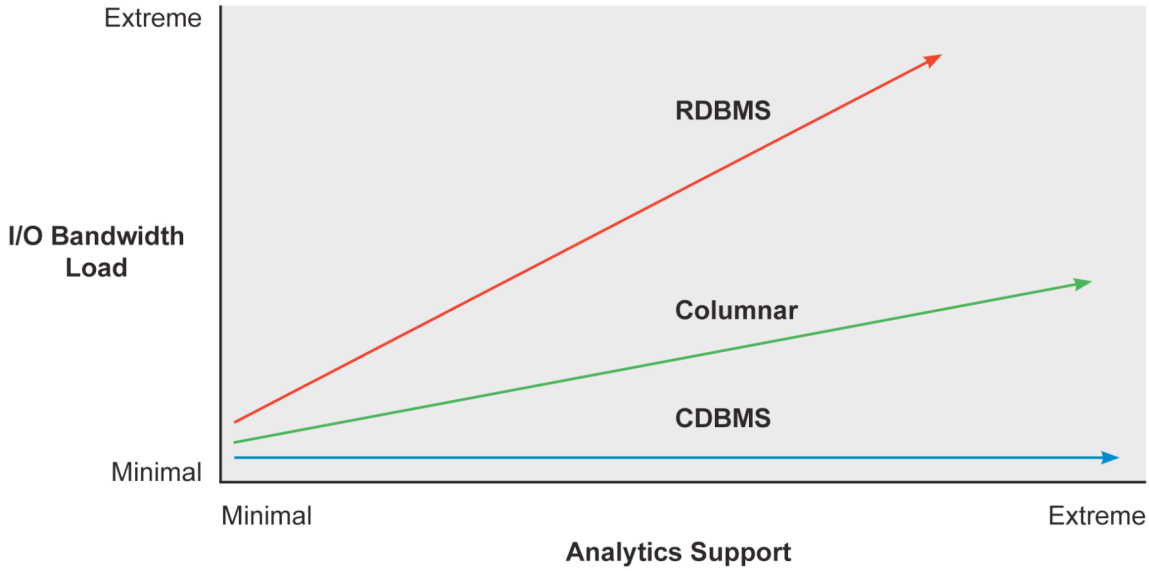
While not a significant problem with small databases, this can result in extremely long response times when the data warehouse contains large volumes. This situation is

commonly referred to as a “query from hell.” These queries will consume all available resources, precluding all other users’ access until the query is finished. Such queries severely overload the most restrictive component of the computer hardware, the I/O channels. The time involved can be crippling. For example, when a full table scan occurs on a table with a billion records, a trillion or more bytes may be moved. Even in a large and expensive computer system, it would take more than 10 minutes to move this volume of data.

Since there is no transfer of unneeded data with columnar or correlation DBMSs, the I/O load is reduced substantially. The worst-case scenario in a columnar database is when it must read an entire column to perform a selection, which happens when a column is not sorted and is ordered in entry sequence. Selection criteria that call values from that column require the entire column to be read to ensure that all possible records are found. The CDBMS doesn’t have this problem since all data values are separated from the columns and stored in sorted order. Every selection process reads only the values that are needed and which are found in a small-ordered set, regardless of the size of the database.

The CDBMS fits perfectly with the way data warehouses are designed and built for selection and retrieval. In a typical situation, a CDBMS will store almost all unique values once a small portion of the data set has been read. After reaching about 20 gigabytes, the data value storage contains almost all of the unique values it will ever need and will grow very little from that point forward, regardless of how large the entire data warehouse becomes. This provides an additional speed improvement in an active system, since most or even all of the data value set may be held in RAM. In fact, the value selection step of the query process can frequently be completed without performing a single physical I/O.

The illustration below shows the superior efficiency that can be expected from the CDBMS.



Detailed Comparison Example—The typical schema in either third normal form or snowflake schema arrangement will contain one table with a very large number of records; a few tables with many records; and many tables with few records.

Table	Records	Columns	Gigabytes
Table 1	1,500,000,000	20	720
Table 2	100,000,000	75	180
Table 3	5,000,000	1000	120
Table 4	100,000	75	0.18
Table 5	25,000	75	0.045
Table 6	15,000	75	0.027
Table 7	10,000	75	0.018
Table 8	7,500	75	0.0135
Table 9	5,000	75	0.009
Table 10	500	75	0.0009
Table 11-99	5,000	75	0.009
Total Size			**1020**

This represents one terabyte of *input data*, not one terabyte of *database size*. The size of this database in an RDBMS would range between five and 20 terabytes depending upon the number of indices, aggregations, OLAP cubes and materialized views in the final schema. In a columnar database, the final database size would be three to five terabytes. A CDBMS would require only one to 1.5 terabytes.

In this example, table 1 would be the fact table in a star schema or the transaction table in third normal form. The other large tables would be the larger dimensions like customer or order header. The remaining tables would be the smaller dimensions such as region, store, etc. In third normal form there could be more than 100 tables, while a well designed star schema would have fewer than 100. The size of these tables, however, will be small enough to be insignificant in evaluating performance.

A typical business request such as, “show me all customers who bought product A” would result in a query approximately like this:

```
Select name, address, city, state, ZIP
        from customer, product, transaction
        where customer.customer_id = transaction.customer_id and
          transaction.product_id = product.product_id and
          product.name = “Product A”
```

The optimum physical I/O needed to answer this query would involve reading one field from one record of the product table, two fields from many records from the transaction table, and several fields from many records from the customer table. The actual I/O would, of course, be more than this but would vary considerably depending on the physical structure of the database.

The hypothetical query above assumes the table shown to be the transactions table or the fact table. Table 3 contains customers. Table 5 contains product. For comparison, consider selection of one product that was purchased 1,000 times by a total of 500 customers.

Ignoring blocking and assuming correct indexing, the RDBMS would require reading one product record, 1,000 detail transaction records, nearly 1,000 order records and 500 customer records to answer the query—a transfer of more than 550,000 bytes of data to retrieve about 50,000 bytes of desired information.

Using a columnar database, actual I/O load would depend upon how the columns were sequenced when the data was loaded. Assuming reasonable planning was put into the creation of the system, the product column would be in sorted order and a single value of a single column would need to be read along with the record identifier associated with that value to find the correct product record. The record identifier would be used to find the correct value in the product ID column. This product ID would be used to find the record identifiers in the transaction table where the product was purchased. These record identifiers would then be used to locate the associated order ID values from the transaction table. The order records would be used to find the customer IDs that would then be used to locate the correct customer records. Finally, the desired information would be read from the selected columns of the customer table.

While this process is more complicated than with an RDBMS, it actually uses more processor cycles (which are cheap and plentiful) but less I/O (which is scarce and expensive). In this scenario, perhaps 60,000 bytes of I/O would be required to retrieve the 50,000 bytes of desired information. However, a compromised design could increase the I/O load to a level comparable to the record structure. For example, if the product ID column in the detail transaction record had not been sorted when it was loaded, the entire column—as many as 1.5 billion values—would have to be read. For a 10 byte ID field, this would increase the physical I/O to as much as 15 gigabytes.

Using a CDBMS to answer the query minimizes I/O load. Again, ignoring blocking, the required physical I/O would be to read one value from the product name column and then read the product ID contents to find the associated ID. This would then be used to find the 1,000 detail transactions, and those record identifiers would be used to find the order IDs. These would be used to find the order header records and the process would be repeated to find the customer records. Finally, the customer information would be returned to complete the query.

Like the columnar database, CDBMSs use more processor power but they require less physical I/O. In this example, the physical I/O would be about the same as with the columnar database: 60,000 bytes read to return the 50,000 bytes needed. However, there's never a compromised structure in a CDBMS since the values are

stored independently from the record identifiers and always in ordered sequence. Consequently, there's never a case wherein the I/O will be increased by any significant amount.

In summary, selection and retrieval for very large data sets in BI applications or analytics and information discovery, the RDBMS is the least efficient option; the columnar database is significantly more efficient; and the CDBMS provides the highest level of efficiency.

Comparison of Realistic Indexing Capabilities

A data warehouse *design* remains relatively stable for years, while analytical *information needs* change daily. The business or other analytical requirements that will be important a year after the warehouse is operational can only be guessed at during the design stage. For example, data volumes can change dramatically in a short time if new technology like RFID is adopted. Meanwhile, content and relationships can change with a merger or acquisition. Most common, though, are continually changing information access needs.

As a result, a data warehouse must have the capability to answer queries that were not anticipated when it was designed. The chief technical challenge created by evolving requirements is the need for a continually changing indexing strategy—indexing requirements change along with business requirements, user communities and operational parameters. Unless every column of every table is indexed, there will be a constant need to modify the indexing strategy, usually by adding more indices.

RDBMS Indexing

The indexing strategy for an RDBMS can't change as quickly as business needs, nor can a strategy be developed that will satisfy *all* needs, even with no change of requirements. All RDBMS systems have a practical limit on the number of indices that can be built for one table. More indexing slows updates, and excessive indexing causes query optimizers to fail. When the query optimizer fails, the RDBMS must read every record of a table rather than using indices to locate specific records quickly. The difference in response times is in orders of magnitude; seconds can become hours when the optimizer fails.

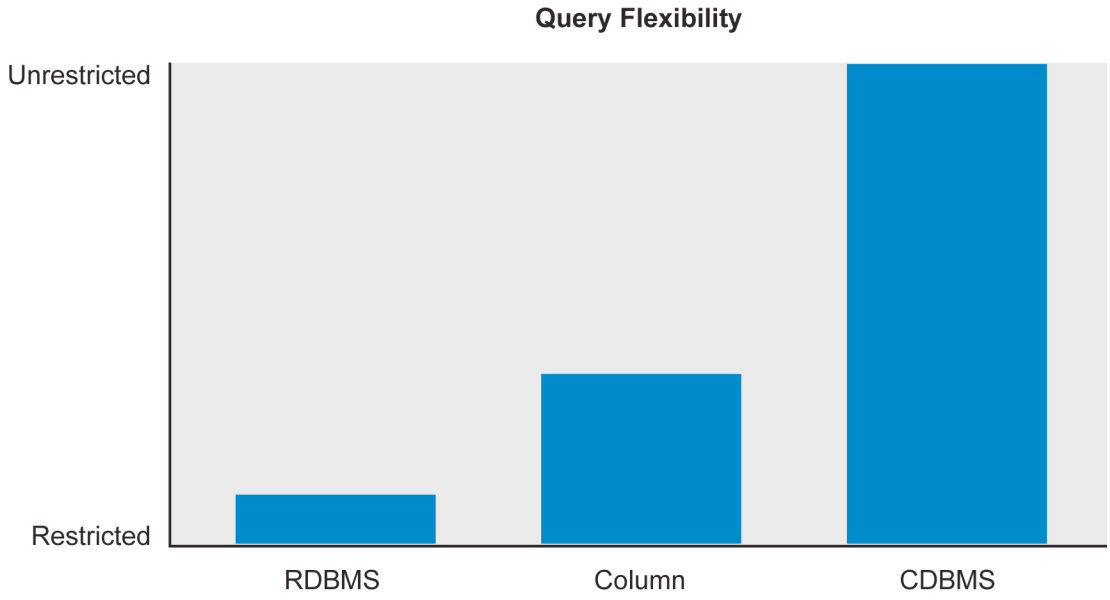
The indexing issue severely limits the utility of RDBMS systems in a discovery and analytical environment. Furthermore, indexing strategies for various analytical requirements are often mutually exclusive; that is, adding the indexing needed for a new requirement can destroy the optimization for existing queries.

Columnar Database Indexing

Columnar databases don't share this problem since every column is independently indexed and all query processes can only access data through an index. Storing column data in an entry sequence versus an ordered sequence will affect response times, though not nearly as significantly as in an RDBMS. When a column is stored in an entry sequence, data loading is faster but query access is slower. Entry-sequenced data also consumes more storage space since repeating values will not occur as frequently. In an ordered sequence, load time increases, but queries are faster and less disk space is used. This trade-off must be considered by the DBA when designing the data structure of the warehouse.

Correlation DBMS Indexing

Like columnar databases, CDBMSs don't suffer from the indexing problem. However, since data values are completely abstracted from the physical record, there is no issue of entry or sorted sequence to consider in a design process. All values are stored in sorted sequence for the data type of the value. That is, all text values are sorted in ASCII sequence, numbers are sorted in numeric sequence and dates are in date sequence. Value-based storage is always optimized for query speed. This also minimizes disk space requirements since repeating values are never stored regardless of the record load order.



Unlike RDBMS and columnar databases, a CDBMS provides the ability to perform two unique types of queries. An *associative* query—a search for an unqualified value—is a simple and fast query that's not possible with other structures. For example, the following query can be answered instantly using a CDBMS.

*Select * from * where * = "France"*

This query, which simply can't be executed by other database structures, would return all records from all tables where any stored value was "France." This could be the name of the country but it could also be a company or customer name, a street name or an entry in a comment field.

Another feature of this structure is the ability to perform *incremental* queries. The result of any query step in this structure is a list of the qualified instance identifiers. It's a simple matter to restrict subsequent query steps to the instances that have already been selected.

Conclusion

Despite their limitations, rrecord-based relational databases have long been the prevailing data structure used in data warehouse systems. However, as data volumes have increased and analytical needs have become more sophisticated, the shortcomings of the RDBMS—most significantly, the design compromise required between optimizing for performance versus query flexibility—have spurred the development of alternative data structures.

Columnar databases offer faster query performance and require less disk storage space than do RDBMSs, but they force their own compromise between optimizing for new record insertion versus record selection and retrieval.

The newest data warehouse database, correlation DBMS, uses a value-based storage (VBS) model that eliminates design compromises. The upfront physical design effort is minimized since the CDBMS is optimized for both query performance and flexibility automatically as data is loaded. The correlation DBMS provides load speeds comparable to RDBMS and columnar systems, minimizes disk space requirements, optimizes query performance, and enables unique types of queries—such as associative and incremental queries—that are impractical or impossible with other data structures.

About illuminate

illuminate Solutions, headquartered in Barcelona, Spain, is the pioneer of the correlation DBMS (CDBMS) for building data-driven data warehouses. The CDBMS is a radical departure from how information management infrastructures are built and accessed. By automatically creating a data-driven schema during the raw data loading process, the need for predesign is virtually eliminated. A CDBMS data warehouse is a fraction of the size of others due to its unique value-based storage™ (VBS) model, which indexes 100 percent of the raw data on-the-fly during the loading process and stores each unique data value only once. A CDBMS eliminates the trade-off between performance and flexibility that so often frustrates IT and business users alike, and dramatically lowers the time/cost of data warehouse deployment and management.

illiminate's products include iLuminate, its CDBMS engine, and a full tool suite for accessing the data warehouse: iCorrelate, for exploration; and iAnalyze, for light analytics, dashboards and geographic mapping. The company also offers the iLuminate SDK to qualified customers free of charge.

illuminate sells its products exclusively through IT consulting services firms and management consulting companies in the Americas. In Europe, its products are sold direct and through a partner network. European customers include BBVA, Bon Preu Group, ZURICH, infojobs, Telefonica and RORI, among others.

For more information, visit www.i-lluminate.com.

About the Author

Chief architect and co-founder of illuminate Joseph Foley, is responsible for leading the company's technology vision and strategy. His expertise encompasses both the theoretical aspects of and the practical application for information structure, management and analysis. Mr. Foley co-founded Illuminate Solutions after starting and growing two successful technology companies that pioneered associative databases and other

systems. With more than 30 years of experience in data management, his research and development spans industries, including retail, insurance, manufacturing, engineering, communications, transportation and more. Mr. Foley is a recognized expert in information analysis; has presented at venues around the world, including several Business Intelligence Europe forums and the DAMA/Wilshire Meta-Data Conference; and writes the Queries from Hell blog. Joe holds a BS degree in business administration with a minor in computer science.