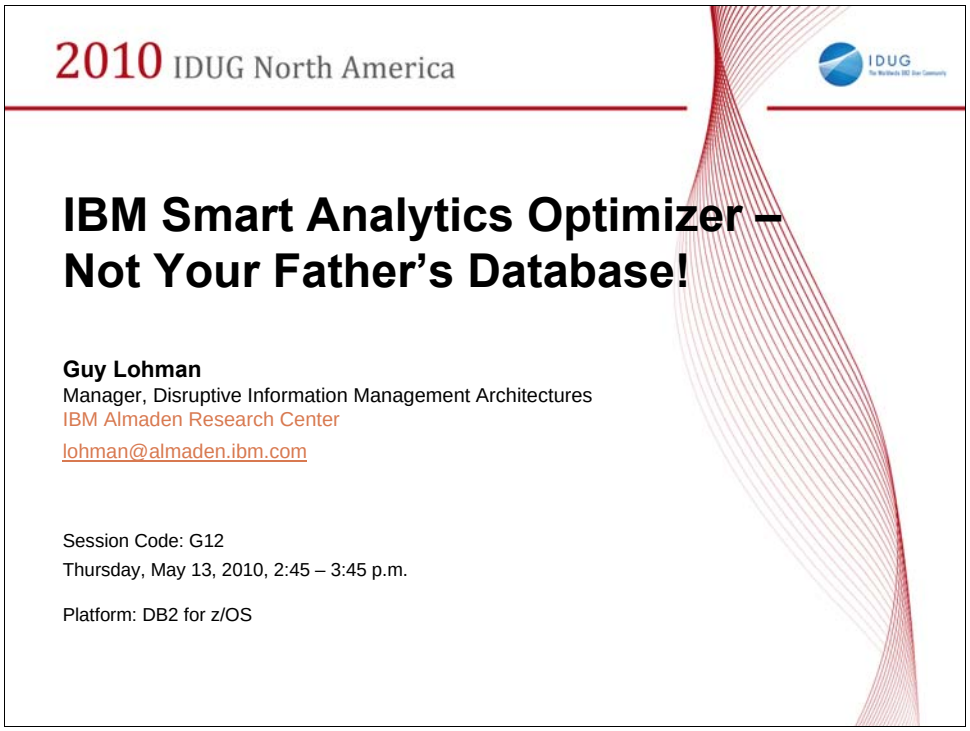


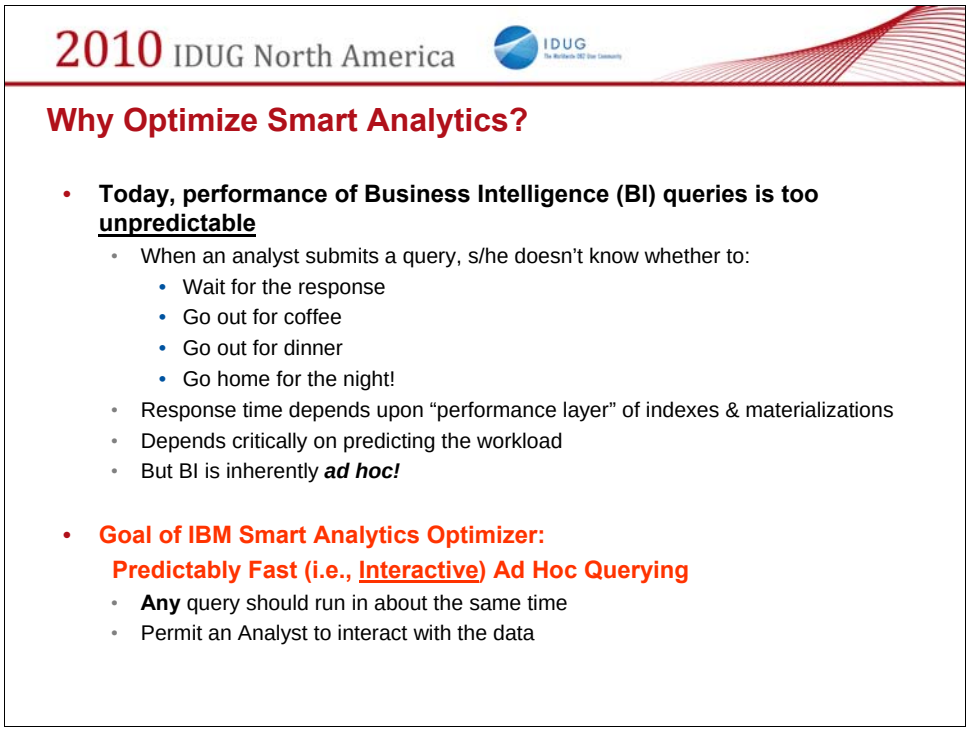
Presentation Abstract:
The IBM Smart Analytics Optimizer for DB2 for z/OS, V1.1 is a revolutionary special-purpose, main-memory database engine that was invented by IBM Research for accelerating Business Intelligence queries to a portion of a data warehouse on DB2 for z/OS. Come find out how the IBM Smart Analytics Optimizer exploits disruptive industry trends such as multi-core architectures and inexpensive main memories in often counter-intuitive ways, enabling massive parallelism and significantly speeding Business Intelligence SQL queries without any need for the usual tuning of a "performance layer" (indexes and materialized views) or even a query optimizer!

IBM Smart Analytics Optimizer (ISAO) – Agenda

- **Why and What is the IBM Smart Analytics Optimizer?**
- **ISAO Market – Business Intelligence**
- ISAO Architecture
- It's All About Performance!
- From the User's Perspective
- What's the Big Deal?
- Behind the Curtain – The Query Engine Technology
- ISAO vs. Column Stores
- Conclusions

Objective 1: What and Why IBM Smart Analytics Optimizer is, and what its target market is (BI)
Objective 2: Architecture of the IBM Smart Analytics Optimizer
Objective 3: What is so disruptive about the technology used in the IBM Smart Analytics Optimizer
Objective 4: A basic understanding of how the IBM Smart Analytics Optimizer exploits industry trends such as multi-core processors and large main memories, and proprietary compression, to speed queries by 10x - 100x.
Objective 5: Startling performance results!

I'll first discuss the motivation for the IBM Smart Analytics Optimizer, and describe what it is. Then I'll describe the market that the IBM Smart Analytics Optimizer targets – Business Intelligence queries.



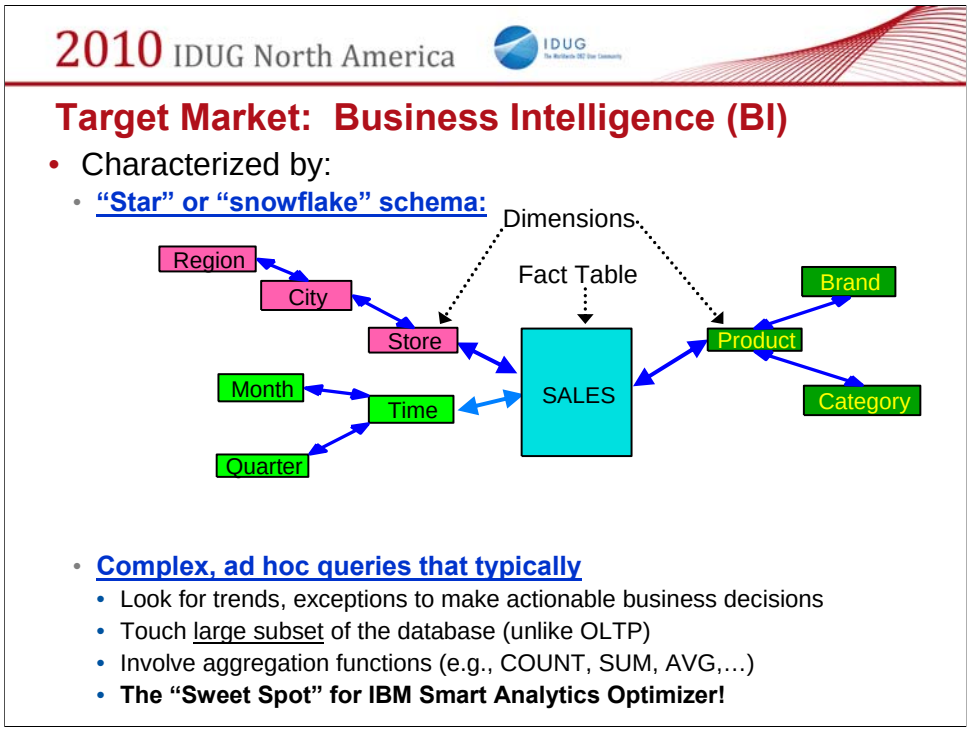
The problem with today's processing of Business Intelligence (BI) queries is that performance is too unpredictable. When an analyst submits a BI query, she doesn't know whether it will run for a few seconds or a few days. If the right performance layer of indexes and materializations (e.g., Materialized Query Tables (MQTs)) has properly anticipated that query, it may run in seconds, whereas without the right indexes or MQTs, the query may take hours or even days, depending upon the data volumes. But BI querying is inherently ad hoc – an analyst will typically formulate her next query based upon the results of the previous query. So it's almost impossible to anticipate the workload by looking at past queries, and hence to anticipate the performance layer that will handle all possible queries the analyst might submit in the future.

The goal of the IBM Smart Analytics Optimizer is to rectify this high variance in the response time of BI queries and achieve predictably and almost uniformly fast ad hoc querying, so that any query will run in approximately the same small time – in seconds or tens of seconds – a timeframe that permits the analyst to truly interact with the data.

What Is the IBM Smart Analytics Optimizer?

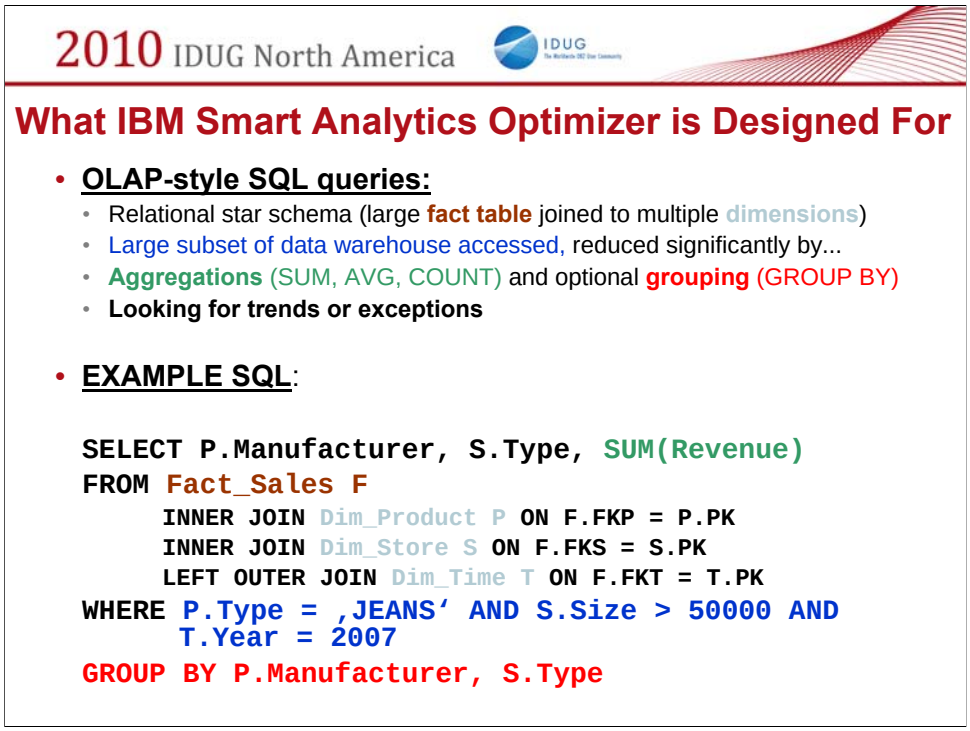
- **IBM Smart Analytics Optimizer for z/OS (ISAO)** is:
  - Network-attached accelerator to DB2 on z/OS
    - (Future: also DB2 for Linux, UNIX, and Windows and Informix IDS)
  - Contains a compressed copy of a (portion of a) data warehouse
  - Exploits:
    - Large main memories
    - Commodity multi-core processors
    - Proprietary compression
  - Speeds up typical Data Warehouse / Business Intelligence SQL queries by **10x to 100x**
  - Without requiring tuning of indexes, materialized views, etc.

So what is the IBM Smart Analytics Optimizer? It is a network-attached accelerator to DB2 for z/OS (and, in the near future, to other IBM products such as DB2 for Linux, UNIX, and Windows and Informix IDS) that caches a compressed copy of a portion of the data warehouse in main memory and exploits the hardware of modern commodity multi-core processors and proprietary compression techniques to speed up query processing by at least an order of magnitude, without having to "tune" the queries by selecting the best indexes, materialized views, or other performance enhancers. Later in the talk, I'll describe the exciting technology by which this magic is achieved, a combination of exploitation of industry trends and clever proprietary engineering.



The target market for the IBM Smart Analytics Optimizer is the Business Intelligence (BI) query market, often called OLAP (On-Line Analytics Processing). Databases in this market are typically characterized by a „star" or „snowflake" schema, referring to the shape of the schema graph. At the center of the star or snowflake is typically the largest table in the schema, called the „fact table", containing millions to 100s of billions of rows. Often the fact table contains information about sales transactions, where each row has information about a particular transaction, such as „metrics" such as the sales price, and various „dimensions" that characterize that transaction, such as the date and time of the sale, the product sold, and the geographical location of the store that sold it. To save space and avoid redundancy that would be expensive to update, the fact-table rows store only a foreign key to the full dimensional data, which is stored in dimension tables. Those dimensions can, in turn, have dimensions (often forming a hierarchy), forming a schema that more resembles a snowflake than a star.

BI queries are often quite complex, ad hoc queries that typically are looking for trends or exceptions in the data, in order to make actionable business decisions. They usually access a large subset of the database, unlike On-Line Transaction Processing (OLTP) workloads, which access only a handful of rows. In order for a human to be able to see trends or exceptions, the large volume of data must be summarized, by grouping the data along some dimension (or dimensions) and aggregating one or more metrics with functions like COUNT or SUM or AVERAGE. This sort of query is the „sweet spot" of the IBM Smart Analytics Optimizer.

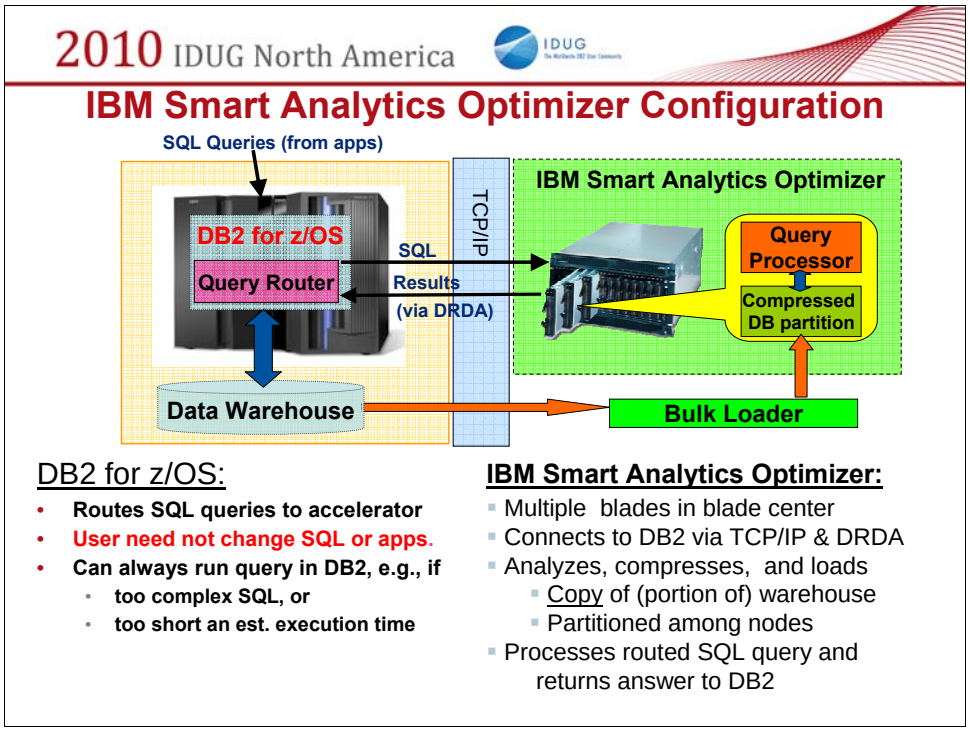


If you speak SQL, this very simple example BI query illustrates the previous ideas in a more concrete way on the schema of the previous slide. The fact table Fact_Sales is being joined with three dimension tables, Dim_Product, Diim_Store, and Dim_Time. We use the INNER JOIN syntax here to emphasize that it‘s a join, but you can also just list the tables in the FROM-list, and provide the join predicates in the WHERE-clause. Note that the IBM Smart Analytics Optimizer supports left outer joins, but only those in which the fact table is the preserved side and the dimension is the null-producing side. This permits aggregations that include fact-table rows having unknown values for some dimensions (in this case, for time). Each dimension is limited by a selection predicate in this example, limiting our query to jeans products sold in stores of size greater than 50000 feet in the year 2007. This query is grouping the resulting rows by the product manufacturer and store type, and summing the revenues for each manufacturer-type pair.

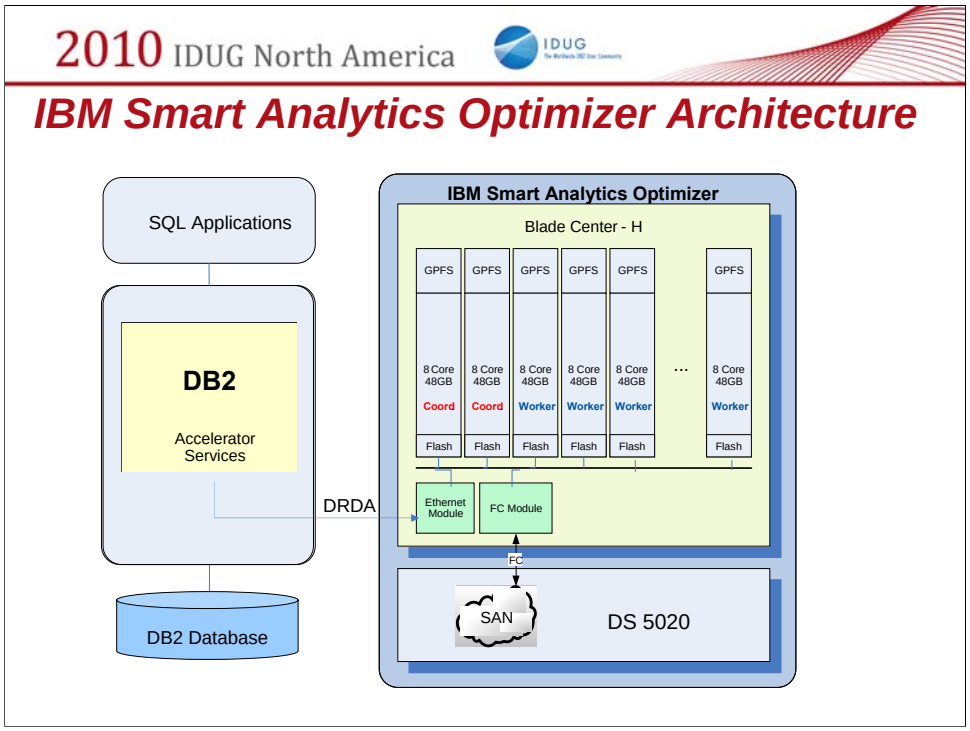
IBM Smart Analytics Optimizer (ISAO) – Agenda

- Why and What is the IBM Smart Analytics Optimizer?
- ISAO Market – Business Intelligence
- **ISAO Architecture**
- It's All About Performance!
- From the User's Perspective
- What's the Big Deal?
- Behind the Curtain – The Query Engine Technology
- ISAO vs. Column Stores
- Conclusions

Let's now look at the architecture of the IBM Smart Analytics Optimizer.



The IBM Smart Analytics Optimizer is configured as an accelerator appliance that is network-attached to DB2 for z/OS. We start with a standard data warehouse managed by DB2 for z/OS, and add an IBM Smart Analytics Optimizer that runs on an IBM Blade Center, which is connected to the z platform via TCP/IP. However, the user sees the accelerator as a System z resource – there are no externalized interfaces to the accelerator! Once the user has defined the data of interest to be accelerated, a bulk loader extracts a copy of that data automatically from the data warehouse by DB2, pipes the data to the accelerator, analyzes it, and compresses it for storage on the blades of the accelerator. The assignment of data to individual blades is arbitrary and not controllable by the user. Once the data has been so loaded, SQL queries coming into DB2 for z/OS that reference that data may be routed automatically by the DB2 optimizer to the IBM Smart Analytics Optimizer, where it will be executed on the accelerator's compressed data, rather than DB2. The SQL query is first parsed and semantically checked for errors by DB2 before being sent to the accelerator via the DRDA protocol in a pre-digested subset of SQL, and results are returned to DB2, and thence to the user, via DRDA, as well. Note that the user need not make ANY changes to their SQL queries to get the router to route the SQL query to the accelerator – it simply has to reference a subset of the data that has been loaded. More on that later…

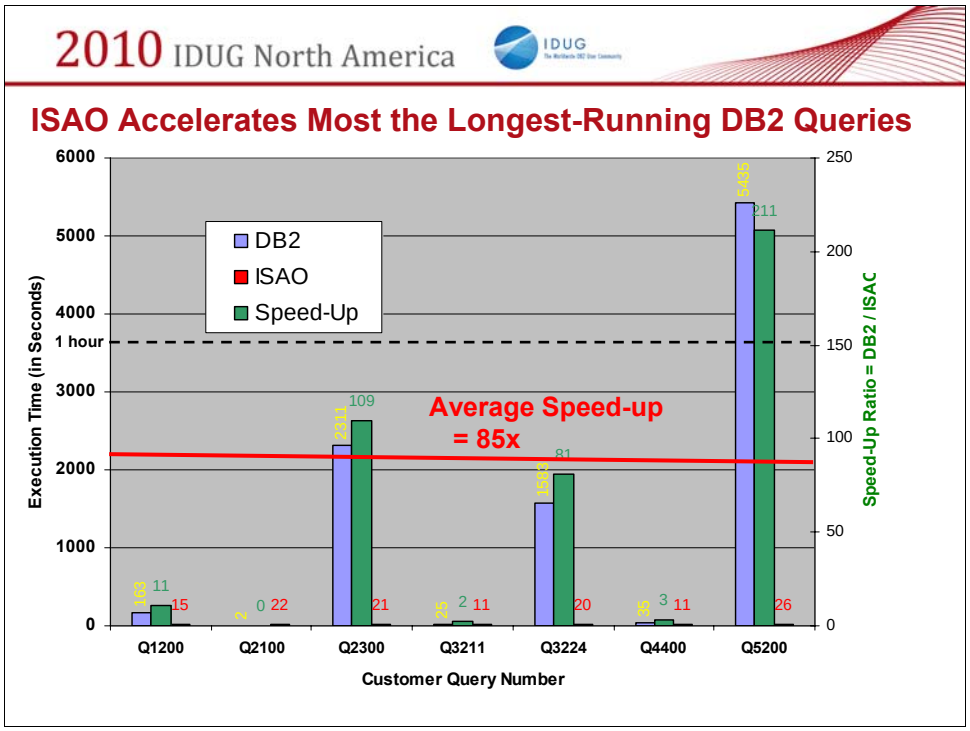


Looking a little closer at the architecture of the IBM Smart Analytics Optimizer, and the IBM Blade Center H on which it runs, we see here an example of the „Small“ configuration, with one fully-populated Blade Center containing 14 blades. (The IBM Smart Analytics Optimizer comes in 5 „T-shirt“ sizes, „Extra-Small“ (1/2 Blade Center), „Small“ (1 Blade Center), „Medium“ (2 Blade Centers), „Large“ (3 Blade Centers), and „Extra-Large“ (4 Blade Centers).) All blades contain two sockets, with a quad-core Nehalem chip in each socket, for a total of 8 cores per blade, and 48 GB of real DRAM and some flash memory used for storing intermediate results. Blades are (soft-) designated as either coordinators or workers. Coordinator blades receive queries from DB2 and broadcast them to the workers, then receive partial answers from the workers, merge the results, and return them to DB2. Only worker blades store data that has been replicated from DB2, and dedicate up to 32 GB of DRAM for storing data; the rest is working memory used for storing the system code and intermediate results. A DS 5020, connected to the Blade Center via the General Purpose File System (GPFS), is needed only to back up the accelerator system code and the compressed data of each worker node – no data is accessed on the DS 5020 during normal query operation (remember, the accelerator is a main-memory database!!). There are always at least 2 active coordinator blades to avoid a single point of failure, plus one held in reserve that can take over for any worker blade that might fail by simply loading its image from the DS 5020. While waiting for such an unlikely failure, however, the reserve blade can also act as a coordinator. Since IBM Blade Centers when fully populated hold 14 blades, this means that there can be 11 worker blades per Blade Center. If you do the math, this means that one IBM Blade Center can hold at most 11 * 32 GB = 352 GB of data. Since the IBM Smart Analytics Optimizer can achieve compression ratios of 2X to 8x, or even more, depending upon the data, we can conservatively estimate a simple rule of thumb that each Blade Center can hold at least 1 TB of raw (pre-load) data, and probably a good deal more. This means that the maximum configuration of the „Extra-Large“ configuration can hold at least 4 TB of raw data in memory.

IBM Smart Analytics Optimizer (ISAO) – Agenda

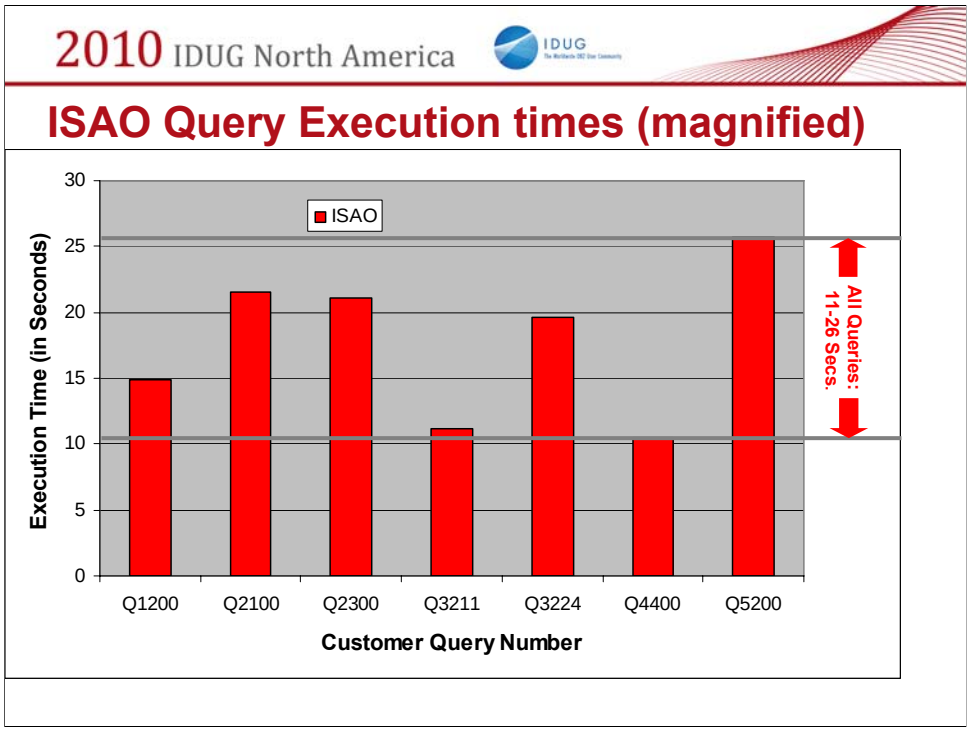
- Why and What is the IBM Smart Analytics Optimizer?
- ISAO Market – Business Intelligence
- ISAO Architecture
- **It's All About Performance!**
- From the User's Perspective
- What's the Big Deal?
- Behind the Curtain – The Query Engine Technology
- ISAO vs. Column Stores
- Conclusions

Since an accelerator is all about performance, just how good is the performance of the IBM Smart Analytics Optimizer? Well, in fact, it's really good!

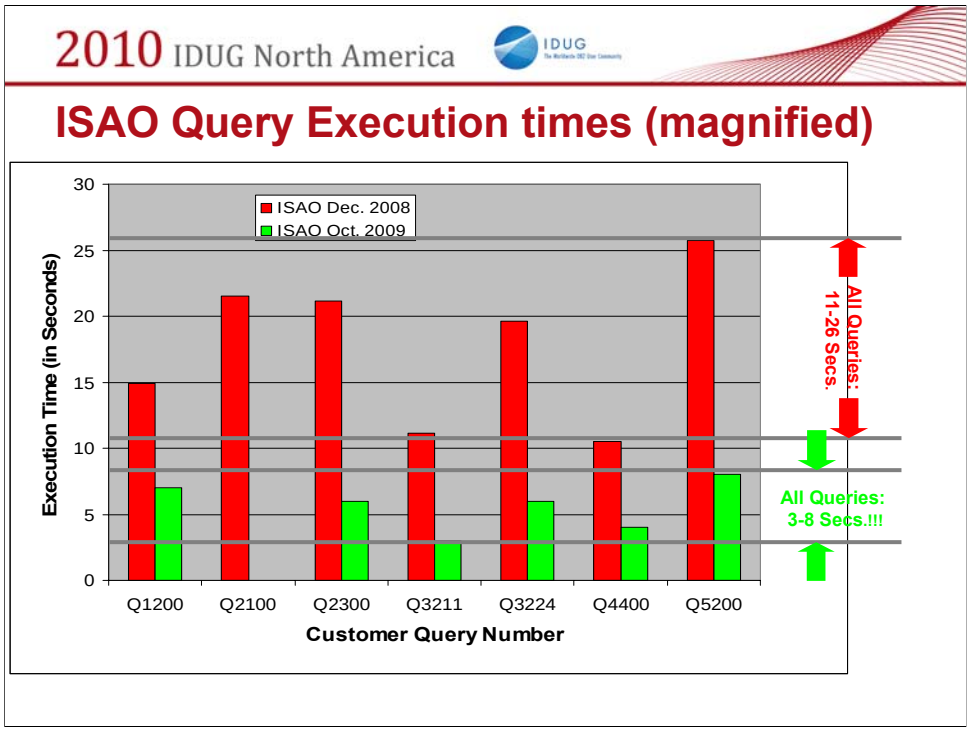


This rather complex chart shows the absolute run times in seconds (on the y-axis at the left) of queries from our alpha customer run on both DB2 for z/OS and the IBM Smart Analytics Optimizer (abbreviated hereafter as ISAO). The DB2 times are in blue, and the ISAO times are in red. Since some run times (esp. for ISAO!) are hard to see, I've put the exact time in numeral form on top of each bar, as well. In addition, we show the speed-up ratio -- defined as the ratio of the DB2 time to the ISAO time – in green bars and measured on the y-axis to the right. DB2 for z/OS performance experts at IBM's Silicon Valley Lab chose the best indexes for, and tuned, each query, in consultation with the customer. Remember that DB2 for z/OS is a disk-based system, whereas ISAO has all the data in memory!

On average, ISAO sped up this set of user queries by an average of 85 times! However, the key thing to notice from this chart is that the speed-up ratios are proportional to the run time on DB2 for z/OS, i.e., was greatest for the queries that ran the longest on DB2, This means that ISAO accelerated the problem queries the most! This is a DBA's dream! How could this be? The next chart reveals the secret – the denominator of the speed-up ratio is almost constant!

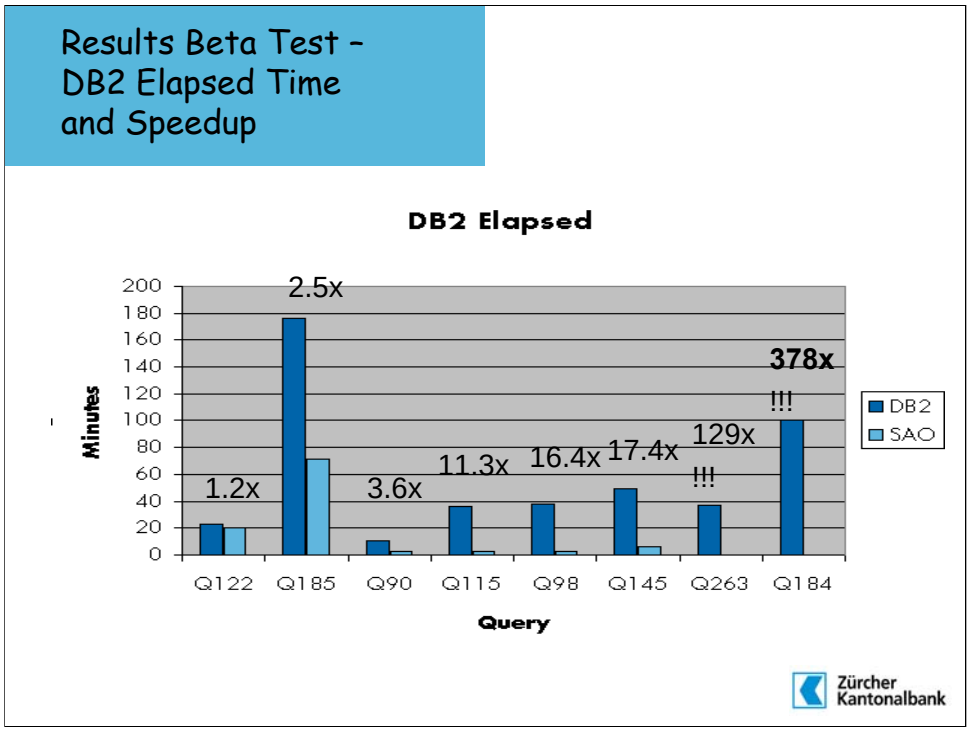


If we zoom in on the previous graph, and just look at the ISAO times (in seconds), we see the secret to why ISAO speeds up the problem queries the most – its execution time varied little from one query to the next, with all queries completing in 11 to 26 seconds, depending upon the complexity of the query. This seems remarkable, as one of the queries (Q5200) ran on DB2 in about an hour and a half, until you realize that ISAO uses essentially the same simple scan plan for every execution of any query. So in this way, execution times vary only depending upon the number of predicates, the number of GROUP BY columns, and any ORDER BY sorting.



I'm afraid I lied to you (a bit) in the previous slide, because those are actually old results, obtained in December 2008. Since that time, we've made some performance improvements and rerun the alpha customer's queries on ISAO as of October 2009. Now all the queries run in 3 to 8 seconds, an improvement of over 3x from our earlier results!

All, that is, except query Q2100. If you go back two slides, you'll see why Q2100 is missing an ISAO execution time now. The December 2008 results show that query Q2100 actually ran faster on DB2 (only 2 seconds, vs. 22 on ISAO!). So it would be better to NOT route it to ISAO, because this particular query is more of a „point query" that touches little data and probably uses the perfect index for that query. The DB2 for z/OS Optimizer estimates the DB2 execution time as part of its optimization, so it can -- and should! – avoid routing such queries to ISAO that do not fit our „sweet spot". We fixed this so that the DB2 Optimizer now makes this determination, and so when we reran the alpha customer's workload in October 2009, DB2 correctly never routed this point query to ISAO. So we don't have a measurement of ISAO's execution time for this query.



In case you thought we just got lucky in our choice of alpha customer workloads, I've included a slide proudly shown by our beta customer at last fall's Information On Demand Conference (IOD 2009), in which speed-ups ranged from 1.2 times to over 378 times! It's important to note that the speed-ups you enjoy with ISAO will depend greatly upon the schema, the data, and the query, as this chart graphically illustrates. Note that the y-axis in this graph is in minutes, not seconds, and that the long execution times for ISAO were measured early in the beta and revealed some problems when dimension tables were bigger – relative to the fact table – than we'd assumed they'd be. Once we addressed these issues, the speed-up ratios improved significantly, but the 378 times speed-up was on a different schema, in which the dimensions were several orders of magnitude smaller than the fact table.

IBM Smart Analytics Optimizer (ISAO) – Agenda

- Why and What is the IBM Smart Analytics Optimizer?
- ISAO Market – Business Intelligence
- ISAO Architecture
- It's All About Performance!
- **From the User's Perspective**
- What's the Big Deal?
- Behind the Curtain – The Query Engine Technology
- ISAO vs. Column Stores
- Conclusions

Let's now look at the IBM Smart Analytics Optimizer from the user's perspective. What does it take to set it up, and what kinds of queries will it be able to accelerate?

Getting Started: Loading Data into ISAO

A. DBA Defines Mart (Data to Accelerate)

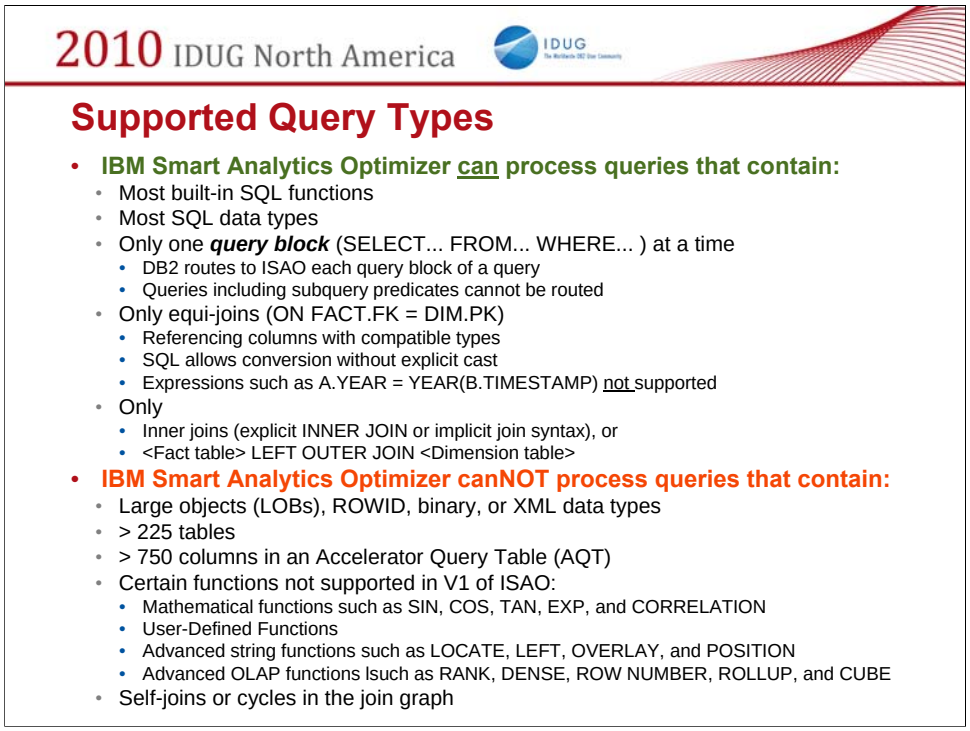
- A ***mart*** is a logical collection of tables which are related to each other.
  - For example, all tables of a single star schema would belong to the same ***mart***.
- The administrator uses a rich client interface in Data Studio to define the tables that belong to a ***mart***, together with information about their relationships.

B. DB2 Automatically Copies *Mart*

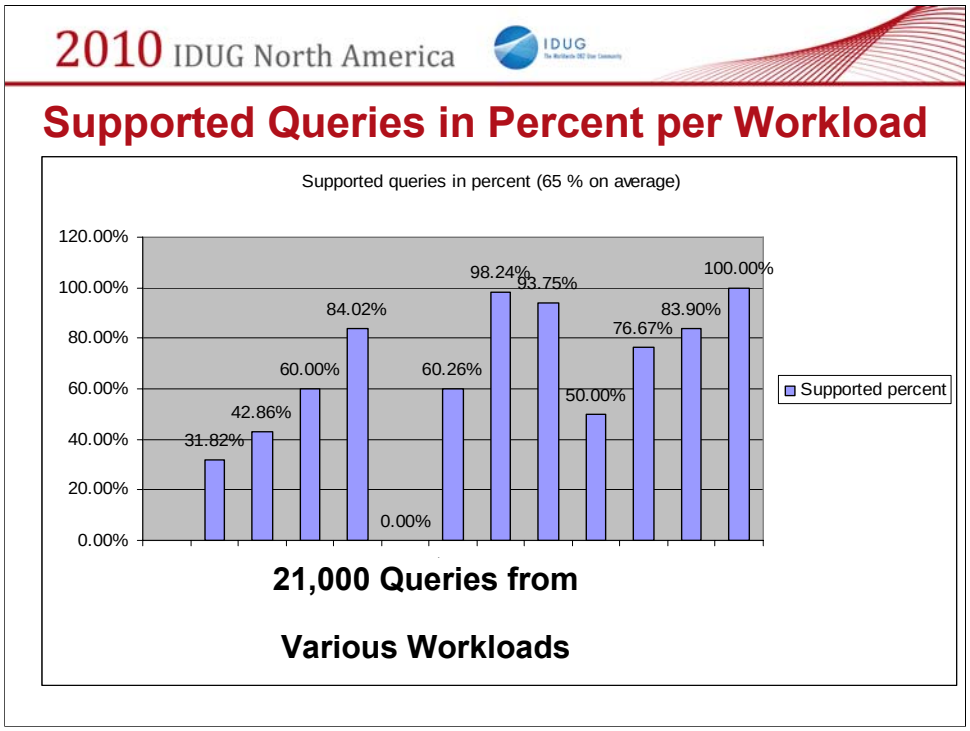
- DB2 creates definitions for these ***marts*** in its own catalog.
- The ***mart's*** data is read from the DB2 tables and transferred to ISAO.

C. ISAO Automatically Transforms & Loads *Mart* Data into a highly compressed, scan-optimized format that is kept locally (in memory) on ISAO

To get started, the user first defines a mart within the data warehouse that he or she wants to accelerate. A mart is defined as a fact table and its corresponding dimension tables, so think of it as a single star or snowflake schema. Using a rich client interface in Data Studio, the user identifies the tables involved, as well as the join relationships that related them, including whether they're one-to-many (1:n) or many-to-many (m:n) relationships. This information is then stored in the DB2 catalogs. Stored procedures are then used to populate ISAO by extracting the mart's data from DB2, piping it to ISAO, and analyzing, compressing, and loading the compressed version in ISAO. ISAO also keeps a version of the catalog information, sent in XML form, in its catalogs. You can think of this process as a form of ETL (Extract, Transform, and Load) for the ISAO mart; the transformation is the encoding that is done in parallel on ISAO. Almost all the heavy lifting on the DB2 side is done by stored procedures.



All the performance gains we highlighted in the last section won‘t be much help if most queries can‘t be accelerated, so the question arises: What percentage of *my* queries will the IBM Smart Analytics Optimizer be able to handle? That largely depends upon what data types your database contains, and what constructs your queries use. Since the IBM Smart Analytics Optimizer is a brand new query engine, in its first release we couldn‘t implement all of SQL, which is a very rich language. We have supported most built-in SQL functions and most SQL data types, but not all. And because the query router re-uses the logic of query matching used for materialized query tables (MQTs) by defining an almost identical object, called an Accelerator Query Table (AQT), it inherits some of the limitations of MQT matching. For example, the DB2 optimizer routes each query block (SELECT...FROM...WHERE...) individually. Nested correlated subqueries cannot be routed. Only very simple column-equals-column equi-join predicates can be matched – any expressions or inequalities will not match the AQT. And only inner joins and left outer joins in which the fact table is the preserved side (i.e., the dimension table side is the null-producing side) are permitted. Large objects of any sort, including LOBs, BLOBs, CLOBs, and XML data types, cannot be matched. A single AQT cannot reference more than 225 tables containing 750 columns in aggregate. Based upon an analysis of typical Business Intelligence queries, we determined that certain functions were infrequently used, such as the trigonometric functions and user-defined functions. We implemented the most commonly used string functions, but certain advanced string functions such as LOCATE, LEFT, OVERLAY, and POSITION will have to await a later release, as will the more complicated OLAP functions of RANK, DENSE, ROLLUP, and CUBE. Lastly.complicated join graphs involving cycles or tables joined to themselves are disallowed.



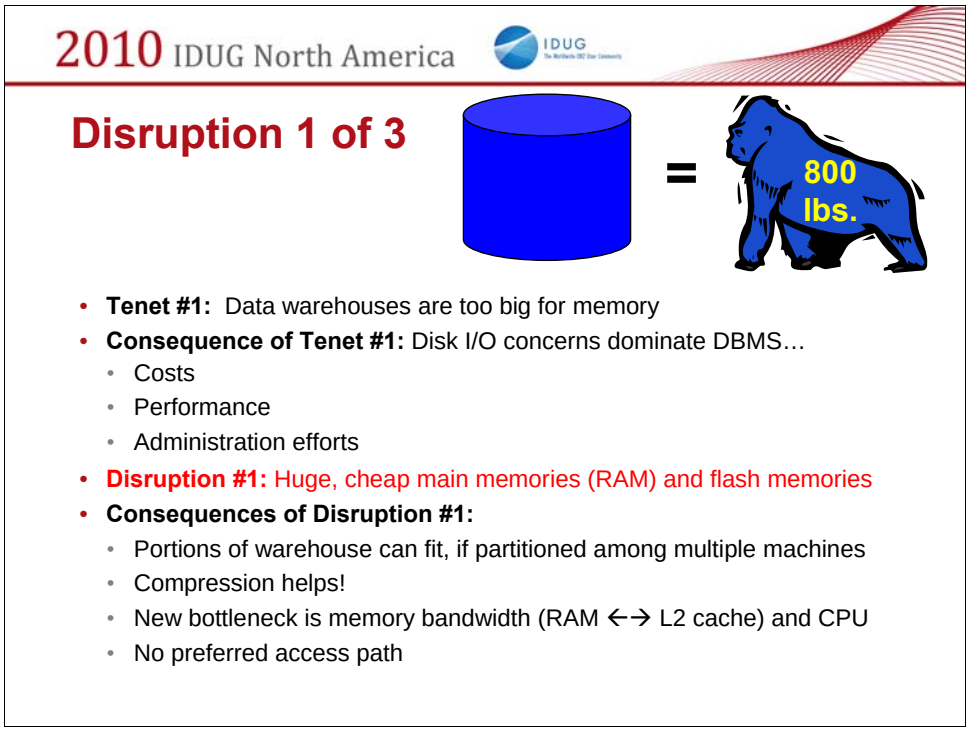
Early in the development of the IBM Smart Analytics Optimizer, we did an analysis of over 21,000 queries in the workloads of several major customers, to determine what percentage of those queries could be routed to ISAO, given the constructs we intended to support. We found that the percent of queries that could be routed varied quite a bit, but on average over 65% of the queries could be routed. "Your mileage may vary", too! For this reason, we have developed a program that will analyze your workload and data characteristics, and estimate for you how many of your queries are likely to be routable to ISAO. In this way, you can determine whether your workload will benefit from ISAO before you buy it. Keep in mind that this chart is based upon the number of queries, not the amount of time each query runs. Remember from the performance charts that queries running the longest will benefit the most, so the gains in terms of MIPS saved could be significantly more, as some long-running queries will dominate the MIPS used. Also, this was a paper study that was accomplished before we had the actual system. We hope to re-visit this study and measure the actual time saved by off-loading queries to ISAO, but for many of these workloads we don't have the actual data – just the schema and queries. We'd be delighted to evaluate your schema, data, and queries.

IBM Smart Analytics Optimizer (ISAO) – Agenda

- Why and What is the IBM Smart Analytics Optimizer?
- ISAO Market – Business Intelligence
- ISAO Architecture
- It's All About Performance!
- From the User's Perspective
- **What's the Big Deal?**
- The Query Engine Technology
- Behind the Curtain – The Query Engine Technology
- ISAO vs. Column Stores
- Conclusions

Now let's understand how all this was accomplished, and why we think it's a big deal, a disruption to the database market.

The IBM Smart Analytics Optimizer was designed from the ground up to exploit several important trends in hardware technology that are quite disruptive to database technology. Though Moore's Law continues to pretty much hold true, the increased density of chips is no longer resulting in ever-increasing clock rates, because the power consumption that that would require would result in power densities so great that chips would literally melt. So the industry has shifted toward using the increasing density of transistors on chips to have multiple CPUs, or cores. (We may have to come up with a new acronym, since *Central* Processing Unit seems quite anachronistic in this multi-core world!) to achieve greater speed via parallel execution. Unlike most legacy software, ISAO was designed explicitly to capitalize upon multi-core processors by chopping the data into small partitions, each of which can be independently processed by a core. As a main-memory database, we are of course exploiting the ever-increasing sizes, and decreasing costs, of commodity DRAM. And by spreading the data over multiple such processors and connecting them with a fast interconnect between the blades, we can accumulate quite large real memories in aggregate. Today's disk-based DBMSs were designed at a time when the latency gap between the time to randomly access something in main memory (DRAM) was only a few orders of magnitude from the time to access something on disk, As DRAMs have shrunk, so too has the time to access a particular byte in memory, while the time to move the disk arm to a particular piece of data on disk has NOT appreciably improved in decades. This means that there is an increasing gap of many orders of magnitude between DRAM and disk latencies, which propel us toward a main-memory DBMS. Hard disk drives are increasingly becoming archive storage! The architecture of ISAO exploits all these trends by starting with a clean slate, departing from 3 major tenets that database system designers have pretty much held sacrosanct for the last 4 decades. Given the extreme changes in hardware capabilities over the last 4 decades, it's actually amazing how well DBMS architectures have "weathered the storm", but we felt that the traditional architecture had been stretched to the breaking point, and it was time to rethink the whole architecture.



The first major tenet is that one couldn't fit a useful amount of any data warehouse into main memory. So it was presumed that data warehouses had to be on disk, and disk dominated our thinking about costs, performance, data placement, and administration. Disk was the "800-pound gorilla" that could not be ignored, and we put all our energies into trying to minimize it, for good reason, as the gap between DRAM and disk arm latencies grew.

The disruption is that cheap main memories and, increasingly, flash memory, when aggregated over a large number of commodity machines, start to look pretty substantial, particularly when the data is compressed. As we calculated earlier, a Blade Center can hold over a terabyte of data in real memory, and that's using the most cost-effective memory DIMMs, not the largest ones possible (which are more expensive per byte). There seems to be no immediate barrier to this trend continuing. As a consequence, many data marts in today's enterprises, and some entire data warehouses of smaller enterprises, can begin to fit in a main-memory database! And when the disk bottleneck is removed, it is of course replaced with a new bottleneck, which seems to be the memory bandwidth, that is, the speed of moving things from DRAM into the L2 cache, as well as the CPU speed itself. However, newer chips promise to have ever more cores, so probably memory bandwidth will increasingly be the new bottleneck. And since DRAM is byte-addressable, there is no preferred access path. In contrast, on disk data is mapped to a linear address space, and moving to a new location involves an increasingly expensive (relatively) disk arm movement.

2010 IDUG North America

Disruption 2 of 3

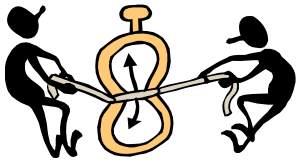
- **Tenet #2:** Need many Indexes & MQTs for scalable OLAP performance
- **Consequences of Tenet #2:**
  - Need an optimizer to choose among access paths
  - Need a ✶✶✶✶ wizard to design "performance layer" (expensive!)
  - Must anticipate queries
  - Large time to update performance layer when new data added
- **Disruption #2:** Massive parallelism achieves DB scan in seconds!
  - Arbitrarily partition database among nodes (32–64 GB RAM / node)
  - Exploit multi-core architectures within nodes (1 user or DB cell / core)
- **Consequences of Disruption #2:**
  - Only need to define 1 AQT in DB2 to satisfy many queries on the accelerator
  - Always scan tables!!
  - Accelerator automatically does equivalent of partition elimination
    - If literal is not in dictionary of that partition
  - Accelerator itself doesn't need
    - Performance layer (indexes or materialized views)!
    - Optimizer!
  - **Simpler! (no need for 4-star wizard)**
  - **Lower TCO!**
  - Consistent response times

The second major tenet of databases, held for decades, was that the only way to scale up to really large databases was via indexes and pre-materializing frequently-accessed data in MQTs, cubes, etc. As a consequence, we needed an optimizer to choose among these various performance enhancers, and we needed a 4-star wizard to choose which of the many such enhancers to create and maintain. To do that, we had to anticipate what queries might be posed in the future, usually using historical workloads that were assumed to be indicative of future workloads, a questionable assumption in BI, which is inherently ad hoc.

The disruption is that massive parallelism made possible by legions of commodity multi-core processors can divide and conquer large databases, so that scanning the data for every query becomes not only possible but desirable when accessing more than a few records. As a result, we need only define one AQT that delimits what data is in the accelerator, rather than the dozens of MQTs and indexes that a standard database requires. We always scan the database for every query, although, as we'll see in a minute, entire portions of those scans can be eliminated when the literal in a predicate doesn't exist in the dictionary for that portion. Always performing the same query execution plan (a scan) significantly simplifies the accelerator, eliminating the need for this "performance layer" and an optimizer to choose among them, significantly reducing the total cost of ownership and the variance in response times.

Disruption 3 of 3

- **Tenet #3:** Main-memory DBMSs are the same as a big buffer pool
- **Disruption #3:** Clever engineering can save lots more!
- **Examples of Disruption #3:**
  - Specialized order-preserving & fixed-length compression within partitions permits:
    - Faster access
    - Performing most operations **on encoded values (saves decoding cost)**
    - **Simultaneous application of predicate conjuncts (1 compare!)**
  - Cache-conscious algorithms make max. use of L2 cache and large registers
  - Exploit multi-core processors
  - Hash-based grouping avoids sorting

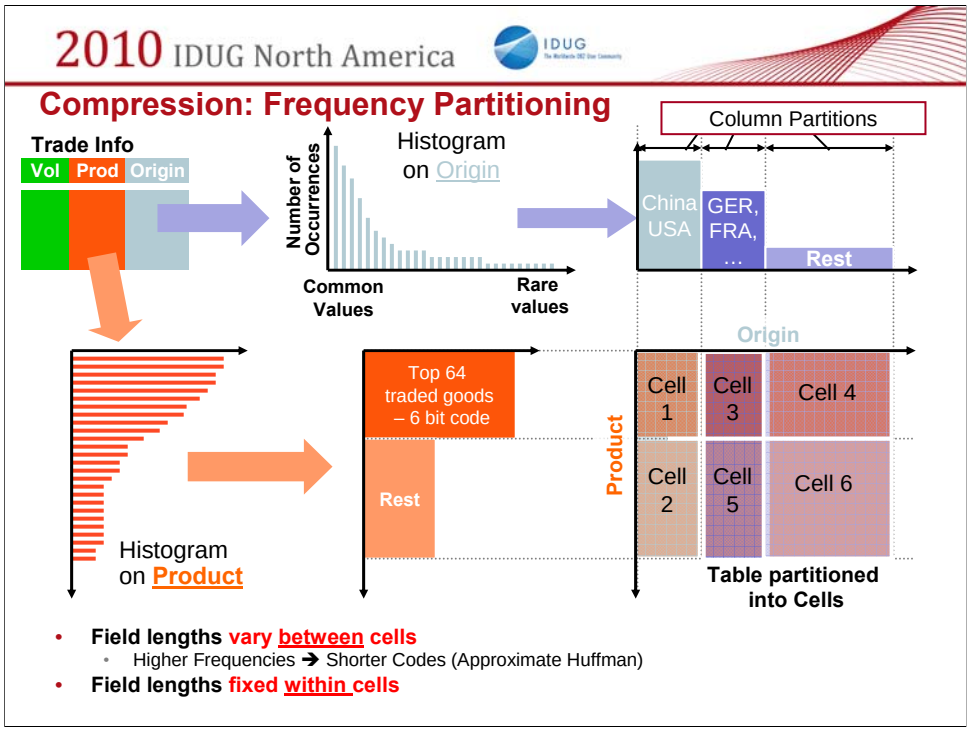


For many years, we thought that a main-memory database system was little different from a standard disk-based DBMS with a buffer pool large enough to obviate any disk I/Os. But what we have found in building the IBM Smart Analytics Optimizer is that clever engineering can do far, far better. Examples of this are described in the following slides, but include the exploitation of a proprietary compression technique that preserves the order of the underlying domain and uses fixed-length codes within database partitions, allowing us to perform most SQL operations on the encoded (compressed) values. This in turn allows ISAO to fit multiple columns into a single register, so that predicates can be applied on all those columns simultaneously. We also exploit cache-conscious algorithms and special vector operations on multiple rows at a time. Lastly, we use hashing rather than sorting to perform grouping.

IBM Smart Analytics Optimizer (ISAO) – Agenda

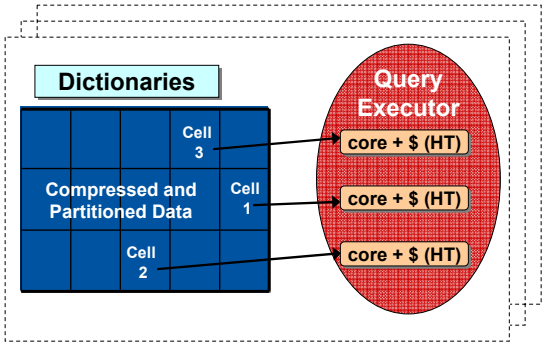
- Why and What is the IBM Smart Analytics Optimizer?
- ISAO Market – Business Intelligence
- ISAO Architecture
- It's All About Performance!
- From the User's Perspective
- What's the Big Deal?
- **Behind the Curtain – The Query Engine Technology**
- ISAO vs. Column Stores
- Conclusions

Let's now take a look at the technology that makes this revolution happen.



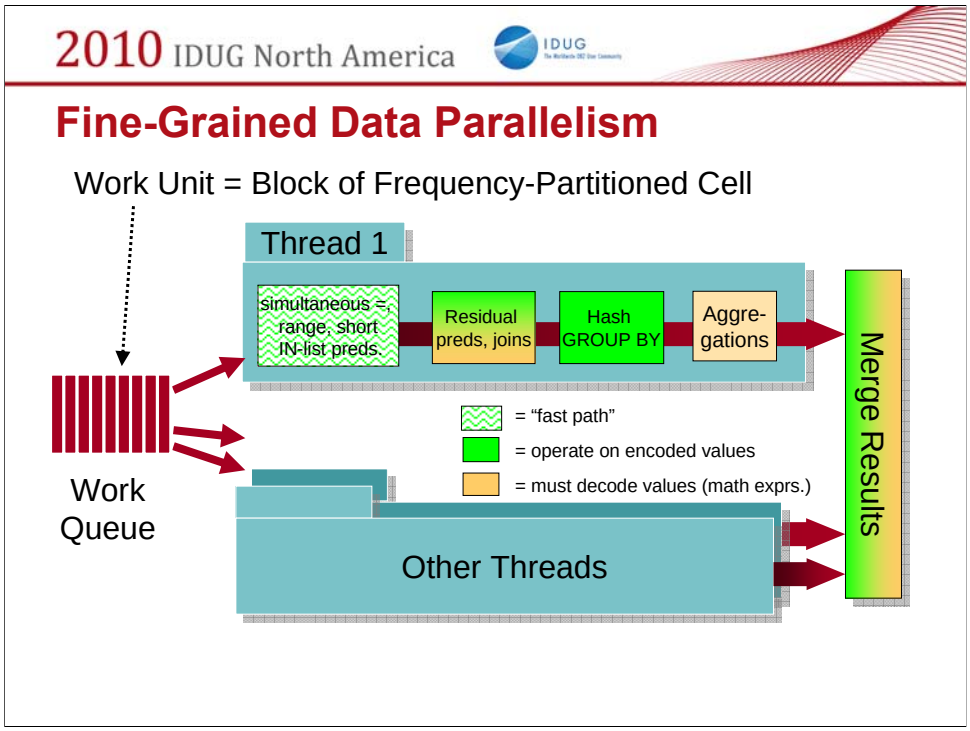
This rather busy slide is nonetheless very important, because it describes our proprietary compression scheme, called frequency partitioning. For illustration purposes, consider a fact table of sales of products having various countries of origin, shown in the upper left hand corner. At load time, we independently histogram the values of each column, in this case the Origin and Product columns. We then partition each histogram into partitions whose values all have the same fixed length. In our example, China and the USA are the most common values, and need only one bit to represent those two values. Suppose the European Union countries are next most frequent; they can all be represented in 5 bits. And all the rest of the countries could be represented in perhaps 7 bits. Similarly, we can partition the Product column into the top 64 products, representable with 6 bits, and all the rest, perhaps requiring 11 bits. The intersection of these partitions define what we call "cells" of all rows having one of the values in that partition and represented by a fixed-length code for each column. In our example, all the rows in Cell 1 have a country of origin of either China or the USA, represented in 1 bit, and one of the top 64 products, represented in 6 bits. But rows in Cell 6 represent countries other than China, the USA, or the European Union countries, in 7 bits, and an uncommon product, represented in 11 bits. If you are familiar with compression, you'll recognize that this scheme is an approximate Huffman scheme, in which the most frequent values are represented in the fewest number of bits. What's important for our purposes is that field lengths are fixed within cells, but can vary from cell to cell. So queries must be recompiled for each cell to fit the lengths of that cell. And each cell has a dictionary that maps the values in that cell to the encoded values. By construction, encoded values are assigned in the same order as the original values of the domain in that cell, so that range predicates can also be applied on the encoded values. So in our example, China would be assigned the value 0, and the USA the value 1.

Query Processing

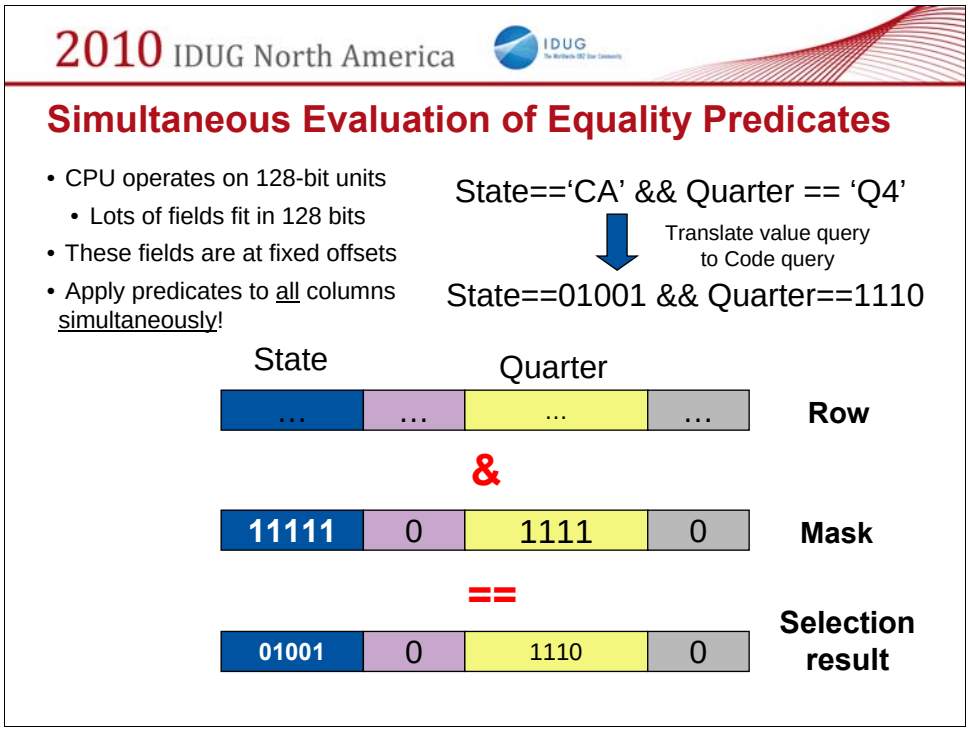


- **Cell is also the unit of processing, each cell…**
  - Assigned to one core
  - Has its own hash table in cache (so no shared object that needs latching!)
- **Main operator: SCAN over compressed, main-memory table**
  - Do selections, GROUP BY, and aggregation as part of this SCAN
  - Only need de-compress for arithmetic operations
- **Response time ∝ (database size) / (# cores x # nodes)**
  - Embarrassing Parallelism – little data exchange across nodes

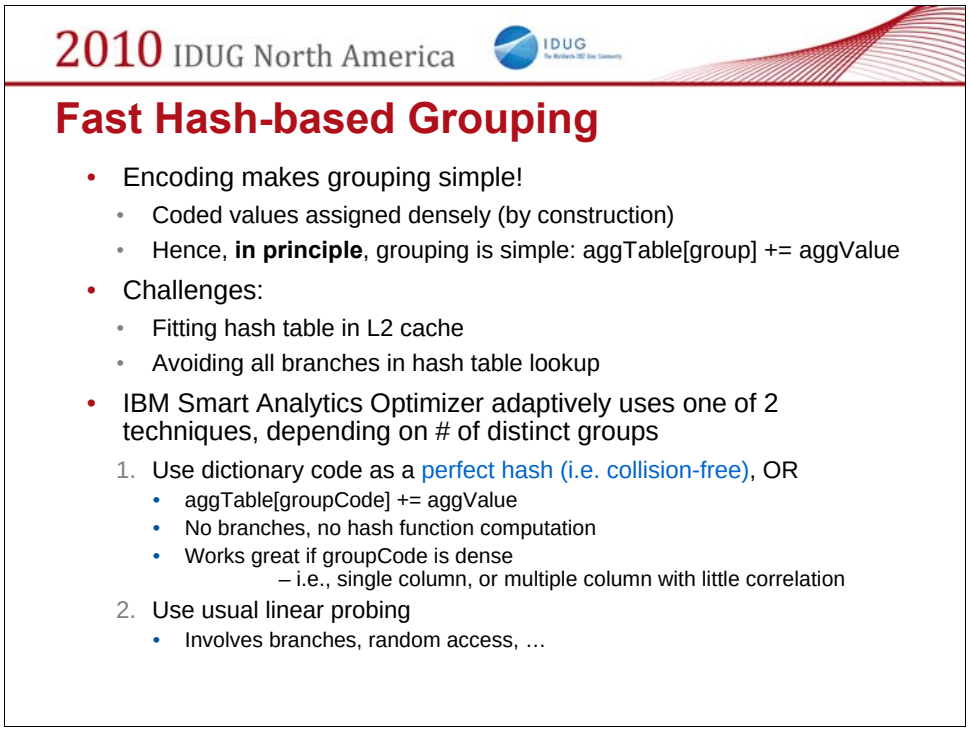
The scheme we just discussed for encoding data is also the scheme for dividing the data finely into chunks for processing. The cell is not only the unit of encoding, but also the unit of processing. Each cell is assigned to one core, and has its own hash table in cache. This ensures that each core can process its data with no shared objects, obviating the need for any locking or latching or mutexes. Processing of a cell is all done in the context of a scan over all rows in that cell, applying predicates, hashing the GROUP BY column(s) to group, and then performing the aggregate functions. These operations can all be performed on the encoded values; only arithmetic operations (e.g., in predicates and aggregate functions) require that data be decoded. This means that each cell can operate completely independently on its piece of the database, and the response time is therefore roughly proportional to the database size divided by the aggregate number of cores across all nodes – "embarrassingly parallel".



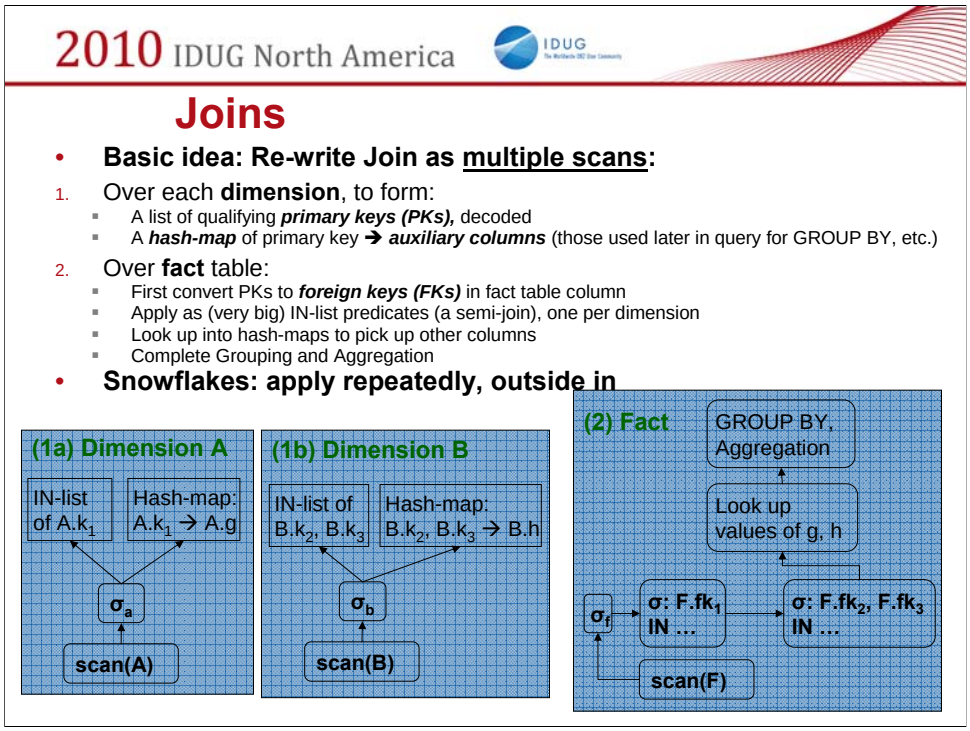
Looking a little closer at how this processing happens: When a query begins execution, all the cells for the table being scanned are put in a queue. Each cell in the queue is assigned to a thread running in a core. First to be executed are so-called "fast-path" predicates, which include equality, range, and short IN-list predicates, which can be performed in just a few instructions, as we shall see in the next slide. Any other predicate is called a "residual" predicate (these terms are not to be confused with similar terms in IMS and DB2). Next the GROUP BY columns are hashed to find the group, and lastly aggregate functions are calculated. The color scheme shows that most operations can be performed on the encoded values, denoted by green, but any arithmetic operations require decoding, shown in beige. When all rows have been so processed for a cell, the hash tables and aggregate functions for each thread are merged on a single (worker) node, and the resulting merged hash table is sent to the coordinator, where the results of all workers are finally merged, and final processing, such as ORDER BY, is performed before returning the results to DB2 and the user.



Fast-path predicates can be applied simultaneously to multiple columns that remain encoded, so many of them can fit into a 128-bit register, as shown in this slide. At compile time, the literals are encoded by looking them up in the dictionary and converting the literal to its encoded value. Here, for example, the literal 'CA' is looked up in the dictionary for the state column, and is converted to '01001'. Similarly, 'Q4' is encoded into '1110' in the dictionary for the Quarter column. Also, a mask is constructed that masks out the columns not having predicates. Then, at execution time, each row is ANDed with the mask, and compared to the encoded literal. In this way, as many predicates as columns fit into a register can be applied simultaneously. This is much faster than how a regular DBMS applies predicates to columns, loading in succession each (un-encoded) column value and comparing it to its (un-encoded) literal. Our technique for fast-path predicates can also be used for applying multiple *range* predicates simultaneously, as shown in a slide in the backup slides, but it is too time-consuming to describe it here. For details, see the VLDB 2008 paper entitled "*"Row-Wise Parallel Predicate Evaluation"*.



Rather than sorting on the GROUP BY columns, the IBM Smart Analytics Optimizer uses hashing to group rows. Usually, hashing is needed to map an inherently sparse (un-encoded) domain into a more dense hash table. However, by construction, the encoded values of the GROUP BY column(s) are assigned densely. So the encoded value can be used directly, without any hash function computation, as an index (a "perfect hash") into a "hash" table without any concern for collisions, linked lists of values, etc. and the branches they entail that might cause instruction-cache misses. However, when there is more than one GROUP BY column, the mapping may be less dense, so we still need to use the usual linear probing hash tables. While this involves computing a hash function on the GROUP BY columns and following a linked list of collision values, it still is much faster than sorting, particularly when we can keep the hash table in the L2 cache.



Since the IBM Smart Analytics Optimizer is good at doing scans, joins are performed by re-writing each join into a succession of joins. In the first step (diagrams 1(a) and 1(b)), each dimension is scanned, applying predicates local to that table. Qualifying rows then add their primary-key values to an IN-list that will be used in the fact-table scan, and their "auxiliary columns" (columns used later in the query) inserted into a hash map. This is repeated for each dimension. The fact table is then scanned (as shown in step (2)), where the IN-lists from each dimension are used to filter the corresponding foreign keys of the fact table, along with any predicates local to the fact table (shown as sigma-sub-f). Then, the auxiliary columns are fetched from the hash table, and used to do grouping and aggregation. Note that the primary key and foreign key columns, though drawn from the same domain, are encoded independently, so in reality the primary key values must be decoded and re-encoded with the dictionary of the foreign key before being added to the IN-list. For schemas having more than one level of join, this technique is applied recursively, starting from the outermost dimension tables and joining inward. In such cases, a table not outermost or the central fact table can act as both a "fact" table and as a "dimension" table in successive joins.

What About Updates?

- ISAO uses **snapshot semantics** (batch updates), common in BI
- System maintains a **currentEpoch** number (monotone increasing)
  - Think of it as a batch or version number
  - Prevents seeing incomplete updates, without needing locking
  - Bumped (N++) atomically after each batch of inserts & deletes completes
- Tables have two new columns
  - **startEpoch** – epoch in which that row inserted
  - **endEpoch** – epoch in which that row deleted (initially Infinity)
- Queries are automatically appended with two predicates:
  - startEpoch < currentEpoch AND
  - endEpoch > currentEpoch
- Encoding of updated values
  - If value is in dictionary, use that encoding
  - Otherwise, store unencoded in a special cell, called the "catch-all" cell

The IBM Smart Analytics Optimizer uses snapshot semantics, common in data warehouses, rather than transactional semantics. This means that updates are made in batches, and all updates in the batch are treated as a commit unit. ISAO accomplishes this by maintaining a "currentEpoch" parameter that increases by one after the completion of each batch update. Every table is augmented with two columns, the startEpoch, the epoch in which that row was inserted, and the endEpoch, the epoch in which that row was deleted. Each row inserted in a batch uses the currentEpoch for its startEpoch, and the endEpoch is initially assigned an infinite value. Additionally, each query is augmented with two predicates to ensure that deleted rows are omitted and rows in the process of being inserted are not partially included. In this way, we need not quiesce the database while inserting new rows, and we can still preserve the correct snapshot semantics.

The value of each column in inserted rows is looked up in its respective dictionary. If the value is found in the dictionary, that encoding is used. Otherwise, the row is put un-encoded in a special cell, called the "catch-all" cell.

IBM Smart Analytics Optimizer (ISAO) – Agenda

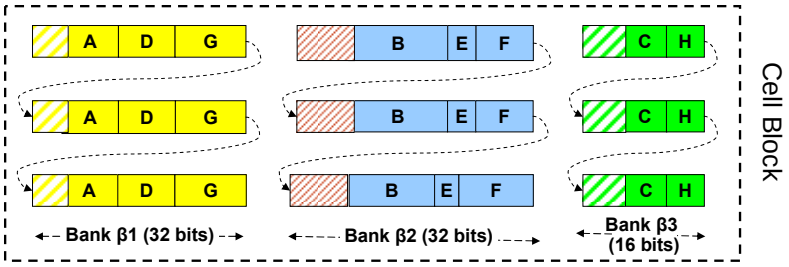
- Why and What is the IBM Smart Analytics Optimizer?
- ISAO Market – Business Intelligence
- ISAO Architecture
- It's All About Performance!
- From the User's Perspective
- What's the Big Deal?
- Behind the Curtain – The Query Engine Technology
- **ISAO vs. Column Stores**
- Conclusions

There has been a lot of hype of late surrounding the benefits of column stores to BI queries, but column stores do have some merits. ISAO is actually a hybrid of a traditional row store and a column store. We now describe this hybrid architecture, and show why we think ISAO is better than a pure column store.

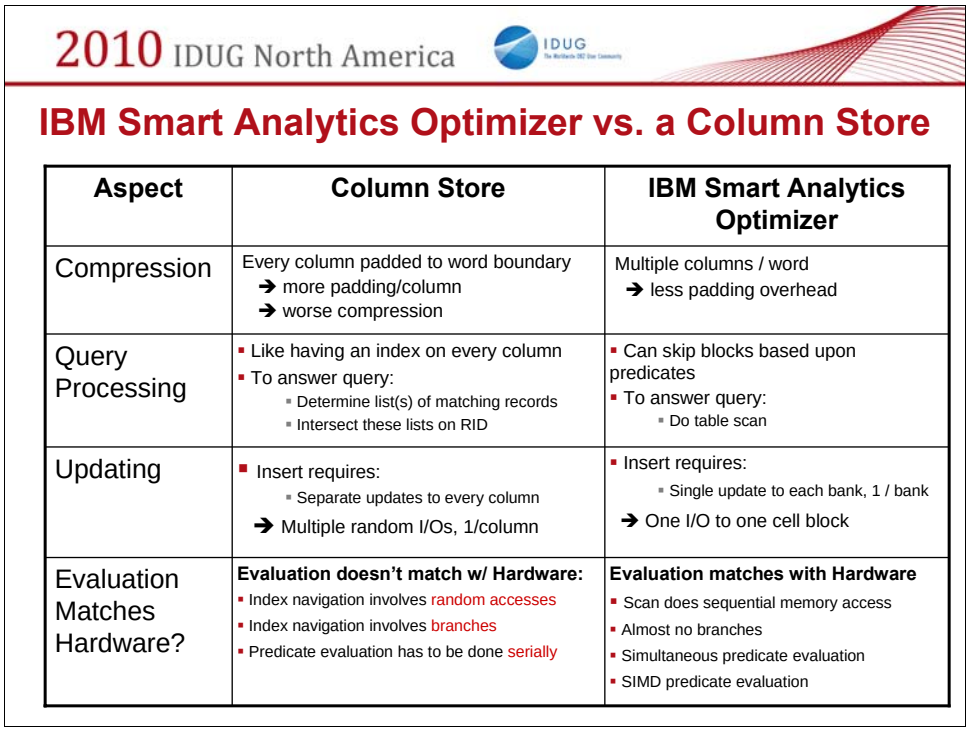
2010 IDUG North America

Banks and Tuplets

- **A *bank* is a vertical partition of a table, containing a subset of its columns**
  - Assignment of columns to banks is cell-specific, since column's length
    - Varies from cell to cell, but
    - Is fixed within a cell
  - Banks contain
    - Concatenations of the fixed-length column codes
    - Padded to the nearest fraction of a word length (8 / 16 / 32 / 64 bits).
    - We call these word-sized units ***tuplets***.
- **ISAO's bank-major layouts are a hybrid of row-major and column-major**

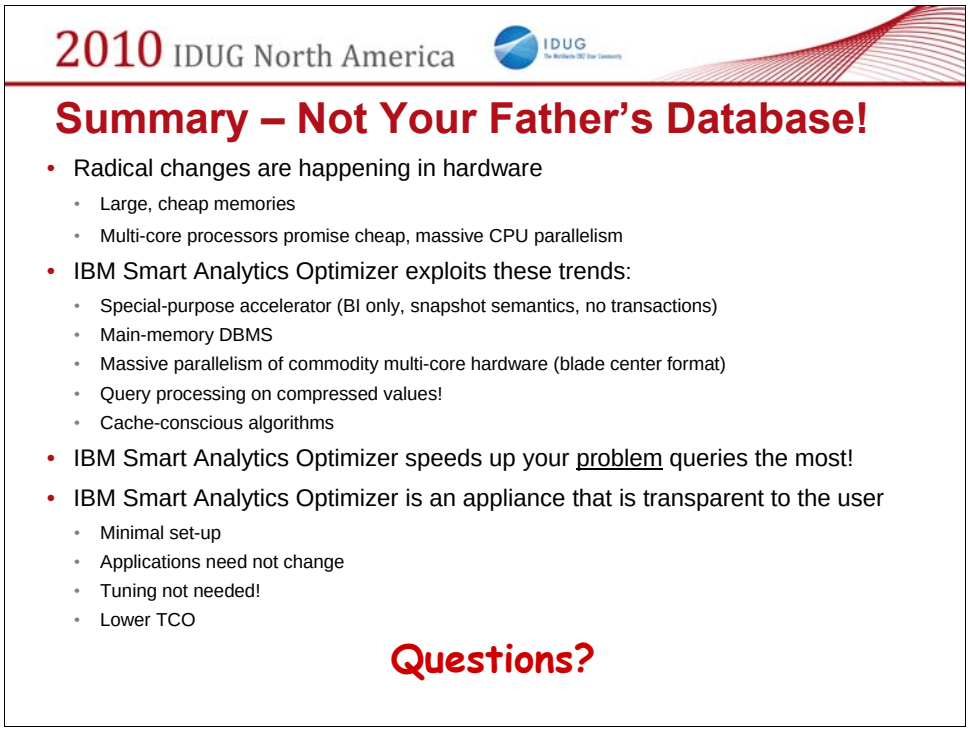


ISAO stores columns in groups, or vertical partitions of the table, called "banks". The fraction of a row (or tuple) that fits in each bank is called a "tuplet". The assignment of columns to banks is cell-specific, because the column lengths vary from cell to cell. The assignment uses a bin-packing algorithm that is based upon whether the column fits in a bank, whose width is some fraction of a word, rather than usage in a workload. In this way, multiple tuplets can be assembled in a register side-by-side so that multi-row parallelism is also achieved in SIMD (single-instruction, multiple data) fashion. Also, scans need only access the banks that contain columns referenced in any given query, saving scanning those banks with no columns referenced in the query. This projection is similar to the way pure column stores avoid disk I/Os to great advantage. The savings in ISAO aren't as dramatic because there are no disk I/Os with ISAO anyway (remember, it's a main-memory database!), but it still saves considerable CPU. All the banks for a set of rows are grouped into a large (1 MB) block called a "cell block". This organization is known as the PAX organization, as detailed in a paper by Ailamaki et al.



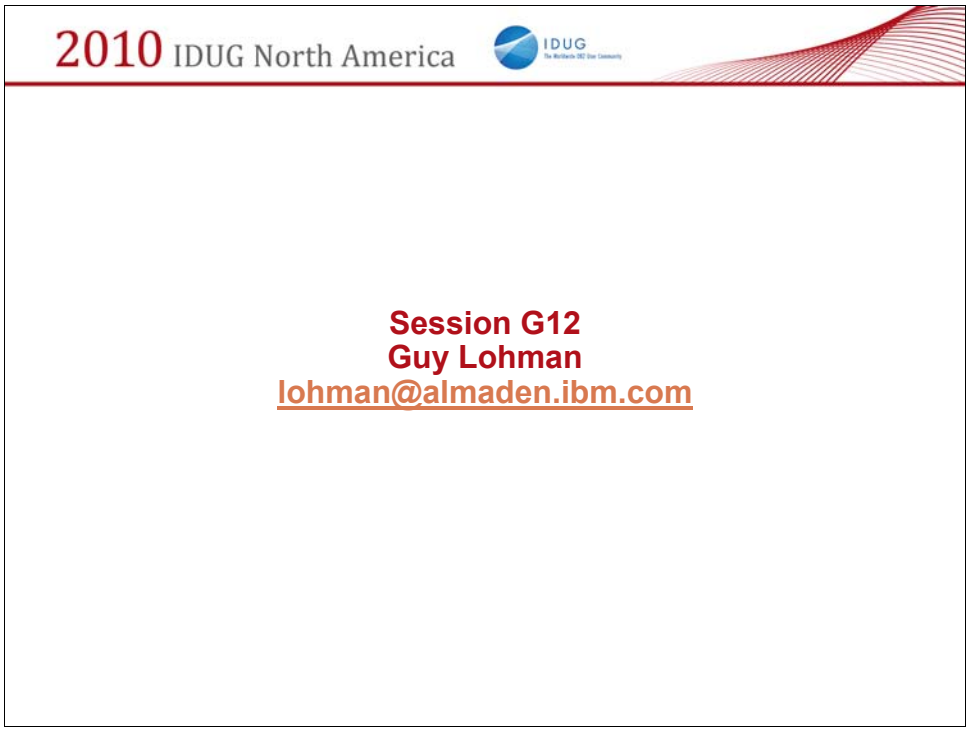
Let us compare column stores to the hybrid approach of the IBM Smart Analytics Optimizer. Column stores store each column separately, so encoded values in each column must be padded with useless bits, whereas ISAO fits multiple columns per fraction of a word, saving padding overhead. When processing a query, a column store determines a list of qualifying rows, and has to intersect these lists on the RID. However, ISAO always does a simple scan (resulting in far fewer cache misses) and can skip entire cells when literals are not found in their dictionary. Updates in a column store require one random I/O per column, whereas ISAO requires updating only each bank, but all banks for the same set of rows are stored together in one cell block in ISAO. In summary, query evaluation in a column store involves the use of index-like structures that require random accesses, and hence branches in the code that cause cache misses. This process in column stores doesn't match the hardware. But in ISAO, the scan of banks does sequential memory access with almost no branches, and evaluates predicates on multiple rows of multiple columns simultaneously, utilizing the hardware as it was designed for maximum efficiency.

If you'd like more details about ISAO, these scholarly papers (NOT the "white papers" of our competition) in the top 3 professional database conferences have been refereed by the world's leading experts, and contain a more thorough explanation of the Blink research prototype upon which ISAO is based. I recommend starting with the ICDE 2008 paper for the best overview. Note that not all features described in these papers have found their way into the IBM Smart Analytics Optimizer, but they'll give you some idea of our technology pipeline for this product developed by IBM Research.



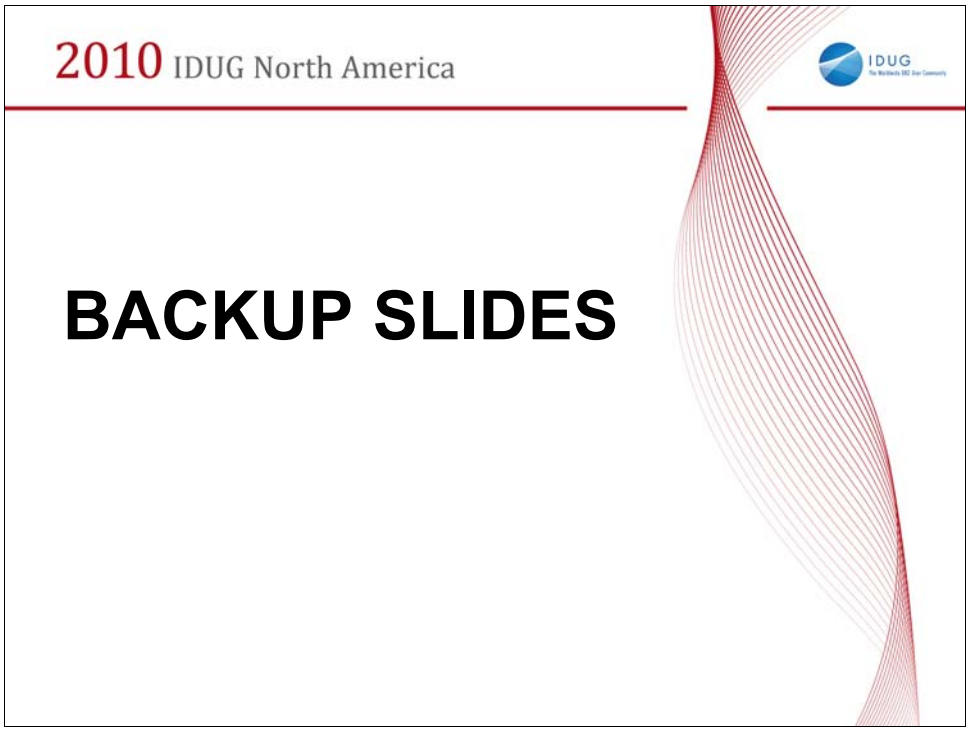
In conclusion, radical changes in hardware necessitate radical changes in software architecture. The IBM Smart Analytics Optimizer is such a radically novel architecture -- a main-memory, special-purpose accelerator for SQL querying of BI data marts that exploits these hardware trends. It also exploits proprietary order-preserving compression techniques that permit SQL query processing on the compressed values, and evaluating multiple predicates on multiple columns simultaneously, using cache-conscious algorithms. As a result, ISAO can process queries in simple scans that achieve near-uniform execution times, thus speeding up the most problematic queries the most, without requiring expensive indexes, materialized views, or tuning. This simplification is the best way to lower administration costs, and hence the total cost of ownership.

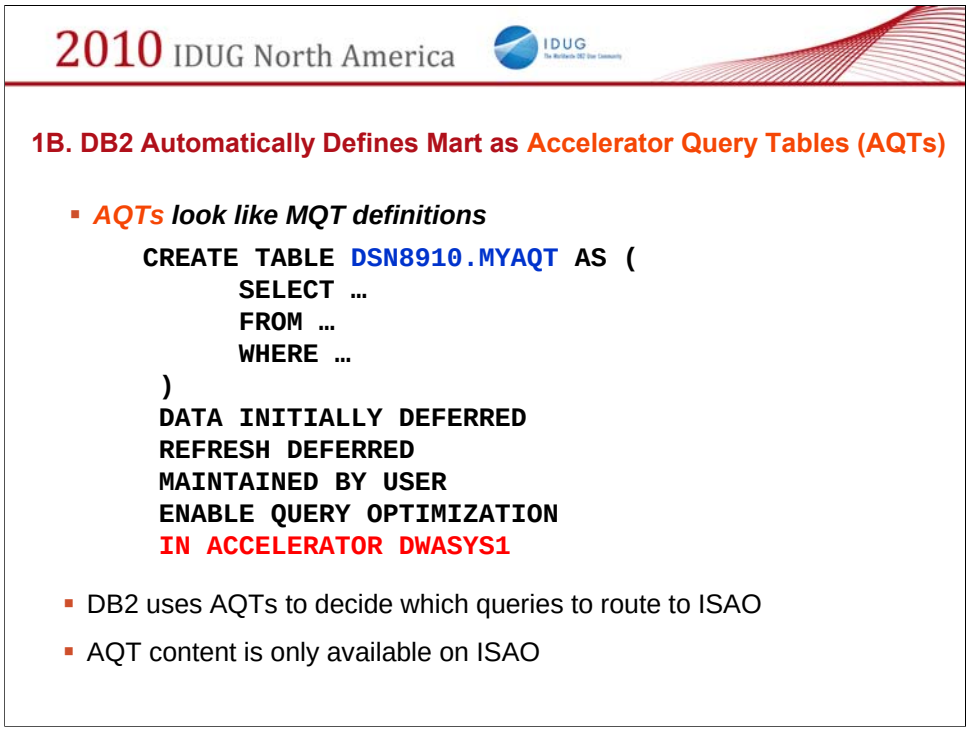
THANK YOU for your interest in the IBM Smart Analytics Optimizer!



Guy Lohman - IBM

Speaker Biography
Dr. Guy M. Lohman is Manager of Disruptive Information Management Architectures in the Advanced Information Management Department at IBM Research Division's Almaden Research Center in San Jose, California, where he has worked for 27 years. He was the architect of the Query Optimizer of DB2 on the Linux, UNIX, and Windows platforms, and was responsible for its development in Versions 2 and 5, as well as the invention and prototyping of Visual Explain and efficient sampling in DB2. During that period, Dr. Lohman also managed the overall effort to incorporate into that DB2 product the Starburst compiler technology that was prototyped at the Almaden Research Center. More recently, he was a co-inventor and designer of the DB2 Index Advisor (now part of the Design Advisor), and co-founder of the DB2 Autonomic Computing project, part of IBM's company-wide Autonomic Computing initiative. Most recently (2004-2006), he was responsible for the design of the extensions to DB2 to optimize XQuery queries in DB2 V9. In 2002, Dr. Lohman was elected to the IBM Academy of Technology. His current research interests involve disruptive machine architectures for Business Intelligence, query optimization, self-managing database systems, information management appliances, database compression, and problem determination.


2010 IDUG North America
IDUG
BACKUP SLIDES



Explain the DB2 side similarities between DWA/AQT and MQT

The data is denormalized

DB2 has to find out when or when not to use the data

Data has an age

Difference: Data is outside of DB2 and it is NOT precalculated

2. SQL Routing: DB2 Automatically Routes Queries to ISAO

- DB2 Optimizer uses AQTs (like MQTs) to decide which queries can be routed to ISAO, and which cannot.
- **Automatic -- User <u>need not</u>:**
  - Change SQL in the application
  - Be aware of whether DB2 routes an individual query

<u>Query:</u>

```
SELECT SUM(REV)
FROM SALES LEFT OUTER JOIN DIM1
   ON...
   INNER JOIN DIM2 ON...
   LEFT OUTER JOIN DIM3 ON...
WHERE DIM1.Brand = ‚Levis‘
   AND DIM3.Year = 2009
GROUP BY P_Group, Quarter, Region
```

Query-time routing

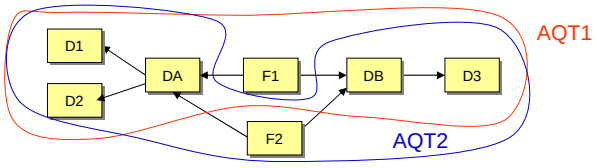
<u>Routed Query:</u>

```
SELECT  SUM(REV)
FROM MYAQT A.
WHERE A.Brand = ‚Levis‘
   AND  A.Year = 2009.
GROUP  BY A.P_Group, A.Quarter, A.Region
```

Supported Schemas

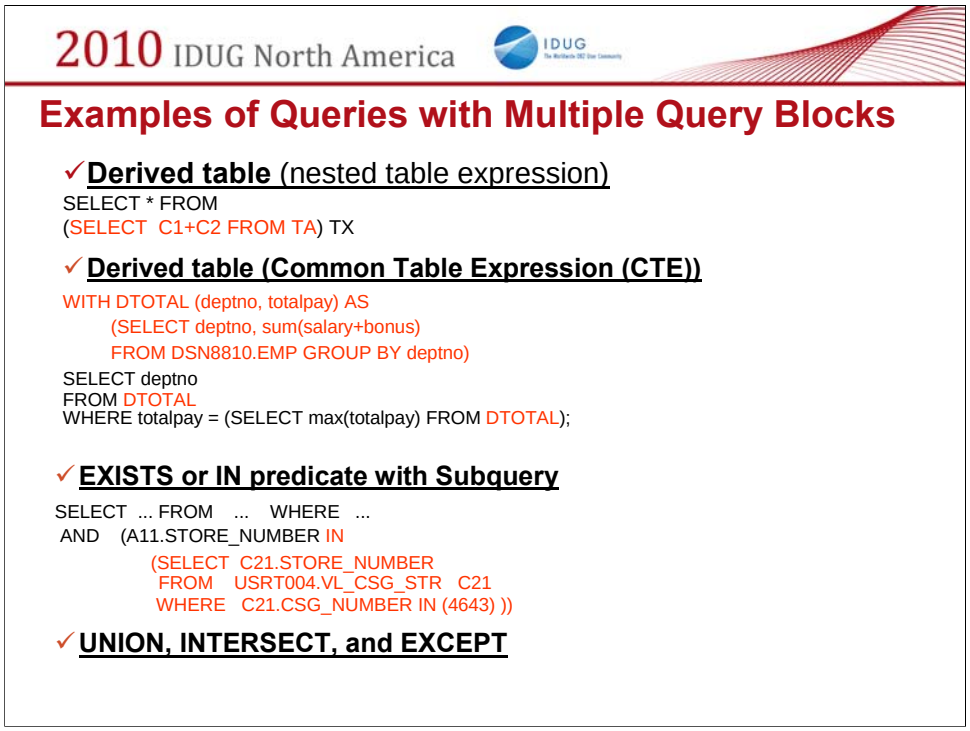
- A ***mart*** consists of a set of tables, together with their referential constraints
  - Fact tables are considered to be the tables having the highest join depth
- A mart may contain multiple fact tables
- However, ISAO cannot handle queries that cross mart boundaries
- Load-time (de-normalization) join requires:
  - Dimension join cols **must have unique constraint / primary key**
    - ➔ **Dimension : Fact cardinality is 1 : 0..n**
    - ➔ **n:m joins are supported as run-time joins only (cannot be pre-joined)**

    WHY: require loss-less join in case a dimension table is omitted from the query



Introduction of the logical construct MART

All tables of a query have to be in the same MART

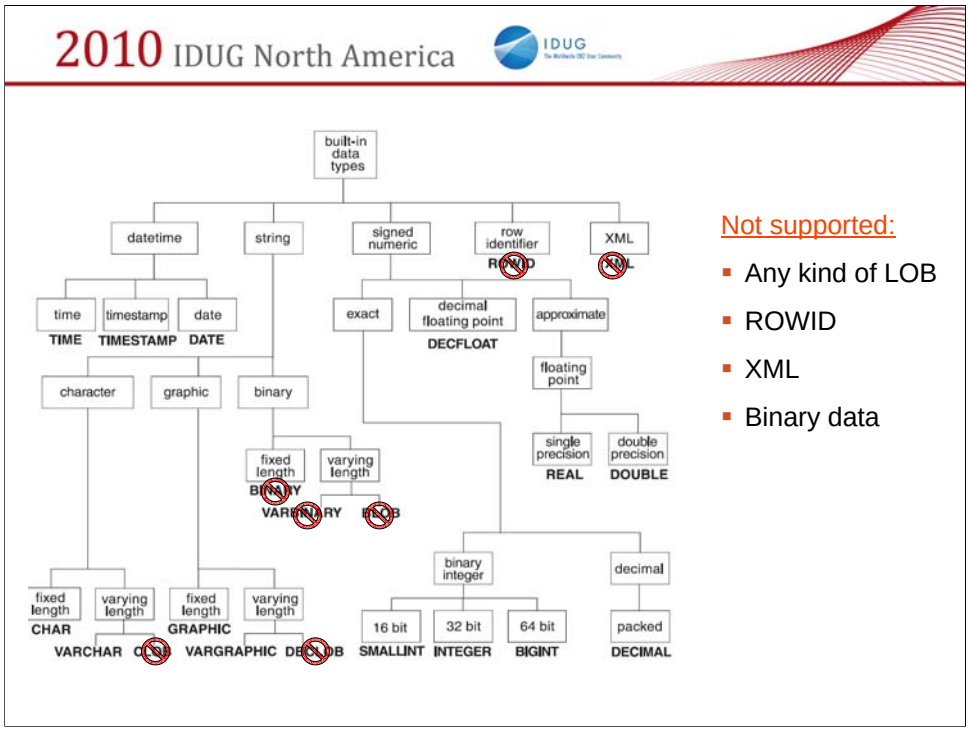


DB2 optimizer rewrites query

It tries to merge table expressions into outer query block whenver possible

Sometimes it is not possible and table expressions must be materialized into work files

Some of these query transformations (pass 2 transformations) happen after AQT/MQT matching in R1 – however merging of table expressions happens before AQT matching



Most decisions on supported types were made on priority base

We do not think that DECFLOAT is widely used.

Also binnary data seems to be rare case within a warehouse

Same goes for XML data type

Tables including such columns can still be accelerated through projection

Reasons a query might not be routed to ISAO

- Because the accelerator or AQT is currently disabled
- Because a query contains
  - References to a table or column not in the accelerated mart
  - Unsupported data type
  - Unsupported syntax
    - E.g., Subselect or FULL OUTER JOIN
  - CURRENT REFRESH AGE = 0
- Because the query does not reference a fact table
- Because the DB2 Optimizer decides that DB2 can do better
  - DB2 has a cost-based threshold
  - E.g., Query with selective predicate on indexed column is executed in DB2

EXPLAIN Will Tell You Why

- **A new EXPLAIN table is added to show:**
  - Whether or not a query block is eligible for automatic query rewrite
    - If not eligible, why it's not eligible
  - If eligible for automatic query rewrite:
    - Which materialized / accelerated query tables were considered
    - For each one not chosen, why not chosen.
- **DDL for this new EXPLAIN table:**

```
CREATE TABLE DSN_QUERYBLOCKINFO_TABLE(
   QUERYNO INTEGER NOT NULL WITH DEFAULT,
   QBLOCKNO SMALLINT NOT NULL WITH DEFAULT,
   ...
   QB_REASON SMALLINT NOT NULL WITH DEFAULT,
   QB_INFO CLOB(2MB) NOT NULL WITH DEFAULT,
) CCSID UNICODE;
```

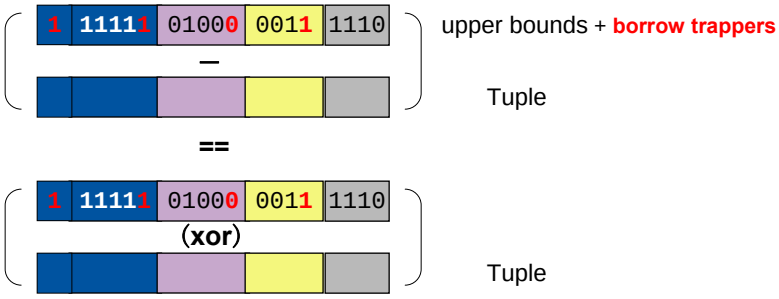
  - Column "QB_INFO" contains (in XML format) objects, functions, etc. that caused an AQT to not be chosen.

Evaluation of Range Predicates on Encoded Values

B<='CA' and C < 17 and D <= 'Q4'

Translate value query to encoded query
(exploits ***order-preserving code***)

A <= 11111 and B<=01000 and C <=0011 and D<=1110



General result: (UB – Tuple) xor (Tuple – LB) == UB xor LB