Design Advisor: Discover the brain behind the brawn

By Jun Rao and Danny Zilio

Technology advances and competition continue to significantly reduce the cost and increase the capacity of database systems, making large, complex database applications commonplace. For example, popular database applications such as SAP typically contain over 30,000 database objects, which include tables and indexes. Concurrently, the industry sustains the relentlessly rising costs of employing skilled database administrators (DBAs) to manage those increasingly complex systems. These economic trends have driven the total cost of ownership of systems today to be dominated by the cost of people, not hardware or software. This new reality has sparked recent interest in developing self-managing, or autonomic [3], systems that can relegate many of the DBAs' more mundane and time-consuming tasks to automated tools. Perhaps the best candidate to date for such automation is physical database design. DBAs have, for years, systematically tuned applications by time-consuming trial and error. They toil to methodically create each index, collect statistics on it so that the query optimizer knows its properties, recompile every query that might benefit, and then evaluate whether the index was, in fact, exploited to improve performance for that workload. Each iteration of this painstaking process could take minutes or even hours on today's terabyte-sized databases, and there is no way to assure convergence to something considered optimal.

New database features to enhance performance - such as materialized views, parallelism, and different ways to cluster data - have only compounded this frustrating process by providing more options that the beleaguered DBA must consider when designing and tuning an application. To make matters worse, these features interact in complex ways. For example, materialized views (being stored tables) require indexes. And one partitioning option is to replicate smaller tables among all nodes, resembling materialized views.

This paper describes the Design Advisor in DB2 UDB Version 8.2, the first integrated commercial tool to automatically determine aspects of physical database design, including indexes, materialized query tables, database partitioning, and multidimensional clustering of tables. Given a workload consisting of a set of SQL statements, any subset of the four supported features, and a disk space constraint, the Design Advisor provides a set of recommendations for selected features that maximally reduces the total cost of workload, while using no more space than the specified constraint. There are three key capabilities of the Design Advisor:

- Exploitation of the query optimizer to perform "what-if" analysis
- Integrated selection of multiple features

- Built-in workload compression module for scalability

Terminology

This section provides definitions for some of the terminology used in DB2 UDB.
A materialized query table (MQT) is a stored and maintained query result, more commonly known in the literature as a materialized view. MQTs were known as automatic summary tables before Version 8.1 of DB2 UDB, when support for join-only views was added. With an MQT, similar queries do not have to recalculate the same information every time they are run; this can dramatically improve overall query performance time.
A multidimensional clustering (MDC) table [2] organizes a table in a multidimensional cube. Each unique combination of dimension attribute values is associated with one or more physical regions, called blocks. A block is a basic unit of clustering and typically contains tens of pages. MDC block indexes are created at the block level for fast access. Since data is clustered in an MDC table, range queries (especially with more than one dimension) can be answered much more efficiently than secondary indexes. The design of an MDC table involves choosing the dimensions as well as the granularity (that is, the number of distinct values) of each dimension.
The DB2 Enterprise Server Edition has a data partitioning feature (DPF) that enables a shared-nothing parallel architecture [1], where independent processors are interconnected using high-speed networks. Each processor stores a horizontally hash-partitioned (referred to simply as partitioning in the rest of the paper) portion of the database locally on its disk. In DB2 UDB, an index does not have its own partitioning, but rather shares the partitioning with the table on which it is defined.

"What-if" analysis architecture

One of the key aspects of the Design Advisor is that it exploits the query optimizer to perform "what-if" analysis. Normally, the query optimizer will determine a query execution plan that leads to the best performance given the physical database design that is defined in the catalog tables. For the Design Advisor, new explain modes were added and the query optimizer was enhanced such that the query optimizer can carry out two tasks for the advisor. First, under RECOMMEND explain modes, the query optimizer can create new virtual design structures to improve a query's performance. These new explain modes will cause the query optimizer to output any of these virtual design structures into ADVISE tables that were found to improve the query's performance. Second, under EVALUATE explain modes, the query optimizer reads in virtual design structures from the ADVISE tables to extend the database design defined in the catalog tables, and treat the new structures as if they were physically present. The query optimizer can then generate query execution plans using virtual structures and provide an estimate of the plan cost.

There are several important benefits of this approach. First, it allows you to simulate the effect of a specific design structure without actually creating it in the database. Second, it guarantees that all recommended design structures (such as new indexes and MQTs) are actually used in the query execution plans generated by the query optimizer, and thus avoids the trial and error process. Finally, by using cost estimates from the query optimizer, you can quantify the potential benefit of a specific design structure to the workload.
The "what-if" analysis support in the query optimizer is implemented as a special explain mode for each design feature, except for MQT, which uses "simulated catalog" and is explained in more detail in Section 4. For example, to evaluate virtual indexes, an explain mode EVALUATE INDEXES is added. Under such a mode, the query optimizer reads virtual index definitions from an ADVISE_INDEX table and treats them as real indexes. The query optimizer then generates an execution plan, possibly using those virtual indexes. In other words, the EVALUATE mode causes the virtual indexes in the ADVISE_INDEX table to act as a temporary extension to the database catalog information.

Let’s consider an example using the SAMPLE database, and the query:
SELECT EMPTIME FROM EMP_ACT WHERE EMPTIME = 5
Without indexes, we get a query execution plan that consists of a table scan on EMP_ACT at a cost of 21 timerons. When we compile the query under the RECOMMEND INDEXES explain mode, we get a new index recommended and put into the ADVISE_INDEX table that leads to a query cost of 2 timerons. We could then set the USE_INDEX field to ‘Y’ in ADVISE_INDEX and compile the query again under EVALUATE_INDEXES mode to get the same plan and cost as we do under the RECOMMEND_INDEXES mode. The only difference is that instead of the query optimzer generating the index as in RECOMMEND_INDEXES mode, the query optimizer merely uses the index in ADVISE_INDEX to extend the database design from the catalogs to include the new index. Thus, we can use the query optimizer to generate design candidates as well as to evaluate the cost of queries using subsets of the candidates.

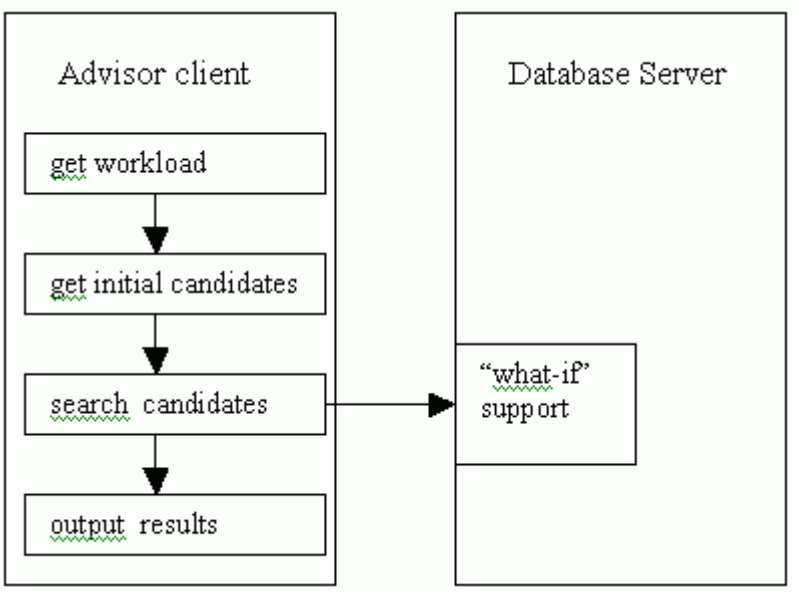

Figure 1. "What-if" architecture of the Design Advisor

The Design Advisor then exploits the database server extension to perform "what-if" analysis, as depicted in Figure 1. It first computes an initial candidate set that includes all design candidates that are potentially useful for the given workload. The computation of the initial candidate set is specific to each feature and is described in detail in the next section. The initial candidate set normally takes much more space than the space constraint. Based on estimated benefit and space consumption of each candidate, the Design Advisor iterates through promising subsets and obtains the workload cost of each subset using the "what-if" support in the database server. The subset with the lowest total cost is returned as the best solution for the workload.

Generating candidates for each feature

The Design Advisor generates candidates for each feature, as described below:

Index:

Indexes are auxiliary structures that are critical for good performance. Useful for not only evaluating equality and range predicates, indexes also allow row ordering, enforce uniqueness, and provide index-only access. On the other hand, indexes need maintenance and therefore make updates slower. By exploiting the "what-if" analysis, the Design Advisor can quantify such trade-offs for a workload that includes both queries and updates.

The initial candidate set for indexes is formed by accumulating the best indexes for each statement in the workload. The intuition here is that for an index to be useful for a workload, it normally has to be useful for at least one statement in the workload. To obtain the best indexes for a statement, the query optimizer compiles it under another explain mode RECOMMEND INDEXES. In such a mode, the query optimizer automatically creates virtual indexes that can potentially improve the SQL statement by analyzing the set of predicates applied, the set of columns referenced and the ordering requirement for each table. Those indexes used in the execution plan generated by the query optimizer are considered as the best indexes for a single statement.

Materialized Query Table (MQT):

Similar to indexes, MQTs take extra space and need maintenance. Therefore, it's important to define MQTs that can be exploited by multiple statements in the workload. The Design Advisor obtains the initial MQT candidate set through a sophisticated multiple-query optimization (MQO) that finds common sub-expressions (CSE) for a set of SQL statements. MQO first generalizes each SQL statement by changing local predicates to additional grouping columns. For example, a query with predicate "COL1=5" is transformed to a new query by removing the predicate and adding a "GROUP BY COL1". Notice that the original query can still be answered by the new one. By removing local predicates, an MQT is more likely to accommodate commonalities in the workload. MQO then traverses through the internal representations of all statements and computes the maximum CSEs, each of which is then converted to an MQT definition. The following example shows the MQTs generated by MQO using two queries Q1 and Q2 as the input. Note that the first candidate MQT1 is derived from the common subquery, and the second candidate MQT2 merges the two queries.

```
Q1:   SELECT A FROM R WHERE B>5 AND C IN (SELECT D FROM S
      WHERE E<10)
Q2:   SELECT A FROM R WHERE B<25 AND C IN (SELECT D FROM S
      WHERE E<10)
MQT1: SELECT D,E FROM S GROUP BY E
MQT2: SELECT A,B FROM R WHERE C IN (SELECT D FROM S
      WHERE E<10) GROUP BY B
```

In general, the MQTs generated by MQO are much more effective than those generated by hand for several reasons. First, because MQO analyzes the internal representation of SQL statements, it's able to identify commonalities that are syntactically different. Second, MQO exploits advanced SQL functionalities such as CUBE BY and GROUPING

SETS when generating MQT definitions. Last but not least, the MQT definition is automatically adjusted to support incremental maintenance. For example, if an MQT computes the sum of a column and is defined as refresh immediate, a COUNT(*) is automatically added to the select clause in the MQT definition.

Finally, the "what-if" support for MQTs is achieved through "simulated catalog" instead of explain modes. The former can simulate more complex design structures, but takes some extra setup time. When starting, the Design Advisor makes a copy of the current catalog and simulates MQT candidates by modifying the catalog copy. The simulated catalog is only available to the connection from the Design Advisor and is automatically deleted when the Design Advisor finishes.

Partitioning:

Partitioning design includes selecting a set of columns as the partition key, as well as a set of nodes to which the data will be distributed. A good partitioning design can significantly improve workload performance. For example, if the partitioning key of a table T matches the join key in a query, data in table T does not have to be moved to other nodes, which reduces the communication overhead. Similarly, if the partitioning key of a table T is a subset of the grouping columns, the aggregation operation can be performed locally at each node.

The initial candidate partitioning set is generated in a similar fashion as indexes by accumulating the best partitioning that minimize the data movement for each SQL statement in the workload. The Design Advisor also ensures that each partitioning candidate does not introduce too much skew; that is, where one node has significantly more data than others. For example, the Design Advisor will reject a candidate partitioning key if it has very few distinct values (for example, gender) or it has significantly uneven distribution (for example, 90% of data has the same partitioning key value). Normally, the Design Advisor detects skew by checking the statistics tables in the catalog. If the sampling option is specified, it will sample the base tables to obtain statistics on the fly.

Multi Dimensional Clustering (MDC):

The key to MDC design is to control the number of cells. Too many cells introduce significant storage overhead and too few cells degrade query performance. The number of cells of an MDC is determined by the number of dimensions as well as the degree of coarsification at each dimension. A dimension can be coarsified through a mathematical expression. For example, INT(COL1/k) coarsifies COL1 (assuming of INT type) by a degree of k because the former produces k times fewer distinct values than the latter.

Depending on the workload, one degree of coarsification might provide better clustering than others. The Design Advisor exploits a sampling-based technique to select appropriate dimensions as well as their coarsification degree. A coarsification is implemented as an extra generated column expression in the table definition. When Design Advisor searches for MDC recommendations, it also considers cases where a traditional single dimensional clustering RID index would be preferable over an MDC solution. The Design Advisor will consider the workload and choose an implementation that gives better overall performance.

Integrated feature selection

Different features tend to interact with one another. For example, an MQT, like a regular table, normally needs indexes defined on itself in order to be attractive to a query. The selection of an MQT, on the other hand, can also make an index useless, and vice versa. Therefore, when designing multiple features, it's critical to consider the overall impact. For example, searching the best indexes, followed by searching the best MQTs does not necessarily give the best set of indexes and MQTs for a given workload.

The Design Advisor uses an integrated search algorithm to ensure that important interactions among features are captured. For example, when recommending indexes and MQTs, the Design Advisor searches the initial candidate sets for both indexes and MQTs jointly to identify the best combination.

In addition, the Design Advisor can automatically include additional necessary features, even if the user does not specify them explicitly. For example, if a user only requests MQT selection, the Design Advisor automatically recommends indexes and partitioning (if in DPF-enabled database) on MQTs to make them more usable.

Built-in workload compression

The Design Advisor supports several methods of obtaining a workload automatically, such as from the Query Patroller or from the cache of recently executed queries. However, a workload retrieved using these methods could yield many queries. A key factor that affects the scalability of the Design Advisor is the size of the workload. Since a significant amount of work in the Design Advisor involves compiling statements in special explain modes, the larger the workload, the longer it takes the advisor to run. The time to run the Design Advisor typically grows exponentially with a linear increase in the workload.

The DB2 Design Advisor has a built-in workload compression module. It will invoke the module if it detects that the workload is too large and the analysis cannot finish in a reasonable amount of time. One important requirement for the compression module is efficiency. Only a relatively small fraction of the total amount of processing time can be spent on compressing the workload. Therefore, the Design Advisor takes an approach that only keeps the top K most expensive statements, whose total cost is no more than a certain percentage of the original workload cost. The cost of each statement is obtained from the query optimizer's estimates. The advisor then continues to work on the reduced workload, and thus can finish the analysis much sooner. Because most of the expensive statements are preserved, the quality of the overall design solutions will not degrade significantly.

The following section presents two sets of experiments. All experiments were conducted on DB2 UDB Version 8.2. The first experiment shows a test of the Design Advisor that involves selecting all four features. The second experiment focuses on the selection of indexes and MQTs only, and demonstrates the benefit of the built-in workload compression module. A summary of the experimental results is included.

Sampling usage in decision making

Another significant feature that distinguishes the Design Advisor from other vendors' products is the use of sampling from a real database to assist the query optimizer with making decisions. For example, the Design Advisor uses sampling to determine realistic statistics for its MDC tables (such as cell cardinalities) and virtual MQTs (such as table and column cardinalities). These sampled statistics are used for the virtual design structures to provide the query optimizer with useful information for making its decisions and for deriving accurate query execution plan costs. Sampling is also used by the Design Advisor to carry out data mining on the virtual design structures, including column correlation between MDC dimensions and correlation between columns in a partitioning key.

Capturing a workload for the Design Advisor to analyze

In previous versions, the Design Advisor obtained workloads from the command line (using a single statement), from a file, or from the ADVISE_WORKLOAD table. With Version 8.2, the Design Advisor also captures or can import workloads from the dynamic SQL snapshot, from the package cache, from explain tables, from an event monitor, and from Query Patroller. Each of the new types of workloads are briefly described below:

- The dynamic SQL snapshot avoids recompilation by storing the plans for all dynamic SQL queries submitted to the database engine. Each dynamic SQL statement is cached, together with the frequency of execution. Therefore, the dynamic SQL snapshot serves as a good source for a typical workload. With this new option, you can first run your typical applications and then invoke the Design Advisor, which automatically collects the SQL statements and associated frequencies from the cache. (From the Command Line Processor (CLP), the -pkg or -g option is used to capture the workload from the dynamic SQL snapshot. From the GUI, simply select it from the dropdown list.)

- Query Patroller is a powerful query management tool that provides the capability to classify queries into classes, prioritize queries, and track runaway queries. With Version 8.2, the Design Advisor can fetch all statements passed through Query Patroller. To do this, use Query Patroller to select from the historical record of past queries. Select a sampling of data by using some of the following strategies (there are many other strategies not listed here):
    - Select from the top 20% largest queries
    - Select a random sample
    - Select a stratified sample that includes queries that access each table
    - Select a stratified sample that includes queries from each tool (e.g., Business Objects versus SAS)
    - Include the entire set of queries that could be in the 10,000 to 1,000,000 range (this is possible, but not recommended).

    After you select the strategy, query the Query Patroller tables and export the queries to a file. You can then direct the Design Advisor to read your input file by using the -qp option, when using the CLP, or by selecting it from the dropdown list, when using the GUI.

You can capture the following types of workloads only from the Design Advisor GUI by selecting it from the dropdown list:

- The package cache contains package and section information required for the execution of static SQL statements that are placed in the catalog. Any static SQL statements that are found in application packages are available for import into the Design Advisor.
- The explain tables, EXPLAIN and ADVISE, contain all the explained SQL statements.
- When you create an event monitor, the SQL statements gathered in the statement table are available for import into the Design Advisor.

Experimental results

The Design Advisor was tested using a 1 GB warehousing database stored on an 8 CPU AIX(R) 5.2 system with 4 logical partitions. The workload contains all the 22 queries used in the TPC-H workload and all four features are selected.

The test started with a baseline design that stored the tables across all 4 partitions and used the primary key as the partition key for all tables except for LINEITEM, which was partitioned on L_PARTKEY, part of the primary key. The LINEITEM partitioning was chosen based on the fact that L_PARTKEY is used in quite a few of the 22 queries. The rest of the baseline physical DB design is derived from the TPC-H [4] database schema.

Design Feature	Number Recommended
Indexes	20
MDC dimensions	6
Partitioning Changes	4
Materialized Views	2

Figure 2. Design Advisor recommendations for a warehousing database

The Design Advisor was able to finish the design of all features in about 10 minutes. Table 1 shows how many recommendations the advisor made for each design feature. For example, the following results show one of the MQTs recommended (MQT2) that matches query Q18 in both the outer query and the inner query. In this MQT, the partitioning key was also properly selected on C1, because the MQT needs to be further joined with the ORDERS table to compute Q18. An index IDX3 was also recommended on MQT2. Notice that the index key is ordered in (C0,C1). This is because the subquery result in Q18 was subsequently filtered though a range predicate on L_QUANTITY.

```
CREATE SUMMARY TABLE MQT2 AS (
     SELECT SUM(L_QUANTITY) AS C0,
          L_ORDERKEY AS C1
   FROM TPCD.LINEITEM
   GROUP BY L_ORDERKEY)
DATA INITIALLY DEFERRED
REFRESH DEFERRED
PARTITIONING KEY (C1)
IN TPCDLDAT

CREATE INDEX IDX3 ON MQT2
(C0 DESC,
 C1 DESC)
ALLOW REVERSE SCANS
```

TPCH Q18:

```
select c_name, c_custkey,o_orderkey,o_orderdate, o_totalprice, sum(l_quantity)
from tpcd.customer,tpcd.orders,tpcd.lineitem
where o_orderkey in ( select l_orderkey
        from tpcd.lineitem
        group by l_orderkey
            having sum(l_quantity) > 300)
  and c_custkey = o_custkey
  and o_orderkey = l_orderkey
group by c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice
order by o_totalprice desc,o_orderdate
```

The following results also show an MDC recommendation from this experiment. The recommendation was for the PARTSUPP table to be MDC clustered (as shown by the ORGANIZE BY clause) based on a newly generated column. This column groups the values of PS_PARTKEY in such a way that each group falls into a clustered block of the MDC. In this particular case, a singledimensional MDC is better than a conventional clustered index because the generated column condenses the value domain.

```
CREATE TABLE TPCD.PARTSUPP (
    PS_PARTKEY INTEGER NOT NULL ,
    PS_SUPPKEY INTEGER NOT NULL ,
    PS_AVAILQTY INTEGER NOT NULL ,
    PS_SUPPLYCOST DOUBLE NOT NULL ,
    PS_COMMENT VARCHAR(199) NOT NULL,
    MDC040303204738000 GENERATED
        ALWAYS AS (
            INT((PS_PARTKEY-11)/(792))) ) PARTITIONING KEY (PS_PARTKEY)
IN TPCDTDAT
ORGANIZE BY (MDC040303204738000 )
```

Finally, note that besides the partitioning recommended for MQTs, the partitioning key of LINEITEM is also changed from L_PARTKEY to L_ORDERKEY. The latter is useful for fewer, but much more expensive, queries.

The DB2 Design Advisor makes its recommendations based on estimated response times for workloads using the cost model in the query optimizer. This experiment achieved an estimated

response time improvement over the baseline (without recommendations from the Design Advisor) of 88.01%. All recommendations made by the Design Advisor were implemented and the actual cost of the workload was measured. Figure 3 the real performance improvement as 84.54% (the baseline is normalized to 100%), which is very close to the query optimizer's estimation.

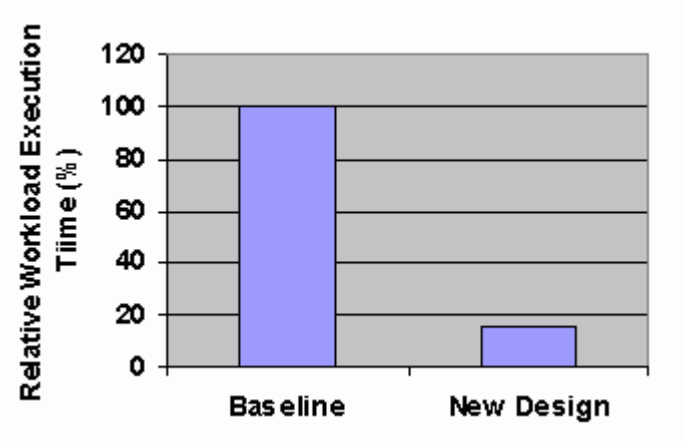

Figure 3. Workload performance improvement

Conclusion

The DB2 Design Advisor is the first comprehensive physical database design tool to recommend indexes, MQTs, partitioning, and clustering for multiple dimensions in an integrated and scalable fashion. Using the "what-if" analysis, the Design Advisor is able to accurately and efficiently recommend physical design that significantly improves workload performance. The Design Advisor also has built-in workload compression for reducing the execution time of the advisor without sacrificing quality in the solution. Initial experimental results verify that solutions selected by the Design Advisor improve by almost 100% the performance of workloads of hundreds of queries after running for under three hours, less time than it would take a human DBA to evaluate a handful of possible solutions. This achievement represents a major advance in automating perhaps the most complex and time-consuming task that DBAs now perform.

References

1. Chaitanya K. Baru, Gilles Fecteau, Ambuj Goyal, Hui-I Hsiao, Anant Jhingran, Sriram Padmanabhan, Walter G. Wilson: An Overview of DB2 Parallel Edition. SIGMOD Conference 1995: 460-462
2. Sriram Padmanabhan, Bishwaranjan Bhattacharjee, Timothy Malkemus, Leslie Cranston, Matthew Huras: Multi-Dimensional Clustering: A New Data Layout Scheme in DB2. SIGMOD Conference 2003: 637-641
3. http://researchweb.watson.ibm.com/autonomic/ manifesto/
4. TPC-H benchmark, http://www.tpc.org/
5. **Danny Zilio (IBM Toronto Lab), Jun Rao (IBM Almaden), Sam Lightstone (IBM Toronto Lab), Guy Lohman (IBM Almaden), Adam Storm, Christian, Garcia-Arellano (IBM Toronto Lab), Scott Fadden (IBM Portland): DB2 Design Advisor: Integrated Automatic Physical Database Design. VLDB 2004, Toronto.**