CODD: COnstructing Dataless Databases

Rakshit S. Trivedi I. Nilavalagan Jayant R. Haritsa*
Database Systems Lab, SERC/CSA
Indian Institute of Science, Bangalore 560012, INDIA

ABSTRACT

Effective design and testing of database engines and applications is predicated on the ability to easily construct alternative scenarios with regard to the database contents. A limiting factor, however, is that the time and/or space overheads incurred in creating and maintaining these databases may render it infeasible to model the desired scenarios. In this paper, we present **CODD**, a lucid graphical tool that attempts to alleviate these difficulties through the construction of "dataless databases". Specifically, CODD implements a unified visual interface through which databases with the desired meta-data characteristics can be efficiently simulated without persistently generating and/or storing their contents. Metadata validation is incorporated to ensure that the simulated database is both legal and consistent. CODD is currently operational on a rich suite of popular database engines, and introduces two additional facets of relevance to test teams: First, it supports a cost-based database scaling model, in addition to the size-based scaling models that have long been in vogue. Second, it provides for largely automated meta-data transfer across different engines, facilitating the comparative study of systems. We showcase here the ability of CODD to elegantly simulate a variety of testing scenarios ranging from legacy applications to Big Data environments.

Categories and Subject Descriptors

H.2.7 [**Database Management**]: Database Administration—*Data dictionary/directory*

General Terms

Experimentation, Reliability

Keywords

Relational Databases, Metadata Construction, Metadata Scaling

*Contact Author: haritsa@dsl.serc.iisc.ernet.in

DBTest'12, May 21, 2012 Scottsdale, AZ, U.S.A.

1. INTRODUCTION

The effective design and testing of database engines and applications is predicated on the ability to easily evaluate a variety of alternative scenarios that exercise different segments of the codebase or profile module behavior over a range of parameters [7, 11, 12]. A limiting factor is that the time and space overheads incurred in creating or maintaining these databases may render it infeasible to model the desired scenarios. In this paper, we present a graphical tool, called **CODD** [1], that supports both (a) the *ab initio* creation of metadata, and (b) the reclamation of the space occupied by an existing database *without* impacting its meta-data, inclusive of indexes. These functionalities are useful for testing modules such as query plan generators, system monitors, and schema advisors, whose inputs are comprised solely of meta-data.

To make the above concrete, consider the situation where a query optimizer developer wishes to evaluate a futuristic Big Data setup featuring *yottabyte* (10^{24}) sized relational tables. Obviously, just generating this data, let alone storing it, is practically infeasible even on high-end systems. However, with CODD, the associated metadata can be easily constructed within a few minutes, including defining the desired attribute-value distributions through visual histogram constructions. Further, CODD incorporates a graph-based model of the structures and dependencies of metadata values, implementing a topological-sort based checking algorithm to ensure that the metadata values are both *legal* (valid range, correct type) and *consistent* (compatible with the other meta-data values).

CODD's ability to model essentially arbitrary database scenarios also comes in handy for debugging legacy applications, or identifying hidden constraints in database engine code. As a case in point, by iteratively executing CODD on a popular commercial query optimizer, with the database size increasing in each iteration, we quickly discovered that the cardinality estimation module "saturated" when the input data size exceeded 10 exabytes – no mention of this threshold was found in the publicly available documentation of the system.

Even for environments in which meta-data information is required to be sourced only from the original data itself, CODD makes it feasible to subsequently *drop* the raw data in a manner that is totally *opaque* to the meta-data, including information related to physical schema constructs such as indexes. This facilitates testers to temporarily load real-world database scenarios without having to incur the storage and maintenance overheads of retaining the data during the ensuing testing process. The ability to retain the physical schema as-is, in spite of the data removal, is an important semantic difference as compared to the data truncation facilities natively provided by current database engines.

[1] In archaic English, *cod* means "empty shell", symbolising our dataless context.

Another special feature of CODD is its support for *automated scaling* of meta-data instances, obviating the need for loading fresh data or meta-data to test variants of datasets. For example, assume that the meta-data for the baseline 100 GB TPC-DS benchmark is available, and we now wish to boost it to the benchmark's maximum size, namely 100 TB. This objective is easily achieved in CODD through the incorporation of space-based scaling models that mimic the TPC-H and TPC-DS data generators [15].

As a novel addition to the above, CODD provides *time-based* scaling models – here, the objective is to scale the meta-data such that the overall estimated execution time of a test query workload is scaled by a user-specified factor. We begin by modeling the optimizer's plan costs for the query workload as algebraic functions of the scaling factors of the relations featuring in the queries. Then, an inverse minimization function is computed to determine the factor values that are expected to produce the desired time scaling. We expect that this feature could serve as a potent complement to the prevalent space-based scaling techniques since it ensures compliance of testing overheads with time budget constraints.

Finally, to facilitate comparative studies of different systems, CODD supports, to the extent possible, the *automated porting* of meta-data across database engines. Specifically, a tester can export most of the meta-data of a given database engine in a format that is compatible with the import interface of an alternative engine, and explicitly input only the engine-specific idiosyncratic information. Another useful application of this feature is that it can be employed to assess, in advance, the potential impact of a *data migration* exercise without having to load the data on the target engine.

In a nutshell, CODD is an easy-to-use graphical tool for the automated creation, verification, retention, scaling and porting of relational meta-data configurations. It is completely written in Java, running to over 10K lines of code, and is operational on a rich suite of industrial-strength database systems including DB2, Oracle, SQLServer, Sybase and PostgreSQL. For the commercial engines, it functions solely through the database APIs in a non-invasive manner, while for PostgreSQL, a few extra functions have been incorporated in the engine. Further, a conscious attempt has been made to design the interface such that the user can focus only on the logical meta-data semantics, and not have to contend with understanding the implementation specifics of individual engines. The tool is freely downloadable at [13], and its complete technical details are described in [10].

2. METADATA GENERATION

Meta-data information in modern database engines covers a variety of aspects, including schema organization, query processing, workload management and performance tuning. Our focus here is on the statistical metadata related to query processing – the extensions to the other aspects is straightforward. In particular, our meta-data is comprised of statistics on the following entities: (a) *relational tables* (row cardinality, row length, number of disk blocks, etc.); (b) *attribute columns* (column width, number of distinct values, value distribution histograms, etc.); (c) *attribute indexes* (number of leaf blocks, clustering factor, etc.); and (d) *system parameters* (sort memory size, CPU utilization, etc).

Given this framework, CODD supports two dataless modes called **ConstructMode** and **RetainMode**, described below.

2.1 Metadata Construction (ConstructMode)

Here, the objective is to allow users to directly create or edit the statistical meta-data without requiring presence of any prior data instance. All the commercial database engines do provide techniques to manually update the statistics – some of them support direct update statements on the catalog tables, while others permit creation of update procedures to achieve the same end. CODD leverages these existing techniques to create the initial metadata shell, but adds value in (a) packaging them in a largely vendor-neutral interface, and (b) camouflaging the details through dynamically created SQL procedures.

As a case in point, the following scheme is implemented for the Oracle engine: We use the statistics setting features provided by the dbms_stats package [16], which can be operated at table, column and index levels. The setting of table and index statistics is straightforward, only requiring a set of SQL commands to execute on the server after obtaining the inputs from the user interface. However, setting column and distribution statistics is more involved since this information, in particular the histograms, is stored in a special internal representation. CODD converts the user input to this internal representation using the prepare_column_stats utility, which is customized for each data type. After obtaining the internal representation, it is passed to the set_column_stats utility which stores these values in the catalogs. To combine this sequence of tasks, we define SQL procedures which carry them out in the required order. These procedures are created and executed on the fly and are not persistently stored in the database.

While existing facilities were leveraged for meta-data construction in the commercial engines, PostgreSQL posed idiosyncratic difficulties in this regard. Specifically, one of the metadata tables, PG_STATISTIC, includes two columns – MOST_COMMON_VALS and HISTOGRAM_BOUNDS – with type *anyarray*, a pseudo-type. PostgreSQL does not allow external updation of pseudo-type attributes. Therefore, we had to perforce create a clone of the in-built analyze command intended for statistics updates, such that the statistics collector now reads the inputs from an input file rather than directly computing from the database. Finally, for SQLServer, we were unable to support ConstructMode since it stores distribution statistics in a proprietary internal format.

A task in ConstructMode that may turn out to be laborious is to enter or update the detailed distribution statistics of relational attributes. To mitigate this overhead, the CODD interface allows the histogram values to be uploaded from a file, and this histogram can then be viewed graphically. Alternatively, users can create a new histogram by selecting from a pre-defined menu of classical distributions, and providing summary information about its distributional properties such as the mean, value range, skew, etc. Further, a *graphical* histogram editing interface, shown in Figure 1, is included in CODD, whereby the current histogram's layout can be visually altered to the desired geometry by simply reshaping the bucket boundaries with the mouse.

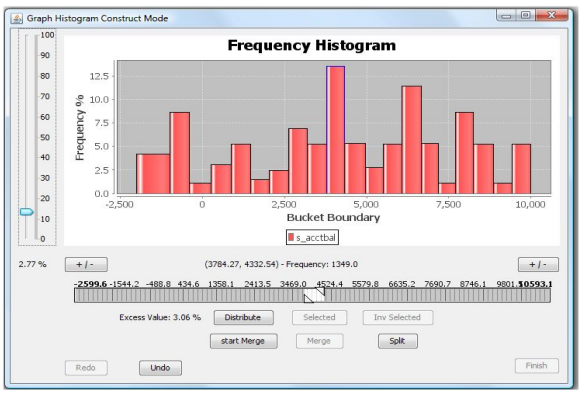


Figure 1: Graphical Histogram Interface

2.2 Metadata Retention (RetainMode)

In environments where the meta-data is required to be explicitly created from a database instance, CODD supports the subsequent dropping of some or all of the raw data, permitting reclamation of the storage space occupied by these contents. The challenge, of course, is to do so without this removal being reflected in the meta-data, since data updates automatically activate engine triggers that refresh the catalogs.

The triggers associated with data deletion/truncation are handled directly by the background processes and their maintenance varies across engines. Our first task was to bypass these triggers, and while this goal can be relatively easily met in some engines (e.g. SQLServer), the procedure is more convoluted with others (e.g. Sybase). The next, and more difficult task, is maintaining the *physical* schema (e.g. indexes) of the database. To accomplish this, a careful handling of the key constraints and the order of truncating relations is required. Physical schema retention is what separates CODD's functionality from the conventional meta-data scripting features available in most engines (e.g. optdiag [6] in Sybase).

The dataless modes available with CODD on different engines are summarized in Table 1. In the RetainMode column of this table, the annotation "fresh schema" indicates that the metadata update cannot be directly done in-place but indirectly through recreation of the schema, while "entire database" signifies that data dropping cannot be implemented at the level of individual relations, but for the entire schema as a whole. Finally, with regard to computational effort, setting up the dataless modes takes almost no time, completing in less than a second on all the engines.

Table 1: Dataless Modes on DB Engines

Engine	ConstructMode	RetainMode
DB2	Y	Y
Oracle	Y	Y
SQLServer	N (internal format)	Y (fresh schema)
Sybase	Y	Y (entire database)
PostgreSQL	Y (code addition)	Y

2.3 Inter-Engine Metadata Portability

An attractive feature of CODD is that it supports automatic porting, to the extent possible, of the statistical metadata across database engines, thereby facilitating comparative studies of systems as well as early assessments of the impact of data migration. While most database engines do provide the facility to transfer the metadata from one database to another on their *own platform*, none of them support (to our knowledge) porting metadata across different engines. To achieve this goal in CODD, a semantic mapping has been carefully worked out between the statistical information appearing in the various engines.

Although each engine has its own idiosyncratic metadata, we have found that most of the table level, column level and index level statistics are fully portable across all the engines. However, the transfer of distribution statistics is only partially feasible across some pairs of engines. To achieve the maximum possible fidelity, we first convert the source distribution statistics to a canonical form (which resembles the style of DB2), and then convert this information to a format compatible with the target engine.

The overall feasibility of the metadata transfer across different pairs of engines is summarized in Table 2. In this table, a Y entry signifies that more than 95% of the metadata can be translated, whereas a Partial entry means that about two-thirds of the data can be populated, while the N entry indicates that the transfer is infeasible. As can be seen, it is only with SQLServer that conversion is not possible due to its proprietary format for communicating statistics.

Table 2: Inter-Engine Metadata Transfer

Source / *Target*	DB2	Oracle	SQLServer	Sybase	PostgreSQL
DB2	-	Y	N	Partial	Y
Oracle	Partial	-	N	Partial	Partial
SQLServer	Y	Y	-	Y	Y
Sybase	Y	Y	N	-	Partial
PostgreSQL	Y	Y	N	Partial	-

3. METADATA VALIDATION

In RetainMode, the meta-data is guaranteed to be valid since it is sourced from a real database instance. However, in ConstructMode, since users are directly allowed to enter the meta-data, we need to ensure that the inputted information is both *legal* (valid type and range) and *consistent* (compatible with other metadata values). We now discuss how these issues are tackled in CODD. For ease of exposition, we will restrict our attention to the DB2 engine here – the handling of the other engines is similar in flavor.

Our validation approach is to first construct a *directed acyclic constraint graph* that concisely represents all the applicable constraints. Specifically, each node in the graph represents a single metadata entity that is annotated with associated legality constraints, and the currently assigned value which must adhere to these constraints. The directed edges, on the other hand, are used to represent statistical value dependencies between the metadata entities. Since the dependencies are typically bi-directional, to prevent duplication of edges, we adopt the convention that the edge will be directed from the node at the higher level of abstraction to the lower level node (e.g. from relation to attribute), while for nodes at the same level, the edge goes from the aggregate to the specific (e.g. from cardinality to distributions), and for the remainder, a lexicographic ordering is used. Our choice of convention attempts to reflect the natural manner in which schemas are usually developed by human users.

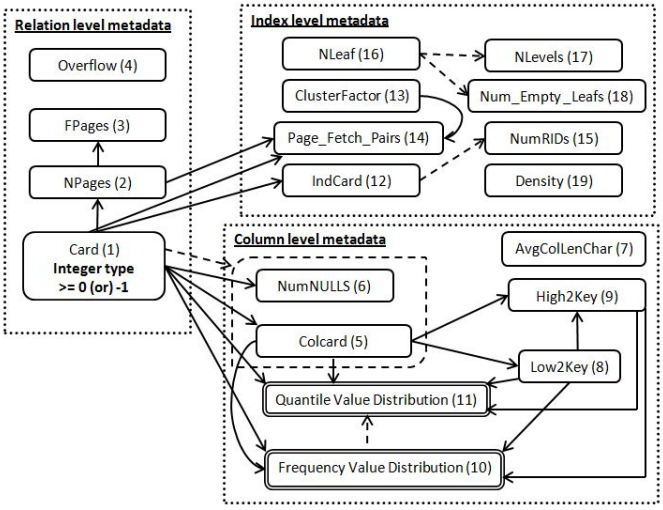


Figure 2: Metadata Constraint Graph (DB2)

As a concrete example, a partial fragment of the constraint graph for DB2 is shown in Figure 2, covering relation, attribute and index-related metadata. Consider the node CARD, representing the number of rows in the relation – here the structural constraint is that the value should be a whole number (or -1 to signify that statistics have not been collected). A similar structural constraint applies to COLCARD, representing the number of distinct values in a relational column, while the edge connecting CARD and COLCARD indicates the statistical constraint that COLCARD $\leq$CARD.

The constraint graph for DB2 was initially populated by creating nodes for the set of fields in the engine's system tables, and then inserting the edges based on the constraints listed in [14]. However, not all applicable constraints are listed and/or enforced in the engine. For example, the following:

- The sum of NUMNULLS and COLCARD must be less than or equal to CARD of the relation.
- The VALCOUNT of a Quantile Histogram bin b must be greater than the sum of all VALCOUNTs in the Frequency Histogram whose COLVALUE is less than b's COLVALUE.

In CODD, these missing constraints are enumerated and added to the constraint graph (shown as dashed edges in Figure 2).

Modeling data distributions usually results in a long chain of nodes that are constrained by a sequential domain ordering, and for compactness, we collapse such nodes into a single "super-node". For example, in Figure 2, the double-bordered nodes QUANTILE VALUE DISTRIBUTION and FREQUENCY VALUE DISTRIBUTION [3] represent super-nodes, and the expansion of these super-nodes is shown in Figure 3. In QUANTILE VALUE DISTRIBUTION, the columns represent the array of buckets used to store distribution statistics, and include information about the bucket boundaries, the (cumulative) bucket frequencies, and the (cumulative) number of distinct values in the buckets. Also, each column is constrained by the values of external nodes as shown by the incoming edges at the bottom of Figure 3. For example, the value of VALCOUNT in the last bucket must be equal to the total CARD of the relation. The dotted edge between the two distributions signifies that the bucket frequencies in the Quantile Distribution are constrained by those appearing in the Frequency Distribution.

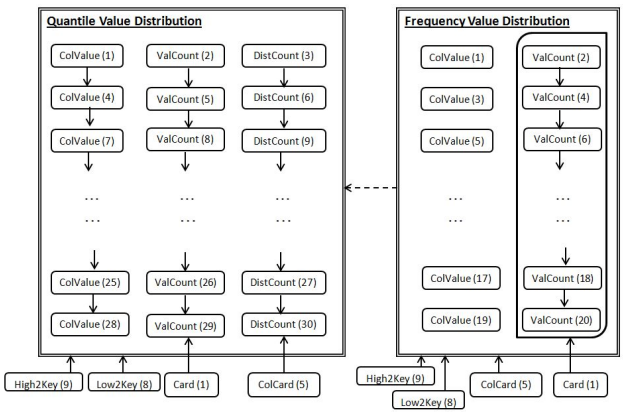


Figure 3: Constraint Graph for Distributions

Finally, after the constraint graph, denoted $CG(V, E)$ has been fully constructed and populated, we run a *topological sort* on CG. The sort provides a linear ordering CG_{linear} of the nodes, and can be accomplished in time complexity $O(|V| + |E|)$ [2]. Then, CODD guides the user through this linear ordering, requesting inputs at each new node, and ensuring that all applicable constraints are met by the freshly added entries. A sample linear ordering is shown through the numbers associated with the nodes in Figure 2, beginning with CARD (1) and ending with DENSITY (19).

4. METADATA SCALING

A common activity in database engine testing exercises is to assess the behavior of the system on scaled versions of the original database. Current benchmarks typically implement a *space-scaling* approach – for example, in TPC-H, the generator can provide databases ranging from 1 GB to 100 TB. It does so by linearly scaling the cardinalities of the large relations, accompanied by domain-cardinality scaling for the primary keys and foreign keys featuring in the scaled tables. In TPC-DS too, the fact table is scaled linearly, but the dimensions undergo sub-linear expansions according to a hard-wired scaling assignment.

CODD natively supports space-scaling models along the lines of TPC-H and TPC-DS. In addition, it also provides a novel *time-based* scaling model. Here, the aim is to scale the baseline metadata $\mathcal{M}$ such that the optimizer's estimated cost (representing response time) of executing a given query workload $\mathcal{Q}$ on the scaled version, $\mathcal{M}^{\alpha}$, is a specified multiple, α, of the cost of executing it on the original database. Initially, we attempt to produce a metadata instance such that the cost of each *individual* query in $\mathcal{Q}$ is scaled by α. However, this may often be fundamentally infeasible, in which case we settle for solving the following optimization problem:

Produce an $\mathcal{M}^{\alpha}$ such that the sum over $\mathcal{Q}$ of the individual squared deviations from α in cost scaling is minimized, subject to the constraint that the overall cost over $\mathcal{Q}$ is scaled by α.

That is, given relations $R_1, R_2, \ldots, R_h$ appearing in $\mathcal{Q}$, identify a relation cardinality-scaling vector $[\alpha_1, \alpha_2, \ldots, \alpha_h]$ such that

$$\sum_{q_i \in \mathcal{Q}} [c^S_{q_i}/c^O_{q_i} - \alpha]^2$$

is minimized subject to

$$\sum_{q_i \in \mathcal{Q}} c^S_{q_i} = \alpha * \sum_{q_i \in \mathcal{Q}} c^O_{q_i}$$

where $c^O_{q_i}$ and $c^S_{q_i}$ represent the costs of q_i in the original and scaled databases, respectively.

Achieving the above objective function appears to be a hard problem since we are faced with the complex proposition of mathematically relating the overall costs of query plans to scaling factors on their base relations. However, if we assumed that (a) the scaling is implemented such that the relative frequency distributions of all non-key columns is *identical* between the original and scaled databases; (b) the choice of query plans is *retained* between the original and scaled databases; and (c) simple polynomial expressions are sufficient to model the relationships between operator costs and their input sizes, then generating a satisfying scaling vector is readily feasible. This is because of Lemma 1 in Figure 4, which allows for constructing each plan operator's output cardinality in the scaled database solely in terms of the corresponding cardinality in the original database and the relation scaling vector – the proof is available in [10],

Figure 5 delineates the summary procedure to compute the total cost of a query on the scaled database, expressed in terms of the scaling factors of the relations appearing in the query. For example, with TPC-H query Q14, which features the PARTS and LINEITEM

Lemma 1. Let $R_1, R_2, \ldots, R_h$ be the input relations to operator *op* and $\alpha_1, \ldots, \alpha_h$ be their scaling factors respectively. Then, if the relative frequency distributions of the scaled database (SD) and the original database are identical for non-key columns and if the domain is scaled for the key columns of the SD, then the output cardinality of each operator *op* in the plan tree for the SD is expressible as

$f(\alpha_m, \ldots, \alpha_n) \times Original\ output\ cardinality$

where $\alpha_m, \ldots, \alpha_n$ are the subset of scaling factors such that $\forall \alpha_i \in (\alpha_m, \ldots, \alpha_n)$, the relation R_i is not referenced by any other relation $R_j \in \{R_1, R_2, \ldots, R_h\} \setminus R_i$; and f is a function on this subset of scaling factors. Further, the relative frequency distribution of the scaled output is identical to that of the original output.

Figure 4: Scaled output size and distribution

Input: Query q_i **Result:** Cost function $c^S_{q_i}$

1. Obtain the query execution plan for the given query.

2. Determine the cost function for each operator in the execution plan with respect to the sizes of the inputs.

3. Using Lemma 1, determine the scaled output size in terms of scaling factors for each operator in the execution plan.

4. Calculate the cost of each operator for scaled inputs using the cost functions obtained in Step 2.

5. Compute the total cost of the query as the aggregate of the costs of the operators present in the execution plan.

Figure 5: Query Costs in Scaled Database

relations, the following expression was obtained on DB2:

$$c^S_{Q_{14}} = 435\ \alpha_P + 17865\ \alpha_L$$

where α_P and α_L are the scaling factors for the PARTS and LINEITEM relations, respectively.

Armed with these individual query cost functions, we now compute the scaling vector that would provide the best scaling configuration, using the optimization procedure enumerated in Figure 6, which typically converges in less than a minute. When multiple solutions are available for the scaling vectors, the final choice made is to pick the solution that is closest to a traditional space-based scaling approach (Step 4 in Figure 6), since it is our expectation that this would result in more robustness with regard to (a) addition of new queries to the workload, and (b) retention of the same plans across the scaled databases.

As an example result, given a TPC-H workload consisting of queries Q1, Q14 and Q17, with appearance frequencies of 0.5, 0.48 and 0.02, respectively, and desired cost-scaling factor α = 2, our algorithm recommended doubling the size of the LINEITEM relation while keeping the PART relation constant. This resulted in an overall cost increase of 1.98, with the individual query cost-scaling ratios being *Q1: 1.99*, *Q14: 1.96* and *Q17: 1.99*.

5. CODD IN ACTION

In this section, a sample scenario that highlights CODD's utility is presented. For this purpose, we will use the notion of "plan diagrams" from [9]. Specifically, given a parametrized SQL query template that defines a relational selectivity space, and a choice of database engine, a plan diagram is a visual representation of the plan choices made by the optimizer over this parameter space. In a nutshell, plan diagrams visually capture the optimality regions of the parametric optimal set of plans (POSP) [5].

To make this notion concrete, consider QT9, the parametrized SQL query template shown in Figure 7, which is based on Query 9

Lemma 2. If the key columns of relations are domain scaled and the primary key columns $(C_a, \ldots, C_n)$ of relation R_i are a combination of foreign key columns, which are referencing the relations $(R_a, \ldots, R_n)$, respectively, then the scaling factor α_k of relation R_k is bounded by the product of $\alpha_a, \ldots, \alpha_n$, where $\alpha_a, \ldots, \alpha_n$ are the scaling factors of relations $(R_a, \ldots, R_n)$, respectively.

Algorithm

Input: Metadata Instance $\mathcal{M}$, Query workload $\mathcal{Q}$, Scaling factor α

Result: Scaled Metadata Instance $\mathcal{M}^\alpha$

1. Determine cost of executing each query in the original database, $c^O_{q_i}$, from the optimizer's execution plan.

2. Determine the cost of each query in the scaled database,$c^S_{q_i}$, as an algebraic function of the scaling factors $[\alpha_1, \ldots, \alpha_h]$, using the procedure of Figure 5.

3. Solve the optimization problem

$$Minimize \sum_{q_i \in \mathcal{Q}} [c^S_{q_i}/c^O_{q_i} - \alpha]^2$$

subject to

$\sum_{q_i \in \mathcal{Q}} c^S_{q_i} = \alpha * \sum_{q_i \in \mathcal{Q}} c^O_{q_i}$ and

for k between 1 and h

$0 < \alpha_k <= Lemma\ 2\ Bound$, if applicable

$0 < \alpha_k < \infty$, otherwise

4. If a set of solutions T is obtained in Step 3, pick a solution $t \in T$ that minimizes $\sum_{\alpha_k \in t} (\alpha - \alpha_k)^2$.

5. Scale the input relations with the scaling factors obtained in Step 4 to get the required cost scaled metadata $\mathcal{M}^\alpha$.

Figure 6: Cost-scaling of Metadata

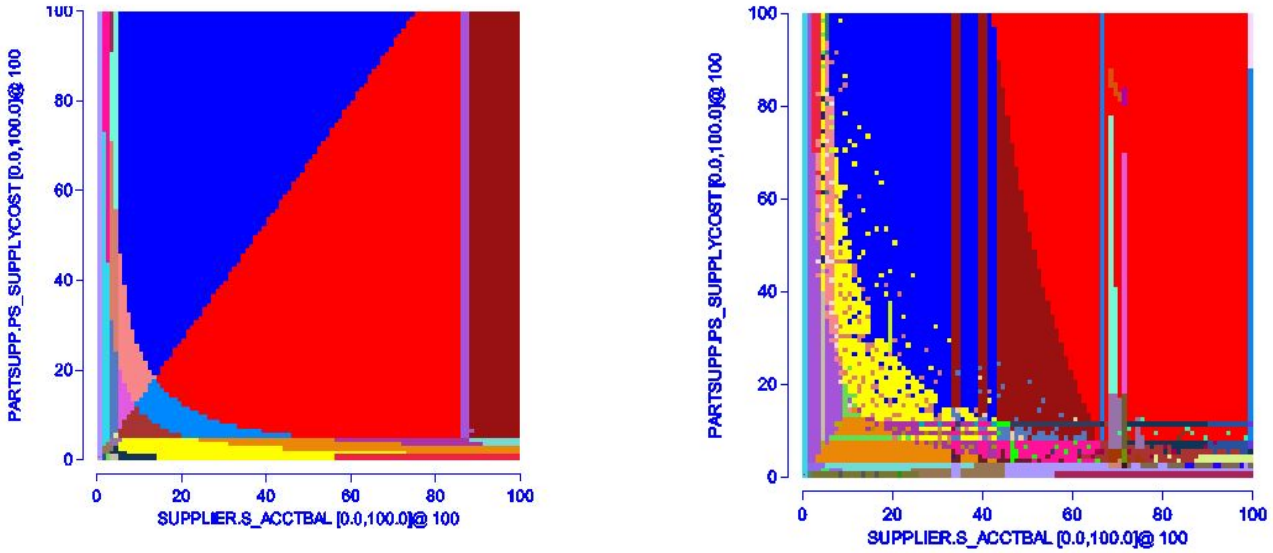
```
select n_name, o_year, sum(amount)
from (select n_name, o_orderdate, l_extendedprice
   from part, supplier, lineitem, partsupp, orders, nation
   where s_suppkey = l_suppkey and ps_suppkey = l_suppkey and
   ps_partkey = l_partkey and p_partkey = l_partkey and o_orderkey
   = l_orderkey and s_nationkey = n_nationkey and p_name like
   %green% and s_acctbal :varies and ps_supplycost :varies
   ) as all_nations
group by n_name, o_year
order by n_name, o_year desc
```

Figure 7: Example Query Template QT9

of the TPC-H benchmark. Here, selectivity variations on the SUPPLIER and PARTSUPP relations are specified through the s_acctbal :varies and ps_supplycost :varies predicates, respectively.

The associated plan diagram produced by Oracle on the baseline TPC-H database of size 1 GB, using a lightweight laptop with a 64GB solid-state hard disk, is shown in Figure 8(a). In this picture, each colored region represents a specific plan, and a set of 32 different optimal plans, P1 through P32, cover the selectivity space.

Now consider the situation where the Oracle optimizer developer wishes to assess its behavior on the highest 100 TB scale of the TPC-H benchmark, *five orders of magnitude* larger than the baseline. Clearly, generation and loading this scenario on the 64GB laptop is completely out of the question. However, using ConstructMode, we can easily create a meta-data shell that represents the 100 TB environment. Further, we can also update provisioning parameters (e.g. max_parallel_servers) and hardware characteristics (e.g. ioseektim) to simulate the enhanced system configurations that may be deployed to cater to the expanded database.



(a) Baseline (1 GB)

(b) Scaled (100 TB)

Figure 8: QT9 Plan Diagrams (Baseline and Scaled)

The plan diagram produced by the Oracle optimizer on the *scaled* database is shown in Figure 8(b). Comparing the two pictures, we observe first that the number of plans has now increased from the 32 of the baseline to 77 in the scaled version! Secondly, the geometries of the optimal plan regions have undergone a significant change. This experiment highlights how we can easily assess, using CODD, the optimizer's altered behavior in response to futuristic scenarios.

6. SUMMARY

In this paper, we presented the CODD tool, which permits users to construct dataless environments and simulate various alternative scenarios that may prove helpful in testing and debugging exercises. While allowing the user to play with arbitrary metadata values, CODD ensures that these values are legal and consistent with the engine requirements. In addition, CODD provides two key features – Cost based Scaling and Inter-Engine Portability – that could be of considerable benefit to testers. Finally, the tool is implemented with a convenient graphical interface that helps users to employ the features while remaining agnostic to the specifics of the underlying database engine. While CODD currently only supports testing of meta-data based modules, we are actively investigating its integration with data generation frameworks (e.g. [8, 1]) to facilitate *execution module* testing as well.

The ability to evaluate "what-if" situations continues to be a potent tool for system designers and has attracted considerable literature – a recent example is the Starfish system [4] which supports evaluating alternative scenarios on the Hadoop framework. While sharing similar objectives, there are key differences between CODD and such systems – they typically focus on the performance and provisioning aspects, use their own optimization models, and do not have to negotiate with internal system constraints. In contrast, ours is a black-box data and system modeling exercise on commercial engines, we are restricted to using the API facilities to achieve the goals, and we have to ensure that stringent systemic and configuration constraints are continuously maintained.

7. REFERENCES

[1] C. Binnig, D. Kossmann, E. Lo and M. Tamer Ozsu, "QAGen: Generating Query-Aware Test Databases", *SIGMOD 2007*.

[2] T. Cormen, C. Leiserson, R. Rivest and C. Stein "Introduction to Algorithms", *MIT Press*, 3rd ed, 2009.

[3] D. Fechner, "Distribution statistics uses with the DB2 optimizer", `www.ibm.com/developerworks/data/library/techarticle/dm-0606fechner/index.html`, *June 2006*.

[4] H. Herodotou et al, "Starfish: A Self-tuning System for Big Data Analytics", *CIDR 2011*.

[5] A. Hulgeri and S. Sudarshan, "Parametric Query Optimization for Linear and Piecewise Linear Cost Functions", *VLDB 2002*.

[6] E. Miner, "Using Optdiag Simulate Statistics Mode", `m.sybase.com/detail?id=20472`.

[7] M. Muralikrishna, "Using the Optimizer to Generate an Effective Regression Suite: A First Step ", *DBTest 2010*.

[8] T. Rabl and M. Poess, "Parallel Data Generation for Performance Analysis of Large, Complex RDBMS", *DBTest 2011*.

[9] N. Reddy and J. Haritsa, "Analyzing Plan Diagrams of Database Query Optimizers", *VLDB 2005*.

[10] R. Trivedi, I. Nilavalagan and J. Haritsa, "Engineering Dataless Databases", `dsl.serc.iisc.ernet.in/publications/report/TR/TR-2012-02.pdf`

[11] F. Waas, L. Giakoumakis and S. Zhang, "Plan Space Analysis: An Early Warning System to Detect Plan Regressions in Costbased Optimizers", *DBTest 2011*.

[12] "Testing and Tuning of Database Systems", *IEEE Data Engineering Bulletin*, 31(1), March 2008.

[13] `dsl.serc.iisc.ernet.in/projects/CODD`

[14] `publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.admin.perf.doc/doc/c0005121.html`

[15] `www.tpc.org`

[16] `docs.oracle.com/cd/B28359_01/appdev.111/b28419/d_stats.htm`