System and Method for Automatically Generating Platform-Specific Database Code from Schema composed of Typed Tables and Formulas

An IP.com Prior Art Database Technical Disclosure

Authors et. al.: Ryan Stephen Ehrenreich
Decia LLC

IP.com Number: IPCOM000255833D
IP.com Electronic Publication Date: October 16, 2018

Title

System and Method for Automatically Generating Platform-Specific Database Code from Schema composed of Typed Tables and Formulas

Inventors

Ryan Stephen Ehrenreich

Background

Field of the Invention

The invention pertains to the field of database schema design and automation of database code.

Description of the Prior Art

Some prior art focuses on using “entity-relational” schema to create database tables. Other prior art focuses on persisting entities to database tables. Other prior art focuses on inferring structure for unstructured data.

Definitions of the Field of Databases

- Schema: A drawing of tables arranged in a manner that is meaningful to the user and where a connection between tables implies relatedness.
- ID: A value that uniquely identifies something. Usually an ID is defined as 128-bit sequence of bits that is automatically generated by a computer and guaranteed to be unique. In this usage, the ID value has no obvious meaning to a human, but is used to guarantee uniqueness.
- Collection: A list of elements.
- Element: A specific item in a collection.
- Table: A collection of rows whose structure is determined by a common set of columns. Usually, one of the columns is an ID column.
- Column: A specification for a single, specific value stored in each row of a table.
- Row: A collection of values where each value corresponds to a single specific column and all columns in the table are represented exactly once.
- Default Value: The value for a specific column that is used when a new row is created.
- Entity: Commonly used to mean any table with an ID.
- Entity-Relational: A methodology for defining related tables such that relationships between tables are clearly specified in a normalized manner.
- Join: The process of combining rows keyed on different IDs so that they can be analyzed as one complete unit of information. This process often uses a shared value that is unique in one of the rows but not the other, although how it happens is flexible.
- Aggregation: The act of reducing a set of values to a single value.
- Null: The absence of a specific value, often reflecting that data is in an uninitialized state.

- Nullable: Capable of remaining in an uninitialized state, such that it holds no meaningful value.
- JOIN: A SQL command that combines sets of rows from different tables bases on a set of criteria, which usually is a shared ID(s).
- FROM: A SQL command that specifies the original table from which JOINs start.
- SELECT: A SQL command that chooses which information to return from 1 or more tables that have been JOINed.
- INSERT: A SQL command that inserts a row in a specific table.
- UPDATE: A SQL command that updates 1 or more values in a set of rows within a specific table, based on some criteria.
- WHERE: A SQL command used for specifying criteria.
- GROUP BY: A SQL command used in aggregation to aggregate rows that share a specific set of values.
- ORDER BY: A SQL command used to order results of a query based on some criteria.
- MapReduce: A specific processing model where problems are divided into an independent part which can be run in parallel (aka Mapping) and a dependent part that requires aggregation of Mapped results (aka Reducing). Often, there is the opportunity to Order the Mapping results prior to Reducing.

Definitions of the Field of Data Structure & Graph Theory

- Graph (aka Network, aka Map): A data structure that consists of nodes and edges, where the configuration of edges is not constrained.
- Node (aka Vertex): A specific location in a graph.
- Edge: A relationship between 2 nodes in a graph that is used for traversing the graph.
- Directed Edge: An edge that only allows traversal to proceed in one direction (denoted by an arrow that points in the allowed direction).
- Disconnected Subset: A subset of nodes that are not connected to the rest of the graph.
- Traversal: The process of visiting all the nodes of a graph, often using a criterion for ordering visitations.
- Cycle: A configuration of nodes and edges where a certain node is reachable from another node and vice versa.
- Cyclic Graph: A graph that contains at least 1 cycle.
- Acyclic Graph: A graph that contains no cycles.
- Directed Graph: A graph that only contains directed edges (i.e. edges that point in only one direction).
- Directed Acyclic Graph (aka DAG): A graph that Is both directed and acyclic.
- Connected Graph: A graph where there are no disconnected subsets of nodes.
- Weakly Connected Graph: A graph where there are no disconnected subsets of nodes AND no cycles between nodes.
- Tree: A directed acyclic graph where each node has at most 1 incoming edge and there is 1 unique root node that all other nodes originate from.
- Dictionary: A data structure that stores values by unique key, using indexed key-value pairs.

Brief Summary of the Invention

The objective of this invention is to unify the fields of spreadsheet, SQL database, and NoSQL database design by offering a common schema that automates the production of native storage structures and code for all three of these data management platforms.

The invention seeks to encode both the data model and the logic of the database, so that from a schema encoded in the system, the user could export a fully functional database to any data management platform, regardless of data representation technology, query language, or data processing approach.

The existing embodiment of the invention accomplishes these objectives by providing a web interface through which users can:

- Visually design a database schema composed of a hierarchy of Typed Tables
- Specify Time Dimensionality of the Variables
- Specify Formulas that encode computational logic
- Create test data to verify the correctness of the schema
- Compute results for the test data
- Export the schema as a fully functional database to a variety of modern data management platforms

Additionally, this embodiment offers users management features, such as versioning of schema and sharing schema with other users based on a dynamic set of permissions.

Unlike prior art, the invention allows users to encode completely functional computational logic without ever specifying any JOIN statements or writing any code that joins values.

Definitions of the Invention

- Entity Table: As opposed to the definition of the field, Entity actually means any concept or thing with a “human understandable” ID that has continuity across time. Just because a table has an ID, does not necessarily make it an Entity. For instance, a Person is an Entity because each Person is uniquely identifiable. For example, each Person has a Social Security Number, which does not change over that Person’s lifetime.
- Relation Table: A table that relates multiple Entity Tables such that the user can store values based on a list of the related Entity IDs. The Relation ID is really just a convenient shorthand representation of the list of Entity IDs that it relates. For instance, a {Person, Car} Relation Table might have an ID, but it is just a convenient single value that actually corresponds to a {Social Security Number, Vehicle Identification Number} pair. The single value for the Relation ID would not be meaningful to a normal human. A Relation Table represents a matrix where the dimensions of the matrix correspond to the IDs of the specific Entity Tables that have been related. Another term for a “Relation Table” would be an “Association Table”.
- Global Table: The Global Table stores information that is universal to the schema. This table needs no ID because it is constrained to always contain exactly one distinct row; although, it may have an ID for compliance with standard database technologies.

- Typed Table: In other systems, “typed table” usually means a table that corresponds to a specific “class” that has been implemented in a modern object-oriented programming language. In the context of this invention, “typed table” specifically means a table that is either a Global, Entity, or Relation table.
- Structural Type: A specific typed table. For example, the Employee entity table could also be called the Employee structural type.
- Instance: The invention uses the terms “Row” and “Instance” interchangeably to denote unique sets of values stored in typed tables. This is so because the set of values need not be constructed as a simple list, as some values may take the form of matrices.
- Dimension: A specific criterion on which data varies. For a simple two-dimensional matrix, the Dimensions would be X and Y.
- Structural Dimension: A Dimension on which data varies due to its correspondence to a specific Entity ID. For example, the Name of a Person varies based on which Social Security Number you are referencing, thus the “Name” field varies on the “Person” Structural Dimension.
- Entity Dimension: Used as interchangeable term for “Structural Dimension”.
- Time Period: A start date and end date that determine a unique period of time on which data can vary.
- Time Dimension: A Dimension on which data varies due to its correspondence to a specific Time Period. For instance, a company’s profit may vary from year to year, where each year is a Time Period whose duration is 365 days.
- Time Period Type: The quantity of time that a Time Dimension varies on. For example, the “Year” time period type indicates that the Time Dimension offers a unique value every 365 days.
- Variable: Instead of using the term “Column”, the invention uses the term “Variable” for fields stored in each row of a typed table. This is so because each Variable can additionally contain multiple Time Dimensions, so that it maps to more than one unique value per instance. For example, the “Company” table’s “Profit” Variable varies by year.
- Variable Type: Whether the Variable derives its value from a user input or a formula. Also, if a formula, whether that formula performs aggregation.
- Operator: A sign that implies a specific transformation. For instance, the “+” sign implies that 2 arguments will be combined via addition into a single resulting value.
- Function: A name followed by parentheses that imply a specific transformation. Sometime Operator and Function are used to mean the same thing, since the only real difference is in the notation that the Operator is written (e.g. “MULTIPLY(X, Y)” and “X*Y” mean the same thing).
- Formula: An object which contains an expression tree composed of arguments and expressions.
- Argument: An indexed parameter passed to an expression that specifies either a constant value, a reference to a Variable, or a nested Expression.
- Expression: A means of specifying that a certain transformation should be performed on a specific set of Arguments. Expressions can use Operators or Functions to signify which transformation is desired.
- Join Path: An ordered set of structural types used to automatically join information.

- Alternate Dimension Number (ADN): The occurrence number of a specific entity dimension. For example, if there is a Relation Table that relates the Product Entity Table to itself, the first related Product ID will have an ADN equal to “1” and the second will have an ADN equal to “2”.
- Unbound Dimension: A dimension that is not limited to values connected via its Join Path. When specifying a query in a formula, the system may treat the relevant Structural or Time Dimensions as unbound so that all possible combinations of relevant values are considered.
- Entity Existence Trees: Specifies the hierarchy of Entities that are connected via non-null edges, such that parent instances are required for child instances to exist. For example, if the tree contains “Manufacturer” as the parent and “Product” as the child, then for a Product row to exist, there must be a Manufacturer row that it corresponds to.
- Entity Relatedness Network: Specifies the hierarchy of Entities connected via both non-null and nullable edges. It contains all the information of the Entity Existence Trees, plus additional “Relatedness” edges that are nullable and do not determine child node existence.
- Structural Map: The data structure that tracks Join Paths and is used to automate the construction of join code that spans multiple tables.
- Dependency Map: The data structure that tracks dependencies between Variables and is used to automate the ordered processing of Variables, such that their logic is properly executed.
- Variable Group: A group of Variables that must be processed step-by-step, as a single unit.
- Parallelizable Variable Groups: A set of Variable Groups that can be computed in parallel because they do not depend on each other completing to start computing. When no cycles exist in the Dependency Map, each Variable will belong to its own Variable Group, so the Parallelizable Variable Groups will correspond to the set of Variables that can be processed independently at each stage of the computation.
- Formula Processing Engine: The class that compiles and computes the formulas specified in the schema.
- Database Exporter: A class that automatically exports the schema and formulas to a specific database technology as native code.

Brief Description of the Drawings

Figure 1. Figure 1 illustrates a valid schema composed of the Global Table and 6 Entity Tables, where there are two Entity Table sub-trees, each composed of a parent with two children.

Figure 2. Figure 2 shows the schema from Figure 1, but with the addition of an edge going from “Entity 2” to “Entity 1”. This is also a valid schema because it meets the constraints that 1) each pair of Entity Tables can only have exactly one directed path between them and 2) there can be no directed cycles in the schema.

Figure 3. Figure 3 shows the schema from Figure 2, but with the addition of an edge going from “Entity 2-A” to “Entity 1-B”. This is NOT a valid schema because there are two directed paths between “Entity 2” and “Entity 1-B”. These two paths are {“Entity 2”, “Entity 1”, “Entity 1-B”} and {“Entity 2”, “Entity 2-A”, “Entity 1-B”}.

Figure 4. Figure 4 shows the schema from Figure 2, but with the addition of an edge going from “Entity 1” to “Entity 2-A”. This is NOT a valid schema because there are two directed paths between “Entity 2” and “Entity 2-A”. These two paths are {“Entity 2”, “Entity 1”, “Entity 2-A”} and {“Entity 2”, “Entity 2-A”}.

Figure 5. Figure 5 shows the schema from Figure 2, but with the addition of an edge going from “Entity 1-B” to “Entity 2”. This is NOT a valid schema because there is a directed cycle in the schema spanning tables {“Entity 2”, “Entity 1” ”, “Entity 1-B”}.

Figure 6. Figure 6 shows the schema from Figure 1, but with the addition of a Relation Table that bridges “Entity 1” and “Entity 2”. This is a valid schema because it meets the same conditions as Figure 1 due to the fact that no edges depart from Relation Tables.

Figure 7. Figure 7 shows the schema from Figure 2, but with the addition of a Relation Table that bridges “Entity 1” and “Entity 2”. This is a valid schema because it meets the same conditions as Figure 2 due to the fact that no edges depart from Relation Tables. However, because there is an edge from “Entity 2” to “Entity 1”, “Entity 1” is subordinate to “Entity 2” in the Join Path. Therefore, the ID of “Entity 2” in “Relation 1” will have an “Alternate Dimension Number” value of 2 (please note that the specific ID that uses “ADN=2” will be determined by which of the Entity Tables is the latter, based on the sorted order of tables).

Figure 8. Figure 8 explains the meaning of “Time Dimensionality”. The “Tax Rate” variable has a Time Dimensionality of “0” because it does not change over time. The “Profit” variable has a Time Dimensionality of “1” because it uses a single Time Dimension that spans the years 2016 to 2020. The “Interest on Debt” variable has a Time Dimensionality of “2” because its value changes over two Time Dimensions. The horizontal Time Dimension reflects the interest actually charged from 2016 to 2020. The vertical Time Dimension reflects the year that the debt was originally incurred. For instance, for debt incurred in 2016, the interest charged in 2017 will be $10,000.

Figure 9. Figure 9 lists the properties that the user can assess for the schema as a whole (i.e. “the Model”).

Figure 10. Figure 10 lists the properties that the user can assess for the Global Table.

Figure 11. Figure 11 lists the properties that the user can assess for an Entity Table.

Figure 12. Figure 12 lists the properties that the user can assess for a Relation Table.

Figure 13. Figure 13 lists the properties that the user can assess for a Variable.

Figure 14. Figure 14 shows the structure of a valid Formula. In particular, it shows the optional Aggregator followed by the mandatory Expression Tree. The Aggregator should exist or not based on the configuration of the Variable’s “Variable Type” property. When set to “Basic Formula”, specifying an Aggregator is not allowed and causes an error. When set to “Structural Aggregation Formula”, any of the Aggregators specified under “Structural Aggregators” or “Structural & Time Aggregators” are allowed, and exactly one MUST be provided. When set to “Time Aggregation Formula”, any of the Aggregators

specified under “Time Aggregators” or “Structural & Time Aggregators” are allowed, and exactly one MUST be provided. It is important to note that the “Structural & Time Aggregators” aggregate structurally, but produce a result that uses 1 or more Time Dimensions, where the Time Period that a value falls into is determined by the value of date Variable(s) that are specified as arguments to the aggregator. These aggregators contain “where” and “order-by” arguments, in case these aspects need to be taken into account when determining results.

Figure 15. Figure 15 shows the structure of a valid Formula, but it focuses on the Expression Tree, which is always available in any type of Formula. Figure 15 lists many of the operators available for use in the Expression Tree; however, it is important to note that any logical or mathematical operator should be capable of being added with relative ease. It is important to note the operators that specify “Dimensional Movement”. For instance, the “Offset by Number of Instances” Operator would change the relevant row for the Variable or Expression provided to it by the specified number of rows, where the rows will be sorted by some criteria (which defaults to the standard sorting order if a custom sort criteria is not provided). Similarly, the “Offset by Number of Periods” Operator would change the relevant Time Period for the Variable or Expression provided to it by the specified number of Time Periods. For instance, offsetting by “-1” Time Periods would cause the system to use the value for the preceding Time Period.

Figure 16. Figure 16 shows the “Entity ID Sets” tracked by the Structural Map for the schema originally depicted in Figure 6. The reason for showing this visually is that it makes it easier to see how the Structural Map determines Join Paths between Typed Tables. Since the Global Table has no IDs in its “Entity ID Set”, it is accessible everywhere in the schema. Because the “Relation 1” table has the “Entity 1” ID in its set, it can be joined to the “Entity 1” table, and vice versa. Similarly, because the “Entity 1-B” table has the “Entity 1” ID in its set, it can be joined to the “Entity 1” table, and vice versa. Because of this, the “Entity 1-B” table can automatically be joined to the “Relation 1” table. However, this join will yield multiple rows because it traverses a downward edge, so aggregation will be necessary to produce a unique value.

Figure 17. Figure 17 shows a valid configuration of Variable dependencies. In this case, the “Payment for Goods” Variable has two dependencies, namely “Price” and “Quantity”, where the two dependencies are both inputs. When loaded into the Dependency Map, this set of Variables will represent a valid configuration. Since the “Payment for Goods” Variable is not involved in a cycle, it will be organized as the sole member of its own Variable Group. During computation, this Variable Group can be run in parallel with any other Variable Group that does not depend on it.

Figure 18. Figure 18 also shows a valid configuration of Variable dependencies. However, at first glance, it may seem invalid because there is a cycle that spans the Variables “Principal, Beginning Balance”, “Interest Charge”, “Change in Principal”, and “Principal, Ending Balance”. However, because the edge from “Principal, Ending Balance” to “Principal, Beginning Balance” shows with “T-1”, this means the edge uses the previous ending balance as the start for the next Time Period, thus creating a valid configuration. In this configuration the Dependency Map will generate a Variable Group for the

Variables involved in the cycle so that they are processed together, using ordered iteration though the Time Periods relevant to the Time Dimension that these Variables use.

Figure 19. Figure 19 also shows a valid configuration of Variable dependencies. As opposed to Figure 18, which uses "T-1" to denote a past value, Figure 19 uses "T+1" to denote a future value. At first, this may seem invalid; however, because the Group of Variables strictly uses only current and future values (i.e. does not use past values), this configuration is valid. The Dependency Map will generate a Variable Group for the Variables involved, and they will be processed in reverse order of the Time Periods (from future to past) using iteration.

Figure 20. Figure 20 shows a configuration of Variable dependencies that is currently invalid, but may be supported in the future using convergent iteration. The cycle in Figure 20 cannot be resolved by Structural-based or Time-based ordering because it involves only current values.

Figure 21. Figure 21 shows the process by which the Dependency Map resolves "Dimensional Movement" and Cycles across Variable dependencies. For example, in Figure 18, the Dimensional Movement specified by "T-1" is that the system should use the previous Time Period value for that Variable. If the Group of Variables contains no Dimensional Movement and no Cycles, it is computable without ordering or iteration. Any Dimensional Movement requires ordering values along that Dimension. For instance, movement across a Time Dimension requires that Variable values are evaluated in the natural order of Time Periods (e.g. from past to future). Using Dimensional Movement to resolve Cycles requires that the Variable values are processed as a Group using iteration over the dimension of movement.

Figure 22. Figure 22 shows the process by which values are computed. Please see the section that describes the Formula Processing Engine below for a complete description of each step.

Figure 23. Figure 23 shows the process by which schemas are exported from the system. Please see the section that describes the export to database platforms below for a complete description of each step.

Figure 24. Figure 24 shows a valid schema of a Sales Order Processing database.

Detailed Description of the Invention and Best Mode of Implementation

(Prior to reading this section, please read "Definitions of the Invention" above to ensure proper understanding of the terminology)

Normally, Relational Databases users draw their schemas as graphs that contain groups of tables with relationship arrows pointing between them. This schema is used to generate SQL code that creates tables and enforces constraints between tables, but does not actually implement any computational logic.

Conversely, Spreadsheet and NoSQL Database users do not even use schema to help them build their logic. They build all the logic by hand, or by using code generators that help handle localized tasks.

The invention solves this problem by generating all computational logic that occurs within the database, regardless of which platform is used as the end target on which to host the database.

The invention accomplishes this by using 3 unique aspects:

1) Typed tables where each of 3 types (Global, Entity, or Relation) imparts unique meaning on the actual table being specified
2) Table schema enforced as "weakly connected directed acyclic graph" (WC-DAG) where data flows clearly in a specific direction
3) Spreadsheet-style Formulas that encode computational logic to be performed

Once the user specifies this schema, the system loads it into proprietary data structures (i.e. the Structural Map and Dependency Map) so that the system can determine the ordered hierarchy of the tables in the schema and the dependencies between individual variables.

Then, when the user chooses to export the schema to a particular target technology, the system uses these data structures to generate the tables (or data stores) required for the database, the ordered steps necessary to implement the logic, the specific join code necessary for each step, and finally translates the Formula into either a native spreadsheet formula, SQL statement(s), or MapReduce code, depending on the target selected by the user.

Creating Typed Tables

As mentioned, the system allows the user to specify each table in their schema as one of three types of tables:

1) Global Table: Contains 1 row of data per schema (this is fixed and invariant)
2) Entity Table: Acts like a normal database table (acts as a list of rows)
3) Relation Table: Takes a set of Entity Dimensions and automatically provides rows for each unique combination of IDs

The importance of the Global Table is that since it always contains exactly 1 row of data, it provides a common top-level place to which to aggregate data from all subsequent tables that the user defines. In other words, the existence of this table enforces that the schema diagram encodes a WC-DAG, because every other table is, at minimum, connected to the Global Table.

Additionally, the user can define any number of trees of Entity Tables to any level of depth desired. In this manner, the user creates schema that contains Entity Trees emanating outward from the Global Table (see Figure 1). The edges within these Entity Trees are termed as "existential" edges because each individual row in a specific sub-table requires that a parent row in the parent table exists so that the row is always in a properly configured state.

Furthermore, the user is able to create edges that cross Entity Trees, so long as they do not result in cycles in the schema diagram or create more than one directed path between any 2 Entity Tables (see Figure 2). These edges that span Entity Trees (change the 2 trees into a directed graph) are termed as “relatedness” edges because they are nullable, can take on extra dimensions of information, and can use a formula to determine their values.

Finally, the user can define Relation Tables on a specific subset of entities. These Relation Tables do not affect the structure of Entity Trees, but they provide useful places where input and output data can be stored in such a manner that it is keyed on a subset of relevant Entity IDs (see Figure 6).

From this schema, the system tracks the Join Paths that users can leverage to automatically join and aggregate information. It accomplishes this by tracking the specific set of Entity IDs relevant to each table (see Figure 16).

For situations where more than 1 of a certain entity dimension is used within a single Join Path (e.g. a Relation Table that contains 2 or more of the same Entity Dimension) the system assigns subsequent instances of the same entity dimension with an Alternate Dimension Number. This also occurs when at least one of the related Entities exist below another in the same Join Path.

Adding Variables to Tables

After creating the table structure of the schema, the user can then add Variables to each table that represent the data they need to record or compute (in reality, however, the user would probably add some tables, then add some Variables, then more tables, then more Variables).

For each Variable, the user can configure its usage by changing the set of values described in Figure 13. Some important values to note include:

- Data Type: What type of information is stored?
- Time Dimensions: How many Time Dimensions does the variable use?
- Variable Type: Is the variable a input or a formula? If formula, does the formula include aggregation?
- Foreign Key Table (for Data Type “Unique ID” only): Which other table this Variable references to obtain the list of valid ID values for this Variable?
- Formula (for Variable Types other than input): The spreadsheet-style expression tree that encodes the logic that should be executed to produce the variable’s value.

An “Existential Edge” mentioned above is created when the “Parent Entity Table” property is set to a non-null value for a specific Entity Table. After doing so, they system will automatically create a Variable of type “UniqueID” that is constrained to be an input value with no time dimensionality.

A “Relatedness Edge” is constructed directly by the user when the user creates a Variable of type “UniqueID” and sets it to reference a non-null Foreign Key Table and specifies to use this Variable in automating joins. This type of Variable behaves as any other Variable, so it may be configured as an input or as using a Formula to compute its value. It may also vary on Time Dimension(s).

Specifying Formulas

When a Variable uses a Formula, the user has the opportunity to define the Formula using many of the Operators that are available in standard spreadsheet applications, plus additional ones unique to the system (see Figures 14 & 15).

In these Formulas, users can reference other Variables in the schema by specifying the Variable in the format:

$”Table Name”:”Variable Name”:”Alternate Dimension Number”

When the user does not specify the Alternate Dimension Number, the system will use the default value of “1”.

The Operators available in these formulas implement many of the standard transformations available is standard database query or spreadsheet formula languages. Many of them provide standard logical or mathematical transformations.

However, some provide the opportunity to aggregate multiple values and return a single resulting value. These aggregators must be specified as the outermost expression in the Formula, so that the Formula is guaranteed to return a unique result (see Figure 14).

Also, additional operators exist for “Dimensional Movement” (which means moving laterally inside specific Structural or Time Dimensions), providing “Type Introspection”, or helping to export well-formed text values.

Furthermore, Formulas can use the “Query” operator, which allows a user to write a SQL-style query inside the Formula. The Query operator takes 3 arguments, the “where clause” argument, the “order by clause” argument, and the “select clause” argument. It operates over the set of relevant Dimensions (treating them as Unbound Dimensions) and uses the arguments to query rows, order results, and return the correct value.

Creating the Structural Map for a Schema Diagram (WC-DAG)

Because of the unique manner in which Typed Tables are used, the resulting schema is guaranteed to represent a WC-DAG. From this WC-DAG, the system generates a specialized data structure called the “Structural Map” that is used to cache the schema in a manner conducive to automating the creation of join code.

In addition to the set of Global, Entity, and Relation tables specified in the schema, the Structural Map also stores for each table the set of Entity IDs that the specific table varies on. This is crucial to automating the traversals up and down the Entity Existence Trees and Entity Relatedness Network so that an ID at the bottom of tree can easily be associated with its parent ID further up the tree, or vice versa.

In this manner, the location of any specific table in the Structural Map is the maximal set of Entity IDs that uniquely determine a specific row for that table (see Figure 16). This is called the "Entity ID Set". This set is treated as unordered for the purpose of determining position in the Structural Map.

For example (see Figure 24):

- The Global Table is located on the Structural Map at the Entity ID Set {}.
- The Customer Table is located at the Entity ID Set {Customer}.
- The Product Category Table is located at the Entity ID Set {Product Category}.
- The Product Table is located at the minimal Entity ID Set {Product} or the maximal Entity ID Set {Product Category, Product}.
- The Sales Order Table is located at the minimal Entity ID Set {Sales Order} or the maximal Entity ID Set {Customer, Sales Order}.
- The Sales Order Line Table is located at the minimal Entity ID Set {Sales Order, Sales Order Line} or the maximal Entity ID Set {Customer, Product Category, Product, Sales Order, Sales Order Line}.
- The "Customer <-> Product Category" Table is located at the Entity ID Set {Customer, Product Category}.

Navigating the Structural Map to Perform Query Automation

Now that the schema has been loaded into the Structural Map, the system can use the Structural Map to navigate the user's schema and to automate the joining of all information. This capability can be used internally within the system to compute results directly, or in the export code to automate the construction of platform specific join code (see Figure 22 & 23).

Common requests to the Structural Map might include:

- Was the Structural Map initialized to a valid state?
- What tables were included in the Structural Map?
- Does the Structural Map include table X?
- Get the minimal Entity ID Set for table X?
- Get the maximal Entity ID Set for table X?
- Does the Structural Map contain a valid Join Path from table Y to table Z?
- Does this Join Path from table Y to table Z yield a single unique row?
- What is the Join Path from table Y to table Z?

To do this, the Structural Map stores dictionaries that contain the relevant Entity ID Sets for each specific table. Additionally, the Structural Map tracks the Time Dimensionality of the Variables that create relatedness edges.

During computation, the actual sets of relevant ID bindings are loaded into the Structural Map, so that in addition to asking for the path between 2 tables, the computation engine can also ask for the specific instances involved in the path for a specific set of ID bindings.

For example, in terms of Figure 24, when computing the “Total Sales” of the Product with ID = 1, it is necessary to ask the Structural Map for all “Sales Order Line” instances (or rows) that are relevant to this situation. To obtain this value, the Structural Map determines the Join Path, then uses the specific binding {Product ID = 1} to obtain the set of “Sales Order Line” instances that refer to this Product ID value.

This is accomplished by first storing the set of instances for each table type in a tree structure, so that the set of instances IDs are readily available. Secondly, the Structural Map stores the specific Entity ID Set Bindings that are relevant to each instance. For example, the Structural Map will store both that:

- Product ID is part of the Entity ID Set for the Sales Order Line table
- The specific Product ID value “1” is part of the Entity ID Bindings for Sales Order Line instances {1,2,5,7,etc.}

In this manner, the Structural Map is responsible for maintaining the complete set of information relevant to automating traversal of the schema and to mapping relationships between all rows (or instances) across time so that the system can compute results.

Creating the Dependency Map from user-defined Formulas

In addition to the Structural Map, which stores relationships between tables and rows, the system also uses the Dependency Map, which stores relationships between Variables, to perform automation. The Dependency Map is created as a directed graph (see Figures 17, 18, 19, 20).

The system uses the Dependency Map to check if the schema contains cycles in the formulas. If it contains cycles, the system will then determine whether each cycle is computable or not. For instance, in computing loan repayment schedules, it is necessary to use the balance at the end of the prior period as the start for the current period. This type of dependency cycle will be resolved by moving laterally across the Time Dimension(s) in question (see Figures 18 & 19). If any cycle encountered is not resolvable, the Dependency Map will provide this information (see Figure 20) as an error message.

Finally, the system will use the Dependency Map to generate a dependency-ordered traversal of the graph, such that the result will determine the order that variables should be processed and whether any variables must be processed iteratively as Variable Groups (see Figure 21).

Computing Results with the Formula Processing Engine

To compute results within the system, the user must choose to show calculated values for the input data they provided. To compute these values, the system uses the Formula Processing Engine to perform the 3 phases of processing: 1) Initialization, 2) Validation, and 3) Computation (see Figure 22).

Initialization Steps

1) Initialize Structural Map
2) Initialize Dependency Map
3) Check Formulas use valid paths to navigate schema
4) Group Variables that must be processed together

5) Check Groups for Cycles
6) Determine if Cycles are computable or not

Validation Steps

1) Ensure Formula is configured correctly for Aggregation (if relevant)
2) Ensure Formula is configured correctly for Dimensional Movement (if relevant)
3) Check each Expression in the Formula uses valid Arguments to its Operation
    - Proceeds from inner-most to outer-most Expression
    - Does it use the correct number of Arguments?
    - Does it use the correct Data Types and Dimensionality for Arguments?
    - Does it use constant values where required?
    - Then propagate the resulting Data Type and Dimensionality to the next outward Expression
4) Use the outer-most resulting Data Type and Dimensionality as the values for the computed Variable
5) Ensure the resulting Data Type and Dimensionality match the values that the user originally configured for the Variable

Compute Steps

1) Use the dependency-ordered traversal of all Variables provided by the Dependency Map to determine how Variables must be Grouped, and which Groups can be batched together to compute in parallel, based on the order of Variable precedence

2) For each batch of Parallelizable Variable Groups, launch all the Variable Groups in the batch to compute in parallel

3) For each Variable Group currently being computed

    a. If the group contains a single Variable, compute for all relevant Dimension values at once, ordering Dimension values if necessary

    b. Otherwise, iterate over the ordered values for each Dimension(s) where Movement is used to resolve Cycles, computing all Variables in the Group at each stage of iteration

        i. For a specific Variable and set of Dimension values (aka Dimension Bindings), compute the Formula

            1. Use the Dimension Bindings to request the relevant set of rows to use from the Dependency Map

            2. Traverse the Expression Tree, applying the transformations from inner-most to outer-most, on the set of relevant rows

        ii. Store the computed results for each set of Dimension Bindings for each Variable that is computed

iii. Propagate the result to the Structural Map when evaluating a Variable that acts as a Relatedness Edge

4) Notify the user of success or failure

Using the Structural Map and Dependency Map in Database Exporters

Given that the system exports the schema to many different types of database platforms; it is useful to group these platforms under 3 categories to help in understanding the export:

1) Spreadsheet: A standard office application for tabulating values in 2 dimension matrices.
2) SQL Database: A database that uses a variant of the SQL programming language to operate on data and produce results.
3) MapReduce Database: A database that uses the MapReduce approach as the primary tool for operating on data and producing results. These databases implement MapReduce as a Mapping function that handles independent tasks and a Reducing function that combines the results of the Mapping function as a single task (often there is also an Ordering function between the Mapping and Reducing). MapReduce platforms may also implement a platform-specific language to operate over data, but developers most commonly use MapReduce processing code to interact with these technologies.

These platforms directly implement many of most commonly used logical, mathematical, and string manipulation Operations that the system provides for use in Formulas, so those Operations export, for the most part, identically.

Where the uniqueness of the system is forefront is in the export of:

1) Formulas that require joining multiple tables to compute
2) Formulas that require access to metadata to compute
3) Formulas that require Dimensional Movement to compute

For each of the 3 categories of database technologies, the system implements a specific Database Exporter that takes as an argument the specific platform to which the database is being exported and whether to export the user's test data, as well as other relevant run-time configuration information.

SPREADSHEET EXPORTER

The Spreadsheet Exporter exports an N-Dimensional matrix for each Variable, where N is determined by the sum of the Structural Dimensions and Time Dimensions relevant to that variable.

When exporting formulas the system will reference the correct value in each preceding matrix, and where necessary, use operators like VLOOKUP AND HLOOKUP to move laterally with the matrix.

SQL EXPORTER

The first step for the SQL Exporter is to export the tables in the schema so that the resulting platform-specific database code contains definitions for:

1) Metadata Tables
2) Tables for Input Values (with extra-Dimensional sub-tables where necessary)
3) Tables for Output Values (with extra-Dimensional sub-tables where necessary)

After this step, it is possible for the SQL Exporter to address all tables necessary to compute results.

At this point, the SQL Exporter generates a procedure that 1) determines which Structural, Time, and Variable values are included in a specific computation (e.g. include all rows or a subset of rows by Entity Table), 2) controls the order in which Variable Groups are computed, and 3) initializes the output tables with the necessary rows.

For each individual Variable Group, the SQL Exporter then creates a procedure that implements the SQL code necessary to compute and store results for each Variable in the Group, and, if necessary, iterate over the Dimension(s) of Movement using a cursor. This procedure starts with code that determines which rows and columns are necessary to compute based on which Structural and Time Dimensions are used and whether they are constrained to specific values due to iteration.

Then, Variable by Variable, the SQL Exporter uses the Structural Map to find the path between tables, and if necessary, uses Time Period specific sub-tables to access information that additionally varies across time. The SQL Exporter exports the Join Path using INNER JOINs for non-null edges, LEFT OUTER JOINS for nullable edges, and JOIN ON 1=1 for unbound edges.

Additionally, for lateral movement within a dimension, the same table is joined again, but with the value offset by the appropriate number of rows.

Finally, the SQL Exporter exports the Formula for each specific Variable by writing its Expression Tree logic as the SELECT statement of the SQL query, writing any conditions included in the Expression Tree in the WHERE statement, and writing any ordering criteria provided in the Expression Tree in the ORDER BY statement. The result of this query is then stored in the corresponding results table.

MAP-REDUCE EXPORTER

The MapReduce Exporter first exports text files (or platform-specific collections) that mirror the table schema created by the SQL Exporter.

It then exports an outer-most class that implements logic similar to the procedure in the SQL Export that orders the computation of Variable Groups (see full description above). The difference between the MapReduce Exporter and SQL Exporter is that the MapReduce Exporter will run Parallelizable Variable Groups at the same time.

Then for each Variable Group, the exporter creates a class that implements the Formulas as 1 or more MapReduce operations, similar to the procedure per Variable Group in the SQL export, using iteration to resolve cycles when necessary.

A major difference, however, is that the code to JOIN data will be implemented as 1 or more MapReduce operations (using ordering of results), instead of using SQL INNER and OUTER JOINS. Also, the exporter will create the necessary classes to encapsulate the joined results from the MapReducing.

Finally, the MapReduce export will export each Variable's formula logic as a MapReduce operation. Upon conclusion, the results are stored in an output text file or collection.

Below is a general list of steps performed in the Export.

Export Steps

1) Create Storage for Metadata, Inputs, and Outputs
    1) Spreadsheet: Cell Ranges
    2) SQL DB: Tables
    3) NoSQL DB: Files or Collections, also create a specific class for each combination of joined values that will be necessary
2) Create code to process all Variable Groups in order
    1) Spreadsheet: This will be handled by cross-linking formulas, so no extra work is necessary.
    2) SQL DB: Create procedure that determines specific Dimension values to pass to each Variable Group computation Procedure
    3) NoSQL DB: Create class that determines specific Dimension values to pass to each Variable Group class and call these classes to perform actual MapReduce operations
3) Create code to compute each specific Variable Group
    1) Spreadsheet: For each Variable in Group, export Variable to range and cross-link Formulas across Variable ranges
    2) SQL DB: In single Procedure, for each Variable in Group (in order of precedence), export inner SQL query that performs JOINs and outer SQL query that computes and aggregates data, using iteration to resolve cycles when necessary. Also, store results as necessary.
    3) NoSQL DB: In single Class, call Variable-specific MapReduce code (in order of precedence). For each Variable, export first MapReduce that performs JOINs and second MapReduce that computes and aggregates data, using iteration to resolve cycles when necessary. Also, store results as necessary.

Possible Additions / Other Incarnations

- Similar to Time Dimension, include Scenario Dimensions that can be used to store conditional values based on different states of the model. Use these states to encode conditional probabilities when necessary.
- Include additional Data Types that Variables can uses to store complex types (and include Operators that are specifically useful to those types), for example:
    - GPS Coordinate
    - Spatial Coordinate
    - Time Period
- Use a script (i.e. programming code) instead of a Formula to specify computational logic in the schema. Allow the user to provide a specification of which Variables are inputs to the script and which Variables are outputs and whether there is Dimensional Movement or Aggregation
- Use iterative calculations (e.g. generate a result, then try for better, and do so until an acceptable margin for error is met) to obtain convergent results for Cycles across Variables that are currently not computable within the system
- For boundary cases of schema that are currently invalid, solve this by allowing user to specify alternate Join Paths based on additional partial schema diagrams
    - This would allow users a way to overcome the constraint that at most one directed path exist between any 2 Entity Tables
    - The user could choose to use an alternate Join Path either for a complete Formula or only specific Variables within the Formula
- Add additional Table-level constraints
    - 1-to-0 Table Edge: Used to encode class inheritance in the schema. This would cause the condition that there only be at most one directed path between any 2 tables to be relaxed so that multiple tables could inherit from a base table.
    - 1-to-1 Table Edge: Used to encode that 2 tables are intertwined such that they act as the same table ID within the Structural Map
    - “Cascading Deletion” on Relatedness Edge: This would allow the deletion of a related row to cause deletion of the row that is related
    - Use ID Variables that are explicitly not configured for join automation to constrain the set of valid parent IDs or related IDs presented to the user for Variables that are configured for join automation
- Table Inheritance(mentioned above)
    - For instance Customer and Employee may both be based on Person
    - Store the common info in Person
    - Use the sub-tables (i.e. Customer and Employee) to determine the validity of the schema. Do not use Person, since doing so might violate the condition that there only be at most one directed path between any 2 tables
- Allow user to define Time Period Type hierarchy as separate schema diagram (i.e. Time Map)
- Allow user to specify Time Dimensions that operate of “Spot Values” instead of “Period Values”, such that changes on the Dimension occur instantaneously at specific points in time

- Allow user to specify how Relation Tables are managed
  - Should the table always contain all combinations of IDs, or should this happen on-demand?
  - For tables that use multiple instances of a specific Entity Type, should combinations that reference same ID be disregarded?
  - For tables that use multiple instances of a specific Entity Type, does order matter in producing a unique set of Ids?
- Consider allowing the user to specify existence hierarchy using Entity Existence Graph instead of Entity Existence Trees
- Track base Units of Measure (e.g. “meters”, “seconds”, etc.) and computed Compound Units (e.g. “meters/second”)
  - Automate conversions between Units (e.g. standard to metric, or vice versa)
- Automate Localization of Time and Language

Claims

A computer-implemented system for defining platform-independent database schema, comprising:

- Typed tables that indicate special behaviors that will be automatically implemented
- Formulas that capture computation logic
- Structural Map that is used to automate all JOINs based on user-provided schema
- Dependency Map that manages dependencies between Variables and is used to validate and order the sequences of logic in the schema
- Formula Processing Engine that compiles schema and computes results within the system
- Visual user-interfaces that aid the user in specifying schema and formulas
- Schema Exporter that translates the user’s schema and formulas to platform-specific database code

Abstract

Commonly, technical workers are faced with the task of constructing a database to store information for a particular purpose. These databases may contain code to compute results or queries to summarize information in a useful manner.

To help users perform this task, a multitude of tools have been developed that help users design database schema and generate tables, usually in a platform-specific manner (i.e. SQL-specific or Document-specific). Many of these tools either use the “Entity-Relational” model or claim to help automate persistence of “Entities” to a database.

However, in these tools, the term “Entity” is used synonymously with any database table that has a unique ID. Since almost all database tables have a unique ID, the term “Entity” loses any meaning or value because then every “table” represents an “Entity”.

The invention uses key distinctions to solve this problem and provide users with a more powerful platform for defining platform-independent database schema and automating the production of platform-specific logic by completely mapping the flow of information between tables.

First, instead of only providing a single type of table, the invention allows users to define "typed tables" in schema so that the default behaviors of the tables better reflect their expected usage. The invention also allows users to define columns (or Variables) in a superior way, such that the data can take on extra dimensions if desired (e.g. column as a single value in a row versus Variable as a matrix of values).

The result of these innovations is that the user is able to specify their schema in a manner that is completely platform-independent and can be automatically exported to a database platform of their choosing, automating the export of both data storage objects (e.g. tables) and logic (e.g. queries & procedures).

The core innovation of the invention is the sub-system for completely automating the joining and aggregating of information. Leveraging the power of this sub-system, the user is free to specify logic in concise spreadsheet-style formulas.

FIG. 1

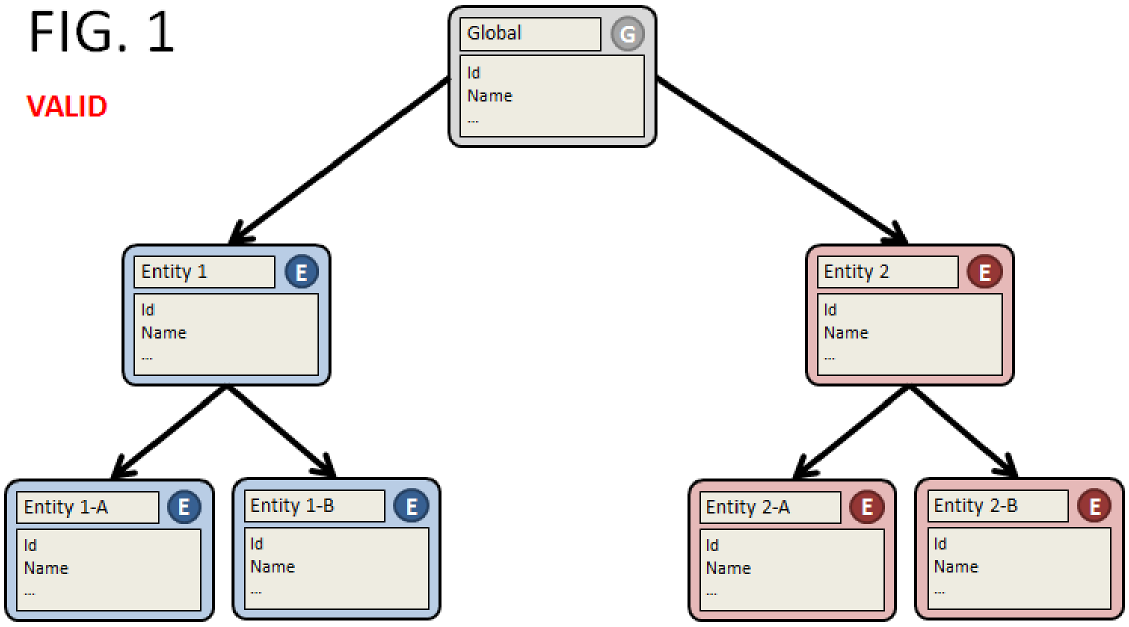


FIG. 2

VALID

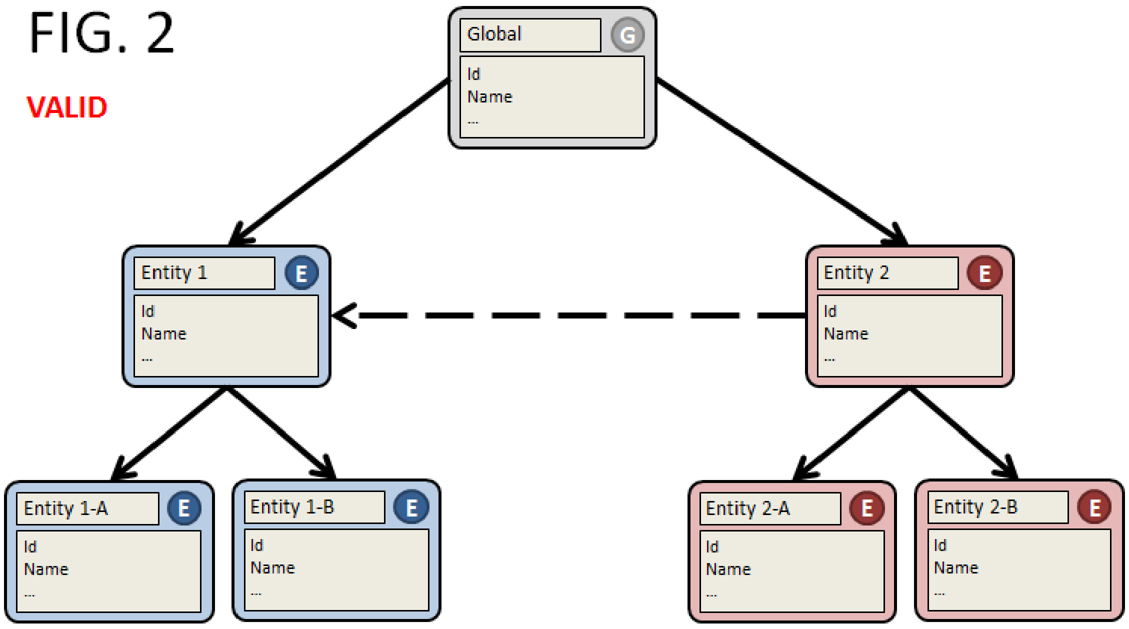

Global
G
Id
Name
...
Entity 1
E
Id
Name
...
Entity 2
E
Id
Name
...
Entity 1-A
E
Id
Name
...
Entity 1-B
E
Id
Name
...
Entity 2-A
E
Id
Name
...
Entity 2-B
E
Id
Name
...

FIG. 3

INVALID

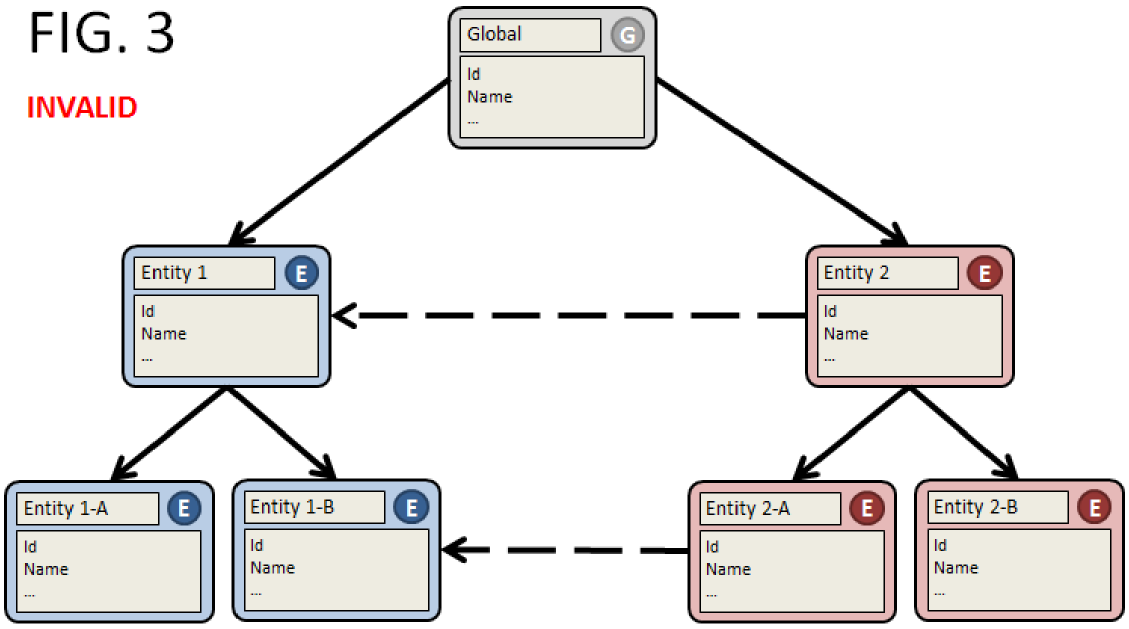


FIG. 4

INVALID

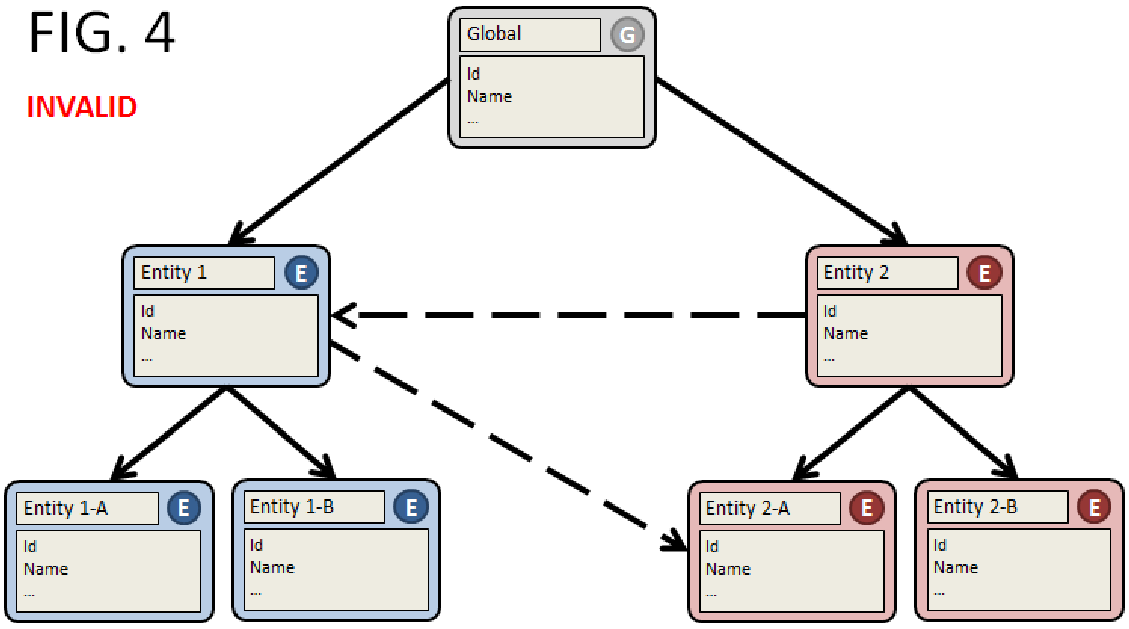


FIG. 5

INVALID

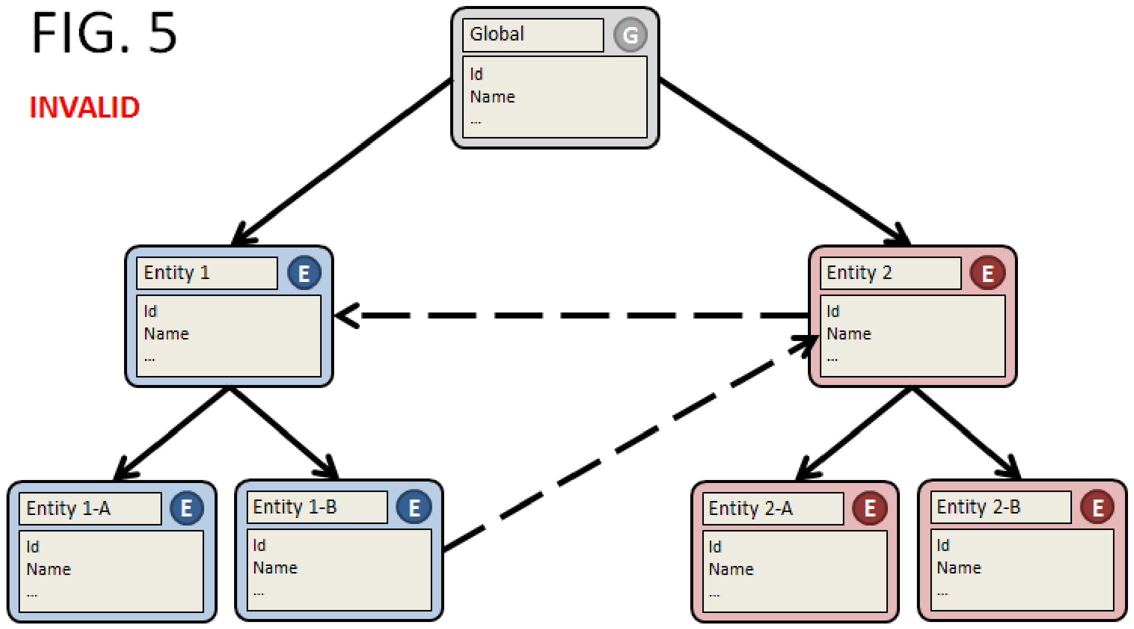


FIG. 6

VALID

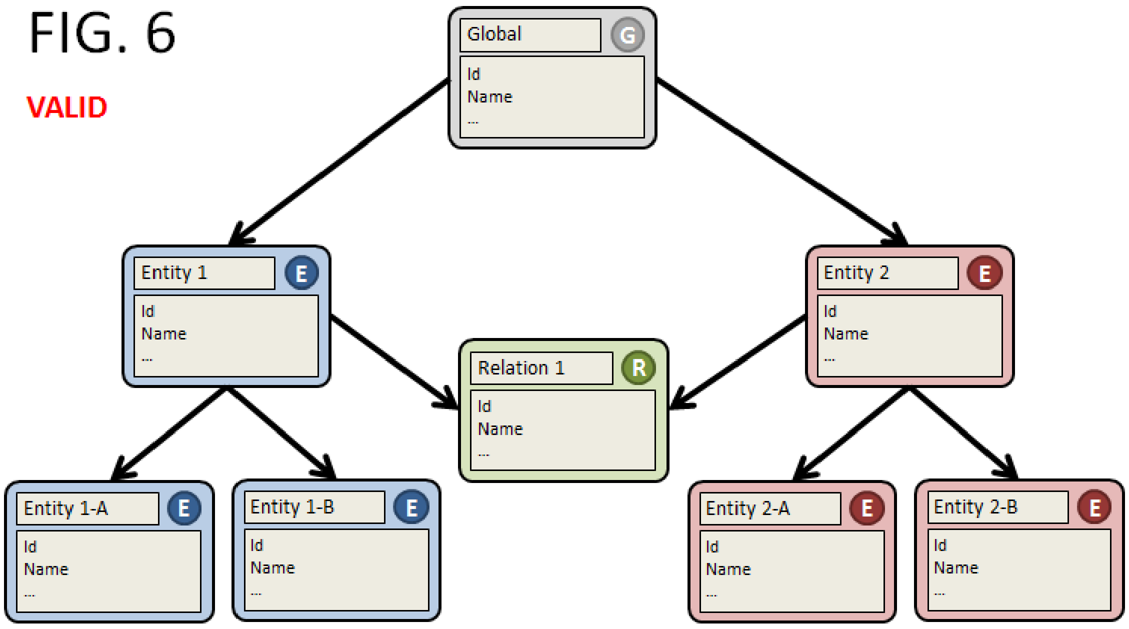

Global
G
Id
Name
...
Entity 1
E
Id
Name
...
Entity 2
E
Id
Name
...
Relation 1
R
Id
Name
...
Entity 1-A
E
Id
Name
...
Entity 1-B
E
Id
Name
...
Entity 2-A
E
Id
Name
...
Entity 2-B
E
Id
Name
...

FIG. 7

VALID

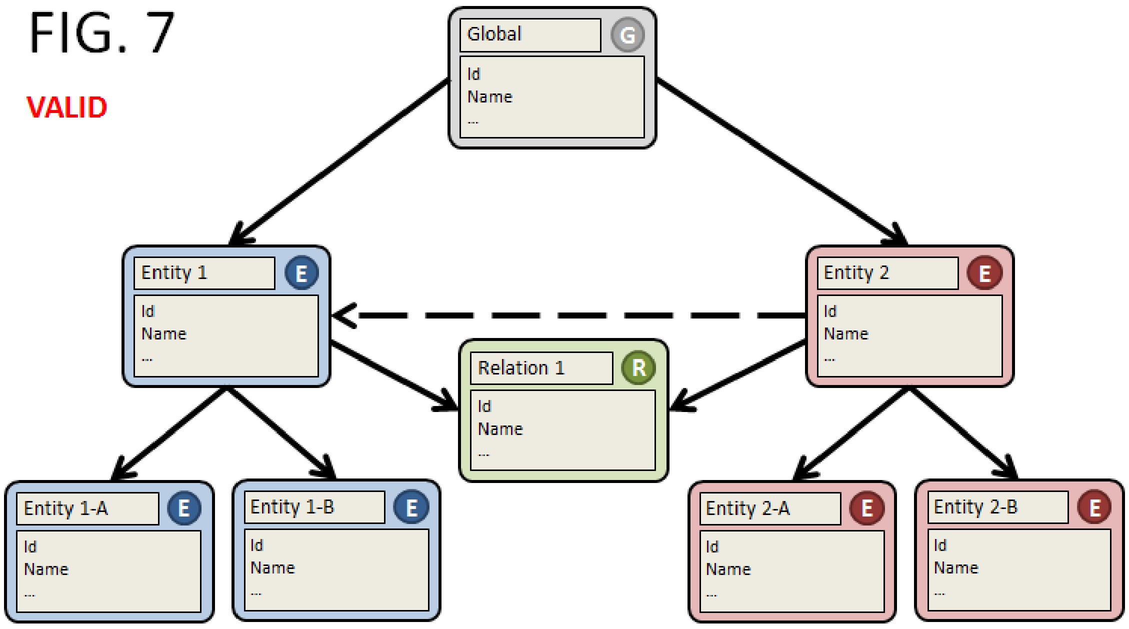

Global
G
Id
Name
...
Entity 1
E
Id
Name
...
Entity 2
E
Id
Name
...
Relation 1
R
Id
Name
...
Entity 1-A
E
Id
Name
...
Entity 1-B
E
Id
Name
...
Entity 2-A
E
Id
Name
...
Entity 2-B
E
Id
Name
...

FIG. 8

Tax Rate	35%

Profit (in $)

2016	2017	2018	2019	2020
100,000	110,000	120,000	130,000	135,000

Interest on Debt (in $)

	2016	2017	2018	2019	2020
2016	10000	10000	10000	10000	10000
2017	-	15000	16000	17000	18000
2018	-	-	20000	22000	25000
2019	-	-	-	30000	33000
2020	-	-	-	-	20000

FIG. 9

Model Properties	**Meaning or Purpose**
Name	Text identifier
Description	Text that describes purpose or usage
Default Time Period Type	A single value from preset list that describes valid Time Period duration {Year, Month, Day, etc.}
Default Start Date	The Date to use as the beginning date of each Time Dimension
Default End Date	The Date to use as the ending date of each Time Dimension
Zoom Factor	Percentage, where 100% shows as normal size
Canvas Size	Width and Height of Designer Canvas in Pixels

FIG. 10

Global Table Properties	Meaning or Purpose
Name	Text identifier
Description	Text that describes purpose or usage
Sorting Number	The index number to use when displaying in ordered list of Tables (if null, use alphabetical value)
Position and Size	Pixel Counts for {Left, Top, Width, Height}
Color	Integer Values for {Red, Green, Blue, Transparency}

FIG. 11

Entity Table Properties	Meaning or Purpose
Name	Text identifier
Description	Text that describes purpose or usage
Sorting Number	The index number to use when displaying in ordered list of Tables
Parent Entity Table	The Table that is used to nest the current Table under, such that each Row of current Table has exactly one Row in Parent
Use Automated Names	Whether to compute the Name of each Row
Automated Name Format	The format text that describes how to compute Name of each Row
Position and Size	Pixel Counts for {Left, Top, Width, Height}
Color	Integer Values for {Red, Green, Blue, Transparency}

FIG. 12

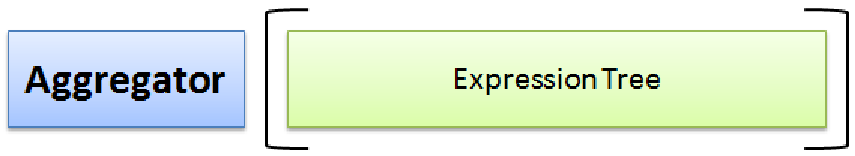
Relation Table Properties	Meaning or Purpose
Name	Text identifier
Description	Text that describes purpose or usage
Sorting Number	The index number to use when displaying in ordered list of Tables
Used Customized Names	Whether to compute the Name of each Row as more than just the comma-delimited list of Entity Names
Customized Name Format	The format text that describes how to compute Name of each Row
Structural Dimensionality	The number of times that each Entity in the schema is used as part of the set of Entity IDs that uniquely identifies rows in the table
Position and Size	Pixel Counts for {Left, Top, Width, Height}
Color	Integer Values for {Red, Green, Blue, Transparency}

FIG. 13

Variable Properties	Meaning or Purpose
Name	Text identifier
Description	Text that describes purpose or usage
Sorting Number	The index number to use when displaying in ordered list of Variables
Containing Structural Table	The Table to which the Variable belongs
Data Type	{Boolean, Integer, Decimal, Date, Duration, Unique ID, or Text}
Time Dimensionality	The number of Time Dimensions for the Variable and which Time Period Type each Time Dimension uses
Default Value	The value to initialize with upon creation of a new row
Variable Type	{Input, Basic Formula, Structural Aggregation, Time Aggregation}
Formula	How to compute values (described elsewhere)
Foreign Key Entity Type	If Unique ID, which Table (if any) this ID must exist in as a Row ID
Use to Automate Joins	If Unique ID, whether to include an edge in the schema diagram that can be used to auto-join data
Alternate Dimension Number	If Automating Joins, specifies whether this ID acts as part of the main Join Path (i.e. "1") or a subsequently numbered Join Path

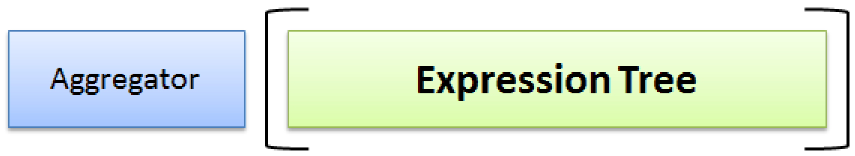
Only available for the "Unique ID" Data Type

FIG. 14



Structural Aggregators:	Time Aggregators:	Structural & Time Aggregators:
Sum	Sum	Sum into Periods
Average	Average	Average into Periods
Median	Median	Median into Periods
Mode	Mode	Mode into Periods
First	First	First into Periods
Last	Last	Last into Periods
Min	Min	Min into Periods
Max	Max	Max into Periods
	Present Value	

FIG. 15



Structural Movement:
Match ID
Match Value
Match Instance Index
Offset by Value
Offset by Number of Instances

Time Movement:
Match Date
Match Period Index
Offset by Duration
Offset by Number of Periods

Introspection:
Project Name
Model Name
Default Time Period Type Name
Model Start Date
Model End Date
Structural Table Name
Variable Name
Time Dimension Name
Time Period Type Name
Time Period Index
Time Period Start Date
Time Period End Date
Time Period Duration

Others:
Query (Where, Order By, Select)
Add, Subtract
Multiply, Divide
Modulo
Power
If-Then-Else
And, Or, Exclusive Or (XOR)
Not
Equals, Not Equals
Less Than, Less Than or Equals
Greater Than, Greater Than or Equals
Is Null, Is Not Null
Floor, Ceiling

FIG. 16

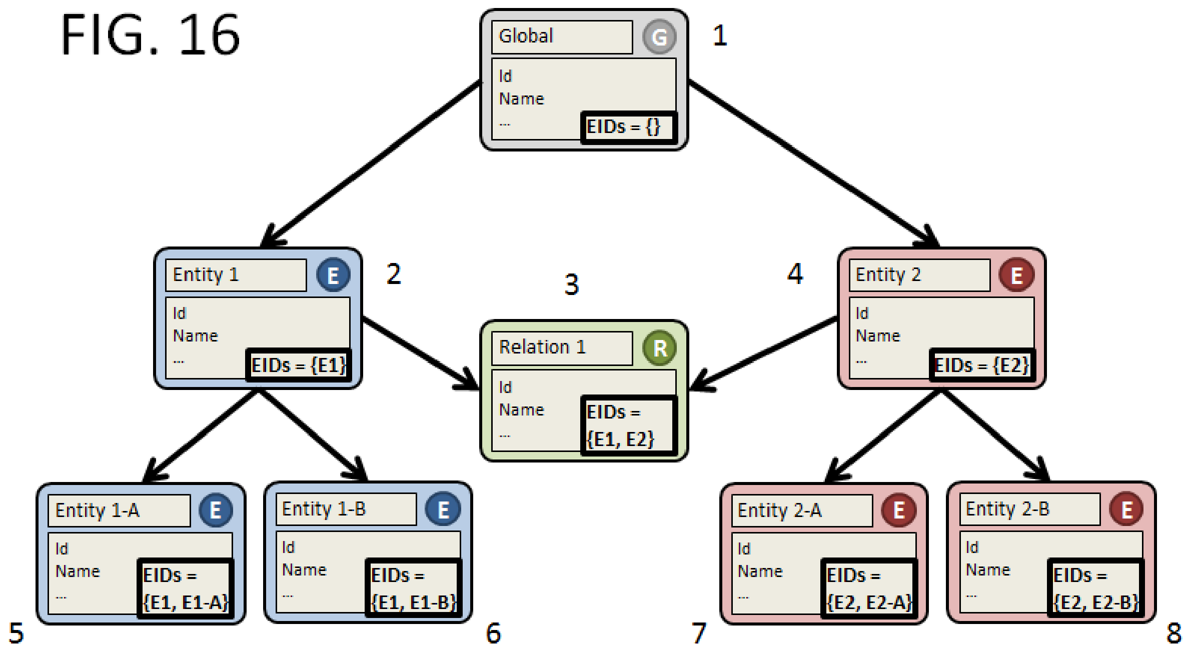

Global
G
Id
Name
...
EIDs = {}
1
Entity 1
E
Id
Name
...
EIDs = {E1}
2
3
Relation 1
R
Id
Name
...
EIDs = {E1, E2}
4
Entity 2
E
Id
Name
...
EIDs = {E2}
Entity 1-A
E
Id
Name
...
EIDs = {E1, E1-A}
5
Entity 1-B
E
Id
Name
...
EIDs = {E1, E1-B}
6
Entity 2-A
E
Id
Name
...
EIDs = {E2, E2-A}
7
Entity 2-B
E
Id
Name
...
EIDs = {E2, E2-B}
8

FIG. 17

VALID

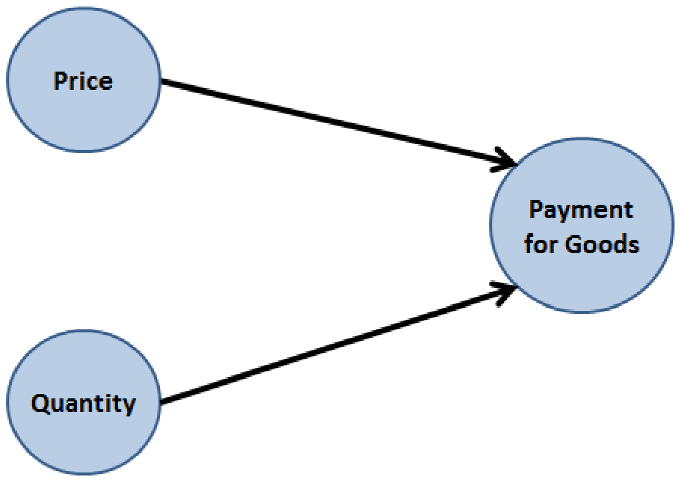


FIG. 18

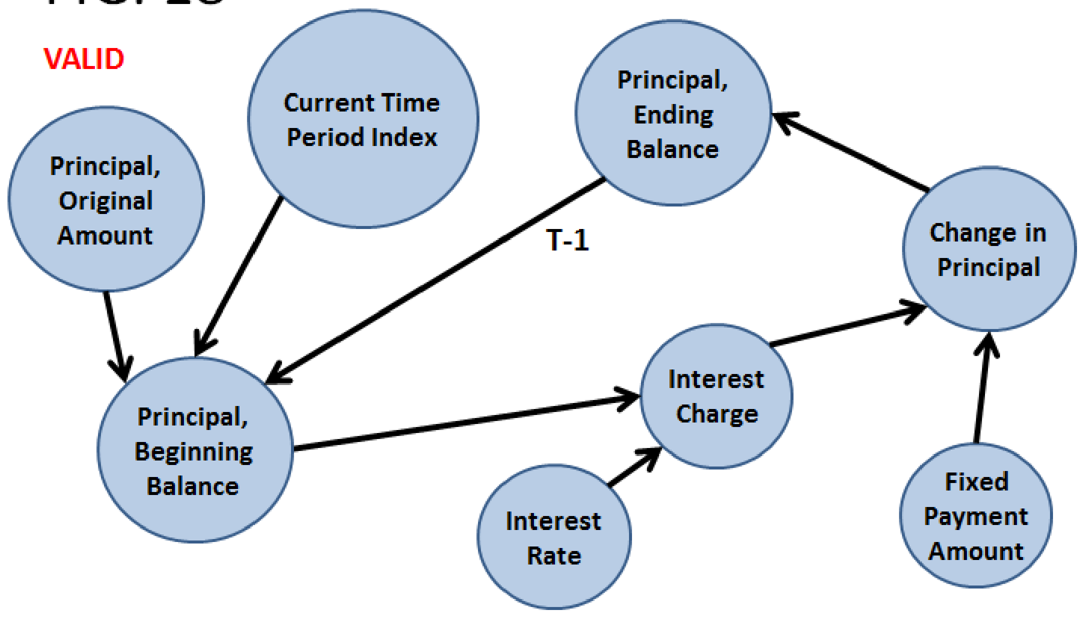

VALID
Principal, Original Amount
Current Time Period Index
Principal, Ending Balance
T-1
Change in Principal
Principal, Beginning Balance
Interest Charge
Interest Rate
Fixed Payment Amount

FIG. 19

VALID

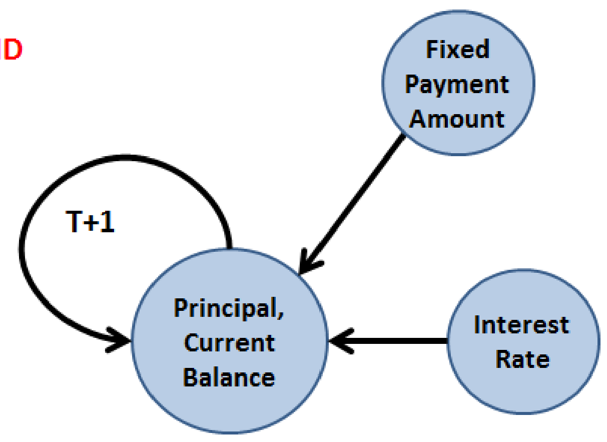

Fixed Payment Amount
T+1
Principal, Current Balance
Interest Rate

FIG. 20

INVALID

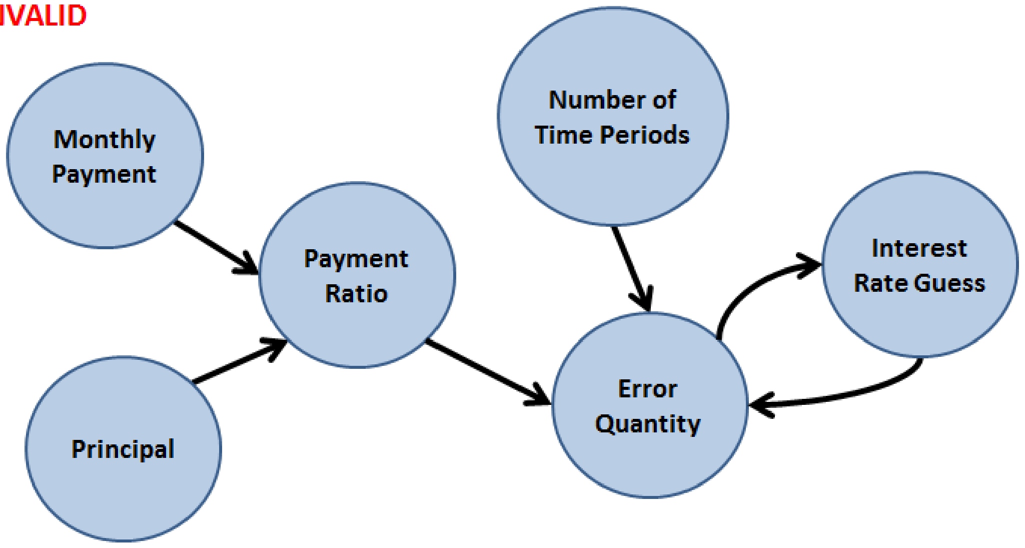


FIG. 21

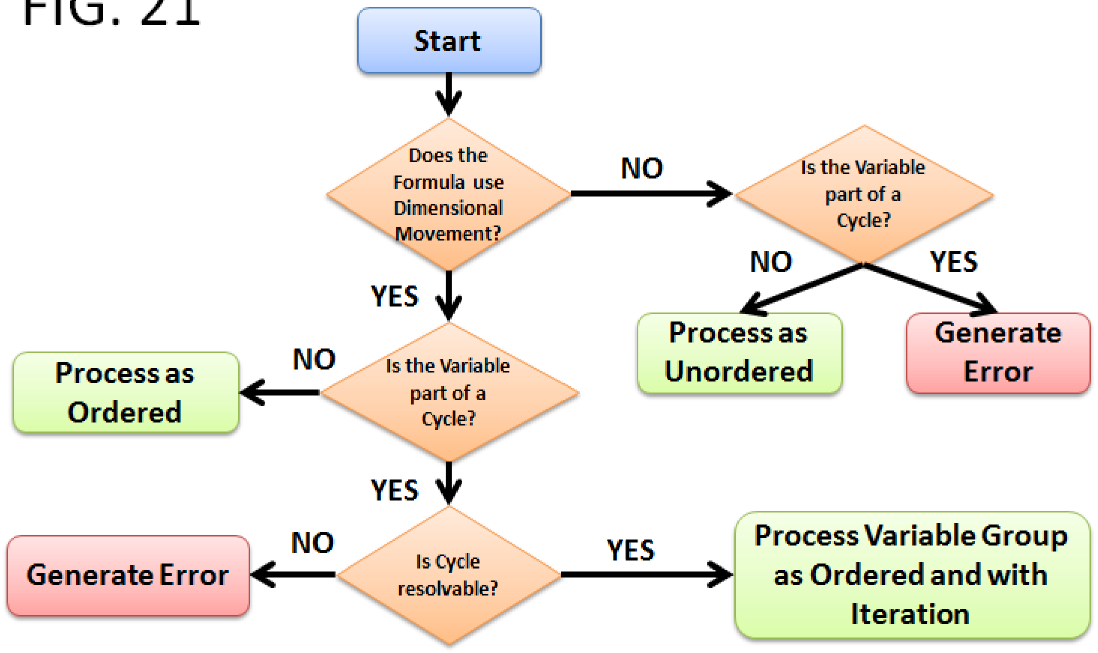

Start
Does the Formula use Dimensional Movement?
NO
Is the Variable part of a Cycle?
NO
YES
Process as Unordered
Generate Error
YES
Is the Variable part of a Cycle?
NO
Process as Ordered
YES
Is Cycle resolvable?
NO
Generate Error
YES
Process Variable Group as Ordered and with Iteration

FIG. 22

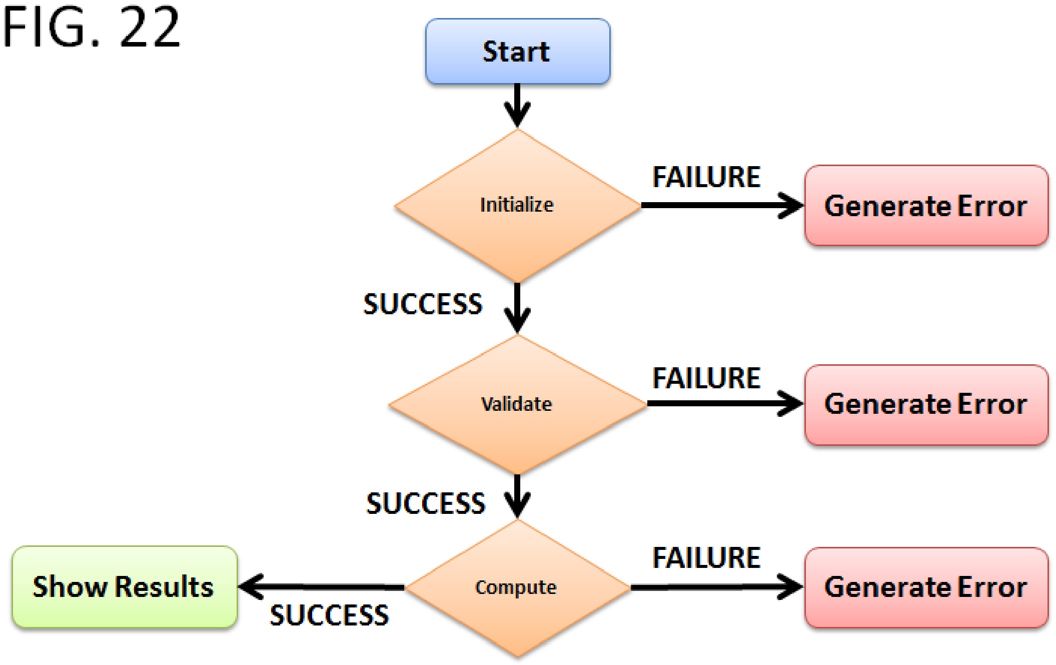

Start
Initialize
FAILURE
Generate Error
SUCCESS
Validate
FAILURE
Generate Error
SUCCESS
Show Results
SUCCESS
Compute
FAILURE
Generate Error

FIG. 23

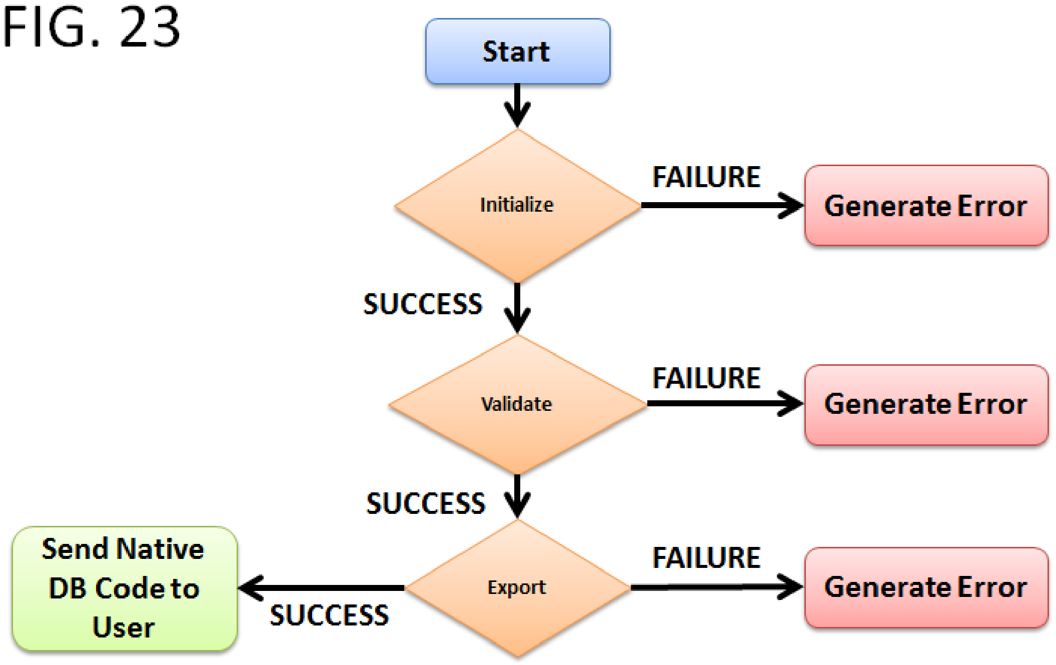
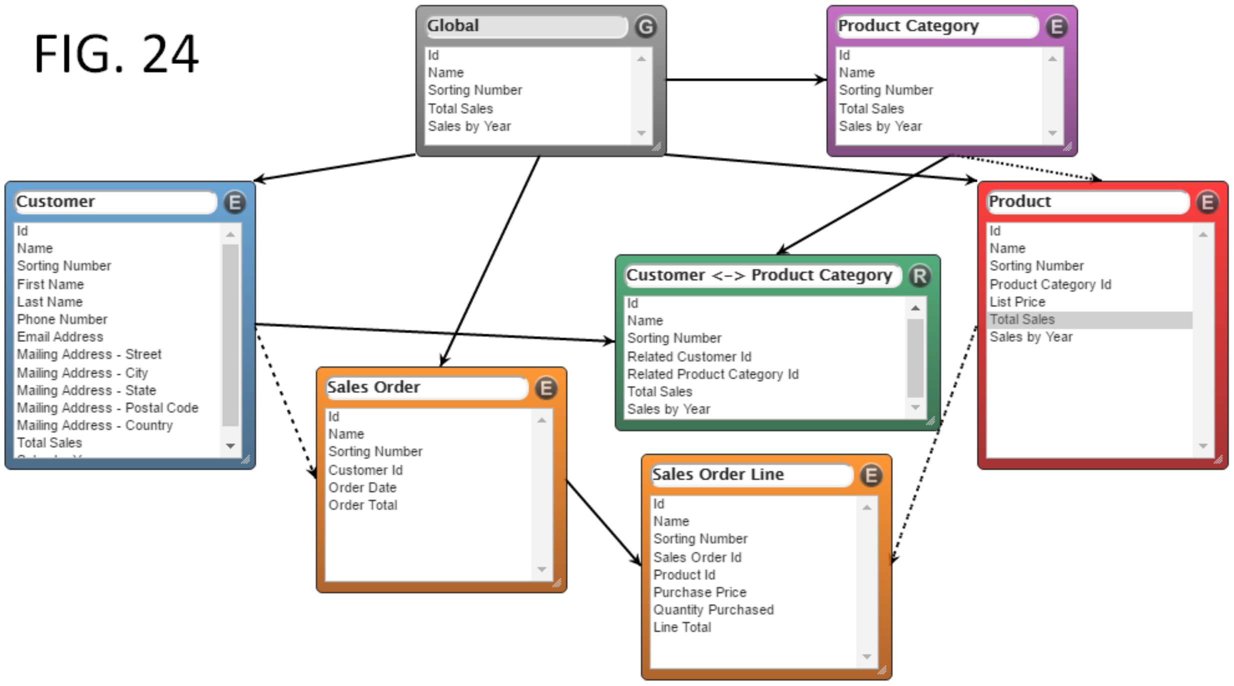

Start
Initialize
FAILURE
Generate Error
SUCCESS
Validate
FAILURE
Generate Error
SUCCESS
Export
FAILURE
Generate Error
SUCCESS
Send Native DB Code to User

FIG. 24


Global
G
Id
Name
Sorting Number
Total Sales
Sales by Year
Product Category
E
Id
Name
Sorting Number
Total Sales
Sales by Year
Customer
E
Id
Name
Sorting Number
First Name
Last Name
Phone Number
Email Address
Mailing Address - Street
Mailing Address - City
Mailing Address - State
Mailing Address - Postal Code
Mailing Address - Country
Total Sales
Product
E
Id
Name
Sorting Number
Product Category Id
List Price
Total Sales
Sales by Year
Customer <-> Product Category
R
Id
Name
Sorting Number
Related Customer Id
Related Product Category Id
Total Sales
Sales by Year
Sales Order
E
Id
Name
Sorting Number
Customer Id
Order Date
Order Total
Sales Order Line
E
Id
Name
Sorting Number
Sales Order Id
Product Id
Purchase Price
Quantity Purchased
Line Total