COMMON DATA MODELING PROBLEMS pdf

1: What is Common Data Modeling | Online Learning

This topic describes common problems that you might encounter when modeling data and explains how to solve them. I can't see any tables or views in Data Modeler If you start Data Modeler and see no tables or views, then one of the following has occurred: There aren't any tables in the database.

Overview[edit] Managing large quantities of structured and unstructured data is a primary function of information systems. Data models describe the structure, manipulation and integrity aspects of the data stored in data management systems such as relational databases. They typically do not describe unstructured data, such as word processing documents, email messages , pictures, digital audio, and video. The role of data models[edit] How data models deliver benefit [8] The main aim of data models is to support the development of information systems by providing the definition and format of data. According to West and Fowler "if this is done consistently across systems then compatibility of data can be achieved. If the same data structures are used to store and access data then different applications can share data. The results of this are indicated above. However, systems and interfaces often cost more than they should, to build, operate, and maintain. They may also constrain the business rather than support it. A major cause is that the quality of the data models implemented in systems and interfaces is poor". This means that small changes in the way business is conducted lead to large changes in computer systems and interfaces". This can lead to replication of data, data structure, and functionality, together with the attendant costs of that duplication in development and maintenance". The result of this is that complex interfaces are required between systems that share data. For example, engineering design data and drawings for process plant are still sometimes exchanged on paper". Typical applications of data models include database models, design of information systems, and enabling exchange of data. Usually data models are specified in a data modeling language. This shows that a data model can be an external model or view , a conceptual model, or a physical model. This is not the only way to look at data models, but it is a useful way, particularly when comparing models. For example, it may be a model of the interest area of an organization or industry. This consists of entity classes, representing kinds of things of significance in the domain, and relationship assertions about associations between pairs of entity classes. A conceptual schema specifies the kinds of facts or propositions that can be expressed using the model. This consists of descriptions of tables and columns, object oriented classes, and XML tags, among other things. This is concerned with partitions, CPUs, tablespaces, and the like. The significance of this approach, according to ANSI, is that it allows the three perspectives to be relatively independent of each other. Storage technology can change without affecting either the logical or the conceptual model. In each case, of course, the structures must remain consistent with the other model. Early phases of many software development projects emphasize the design of a conceptual data model. Such a design can be detailed into a logical data model. In later stages, this model may be translated into physical data model. However, it is also possible to implement a conceptual model directly. History[edit] One of the earliest pioneering works in modelling information systems was done by Young and Kent , [10] [11] who argued for "a precise and abstract way of specifying the informational and time characteristics of a data processing problem". They wanted to create "a notation that should enable the analyst to organize the problem around any piece of hardware ". Their work was a first effort to create an abstract specification and invariant basis for designing different alternative implementations using different hardware components. This led to the development of a specific IS information algebra. According to Leondes , "during that time, the information system provided the data and information for management purposes. Two famous database models, the network data model and the hierarchical data model , were proposed during this period of time". Codd worked out his theories of data arrangement, and proposed the relational model for database management based on first-order predicate logic. Entity relationship models were being used in the first stage of information system design during the requirements analysis to describe information needs or the type of information that is to be stored in a database. This technique can describe any ontology , i. In the s G. In contrast to other researchers who tried to create models that were mathematically clean and elegant, Kent emphasized the essential messiness of the real

world, and the task of the data modeller to create order out of chaos without excessively distorting the truth. In the s, according to Jan L. Harrington , "the development of the object-oriented paradigm brought about a fundamental change in the way we look at data and the procedures that operate on data. Traditionally, data and procedures have been stored separately:

COMMON DATA MODELING PROBLEMS pdf

2: Data modeling - Wikipedia

Data Modeling is actually a vast field but having a Common Data Model for a certain domain can answer problems with many different models operating in a homogeneous environment. To make Common Data Models, modelers need to focus on one standard of Data Abstraction.

A data model is a model that describes how data is represented and accessed, usually for a database. The construction of a data model is one of the most difficult tasks of software engineering and is often pivotal to the success or failure of a project. There are too many factors that determine the success of a data model in terms of its usability and effectiveness. Not all of them can be discussed here. Plus people tend to make different types of mistakes for different types of modelling patterns. Some modelling patterns are prone to some specific types of issues which might not be prevalent is other types of patterns. Nevertheless, I have tried to compile a list of some widespread mistakes that are commonly found in data modelling patterns. Avoid Large Data Models Well, you may be questioning how much large is large. You must ask yourself if the large size of the model is really justified. The more complex your model is, the more prone it is to contain design errors. For an example, you may want to try to limit your models to not more than tables. To be able to do that, in the early phase of data modelling ask yourself these questions â€" Is the large size really justified? Is there any extraneous content that I can remove? Can I shift the representation and make the model more concise? How much work is to develop the application for this model and is that worthwhile? Is there any speculative content that I can remove? If you consciously try to keep things simple, most likely you will also be able to avoid the menace of over modelling. Over modelling leads to over engineering which leads to over work without any defined purpose. A person who does modelling just for the sake of modelling often ends up doing over modelling. Watch carefully if you have following signs in your data model? Lots of entities with no or very few non-key attributes? Lots of modelling objects with names which no business user would recognize? You yourself have lot of troubles coming up with the names of the attributes? All the above are sure signs of over modelling that only increases your burden of coding, of loading, of maintaining, of securing, of using. Lack of Clarity or Purpose Purpose of the model determines the level of details that you want to keep in the model. If you are unsure about the purpose, you will definitely end up designing a model that is too detail or too brief for the purpose. Clarity is also very important. For example - do you clearly know the data types that you should be using for all the business attributes? Or do you end up using some speculative data types and lengths? Modern data modelling tools come with different concepts of declaring data e. So, before you start building â€" pause for a moment and ask yourself if you really understand the purpose of the model. Reckless violation of Normal Form This section is only applicable for operational data models When the tables in the model satisfy higher levels of normal forms, they are less likely to store redundant or contradictory data. But there is no hard and fast rule about maintaining those normal forms. A modeler is allowed to violate these rules for good purpose such as to increase performance and such a relaxation is called denormalization. But the problem occurs â€" when a modeler violates the normal form deliberately without a clearly defined purpose. Such reckless violation breaks apart the whole design principle behind the data model and often renders the model unusable. So if you are unsure of something â€" just stick to the rules. The above figure shows a general hierarchical relationship between customer and its related categories. Traps in Dimensional Modelling When it comes to dimensional modelling, there are some inexcusable mistakes that people tends to make. Theoretically speaking there is no issue with such a model, at least until one tries to create the ETL programming extraction-transformation-loading code behind these tables. This new row will have new surrogate key. But in the Product table, any existing row is still pointing to the old product type record and hence leading to data anomaly. Indiscriminate use of Surrogate keys Surrogate Keys are used as a unique identifier to represent an entity in the modeled world. Surrogate keys are required when we cannot use a natural key to uniquely identify a record or when using a surrogate key is deemed more suitable as the natural key is not a good fir for primary key natural key too long, data type not suitable for indexing etc. But surrogate keys also come with some disadvantages. The values of surrogate keys have no

relationship with the real world meaning of the data held in a row. Even query optimization becomes difficult when one disassociates the surrogate key with the natural key. The reason being â€" since surrogate key takes the place of primary key, unique index is applied on that column. And any query based on natural key identifier leads to full table scan as that query cannot take the advantage of unique index on the surrogate key. Before assigning a surrogate key to a table, ask yourself these questions â€" Am I using a surrogate key only for the sake of maintaining standard? Is there any unique not null natural key that I can use as primary key instead of a new surrogate key? Can I use my natural key as primary key without degrading the performance?

COMMON DATA MODELING PROBLEMS pdf

3: Microsoft Dynamics the Common Data Model explained

The crux of the problem is the age-old chasm trap relational modeling construct. Although you may not have heard of it, it is extremely common and often a challenge for BI projects. Chasm Trap.

The meaning of the record data model as a representation of the physical file design was apparent. It represented a schematic of record interconnections that could readily be translated into code which instructed the DBMS how to construct the files. Data record content was determined by the analyst and designer and only they needed to understand the logic behind the placement of data in specific records. The model was a tool of physical file design and a method of communicating design decisions between the designer and the implementation team. The use of the analysis data model during the design phase as a communication mechanism between designer and user was less effective. Its meaning was also less apparent than that of the record model. The analysis model attempted to address the notion that data from the user perspective were descriptors attributes of things entities , and that by identifying those things, their attributes and their relationships one would be identifying the subjects of data required by the corporation. The activities to produce these new models were positioned at the beginning of the development life cycle before the beginning of the data element identification process and well before the steps of assembling those elements into records. Until the introduction of these models any data analysis activities that were performed attempted to develop the file and record designs by going from the detail elements to the general records and files. The new models took a general to specific approach and added a new component to the model the concept of data subjects. This allowed the designer to proceed with the design logically, beginning with identification of subjects, then developing conceptual data groupings, populating those groupings with data elements and then taking the resultant model and translating it into physical records and files. The specific record layouts and file structures would be dependent upon the specific DBMS or data management technique chosen for implementation. Areas of difficulty The differences between the record models and the ER models stemmed from their content, from their perspective what they were intended to represent from their method of construction and from their use. Differences also arose from the terminology used by each. The following is a summary of these areas of difference, Changes in perspective The first area of difference arose from the change in perspective of the modelers. Prior to the analysis model, data file design tended to follow the pattern set by the forms that preceded them. That is the files were designed to contain data obtained from earlier forms. These forms tended to be utilitarian and rather restrictive or limited in the amount, kind and variety of data they collected. They were also highly standardized. A given form was designed to collect a certain kind of data, for a certain purpose. Any data not anticipated in that form was not intended to be included. Where unanticipated data was added to existing forms, or where form fields designated for one kind of data was used to another kind, confusion occurred, and mis-communication resulted. The data collected by the forms and later by the automated files that replaced them did not allow for variation. Data was strictly formatted, strictly coded, and all data indicated on the form had to be collected. Because forms represented the data to be collected for a specific purpose it was codified for that purpose and rarely for any other. Customer information on an account opening form, was rarely of much use to the marketing department, since it was usually designed by the sales department, sometimes in consultation with the credit department, to determine the credit worthiness of the client and to determine what kind of account was to be opened for that customer. Changes in content assumptions The record models assumed that all data represented by the model had to be collected, although some variation was allowed. Another major assumption of these models was that each occurrence or entry in the file, each set of records was relatively uniform. These types of models allowed for few variations in the kinds of subjects their data represented. They also assumed that for the most part all occurrences would be processed in a similar manner. There was one additional assumption which was key to the record model. Each symbol on the model represented a collection of data elements segment, row or record and each line where they were used represented the physical address or pointers which connected the records to each other. The only additional information contained on these models in most cases represented whether a given record

occurred once or more than once within a "logical" set of records. Changes in terminology Each of the many different types of data model, both record and conceptual, introduced terminology and definitions intended to explain the concepts inherent in these models, and in some respects to differentiate the model from all other models. Each model had proponents who explained, clarified and extended the model beyond its original scope. Many of these models also engendered large groups of practitioners who met regularly to exchange ideas and experience. From these meetings originated more new terminology and concepts. Because the models were used primarily for DBMS file construction, and because most companies tended to standardize on a single DBMS, practitioners tended to be well versed in the terms and concepts of their particular model but relatively ignorant of the terms and concepts of any other model. As firms began to use more than one DBMS, the terminology became confusing, because the same words meant different things for each. This difference between what the designer built and what was presented to the programmer or user led to the rise of a distinction between the "physical" records and the "logical" records, between "physical" database and "logical" database. The conceptual, analytical or logical models models shifted the focus of the designers and analysts from the question "how should the data be grouped and stored? The analytic model became a tool for stimulating ideas, identifying requirements and organizing those requirements in a coherent fashion. This use of the model however introduced many new concepts and perspectives which were only vaguely understood, and directed the analysis into areas which were equally vague and ill-understood. Further confusion was introduced when designers attempted to translate concepts from the record models into the analytic models. The analytic models, because they portrayed concepts, became vague and muddled. This was because in most cases the concepts themselves were vague and muddled. The intent of the model was no longer clear nor was it clear in many cases what the model was trying to communicate. The primary form of the mew or analytic models was the entity-relationship model. As originally proposed it was to be a link between the real world things that the user worked with and the record models that the automated system implementation teams worked with. It was intended to be DBMS independent, and focused on things about which data was collected rather than on how data about things was to be stored. These new models were part of a new approach to design, an approach from the user perspective. These models depicted business entities and their relationship with each other. The concept was stark in its simplicity. All business data is collected and maintained about people, places, things or concepts of interest to the firm. Thus if we model those things we have a strong foundation for determining what data is needed about these things. This new approach however developed still another set of terms to describe its components and to explain its concepts. It was strongly grounded in the relational model, and incorporated many of the concepts of normalization associated with the relational model. Although data is collected about people, places, things and concepts files are created about groups or collections of people, places, things and concepts. Since it is cumbersome to repeat the phrase people, places, things or concepts each time when talking about these groups in general, data modelers use the term generalized entity. Thus an entity is any group of persons, places, things or concepts about which the firm must collect and maintain records records in this sense are business records. Many of the analysts and designers familiar with the record models saw the new models as a new way to portray the record relationships, rather that as a different kind of model. Unlike the record model where the symbols or model components represented a record, which was a representation of a group of data elements, the ER model symbols, or components represented entities. The term entity as defined in the dictionary means the fact of being, existence. There is no representation, implied or otherwise as to the subject of that existence. In other words, without further detailed explanation, the term entity means absolutely nothing. Even if the ER models were limited to people, places and things they would probably have not been easy to build and understand. The greatest difficulty however is engendered by the inclusion of concepts in the definition. First, by including concepts and sometimes even events within the definition of the term entity, we have in effect made an entity equate to anything we want it to be, since the span of the definitions includes everything within our perception. Thus, not only can records become entities, but also roles, and even entities themselves can become part of what is being modeled by the ER models. The latter can be seen most clearly in data dictionary also called repositories, and encyclopedias models when data processing personnel attempt to create entry

types to document entities from their analytic models. The inclusion of concepts and events in the definition, caused many modelers to find themselves in the curious position of having to discuss the "entity entity. The record model described a record structure. It was implicit in the model that each structure or schematic of records represented the data about a given subject and that each iteration of the structure, or each iteration of records represented data about a given instance. The record model however was the final product, it did not decompose into any other type of model, nor did the records described decompose into different kinds of records. The entity-relationship model is a decomposition model. That is it is not intended to be used as a final form, but as an intermediate product. It was a transition product in much the same manner that the requirements documentation of the procedural analysis was not a final product but a reference point for the analysis phase and and the starting point for the detail design phase. Semantic versus structural models The entity-relationship model however was also substantially different from all other models then in use in the analysis and development cycle. That difference arose from the fact that all other models were either schematic such as the data record models or strict decomposition models such as the functional and process decomposition models of the analysis and design processes. It was also different from the flow models as typified by the data flow diagrams used by structured analysis advocates. The difference was that the ER model was what was called a semantic model. Semantics is defined as pertaining to meaning, as in language. The semantic model attempted to translate the meaning of the words used in requirements statements into pictorial representations for analysis. Specifically the semantic modelers attempted to translate the business rules which governed how, when and why the objects of business interest entities related to each other. Previous data models looked at data record relationships, and portrayed the paths that had to be established between one set of records and another. These paths were analogous to the indices and cross references established between sets of paper form files. They attempted to automate the means of associating data in one file with data in another file. The new models were an attempt to step back from the record models and look not at the data about things, but at the things about which the data was collected, and the relationships that had to exist between them based upon the statements of business requirements. Conceptual ambiguity This step back forced both analyst and user to look at and clearly define those things, those entities. The development of the ER models was begun at a higher level and from a different perspective than most previous models. The higher level took it outside a specific business area, outside most operational areas and into the realm of the managerial, and sometimes even into the strategic levels of the firm. The focus shifted from ascertaining the data content of forms to trying to understand, or come to some common agreement about some of the fundamental concepts of the firm - those of product, customer, location, and sometimes even function and process. The new models highlighted many areas of ambiguity, misunderstanding and sometimes disagreement, problems which were masked by lower level models because they did not have to deal with it. These models also highlighted hidden redundancy of concept and data by looking at the business rules which were the foundation of the business processing. In many cases these business rules were never clearly defined, or were defined in several different ways. The models also highlighted another and perhaps more disturbing problem, that of conceptual inaccuracy. Many employees, in most cases highly experienced employees did not have an accurate understanding of those business rules, nor of the legislative and regulatory rules under which the firm must operate. In still other cases, the models highlighted objects of business interest about which data was not being collected, or for which data was being collected in an incorrect manner. The meaning of relationships Another area of problem could be viewed as technical rather than analytical.

4: Modeling Issues

Model location: Article III: Common mistakes in dimensional modeling > INCOMPLETE DIMENSION-FACT RELATIONSHIP. Let's have a peek at the data. We'll imagine that we need to roll up the data in two scenarios.

Why does it take longer? It seems PostgreSQL is not wise enough to select 30 newest records without sorting all , of them. The query plan now is quite different: And estimated query cost is less than 1. You have a performance problem? The first attempt to solve it should be to find long running queries, ask your database to explain them, and look for sequential scans. If you find them, probably adding some indexes will speed things up a lot. Yet, database performance design is a huge topic and it exceeds the scope of this article. You can use this information when you design a model for a new system. If your bookstore is very successful, the volume of data in the purchase table can be very high. The more you sell the more rows there will be in the purchase table. If you know this in advance, you can separate current, processed purchases from completed purchases. Instead of a single table called purchase, you can have two tables: Current purchases are retrieved all the time: On the other hand, completed purchases are only kept as historical data. They are rarely updated or retrieved, so you can deal with longer access time to this table. With separation, we keep the frequently used table small, but we still keep the data. You should similarly optimize data which are frequently updated. Imagine a system where parts of user info are frequently updated by an external system for example, the external system computes bonus points of a kind. Then there are also other information in the user table, for example their basic info like login, password and full name. The basic info is retrieved very often. The frequent updates slow down getting basic info of the user. The simplest solution is to split data into two tables: Separating frequently and infrequently used data into multiple tables is not the only way of dealing with high volume data. Book description is likely to remain unchanged, so it is a good candidate to be cached. Use business, domain-specific knowledge your customer has to estimate the expected volume of the data you will process in your database. Separate frequently updated data from frequently read data Consider using application-level caching for heavyweight, infrequently updated data. Here is our bookstore model after the changes: Customers come from all over the world and use different time zones. Managing time zones in date and datetime fields can be a serious issue in a multinational system. The system must always present the correct date and time to users, preferably in their own time zone. The users must see the promotion date in their own time zone. In a multi-time zone system date column type efficiently does not exist. It should always be a timestamp type. A similar approach should be taken when logging events in a multi-time zone system. Time of events should always be logged in a standardized way, in one selected time zone, for example UTC, so that you could be able to order events from oldest to newest with no doubt. In case of handling time zones the database must cooperate with application code. Databases have different data types to store date and time. Some types store time with time zone information, some store time without time zone. Programmers should develop standardized components in the system to handle time zone issues automatically. Check the details of date and time data types in your database. Store date and time in UTC. Handling time zones properly requires cooperation between database and code of the application. Make sure you understand the details of your database driver. There are quite a lot of catches there. In my opinion, we have a serious problem. Perhaps we have a backup from three months ago, so we can restore the backup to some new database and access the data. Then we will have a good chance to restore it and avert the loss. But to do that, several factors must contribute: And when we eventually restore the data but is it the correct version for sure? Who damaged the data three months ago? How do we check that? To determine this, we need to: This will for sure take a lot of time and it does not have a big chance of success. What our model is missing, is some kind of audit trail. There are multiple ways of achieving this goal: As usual, it is best to keep the proverbial golden mean. You should find balance between security of data and simplicity of the model. Keeping versions and logging events makes your database more complex. Ignoring data safety may lead to unexpected data loss or high costs of recovery of lost data. Changes in the model were as follows on the example of the purchase table: We add it here because we encounter it quite often, and it does not seem to be widely known. Most often we assume that

sorting words in a language is as simple as sorting them letter by letter, according to the order of letters in the alphabet. But there are two traps here: If we have content in one language only, it is clear, but if we have content in 15 or 30 languages, which alphabet should determine the order? We will illustrate this with a simple SQL query on French words: But these words are French, so this is correct: It is a feature of this particular language. So â€" language of content can affect ordering of records, and ignoring the language can lead to unexpected results when sorting data. Here is the final version of our bookstore model:

COMMON DATA MODELING PROBLEMS pdf

5: High School: Modeling | Common Core State Standards Initiative

A model is an abstraction of some aspect of a problem. A data model is a model that describes how data is represented and accessed, usually for a database. The construction of a data model is one of the most difficult tasks of software engineering and is often pivotal to the success or failure of a project.

August, When designing your dimensional model, it is worthwhile to watch out for mistakes that commonly occur during the process. Specifically, they can occur in the relationships between tables, both in fact-to-dimension and dimension-to-dimension relationships. As you start a BI-related project, bulletproof dimensional design is hugely important. What makes a design bulletproof is the early mitigation of common design mistakes. In dimensional modeling, these errors are mostly tied to summary computations. Summary computations avg, sum, median, etc. They also are found in join operations that combine fact and dimension tables. Bad summary computations can lead to: Incorrect results Errors in decision making Unusable performance optimizations These problems occur repeatedly in different dimensional models across all industries. Dimension Summarizability Problems We can classify dimensional modeling problems into two main categories: They are evident in the erroneous cardinalities of summarized data. We will look at three anti-patterns in data warehouse DWH modeling between dimension tables. Our first two anti-patterns relate rolling up and drilling down operations in a DWH. The third involves problems with a many-to-many relationship between dimension levels. But this does not make it any less important to identify and remedy exceptions from the model. We can see an incompleteness when there is an inconsistency between the total shown and the drill-down value. Drill-down incompleteness occurs when at least one of our parent table values does not have any values in the child entity. Summary calculations on parent table dimensional values yield different results than those done on child dimensional table values. They should be the same. Differing summary calculations on measures from the fact table are not erroneous if the connection to the fact table is only through a child table. Note the zero-to-many relationship on the child table. Therein lies the problem. Drill-down incompleteness Moving from left to right, we drill into the data values of all the sectors. When we look at the data, we see that the minimal date value on the parent sector table is different than the minimal date value on the child department table. This is because treasury values are not decomposed in the department table as other sectors are. We need to add a default value for the parent in the child entity and create a connection between them. In our example, we could add a default treasury value in the department dimension. Another solution â€” the more common one â€” is to add a default value for all unallocated sectors. Roll-Up Incompleteness Roll-up incompleteness is the reverse of drill-down incompleteness. Moving from a more granular child dimensional table to a less granular parent table, we get a smaller total. This total indicates that not all child values are allocated to parent values. Example The fact table is connected to the child table in a typical snowflake schema. In this case, we have two dimensional tables: Again, this is the problem. And again although this is a valid modeling construct, it opens the door to errors. Moving from left to right, we go from a finer grain to a coarser grain. We roll up the data. Notice the difference in total values on different grain levels. How to Resolve Roll-Up Incompleteness As roll-up and drill-down incompleteness are mirror images of each other, so are their solutions. We need to add a default value for the child entity to the parent entity, with a connection between them. In this example, it would be an unallocated category value. Non-Strict Dimension Relationships This is another issue that we can easily identify. Unlike roll-up and drill-down incompleteness, a non-strict dimensional relationship problem is pure design error. Watch for it when there are many-to-many relationships in the model: The Vertabelo modeling tool does not allow many-to-many relationships. A good rule of thumb in dimensional modeling and modeling in general is to avoid many-to-many relationships. In this case, I will display the model with two one-to-many relationships. We have a two dimensional tables: The relationship is many-to-many because each month can have many weeks and one week can be in two months. Non-strict Incompleteness We represent the weeks of a year as a sequence of numbers and months in a year as a list of names. The sum of data of sales in weeks is different than the sum of sales in months because there is an overlap in some weeks. This happens when a week falls into two months or when certain months are not in

our data period March. The data displayed is correct, but it is not roll-upable. How to Resolve Non-Strict Dimension Relationships We solve this error by placing the dimensions in different hierarchies. If we look at the model for this solution, we see two hierarchies which are independent of one and other. We mitigated the roll-up operation from week to month. For example, in the above case we would define one major category â€" the major parent â€" out of many categories. Dimension-Fact Summarizability Problems Dimension-fact summarizability problems are found in operations between fact and dimensional tables. Like dimensional summarizability problems, they are evident in the erroneous cardinalities of summarized data. When looking at dimension-fact summarizability problems, we commonly see two modeling anti-patterns. The first relates to the joining of incomplete dimensional table data for all fact table values. The second relates to a non-strict relationship between values in fact and dimensional tables. Incomplete Dimension-Fact Relationships Incomplete dimension-fact relationship problems manifest themselves in join operations between fact and dimensional tables. They occur when the fact table contains measures with no corresponding value in the dimensional table. Summary calculations on the fact table vary depending on the dimensional tables we are using for our calculation. You may have already noticed the incomplete relation to the customer table. As with the incomplete dimensional model, therein lies the problem. In the first scenario, we must display the sum of all balances for customers on a monthly grain. In the second scenario, we must display the complete sum of all account balances on a monthly grain. As some customers in our fact table are not connected to the customer dimension table i. How to Resolve Incomplete Dimension-Fact Relationships Unrelated fact entities should be connected to the default dimensional entity. Non-Strict Dimension-Fact Relationships This problem involves double counting measure values when a many-to-many relationship between the fact table and dimensional tables is present. It usually occurs when one event i. So I will display the problematic model with two relationships, as I did earlier. But this time, one account fact has many customers attached to it. We have two scenarios; the first demands aggregation on the customer level for a set time period. The aggregated data is displayed as: The second scenario requires us to aggregate data on the account level for the same time period. The data looks like this: We can see the difference between totals of the same data range. Since customer-1 and customer-2 share the same balance 10, , the error is in the double counting of the first scenario. How to Resolve Non-Strict Dimension-Fact Relationships Resolving this issue can be harder than resolving other relationship-type problems. If there are a limited or fixed number of many-to-many relationships in the source data, we can adjust our fact and dimensional tables. If the many-to-many relationships in the source system have a variable number of connections, a more complex solution is needed. Limited Resolution In this example, we know that account ownership is shared between only two clients. We can expand our model with an additional column to represent the second owner. We also can add a new relation to cover this new customer. Unlimited Resolution A more general solution to our non-strict issue would be the construction of a new bridge table which is actually a new fact table. This would bridge all possible owners in our model. Besides storing a list of account owners, we could use our bridge table to include any information relevant to the fact-to-dimensional relationship. In our example, this would be percent of ownership. However, it undoubtedly leads to a more complicated ETL process. This is because we have to fill our dimensional model with multiple default values. I will leave you with this thought: It should guide all your modeling endeavors. Without simplicity, common problems have complex solutions. If you have found some interesting anti-pattern popping up time and again, tell us about it!

COMMON DATA MODELING PROBLEMS pdf

6: Data Modeling Problems and Constraints

Microsoft Common Data Model (CDM) is a secure business database, comprising of well-formed standard business entities that can be deployed for use in your organization. CDM is secure - it is encrypted at rest.

What is the Common Data Model? The CDM is a standardized, modular, and extensible collection of data schemas that Microsoft published to help you to build, use, and analyze data. This collection of predefined schemas consists of entities, attributes, semantic metadata, and relationships. The schemas represent commonly used concepts and activities, such as Account and Campaign, to simplify the creation, aggregation, and analysis of data. This graphic shows some elements of those schemas. The CDM simplifies data management and app development by unifying data into a known form and applying structural and semantic consistency across multiple apps and deployments. In other words, if your data is in the CDM, you can use it in many apps, streamline the creation or use of other apps to use that data, and easily build reports for each of those apps or all of them. In addition, data integrators who bring data from a variety of systems can focus on landing the data in the CDM, instead of building a different model for each app. Imagine that you have three business apps â€" one each for materials, manufacturing, and sales. Often each app would be created independently, with different structures that represent an entity, such as Account, in nearly the same way but not quite. With the CDM, you could build your data in a standardized format using the CDM entities, attributes, and relationships , and then each app could use the same data as a basis. Of course, each app could have its own additional data and schemas, based on its functionality. But when it comes to development, your apps and reports could pull the common data elements quickly, cleanly, and with confidence. And what about the need to create a fourth app? Your data is ready, in the CDM schema, so your development efforts can concentrate on business logic, not data quagmires and sticky transformations. In other words, the CDM offers these benefits: Structural and semantic consistency across applications and deployments. A unified shape where data integrations can combine existing enterprise data with other sources and use that data holistically to develop apps or derive insights. You can use the CDM to create data repositories that match the schema, and you can also transform your existing data into the CDM schema. Either way, the efficiency that you get from standardization can expedite and streamline whatever you do next with your data. Who uses the Common Data Model? A variety of customers, partners, and products use the CDM, and all have the same goal of unifying data in a well-known form with semantic meaning. These users are responsible for bringing data from a variety of systems to make it accessible for apps to use. Historically, the work to build an app has been tightly tied with data integration, but with the CDM and the platforms that support it, the two can happen independently. These examples show how organizations use the CDM: In fact, many of the original business entities in the CDM came from Dynamics offerings, such as Dynamics for Sales and Dynamics for Marketing. Industry verticals such as healthcare are working closely with Microsoft to extend the CDM to their specific business concepts, such as Patient and Care Plan. That way, users can can share data and build services so partners can easily exchange data, create interoperable apps and services, and create quick analytics that are easy to share. Next step How to use the Common Data Model:

COMMON DATA MODELING PROBLEMS pdf

7: Data model - Wikipedia

The Common Data Model (CDM) is a standardized, modular, extensible collection of data schemas published by Microsoft that are designed to make it easier for you to build, use, and analyze data.

Print this page Modeling links classroom mathematics and statistics to everyday life, work, and decision-making. Modeling is the process of choosing and using appropriate mathematics and statistics to analyze empirical situations, to understand them better, and to improve decisions. Quantities and their relationships in physical, economic, public policy, social, and everyday situations can be modeled using mathematical and statistical methods. When making mathematical models, technology is valuable for varying assumptions, exploring consequences, and comparing predictions with data. A model can be very simple, such as writing total cost as a product of unit price and number bought, or using a geometric shape to describe a physical object like a coin. Even such simple models involve making choices. It is up to us whether to model a coin as a three-dimensional cylinder, or whether a two-dimensional disk works well enough for our purposes. Other situationsâ€”modeling a delivery route, a production schedule, or a comparison of loan amortizationsâ€”need more elaborate models that use other tools from the mathematical sciences. Real-world situations are not organized and labeled for analysis; formulating tractable models, representing such models, and analyzing them is appropriately a creative process. Like every such process, this depends on acquired expertise as well as creativity. Some examples of such situations might include: Estimating how much water and food is needed for emergency relief in a devastated city of 3 million people, and how it might be distributed. Planning a table tennis tournament for 7 players at a club with 4 tables, where each player plays against each other player. Designing the layout of the stalls in a school fair so as to raise as much money as possible. Analyzing stopping distance for a car. Modeling savings account balance, bacterial colony growth, or investment growth. Engaging in critical path analysis, e. Analyzing risk in situations such as extreme sports, pandemics, and terrorism. Relating population statistics to individual predictions. In situations like these, the models devised depend on a number of factors: How precise an answer do we want or need? What aspects of the situation do we most need to understand, control, or optimize? What resources of time and tools do we have? The range of models that we can create and analyze is also constrained by the limitations of our mathematical, statistical, and technical skills, and our ability to recognize significant variables and relationships among them. Diagrams of various kinds, spreadsheets and other technology, and algebra are powerful tools for understanding and solving problems drawn from different types of real-world situations. One of the insights provided by mathematical modeling is that essentially the same mathematical or statistical structure can sometimes model seemingly different situations. Models can also shed light on the mathematical structures themselves, for example, as when a model of bacterial growth makes more vivid the explosive growth of the exponential function. The basic modeling cycle is summarized in the diagram. It involves 1 identifying variables in the situation and selecting those that represent essential features, 2 formulating a model by creating and selecting geometric, graphical, tabular, algebraic, or statistical representations that describe relationships between the variables, 3 analyzing and performing operations on these relationships to draw conclusions, 4 interpreting the results of the mathematics in terms of the original situation, 5 validating the conclusions by comparing them with the situation, and then either improving the model or, if it is acceptable, 6 reporting on the conclusions and the reasoning behind them. Choices, assumptions, and approximations are present throughout this cycle. In descriptive modeling, a model simply describes the phenomena or summarizes them in a compact form. Graphs of observations are a familiar descriptive modelâ€”for example, graphs of global temperature and atmospheric CO2 over time. Analytic modeling seeks to explain data on the basis of deeper theoretical ideas, albeit with parameters that are empirically based; for example, exponential growth of bacterial colonies until cut-off mechanisms such as pollution or starvation intervene follows from a constant reproduction rate. Functions are an important tool for analyzing such problems. Graphing utilities, spreadsheets, computer algebra systems, and dynamic geometry software are powerful tools that can be used to model purely mathematical phenomena e. Modeling Standards Modeling is

best interpreted not as a collection of isolated topics but rather in relation to other standards.

8: Microsoft's Dynamics Understanding the Common Data Model | ZDNet

Data modeling in software engineering is the process of creating a data model for an information system by applying certain formal techniques.

So, there you have it! Yeah, I actually do have a couple. Right now the only way to push data into CDM is a Flow. Keeping the Sales and Operations apps of D in sync with the customer and transaction data managed in the process of making an delivering a sale involves a fair amount of business logic. The reason CDM exists is that there will be more than one physical database in the Dynamics suite. For the business processes to work seamlessly, someone needs to keep those database closely in sync with one another. From reading through the Common Data Model tutorials , we can see that at least as of now, Flow is not the system that can handle it: Whenever an object is added to one service, it will be imported into the other system. However, that means if an object is deleted from one system it will not be deleted in the other system. Again, it may not sound like such a mission impossible to build. Sure, a collaboration solution like SharePoint has very different security concepts than a system designed for structured business records management like CRM or ERP. The thing here is that unless the solution provided by Microsoft is going to be fairly advanced, it might not be an actual solution. What limitations will this impose on customization? The one reason why many of us love the capabilities of the Dynamics XRM platform is the awesome flexibility it offers us to customize the application to meet the specific needs of customers. ERP is a good generic example of such a system to use when demonstrating the potential pitfalls in planning your integration strategy and why it can have such a big impact on the success of your CRM initiative. If the implementation of a new CRM system starts by replicating the exact customer data model of the ERP system and copying over tons of transactional data, this can severely limit the possibilities of delivering new functionality for the business in the areas where it is needed: Now, having spent a bit over a decade working with the default data model of Dynamics CRM oh dear lord, what have I done with my life?!? Perhaps even a kind of V2 of the CRM data model that would address some of the challenges often encountered when attempting to represent the non-hierarchical real world with the relationships, parties and activities of Dynamics CRM. Fair enough, we are of course looking at a solution here that aims to bridge the gap between CRM and ERP, so this approach makes a lot of sense from a practical perspective. I can also imagine myself being in the role of an ISV coming from outside the Dynamics ecosystem and investigating the logic of how these applications work, so that I could plan how to integrate my own software with these Microsoft applications. Ok, great, so I can store the name, address and Twitter handle of a physical person into 11 different entities now. Well, of course you could, but the question really is what will the customers want? The reality is that the stack off applications that organizations are deploying to support their information workers and enable fully automated business processes is growing taller every day. The trend is moving towards settling for the standard application functionality with no custom code whenever possible, in part to reduce the risk of getting burned when new releases of the product are being rolled out that may not support the unique extensions you might have developed. The Common Data Model version 1. Multiple versions of an entity will be supported at the same time, however, entity versions may be deprecated and removed over time with adequate notice and upgrade guidance. The only thing guaranteed over time is change.

COMMON DATA MODELING PROBLEMS pdf

9: Common Data Model home page (Previous Version) | Microsoft Docs

As the use of data modeling moved from an implementation tool of the physical file designers to a design tool of the systems analysts, systems designers, and more importantly the user, many of the problems inherent in the process began to become apparent.

These articles address various issues in data modeling. Atomicity and Normalization pdf file K Carver, A. A common aim of data modeling approaches is to produce schemas whose instantiations are always redundancy-free. This is especially useful when the implementation target is a relational database. This paper contrasts two very different approaches to attain a redundancy-free relational schema. The Object-Role Modeling ORM approach emphasizes capturing semantics first in terms of atomic elementary or existential fact types, followed by synthesis of fact types into relation schemes. Nonloss decomposition of a relation requires decomposition into smaller relations that, upon natural join, yield the exact original population. Nonloss decomposition of a table scheme or relation variable requires that the decomposition of all possible populations of the relation scheme is reversible in this way. Petit, Namur University Press, pp. A business domain is typically constrained by business rules. In practice, these rules often include constraints of different modalities e. Alethic rules impose necessities, which cannot, even in principle, be violated by the business. Deontic rules impose obligations, which may be violated, even though they ought not. Conceptual modeling approaches typically confine their specification of rules to alethic rules. This paper discusses one way to model deontic rules, especially those of a static nature. A formalization based on modal operators is provided, and some challenging semantic issues are examined from both logical and pragmatic perspectives. A basic implementation of the proposed approach has been prototyped in a tool that supports automated verbalization of both alethic and deontic rules. Objectification pdf file K Halpin, T. Some information modeling approaches allow instances of relationships or associations to be treated as entities in their own right. While this modeling option is rarely supported by industrial versions of Entity-Relationship Modeling ER , it is allowed in several academic versions of ER. Objectification is related to the linguistic activity of nominalization, of which two flavors may be distinguished: In practice, objectification needs to be used judiciously, as its misuse can lead to implementation anomalies, and those modeling approaches that permit objectification often provide incomplete or flawed support for it. This paper provides an in-depth analysis of objectification, shedding new light on its fundamental nature, and providing practical guidelines on using objectification to model information systems. While some information modeling approaches e. There appear to be three main reasons for requiring higher-order types: As the move to higher-order logic may add considerable complexity to the task of formalizing and implementing a modeling approach, it is worth investigating whether the same practical modeling objectives can be met while staying within a first-order framework. This paper examines some key issues involved, suggests techniques for retaining a first-order formalization, and also makes some suggestions for adopting a higher-order semantics. This article argues that this restriction should be relaxed, and replaced by a modeling guideline that allows some n-ary associations to be objectified even if their longest uniqueness constraint spans n-1 roles. The pros and cons of removing this restriction are discussed, and illustrated with examples. Join Constraints pdf file K Halpin, T. Siau, Toronto, Canada, pp. Many application domains involve constraints that, at a conceptual modeling level, apply to one or more schema paths, each of which involves one or more conceptual joins where the same conceptual object plays roles in two relationships. Popular information modeling approaches typically provide only weak support for such join constraints. Three main problems for rich support for join constraints are identified: To address these problems, some notational, metamodel, and mapping extensions are proposed. What is an elementary fact? Sharp, Utrecht, Sep , 11 pp. Database schemas are best designed by mapping from a high level, conceptual schema expressed in human-oriented concepts. While conceptual schemas are often specified using entity relationship modeling ER , a more natural and expressive formulation is often possible using Object Role Modeling ORM. This approach views the world in terms of objects playing roles, and traditionally expresses all information in terms of elementary facts, constraints and derivation rules. Although verbalization in terms

of elementary facts has many practical and theoretical advantages, it is difficult to define the notion precisely. This paper examines various awkward but practical cases which challenge the traditional definition. In so doing, it aims to clarify what elementary facts are and how they can be best expressed. Ross, Database Research Group Inc. Subtyping is an important feature of semantic approaches to conceptual schema design and, more recently, object-oriented database design. However the relational model does not directly support subtyping, and CASE tools for mapping conceptual to relational schemas typically provide only very weak support for mapping subtypes. This paper surveys some of the main issues related to conceptual specification and relational mapping of subtypes, and indicates how Object Role Modeling solves the associated problems. Although entity relationship ER modeling techniques are commonly used for information modeling, Object Role Modeling ORM techniques are becoming increasingly popular, partly because they include detailed design procedures providing guidelines for the modeler. In this paper, we propose an integration of two theoretically well founded ORM techniques: Our main focus is on a common terminological framework, and on the notion of subtyping. Subtyping has long been an important feature of semantic approaches to conceptual schema design. The subtyping issue is discussed from three different viewpoints covering syntactical, identification, and population issues. Finally, a wider comparison of approaches to subtyping is made, which encompasses other ER-based and ORM-based information modeling techniques, and highlights how formal subtype definitions facilitate a comprehensive specification of subtype constraints. Subtyping Revisited pdf file K Halpin, T. Gulla, Tapir Academic Press, pp. In information systems modeling, the business domain being modeled often exhibits subtyping aspects that can prove challenging to implement in either relational databases or object-oriented code. In practice, some of these aspects are often handled incorrectly. This paper examines a number of subtyping issues that require special attention e. However, the main ideas could be adapted for UML and ER, so these are also included in the discussion. A basic implementation of the proposed approach has been prototyped in an open-source ORM tool. Database schema transformation and optimization pdf file K Halpin, T. An application structure is best modeled first as a conceptual schema, and then mapped to an internal schema for the target DBMS. Different but equivalent conceptual schemas often map to different internal schemas, so performance may be improved by applying conceptual transformations prior to the standard mapping. This paper discusses recent advances in the theory of schema transformation and optimization within the framework of ORM Object Role Modeling. New aspects include object relativity, complex types, a high level transformation language and update distributivity. Conceptual Schemas with Abstractions: Making flat conceptual schemas more comprehensible pdf file K Campbell, L. Flat graphical, conceptual modeling techniques are widely accepted as visually effective ways in which to specify and communicate the conceptual data requirements of an information system. Conceptual schema diagrams provide modelers with a picture of the salient structures underlying the modeled universe of discourse, in a form that can readily be understood by and communicated to users, programmers and managers. When complexity and size of applications increase, however, the success of these techniques in terms of comprehensibility and communicability deteriorates rapidly. This paper proposes a method to offset this deterioration, by adding abstraction layers to flat conceptual schemas. We present an algorithm to recursively derive higher levels of abstraction from a given flat conceptual schema. The driving force of this algorithm is a hierarchy of conceptual importance among the elements of the universe of discourse. OTM Workshops, eds. This paper proposes extensions to the Object-Role Modeling approach to support schema transformations that eliminate unneeded columns that may arise from standard relational mapping procedures. This constraint is exploited to enable graphic portrayal of a new corollary to a schema transformation pattern that occurs in many business domains. An alternative transformation is introduced to optimize the same pattern, and then generalized to cater for more complex cases. The relational mapping algorithm is extended to cater for the new results, with the option of retaining the original patterns for conceptual discussion, with the transforms being applied internally in a preprocessing phase. Collection types such as sets, bags and arrays have been used as data structures in both traditional and object oriented programming. Although sets were used as record components in early database work, this practice was largely discontinued with the widespread adoption of relational databases. Object-relational and object databases once again allow database designers to

embed collections as database fields. Should collections be specified directly on the conceptual schema, as mapping annotations to the conceptual schema, or only on the logical database schema? This paper discusses the pros and cons of different approaches to modeling collections. Overall it favors the annotation approach, whereby collection types are specified as adornments to the pure conceptual schema to guide the mapping process from conceptual to lower levels. The ideas are illustrated using notations from both object-oriented Unified Modeling Language and fact-oriented Object-Role Modeling approaches. Springer LNCS , pp. This paper proposes an extension to the Object-Role Modeling approach to support formal declaration of dynamic rules. Dynamic rules differ from static rules by pertaining to properties of state transitions, rather than to the states themselves. In this paper, application of dynamic rules is restricted to so-called single-step transactions, with an old state the input of the transaction and a new state the direct result of that transaction. Such restricted rules are easier to formulate and enforce than a constraint applying historically over all possible states. In our approach, dynamic rules specify an elementary transaction type indicating which kind of object or fact is being added, deleted or updated, and optionally pre-conditions relevant to the transaction, followed by a condition stating the properties of the new state, including the relation between the new state and the old state. These dynamic rules are formulated in a syntax designed to be easily validated by non-technical domain experts. This paper provides formal semantics for an extension of the Object-Role Modeling approach that supports declaration of dynamic rules. In this paper we restrict application of dynamic rules to so-called single-step transactions, with an old state the input of the transaction and a new state the direct result of that transaction. These dynamic rules further specify an elementary transaction type by indicating which kind of object or fact being added, deleted or updated is actually allowed. Dynamic rules may declare pre-conditions relevant to the transaction, and a condition stating the properties of the new state, including the relation between the new state and the old state. In this paper we provide such dynamic rules with a formal semantics based on sorted, first-order predicate logic. The key idea to our solution is the formalization of dynamic constraints as static constraints on the database transaction history. One difficult task in information modeling is to adequately address the impact of time.

COMMON DATA MODELING PROBLEMS pdf

Revisiting workers compensation in Washington Competence Development The Berenstain bears and too much junk food Rashbams commentary on Exodus Predictors of performance and competitive stress among female dancers The Higher Calculus Mens world magazine Humanitarian intervention and medical epidemics Bruce M. Landesman Britain and the ILO Simply Beautiful Boxes Behind The Bar Room Doors A treatise on the law of circumstantial evidence, illustrated by numerous cases Himu by humayun ahmed Isocrates, Volume I Wheelchair sports classification system Corrosion of Electrical Contacts Mercedes benz w126 service manual Big data 2017 Solid waste management in brazil Almost single Wedding album design photoshop tutorials Rotary microtome parts and function Sociology in our times 6th edition The dawn of peace The Best of The Regent of Sydney New trends in employment practices Wittgenstein-Aesthetics and Transcendental Philosophy (Schriftenreihe der Wittgenstein-Gesellschaft) The Usborne Big Book of Things to Spot (1001 Things to Spot) When a Parent Marries Again Max and Moritz and Other Bad Boy Tales Puerto Rican Chicago (IL) Dark Sweat, White Gold Job application for applebees The power of enterprise-wide project management Metacode and cultural code. World of animals annual Chemistry form 4 chapter 2 Oxford c.1571 : Rumors Part III. Design principles for urban villages Paul Brewster and son