DOWNLOAD PDF LOGICAL DATA MODEL EXAMPLE

Chapter 1 : Logical data model - Wikipedia

A logical data model or logical schema is a data model of a specific problem domain expressed independently of a particular database management product or storage technology (physical data model) but in terms of data structures such as relational tables and columns, object-oriented classes, or XML tags.

Primary UID unique identifier. Numeric Id M mandatory, that is, must not be null. Book Id M mandatory, that is, must not be null. Date and time of the transaction. Numeric code indicating the type of transaction, such as 1 for checking out a book. Click OK to finish creating the Transactions entity. The following relationships exist between the entities: Each book can be involved in multiple sequential transactions. Each book can have zero or one active checkout transactions; a book that is checked out cannot be checked out again until after it has been returned. Each patron can be involved in multiple sequential and simultaneous transactions. Each patron can check out one or many books in a visit to the library, and can have multiple active checkout transactions reflecting several visits; each patron can also return checked out books at any time. Create the relationships as follows. When you are done, the logical model pane in the main area should look like the following figure using Bachman notation, which you can change to Barker by clicking View, then Logical Diagram Notation, then Barker Notation: In the logical model pane in the main area, arrange the entity boxes as follows: Books on the left, Patrons on the right, and Transactions either between Books and Patrons or under them and in the middle. If the pointer is still cross-hairs, click the Select icon at the top left to change the pointer to an arrow. Turn off auto line routing for this exercise: Click the New 1: Click first in the Books box, then in the Transactions box. A line with an arrowhead is drawn from Books to Transactions. Click first in the Patrons box, then in the Transactions box. A line with an arrowhead is drawn from Patrons to Transactions. Optionally, double-click a line or right-click a line and select Properties and view the Relation Properties information. Go to Section 2. In the simplified data model for this tutorial, a single relational model reflects the entire logical model; however, for other data models you can create one or more relational models, each reflecting all or a subset of the logical model. To have a relational model reflect a subset of the logical model, use the "filter" feature in the dialog box for engineering a relational model. Develop the relational model as follows: With the logical model selected, click Design, then Engineer to Relational Model. The Engineering dialog box is displayed. Accept all defaults do not filter , and click Engineer. The DDL statements will implement the physical model type of database, such as Oracle Database 11g that you specify. Develop the physical model as follows: Optionally, view the physical model before you generate DDL statements: A dialog box is displayed for selecting the type of database for which to create the physical model. Specify the type of database for example, Oracle Database 11g , and click OK. In the hierarchy display on the left side of the window, a Physical Models node is added under the Library relational model node, and a physical model reflecting the type of database is created under the Physical Models node. Expand the Physical Models node under Library the relational model , and expand the newly created physical model and nodes under it that contain any entries such as Tables and Columns , to view the objects created. Select the database type for example, Oracle Database 11g and click Generate. Accept all defaults, and click OK. Although you can edit statements in this window, do not edit any statements for this tutorial exercise. Click Save to save the statements to a. Click Close to close the DDL file editor. A directory or folder structure will also be created automatically to hold the detailed information about the design, as explained in Section 1. Continue creating and modifying design objects, if you wish.

Chapter 2 : Logical Versus Physical Database Modeling - calendrierdelascience.com

A logical data model describes the data in as much detail as possible, without regard to how they will be physical implemented in the database. Features of a logical data model include: Includes all entities and relationships among them.

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:

DOWNLOAD PDF LOGICAL DATA MODEL EXAMPLE

Chapter 3 : Conceptual, logical and Physical data model - Visual Paradigm

Logical Data Modeling: This is the actual implementation and extension of a conceptual data model. A Logical data model is the version of a data model that represents the business requirements (entire or part) of an organization and is developed before the physical data model.

Data Dictionary is a description of data structures read about two basic types and you can distinguish logical and physical Data Dictionaries. Data Dictionary and Data Model The difference between logical and physical Data Dictionaries is the same as between logical and physical data model: Logical data model is created at the requirements gathering, system analysis and top level design. It is a communication and specification tool for business analysts and business. Physical data model is created when you translate top level design into physical tables in the database. This model is slightly different due to the fact that you have to worry about many details. The same goes for Data Dictionary â€" logical one describes elements of logical data model and physical documents actual tables and columns in your database schema. Different audience and users Logical Data Dictionary is used by business analysts or business users. Physical Data Dictionary is used by database administrators, software developers and database architects. Different authors Not only audience is different, it is also created by different people. Logical Data Dictionary is created by business analysts. Physical Data Dictionary is created and maintaied by database architects and database administrators. Relation to applications and databases Logical Data Dictionary are system agnostic while Physical Data Dictionary represent data in a specific database. Different purpose Logical Data Dictionary is used while business analysis and top level design as communication tool. Physical Data Dictionary serves as a documentation of databases and systems for technical users. Different development stage Logical Data Dictionary is usualy created at the beginning of the system design - top level design, data modeling and gathering of requirements. It is then reviewed at launching and validation. Physical Data Dictionary is created while database desing and implementation and should be maintained thougout the whole system lifecycle, as database schema evolves. Different scope Physical Data Dictionary covers one database or one schema, while Logical Data Dictionary covers one business domain or even entire organization. Therefore, there should be one Logical Data Dictionary covering one domain, but there may be more than one Physical Data Dictionaries since the same data may exist in more applications. Different level of details Logical Data Dictionary focuses on key data elements entities and fields while Physical Data Dictionary describes all tables and columns. A few examples Here are some differences in those two models and dictionaries: Logical model uses user friendly names while physical has some limitations and often uses different notation eg.

Chapter 4 : News, Tips, and Advice for Technology Professionals - TechRepublic

A logical data model, as the name suggests, represents the logical relationships between the data of any organization. The data is placed in an entity. Each entity has various attributes, and each attribute is housed in an individual field.

Collapse the table of content Expand the table of content This documentation is archived and is not being maintained. This documentation is archived and is not being maintained. Logical Model The logical model defines the business entities that will be rendered by the application and what policies and rules will be used to operate on those entities. The questions this model answers are very specific to the business for which you are writing your application. When should a customer be given a purchase discount? When should inventory be restocked? How much sales tax should be charged to an international order? The logical model itself is made up of two relatively independent sub-models, the logical data model and the logical object model. The logical data model is responsible for documenting the business entities that the system manages such as products, customers, and orders and the rules for maintaining them for example, "all new orders must be represented in the Orders table as well as the Shipping table". You also use the logical model to specify how each object fits into the three-tier logical services model. How the logical model interacts with other models As the diagram in the Enterprise Application Model shows, the user model directly interacts with the business model, the logical model, the technology model, and the physical model. The following table characterizes these interactions and gives brief examples of each. Sub-model How the logical model relates to it Example Business model Determines how business policies can be encapsulated. An application that must accommodate rapidly changing tax rates and rules must isolate these rules in separate, easily maintainable components. User model Defines the logical entities the user acts on to perform various business tasks. Logical customer and order objects and credit checking services are needed to enable the user to carry out typical sales tasks in the application. A credit checking component must use technology capable of interfacing with credit reporting agencies. Physical model Determines the features and services that must be deployable using the chosen physical architecture. A funds transfer component must be deployed on an infrastructure that supports reliable security protocols. Since the development model permeates all of the Enterprise sub-models, there are no "typical" interactions because you must account for every design and implementation decision in the development model. How the Internet affects the logical model Examples of the way the Internet has affected the logical model include: The three-tier model of separating business logic from user interface behavior has gained great momentum as businesses strive to encode business functionality onto powerful Internet application servers. The competitive environment fostered by the Internet is also driving an aggressive push by corporations and third party software vendors to implement their new functionality with objects that can be more easily maintained and reused, improving businesses responsiveness. You should note that for the most part, the Internet is a physical phenomenon and will not have a significant impact on the logical model of those businesses that have already moved to object oriented LAN-based development.

Chapter 5 : What a Concept! Is Logical Data Modeling Obsolete?

Conceptual, logical and physical model or ERD are three different ways of modeling data in a domain. While they all contain entities and relationships, they differ in the purposes they are created for and audiences they are meant to target.

What is Data Modeling? Data modeling is the act of exploring data-oriented structures. Like other modeling artifacts data models can be used for a variety of purposes, from high-level conceptual models to physical data models. From the point of view of an object-oriented developer data modeling is conceptually similar to class modeling. With data modeling you identify entity types whereas with class modeling you identify classes. Data attributes are assigned to entity types just as you would assign attributes and operations to classes. There are associations between entities, similar to the associations between classes â€" relationships, inheritance, composition, and aggregation are all applicable concepts in data modeling. Traditional data modeling is different from class modeling because it focuses solely on data â€" class models allow you to explore both the behavior and data aspects of your domain, with a data model you can only explore data issues. However, some people will model database methods stored procedures, stored functions, and triggers when they are physical data modeling. It depends on the situation of course, but I personally think that this is a good idea and promote the concept in my UML data modeling profile more on this later. In fact, my experience is that CRC cards are superior to ORM diagrams because it is very easy to get project stakeholders actively involved in the creation of the model. Instead of a traditional, analyst -led drawing session you can instead facilitate stakeholders through the creation of CRC cards. Although methodology issues are covered later , we need to discuss how data models can be used in practice to better understand them. You are likely to see three basic styles of data model: These models, sometimes called domain models, are typically used to explore domain concepts with project stakeholders. On Agile teams high-level conceptual models are often created as part of your initial requirements envisioning efforts as they are used to explore the high-level static business structures and concepts. Logical data models LDMs. LDMs are used to explore the domain concepts, and their relationships, of your problem domain. This could be done for the scope of a single project or for your entire enterprise. LDMs depict the logical entity types, typically referred to simply as entity types, the data attributes describing those entities, and the relationships between the entities. LDMs are rarely used on Agile projects although often are on traditional projects where they rarely seem to add much value in practice. Physical data models PDMs. PDMs are used to design the internal schema of a database, depicting the data tables, the data columns of those tables, and the relationships between the tables. PDMs often prove to be useful on both Agile and traditional projects and as a result the focus of this article is on physical modeling. Although LDMs and PDMs sound very similar, and they in fact are, the level of detail that they model can be significantly different. This is because the goals for each diagram is different â€" you can use an LDM to explore domain concepts with your stakeholders and the PDM to define your database design. Figure 1 presents a simple LDM and Figure 2 a simple PDM, both modeling the concept of customers and addresses as well as the relationship between them. Both diagrams apply the Barker notation , summarized below. Notice how the PDM shows greater detail, including an associative table required to implement the association as well as the keys needed to maintain the relationships. More on these concepts later. A PDM should also indicate the data types for the columns, such as integer and char 5. A simple logical data model. A simple physical data model. Data models can be used effectively at both the enterprise level and on projects. Enterprise architects will often create one or more high-level LDMs that depict the data structures that support your enterprise, models typically referred to as enterprise data models or enterprise information models. Enterprise data models provide information that a project team can use both as a set of constraints as well as important insights into the structure of their system. Project teams will typically create LDMs as a primary analysis artifact when their implementation environment is predominantly procedural in nature, for example they are using structured COBOL as an implementation language. LDMs are also a good choice when a project is data-oriented in nature, perhaps a data warehouse or reporting system is being developed having said that, experience seems to show that

usage-centered approaches appear to work even better. However LDMs are often a poor choice when a project team is using object-oriented or component-based technologies because the developers would rather work with UML diagrams or when the project is not data-oriented in nature. As Agile Modeling advises, apply the right artifact s for the job. Or, as your grandfather likely advised you, use the right tool for the job. When a relational database is used for data storage project teams are best advised to create a PDMs to model its internal schema. My experience is that a PDM is often one of the critical design artifacts for business application development projects. What About Conceptual Models? The advantage is that the notation is very simple, something your project stakeholders can quickly grasp, although the disadvantage is that the models become large very quickly. ORMs enable you to first explore actual data examples instead of simply jumping to a potentially incorrect abstraction â€" for example Figure 3 examines the relationship between customers and addresses in detail. For more information about ORM, visit www. A simple Object-Role Model. My experience is that people will capture information in the best place that they know. I sometimes user ORMs to explore the domain with project stakeholders but later replace them with a more traditional artifact such as an LDM, a class diagram, or even a PDM. Not only are they tempted to keep the artifacts that they create but also to invest even more time to enhance the artifacts. Generalizing specialists are more likely than specialists to travel light. Common Data Modeling Notations Figure 4 presents a summary of the syntax of four common data modeling notations: Comparing the syntax of common data modeling notations. Discussing common data modeling notations.

Chapter 6 : Logical Data Model Concepts

This feature is not available right now. Please try again later.

Like all good data architects, I want to define the terms I use on this blog, speaking engagements, and on my projects. My friend Graeme Simsion has even done research in this naming conflict. My uses of conceptual, logical, and physical come from the Information Engineering IE methods of data modeling. Other uses and definitions arise from the database schema and academic world. The industry as a whole tends to use the IE definitions, so I tend to stick to them because they are used by the vast majority of practitioner data modelers and other team members. Conceptual Data Model A conceptual data model is a summary-level data model that is most often used on strategic data projects. It typically describes an entire enterprise. Due to its highly abstract nature, it may be referred to as a conceptual model. Common characteristics of a conceptual data model: Enterprise-wide coverage of the business concepts. Designed and developed primarily for a business audience Contains around entities or concepts with no or extremely limited number of attributes described. Sometimes architects try to limit it to printing on one page. Contains relationships between entities, but may or may not include cardinality and nullability. Entities will have definitions. Designed and developed to be independent of DBMS, data storage locations or technologies. In fact, it would address digital and non-digital concepts. This means it would model paper records and artifacts as well as database artifacts. Logical Data Model A logical data model is a fully-attributed data model that is independent of DBMS, technology, data storage or organizational constraints. It typically describes data requirements from the business point of view. While common data modeling techniques use a relational model notation, there is no requirement that resulting data implementations must be created using relational technologies. Common characteristics of a logical data model: Typically describes data requirements for a single project or major subject area. May be integrated with other logical data models via a repository of shared entities Typically contains entities, although these numbers are highly variable depending on the scope of the data model. Contains relationships between entities that address cardinality and nullability optionality of the relationships. In fact, it may address digital and non-digital concepts. Data attributes will typically have datatypes with precisions and lengths assigned. Data attributes will have nullability optionality assigned. Entities and attributes will have definitions. In fact, the diagram of a logical data model may show only a tiny percentage of the meta data contained within the model. A logical data model will normally be derived from and or linked back to objects in a conceptual data model. Physical Data Model A physical data model is a fully-attributed data model that is dependent upon a specific version of a data persistence technology. Common characteristics of a physical data model: Typically describes data requirements for a single project or application. Sometimes even a portion of an application. May be integrated with other physical data models via a repository of shared entities Typically contains tables, although these numbers are highly variable depending on the scope of the data model. Contains relationships between tables that address cardinality and nullability optionality of the relationships. Designed and developed to be dependent on a specific version of a DBMS, data storage location or technology. Columns will have nullability optionality assigned. Tables and columns will have definitions. Will also include other physical objects such as views, primary key constraints, foreign key constraints, indexes, security roles, store procedures, XML extensions, file stores, etc. The diagram of a physical data model may show only a tiny percentage of the meta data contained within the model.

Chapter 7 : Logical Data Model [Enterprise Architect User Guide]

Unfortunately, most modeling tools cannot even draw logical data model. Instead, they use UML class notation and class attributes to represent logical data model and that is the main reason why is it possible to specify data-types in such modeling tools.

Print Article Logical Modeling Logical modeling deals with gathering business requirements and converting those requirements into a model. The logical model revolves around the needs of the business, not the database, although the needs of the business are used to establish the needs of the database. Logical modeling involves gathering information about business processes, business entities categories of data , and organizational units. After this information is gathered, diagrams and reports are produced including entity relationship diagrams, business process diagrams, and eventually process flow diagrams. The diagrams produced should show the processes and data that exists, as well as the relationships between business processes and data. Logical modeling should accurately render a visual representation of the activities and data relevant to a particular business. Note Logical modeling affects not only the direction of database design, but also indirectly affects the performance and administration of an implemented database. When time is invested performing logical modeling, more options become available for planning the design of the physical database. The diagrams and documentation generated during logical modeling is used to determine whether the requirements of the business have been completely gathered. Management, developers, and end users alike review these diagrams and documentation to determine if more work is required before physical modeling commences. The point of the initial ERD is to provide the development team with a picture of the different categories of data for the business, as well as how these categories of data are related to one another. Business process diagrams The process model illustrates all the parent and child processes that are performed by individuals within a company. The process model gives the development team an idea of how data moves within the organization. Because process models illustrate the activities of individuals in the company, the process model can be used to determine how a database application interface is design. User feedback documentation Physical Modeling Physical modeling involves the actual design of a database according to the requirements that were established during logical modeling. Logical modeling mainly involves gathering the requirements of the business, with the latter part of logical modeling directed toward the goals and requirements of the database. Physical modeling deals with the conversion of the logical, or business model, into a relational database model. When physical modeling occurs, objects are being defined at the schema level. A schema is a group of related objects in a database. A database design effort is normally associated with one schema. During physical modeling, objects such as tables and columns are created based on entities and attributes that were defined during logical modeling. Constraints are also defined, including primary keys, foreign keys, other unique keys, and check constraints. Views can be created from database tables to summarize data or to simply provide the user with another perspective of certain data. Other objects such as indexes and snapshots can also be defined during physical modeling. Physical modeling is when all the pieces come together to complete the process of defining a database for a business. Physical modeling is database software specific, meaning that the objects defined during physical modeling can vary depending on the relational database software being used. For example, most relational database systems have variations with the way data types are represented and the way data is stored, although basic data types are conceptually the same among different implementations. Additionally, some database systems have objects that are not available in other database systems. Implementation of the Physical Model The implementation of the physical model is dependent on the hardware and software being used by the company. The hardware can determine what type of software can be used because software is normally developed according to common hardware and operating system platforms. Some database software might only be available for Windows NT systems, whereas other software products such as Oracle are available on a wider range of operating system platforms, such as UNIX. The available hardware is also important during the implementation of the physical model because data is physically distributed onto one or more physical disk drives. Normally, the more

physical drives available, the better the performance of the database after the implementation. Some software products now are Java-based and can run on virtually any platform. Typically, the decisions to use particular hardware, operating system platforms, and database software are made in conjunction with one another. Typical deliverables of physical modeling include the following: Server model diagrams The server model diagram shows tables, columns, and relationships within a database. User feedback documentation Database design documentation Conclusion Understanding the difference between logical and physical modeling will help you build better organized and more effective database systems. This article described both of these models. About the Authors Ryan K. Stephens and Ronald R. Both have numerous years of experience training and consulting at the DBA level.

Chapter 8 : Logical Data Model

The scope and complexity of a logical data model depends on the requirements of the reporting needs of the user community and the availability of source data. The more sophisticated and complex the reporting requirements and source data, the more complex the logical data model becomes.

This is because the broad retail framework is well established and it is in a stage of development where it is being extended and enhanced not created from scratch. A Logical Data Model data model is composed of: Each instance of these object types are uniquely identified and defined in business terms. The definitions supply the semantic content for a data model. Logical Data Model Components Entity Types An entity type is a representation of a person, place, thing, event or concept of interest to a retailer. Within the ARTS data model each entity type is defined in business terms. In an entity diagram, entity types are represented as rectangles. Each entity type has a unique, singular noun phrase assigned as its name. In a relational data model, each entity type instance is uniquely identified by a primary key. A primary key is one or more attributes that have values used to uniquely identify and distinguish each entity type instance from each other. Attributes An attribute identifies, names and defines a characteristic or property of an entity type. For example an Item entity type will have an ItemID attribute to unique identify it. It will have a Name attribute to use in catalogs and labels. It will have a Description attribute, etc. Attributes are the most atomic parts of a data model. They cannot be decomposed into lower level components. In a relational data model, an attribute cannot exist independently from an entity type. Accordingly all attributes are always identified and shown as part of entity types. As discussed under entity type, a primary key is composed of one or more attributes and serves as a unique entity type instance identifier. Attributes used to compose a primary key are listed above a horizontal line in the entity rectangle. Figure 95 - Sample Item Entity Relationships A relationship identifies, names and defines an association between two entity types. A relationship always associates two and only two entity types. Relationship names are represented as verb phrases. A relationship verb phrases may be established for both directions of a relation between two entities. The entity types associated through a relationship fulfill two roles: One entity is a parent entity. The second entity is a child entity. The parent entity shares it identity with the child entity. The child entity inherits the primary key of the parent entity type and is referred to as a dependent entity type. The attribute shared from a parent to a child entity type is called a foreign key. In an entity diagram an attribute name that is a foreign key is designated with a " FK " suffix. In a relational data model there are two ways a parent and child entity type may be related. The first way is an identifying relationship. If the parent entity type is deleted in this scenario, this scenario, the child is deleted. Conversely a child entity type cannot be inserted until the parent it references is inserted. In an entity diagram, an identifying relationship is signified by a solid line between the parent and child entity types. The second way parent and child entity types may be related is through a non-identifying relationship. In a non-identifying relationship, the parent entity primary key is inherited by the child entity as a non-primary key attribute. This means that the child entity references its parent entity type but is not dependent on the parents existence for its own existence. From a relational database point of view this means that the inherited attribute may be null -- that is point to nothing. In an entity diagram, a non-identifying relationship is signified by a dashed line between the parent and child entity types. Relationships incorporate an additional property between parent and child entity types called cardinality. Cardinality expresses the count of child entity type instances that may exist for a parent entity type. For example a Brand may apply to zero, one or many Item entity type instances. Conversely, an Item may be referred to by zero or one Brand. These cardinalities are illustrated in the diagram below. There are a variety of ways cardinality is used to express the relative counts of parent entity types to children entity types and they are presented in the Data Model Methodology and Notation Topic. It also illustrates how foreign keys and cardinality are presented in an entity diagram. Figure 96 - Sample Entity, Attribute and Simple Relationship In addition to cardinality, there is a special type of relationship called a subtype that allows several child entity types to inherit a common parent entity type characteristics. This is illustrated in the next diagram. A retail transaction definition is shown in the yellow block. As modeled here, a

RetailTransaction may have zero, one or many RetailTransactionLineItem entity instances associated with it. The RetailTransactionLineItem entities are dependent entities because a line item cannot exist without a retail transaction header. A retail transaction line item provides a set of attributes including line item number that all subtypes share i. A RetailTransactionLineItem entity type instance may be one and only one subtype. This relational concept of subtype is analogous to the inheritance used to model classes and objects in object oriented design. For this example subtype child entity types efficiently represent a retail transaction and the different kinds of line items needed to capture item, discount, tax and tender data. The sample receipt shows how each subtype of RetailTransactionLineItem reflects different sales receipt line items. Figure 97 - Entity Subtype Relationship Example Domains A domain is a named type of data representation that may apply to one or more attributes. Data representation defines a data type such as integer, string, floating point, date, time or other standard data type or an extended definition that assigns custom properties and constraints to a base data type. Domains enable retail-specific data types to be derived from SQL base data types. The creation of domains can also be used to define constraints that values assigned to an attribute assigned to a domain. Data Model Semantics Semantics is the branch of linguistics and logic concerned with meaning. Logical models, in addition to identifying entities, attributes, relationships and domains define what each instance of these object means. These definitions provide the semantic content of a data model are are essential to the business relevancy of a relational model. The diagram below illustrates the assignment of a definition to the ItemID attribute of Item. Definitions should be expressed in business terms and reflect the business concepts represented by a data model entity, attribute, relationship, domain and other model objects. Data models are not just for information technologists. Information as A Currency and Asset Retailing in the 21st century is as much about managing information as it is about managing cash, merchandise, customers, stores, vendors and other "real world" business assets. Most retail decision makers rely on information to make decisions because they can not personally visit and observe every retail site personally. To be useful, information has to be identified, named, described and organized into a coherent structure so it can be understood by decision makers. Data modeling provides a formal set of tools and procedures to make information useful. The formality and discipline introduced by data modeling is vital in figuring out what retail reports actually are telling decision makers. Consider the terms item, article, product, SKU and merchandise. They each mean different things to different people. The data model by defining each entity type clarifies what each term means. Where some are used as synonyms, they are explicitly referenced as such. This is called a controlled vocabulary and it is a key value-adding feature of data modeling. It establishes a common language for retailer organizations and individuals to communicate using explicitly defined words. Costs of Misinterpretation and Inconsistent Semantics Retailers manage a complex set of interactions between customers, vendors, tax authorities, regulators, employees and a wide range of other kinds of parties. Retailers that do not have a single standard way to identify, name, define and describe items, tender types, tax rules, promotions, vendor deals and the like will encounter transaction processing errors that will cost real money. Data accuracy has a direct, unambiguous impact on the bottom line. If an ordered item is not correctly aligned with the vendors catalog product code and the order is placed some party the customer, retailer, vendor, etc. The data model particularly a third normal form relational model reduces this risk by insisting on a consistent representation of each data element in a single place in the data model. This same issue comes up when developing reports. Retailers without a consistent way of identifying, naming and defining entities, attributes and relationships spend a lot of time and money trying to reconcile conflicting summary reports. In some companies middle and senior managers spend an inordinate amount of time manually reconciling inconsistently defined data. Data models that establish an enterprise-wide controlled vocabulary eliminate one of the root causes of data inconsistency. Data Model as a Reflection of Business Assumptions, Constraints and Rules Data models reflect important retail business assumptions and constraints. For example, the relationship between taxation, merchandise and retailer provided services is explicitly represented in the way items, taxes, tax authorities, retail transactions, inventory control documents, etc. The rules governing the way point of sale discounts are treated by a retailer are likewise reflected in the way price modification rules are related to retail transaction sale return items and promotions. The complex web of relationships that define retailer business rules is explicitly presented

through entity relationship models. The Sarbanes Oxley Act of mandates detailed reporting and tracking of business operational and financial controls. Data for compliance is generated by corporate systems for financials, sales, marketing, inventory, purchasing, and related operational systems. Depending on the size and complexity of the organization, these systems are part of a set of applications whose data must be turned into information, then distilled into knowledge on reports for senior executives. The operational and decision support systems are based on data being collected and stored in data structures such as tables and files, and then transformed and moved â€" to become knowledge in reports that are signed by corporate principals. The business and technical meta data for these systems data constitute an information architecture that can both guide the development of internal controls and give corporate principals the confidence that the reports they sign are valid. The data model concepts discussed here provide the kind of support required to support regulatory reporting compliance. Additional support for data movement and transformation is also required.

Chapter 9 : Data model - Wikipedia

In the simplified data model for this tutorial, a single relational model reflects the entire logical model; however, for other data models you can create one or more relational models, each reflecting all or a subset of the logical model.

Add the following columns into the Order entity: In the Select Parent Model, click on the project root node first. You should see a new ERD formed, which looks quite similar to the logical ERD, except that the entities are in orange and with a primary key column automatically created. First, one Customer can produce multiple Order. Click OK when you are prompted for the creation of foreign key column. Select it and press Delete. Order is a reserved word that cannot be used as table name. Doing so allows us to know the differences between the logical model and physical model. To open it, click on Switch Diagram on the navigation bar and then double click on the logical ERD to open it. Settings for controlling how and what to compare Left hand side: A list of diagrams in the two projects selected on left and right hand side Middle: A pane that has two sides. Each side represents a project and one of its diagram. Comparison is made for the two sides. Difference of the two diagrams are shown here. At the top left corner of the Visual Diff window, select Transitor to be the comparison strategy. There are three types of strategies. Shapes will be matched base on their internal model element ID. This way of comparison is useful when visualizing differences for different stages of design. Shapes will be matched base on their names. This way of comparison is useful when visualizing differences for external works. Typical examples are to compare databases and class models. Shapes will be matches base on their transition established by Model Transitor. This way of comparison is useful when visualizing differences for different Models. Next to the Strategy setting, there is a drop down menu for selecting the things to compare. For View, differences like the coordinate of shape will be reported. For Model Element, differences such as the name of model element or other specification-level changes will be reported. As we are interested in knowing only the differences at schema level, select Model Element. Differences between logical and physical ERD can then be found easily. ID column has been added in physical ERD. The one-to-many relationship has been added. Column order count has been deleted.