Universal E-Catalog pattern

Hesham Saadawi

Carleton University

School of Computer Science
E-mail: hsaadawi@connect.carleton.ca

Abstract

This paper introduces a data modeling design pattern that can help organize and persist information of a catalog in any RDBMS. The catalog would not be bound to a specific business context and does not need any code maintenance to be re-deployed in different business contexts. Hence this pattern describes a universal electronic catalog. The catalog supports unlimited number of categories and their attributes, and facilitates easy searches and comparisons of stored items.

Introduction

In E-Commerce systems, like online shops, there is a need to present a variety of products to online customers. These products can be unrelated like food and books, or related in a hierarchical classification structure. The product information needs to be organized in a way that enables the customer to do searches, matches, and comparisons between different products based on some common product attributes. The common solution for this is to represent all products in a store in a catalog, either in a print format, or in case of online store, an electronic catalog.

To simplify the catalog building process and customer browsing of a catalog, products are often classified into categories where similar products that share common attributes are in one broad category. An example would be footwear, where each product in this category shares some common attributes like size, material, color, gender (male, female, or kids), and then can have more specialized sub-categories like sports footwear. Another example would be an appliances category where all products share common attributes as an operating voltage, color, energy consumption, dimensions (height, width and length), and further have some special sub-categories like refrigerators.

An online store would be interested with categorizing its products to be able to add new products efficiently, as it would only need to add the new product to its sub-category and specify values for its attributes that were defined in the product sub-category and all the parent categories. Therefore, re-using previously defined attributes for a product category.

A typical customer would be interested with selecting a product based on certain attributes, like selecting a stainless steel refrigerator, and then to compare on other attributes like getting the most efficient refrigerator for its capacity group.

Hence, for electronic catalog systems, the requirement is often to internally represent and then persist information about different products or items. Each of these products would have a certain number of attributes that describe it. These attributes are important to enable product searches, comparisons and product classification.

Another context for using a catalog may be a scientific research agency that is building a catalog with different plantation types or species found in certain geography. In this example, the classification of plants in a hierarchy of categories occurs naturally and each of these categories would need to have its

attributes defined. Categories in lower levels would inherit the attributes of their parents, and add more attributes to it.

In those catalog applications however, it should be noted that there is a distinction between the number of product types, i.e. product categories and sub-categories, and the number of concrete products stored in the catalog. This is important as per a typical catalog; we could have hundreds of product types. Each of these types may contain several identical products. A group of identical products needs only to be represented in a catalog once with a quantity attribute that indicates how many of this product is available in stock. This is different from a banking system for example, where it has few bank account types (which are similar to different product categories in a catalog), but those types do not change often and each account type contains millions of different customer accounts. The latter is an example of a high volume database system with few varieties of product types. This distinction would become important later when discussing advantages and consequences of using the e-catalog pattern.

A universal e-catalog pattern is introduced here using the canonical pattern format.

Pattern Name.

Universal E-Catalog

Problem.

An electronic catalog can be defined as an electronic repository of information about items, products, or species. This makes it a general database storage structure that can be used in many applications like storing inventory items, manufactured products and components, or a catalog of some living organisms.

In such a general electronic catalog, we need to dynamically, i.e. at runtime, define item categories and their attributes. This would enable many different application contexts to define their own categories and for each category, its list of attributes.

Normally, categories of products in an inventory, or of some living species would follow a simple hierarchical structure in which a *parent* category may have one or more *child* categories.

The representation of the e-catalog would need to be persisted in a database system in order to be useful. The challenge here is that a store cannot anticipate all the products it would have during its lifetime. Even if they do, it is a waste of storage space to create and populate database tables for products that may come after years if they come at all. The same is true for a species catalog. Further, creating all products, or species categories as concrete tables in the database would make the e-catalog inflexible when used for other types of applications, hence it won't be a universal e-catalog.

This presents a challenge to the Data Model designer as these items and products would need to be persisted in a database, typically using a relational database management system as these currently are the most available and commercially used database systems.

Context

- You are building an e-catalog for a large variety of products or items that need to be stored, searched and compared with each other.
- Different products or items have different attributes.
- You need to add, remove, or modify product attributes at runtime. New products and items would need to be defined with their attributes

whenever they become available. Moreover, there may be a need to add or remove some existing product attributes depending on the market.

- You do not want software system maintenance whenever there is a need to add, remove or change products.
- You need to reuse the catalog in other business contexts without the need to change the code or the database structures.
- You have a large variety of *product types* that may keep growing.

Forces.

- An e-catalog needs to store a wide variety of product categories, to cover all possible products of a business during its lifetime. Many of these may not be known at design time.
- To design a data model for the e-catalog, all information about categories and products would need to be identified at design time
- Changing structure or re-coding the e-catalog, as a software application, is expensive and may introduce new bugs: Thus, this should be avoided as much as possible.
- Different contexts or subject areas would need to store different types of products and items: These are usually unrelated like foods, books and furniture. The e-catalog application should work well within almost any context, i.e. to be able to capture product attributes, store them and display different unrelated products on demand.
- E-Commerce systems need a high availability e-catalog: Any application software accessing RDBMS tables must have a priori information on table structure and data types stored in them. Modifying one or more of these database tables, for example to define a new product type, would usually require modification of code accessing these tables. Therefore, it is not possible to change table structure or add new tables in the catalog

database at application runtime. This must be done off-line and the application code would need to be retrofitted and redeployed.

- Product attributes data need to be stored in atomic form: In order to support accurate product searches and comparisons, these operations need to be done on atomic data values stored in columns. Storing all product information in one large text field would not enable extraction of this information when needed. For example, a customer wants to get a list of refrigerators between the sizes of 18c.f. and 22c.f, with best energy efficiency. This means that we need to base our search on "*Size"* and "*Energy Consumption"* attributes of the *Refrigerator* category. If this information were embedded in a description text field with other properties, it would be a difficult task to extract them for the purpose of this search.

Solution

- **Organize your product types (categories) in a hierarchy of parent-child relationships (use the inheritance pattern):** Each parent would contain common attributes that are common to all of its children, i.e. a child would inherit the attributes of its parent. Choose this scheme to represent the product or item categories in your catalog. Try to capture common attributes in the hierarchy top. For example, a common attribute for all products in a store would be price, quantity, manufacturer, discounts offered, and product name. This scheme would allow new categories to be defined with minimal effort, as they would inherit attributes from existing ones.
- **Allow the administrator system user to define a new category when needed:** Define the category name, its parent category, its attribute names and their types.
- **Store user defined categories in memory in a flexible data structure**: for example, in a Java HashTable.

- **Create database tables and structure**: As shown in Figure 1 to persist the product information. This database structure is represented in a traditional entity relationship diagram. More details about the notation used in this diagram and a reference is given in Appendix 2.

Structure

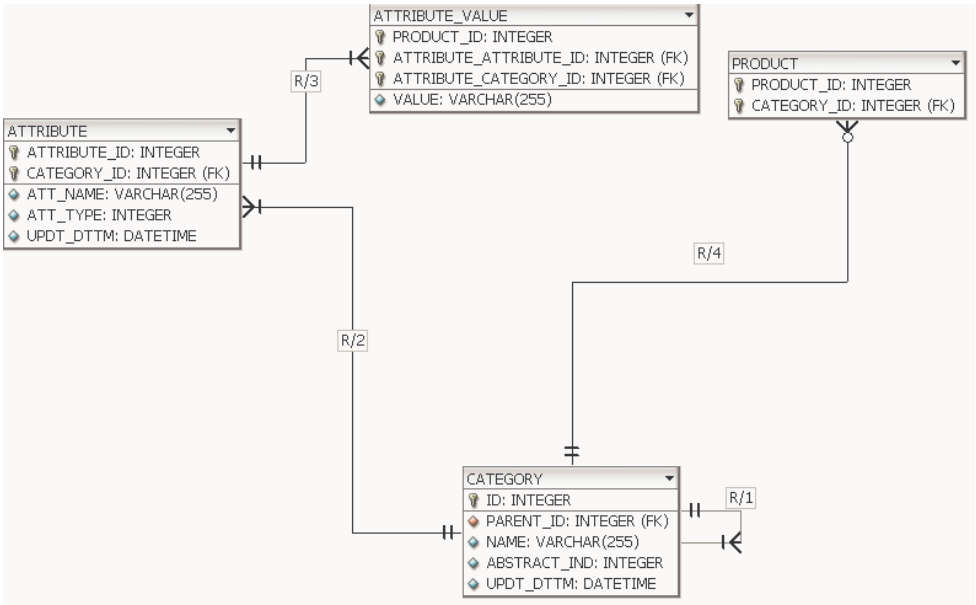

Figure 1: Catalog Data Model

In the model shown in Figure 1:

- CATEGORY: Relational table that stores information about catalog categories. Each row in this table would represent a category. The ID column stores unique id for this category and it is the primary key for the table. PARENT_ID contains the parent category id. This is also a foreign key referencing the ID column, as any parent category would be a category itself. This column would contain a NULL value for the category at the hierarchy top, as it has no parent. NAME contains the name of the category. ABSTRACT_IND is a flag to be set if this category should have

no concrete products in it. UPDT_DTTM contains the timestamp that this row has been last updated. This information would be used to apply an optimistic concurrency control pattern. This table is related with relation R1 to itself as a category may have multiple child categories.

- ATTRIBUTE table: contains the attributes of a category. Each row represents an attribute. The ATTRIBUTE_ID is a unique id for the attribute, and it belongs to the category defined by CATEGORY_ID. ATT_NAME contains the name of the attribute. ATT_TYPE contains the attribute type.
- ATTRIBUTE_VALUE table: contains values of those attributes defined in ATTRIBUTE table. This value is stored in VALUE column. This table would store attribute values ATTRIBUTE_ATTRIBUTE_ID values associated with concrete products defined by PRODUCT_ID that belongs to a category identified by ATTRIBUTE_CATEGORY_ID.
- PRODUCT table: contains the information about which product (PRODUCT_ID) is of which product type (CATEGORY_ID).
- Relation R/2 means that a category would have many attributes, and each attribute belongs to one category.
- Relation R/3 shows that an attribute may have many values stored, each value is associated to a different concrete product.
- Relation R/4 shows that a category may have zero, one or multiple products stored in the catalog of that category type.

- One variation of the implementation would be to define attributes common to all products as columns in the PRODUCT table, instead of defining and storing them as attributes in ATTRIBUTE and ATTRIBUT_VALUE tables. This is shown in the known uses section examples. This would enhance the system performance, however, it would limit the use of the e-catalog system for certain contexts like online stores. Other contexts, for example in a scientific research lab, may be interested with storing other common attributes for its research specimen catalog.

Dynamic behavior

To illustrate the use of the e-catalog to build a list of categories and products, we walk through a typical use case.

- A typical user of the e-catalog would need to add a new product. If no suitable category for this product exists in the catalog, the user would need to define a new category for this product.
- The user would then define a category for the product. This entails defining category name, number of its attributes, their types and their names.
- After the category has been defined, the system would persist this definition in the corresponding database structures (CATEGORY and ATTRIBUTE tables).
- The user would repeat the above process to define a hierarchy of the categories that may exist at the business context.
- The user then proceeds to store the information about the product. The system would present the user with a list of pre-defined categories; the user would pick a category from the list to add the product in that category. The system then gets a list of all attributes defined for this category. This list would be composed of attributes defined for this category and all attributes inherited from its parent categories. The system would present the user with a form with all attributes to be filled. The user fills the information. The system persists the information in ATTRIBUTE_VALUE and PRODUCT tables.

Searching and browsing the catalog would be similar to the Catalog Pattern previously identified at [1].

Example.

An example of a context where the E-Catalog pattern could be used is in an E-commerce application that is displaying a catalog of products for online shoppers. The products are usually ordered in a group of categories for ease of browsing. There may be also a search facility to enable shoppers to find products based on some criteria they enter on the online form.

One of these applications is osCommerce (www.oscommerce.com), which is an open source e-commerce tool. In this tool, an e-catalog is implemented with an administration tool that enables the administration of the catalog and the application. A snapshot of this application is shown in Figure 2.

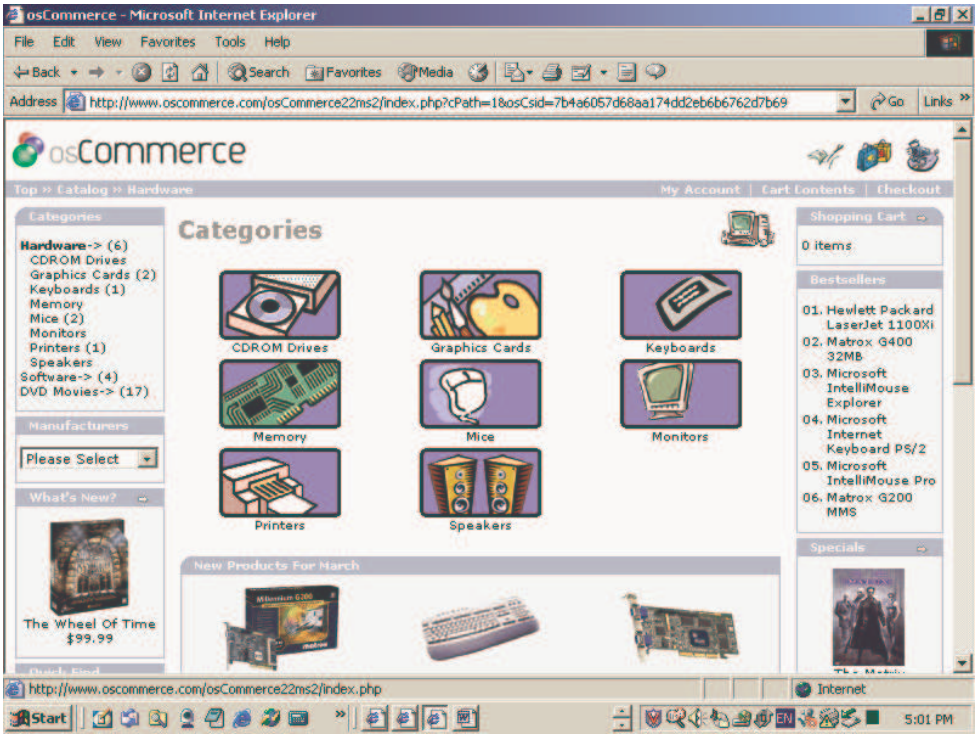

Figure 2: osCommerce demo snapshot

In this e-commerce solution, a catalog administrator can define categories and add products. An online shopper would be able to interact with the catalog to browse, search and select products. This application uses a relational database to persist categories and products.

Another open-source e-commerce system is ofbiz at http://www.ofbiz.org/index.html. A portion of its data model for products and their type is illustrated in Figure 4.

More details about the data models can be found at the documentation on these application web sites as explained in appendix 1.

Resulting Context

This pattern is most similar to the *Name-Value Pair Table* approach presented in [6], with some variation. A formal study on this pattern and other variant structures performance for e-catalogs is presented at [7]. This study shows experimental results for different catalog queries and different catalog structures. However, we present here general advantages and disadvantages for using this pattern over conventional database design models.

By applying the pattern we obtain the following:

Pros.

- The ability to add new product categories and products to the e-catalog even when these are not known at design time.
- Unlimited number of categories can be added in a hierarchical scheme to represent large variety of products and items that may exist in a business context.

- Ability to execute advanced user queries against the catalog information.
- Flexible e-catalog structure that can be used in many contexts without the need to change its code or database structure.
- Effort of defining new categories is minimized, as these would inherit their parent's attributes without the need to redefine them.
- This pattern works well for a database that stores a moderate volume of concrete products, but with large variety of product types that keep changing over time. This is typical in a store catalog, as we may have hundreds of product types, but for each type we store information of one product with a quantity value that reflects store inventory.

Consequences

- Complicated data model that needs complex code to deal with it. This is true with application code as well as Data Management Language (SQL in case of RDMS). For example, to get all the refrigerators with a size of 18c.f. and with a price less that $900, it would need a query joining all of the four tables with a relatively complicated SQL. On the other hand, if the refrigerator product were represented with a single table, as in conventional database design, it would be much easier to write the query against that table.
- When implementing catalogs in RDBMS using this pattern, there would be a gain of flexibility on the expense of performance. Some performance penalty is incurred due to storing product information in many rows and tables, thus the need to join these to get product information, instead of getting one row from a single product table as implemented traditionally. This penalty would grow with the growth of data volume, thus this technique may not be suitable for large databases where we know that product types (categories) would not change in the future, and data volume (i.e. number of actual distinct products) is always high. Such systems, like those found in banking

Deleted: to

Deleted: in a query

Deleted: in

industry for example, would have few bank account types that rarely change, and huge volume of customer account information with high transaction rate.

- This technique would be more complicated to deal with products that may have some relation to each other. Such relations are usually defined in RDBMS with foreign keys and data constraints. In this pattern, these DBMS managed constraints would not work and they would need to be enforced through application code adding much more complexity to the application. A framework that encapsulates the pattern implementation in the DBMS and provides simple API to access its stored information would better handle this complexity.

Rationale

Without using the above pattern, we would need to define a data base table for each product we wish to represent in our catalog. This solution would prove impossible as the number of different products could go to the hundreds. In addition, new products may appear after the system being built, and thus new data base tables would need to be defined and application code would need to be altered to read these new tables. On the other hand, if all products are stored in a generic table with its attributes stored in one large text column in that table, doing searches and comparisons would be difficult as there is no easy way to extract a particular attribute value from the large text field. Also, comparing two attributes for two different products would not be possible unless we know the type of these attributes and make sure they are compatible.

The presented pattern works well in the given context above for the following reasons:

- The solution defines a generic catalog system that can be used in many business contexts without the need to change its code or database structures.
- It provides a basic e-catalog structure that is able to store a variety of products and items that are typically found in a store or other subject areas.

- The e-catalog can keep information about unlimited types of products without the need to change its structure.
- Definition of hierarchical category tree minimizes the effort when populating a new product into the catalog. A user, entering a new product to the catalog, would enter values for those attributes defined for the product category without a need to redefine them with each product.
- Accurate product searches and comparisons can be made, as all product attributes are stored in an atomic format, as opposed to one large text field.
- The performance of the e-catalog would still be acceptable when there is a need to store many types of products, with small data volume and low updates to stored data. This is typically the case with most commercial and manufactured product catalogs, where there is a need to capture the properties of many different product types, but, transactions changing product types are not frequent.

Deleted: ,
Deleted: as
Deleted: a
Deleted:

Known uses

Details of obtaining data models documentation from project web sites in this section are explained in appendix 1. All known uses in this section are from the open source projects. This is due to the availability of the source code including the database design models.

The data models diagrams are presented here in a simplified Entity Relationship notation. In this notation, an *entity* is represented with a rectangular box. The entity name is written at the upper cell of the box. The entity *attributes* are listed at the lower cell of the box. Each attribute has a data type that may be shown on the diagram as in Figure 3, or omitted for brevity. An attribute could also be part of the entity primary key, and in this case it is denoted with PK on the diagram. Any attribute in an entity could also contain values of another entity's primary key. In this case, this attribute is called a foreign key and denoted with FK on the diagram. An arrow from one entity to another shows a reference from the entity

Formatted

at the arrow base (a *child*) to the one at the arrow head (the *parent*). This reference is an indication that a FK (in the child entity) contains values of a PK (in the parent entity). An arrow originating from, and pointing to the same entity relates two attributes in the entity. One of these is acting as a PK and the other as a FK pointing to that PK. This self-referencing usually indicates a hierarchical parent-child structure that can be represented in that entity.

Formatted
Formatted
Formatted
Formatted
Formatted

OsComerece

As per the documentation found at [3]. Products are stored in this e-commerce system in a catalog modeled as shown in Figure 3. In this model a hierarchy of categories is represented as indicated in our e-catalog pattern (in categories table). Storing a row in products_to_categories Table represents product association to a category.

Deleted: Figure 3
Inserted: Figure 3

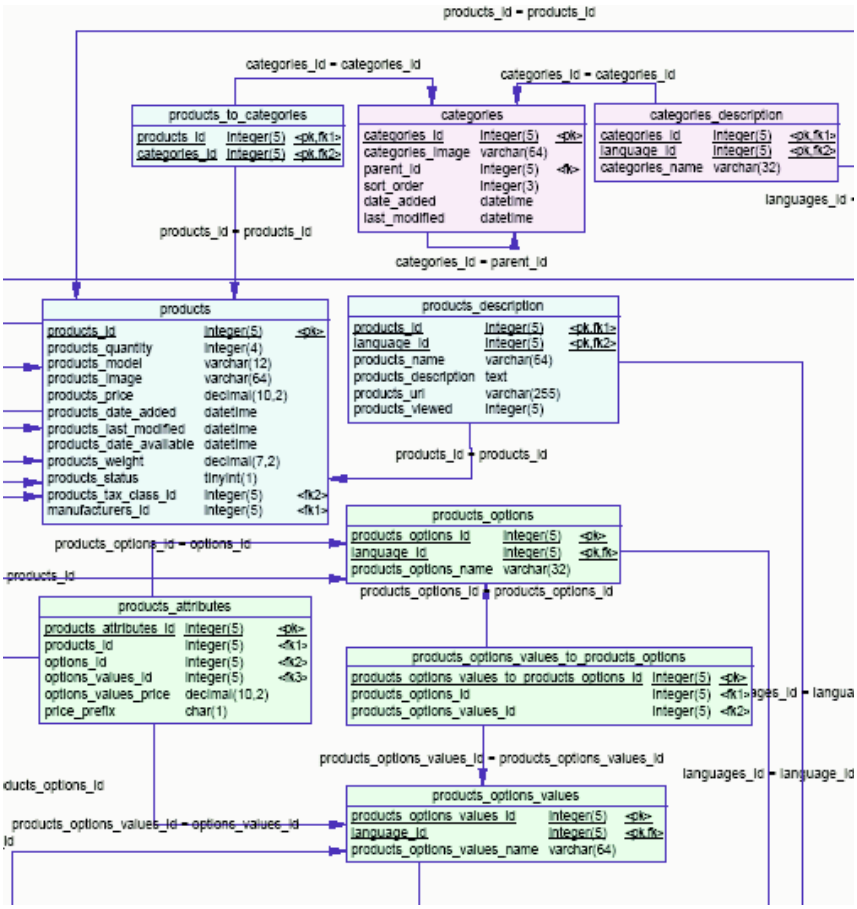

Figure 3: Part of Catalog data model at osCommerce [3].

Open For Business (ofbiz)

Is an open source e-commerce solution [4]. It describes itself as “The Open For Business Project is an open source enterprise automation software project licensed under the MIT Open Source License. By open source enterprise automation we mean: Open Source ERP, Open Source CRM, Open Source E-Business / E-Commerce, Open Source SCM, Open Source MRP, Open Source CMMS/EAM, and so on.” [4].

Part of the data model for Product is shown in Figure 4. In this data model, category hierarchy is captured with multiple rows in PRODUCT_CATEGORY table. Each category has some attributes defined with multiple rows in

PRODUCT_CATYEGORY_ATTRIBUTE table. Different products are associated with a category and stored in the PRODUCT table. Products could also belong to a product_type that is stored in PRODUCT_TYPE table, each poroduct_type contains some attributes defined in PRODUCT_TYPE_ATTR table.

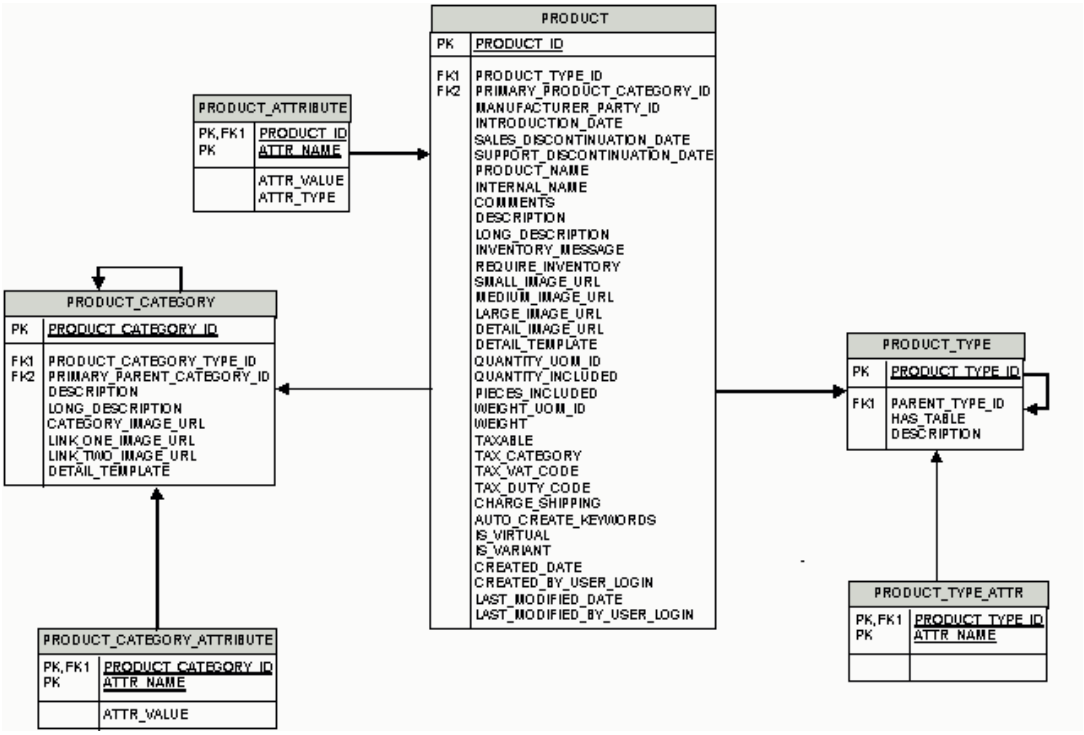

Figure 4: Product data model in ofbiz application [4]

Hipergate

Is an open source Customer Relationship Management (CRM) solution [5], and a well-documented application suite. It has a product catalog component that persists its information using a similar pattern. In this implementation, categories are organized into hierarchies or trees. Each hierarchy starts with a root category. The parent/child relation in the hierarchy is represented in table k_cat_tree as shown in the data model in Figure 5.

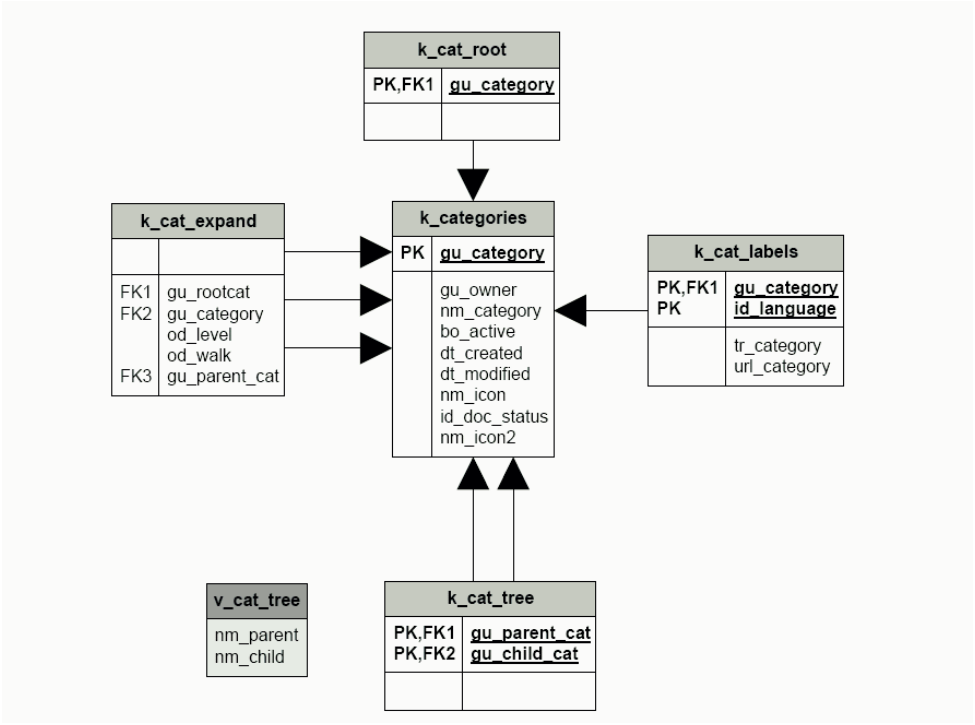

Figure 5: hipergate Categories Sub-model [5]

Product is an object defined in a category. Products share a set of common attributes (defined in k_prod_attr) , and each product could add more custom attributes (defined in k_prod_attrs). The data model is shown in Figure 6. More details can be found at [5].

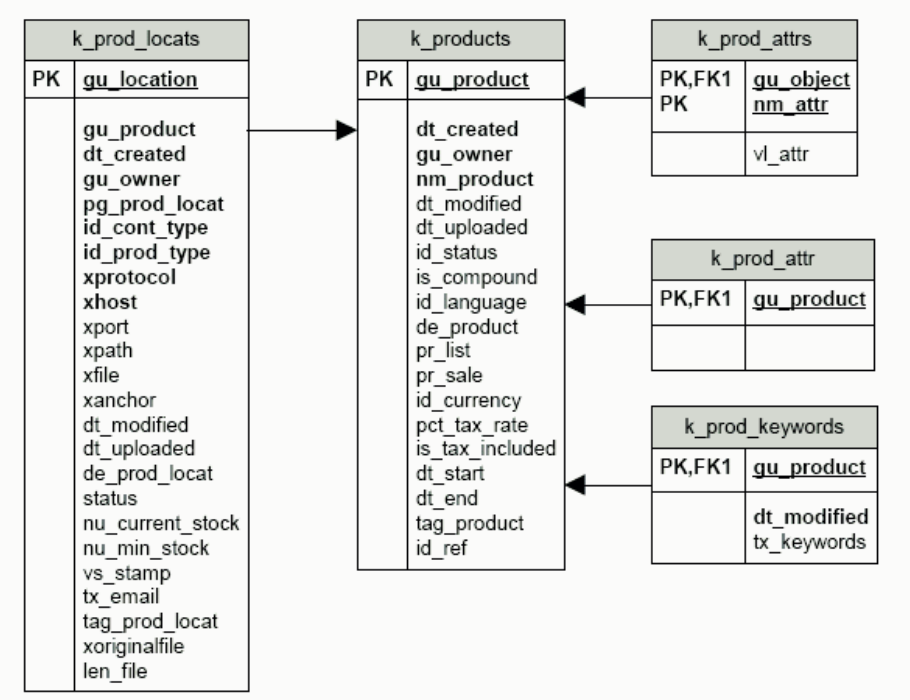

Figure 6: Products sub-model [5]

Related Patterns

This pattern makes use of inheritance pattern. It can also be used with optimistic concurrency control pattern to control concurrent changes to rows of product categories and products in the catalog. Other variations of data models for e-catalogs could be found at [6] and electronic commerce patterns at [8].

Appendix 1

To obtain osCommerce data model shown in Figure 3, download the whole "Windows package" from the project web site download section (at http://www.oscommerce.com/solutions/downloads) that is "osCommerce 2.2 Milestone 2 Update 051113", and un-zip the package into a local folder. In that

folder, the file "oscommerce-2.2ms2-051113\tep_database-pr2.2-CVS.pdf" shows the complete data model for osCommerce. Figure 3 shows the part of the model that implements the pattern presented in this paper.

To obtain Ofbiz data model as shown in Figure 4, go to the documentation web page at http://incubator.apache.org/ofbiz/documents.html, then select "Data Model Documents & Diagrams" which takes to https://ofbiz.dev.java.net/servlets/ProjectDocumentList?folderID=236.
On this page, the file "ofbiz.product.20020826.vsd" is a MS VISIO diagram that contains the complete data model for products and catalogs in ofbiz. Figure 4 shows only the components that implement the pattern under study.

To obtain hipergate data model as shown in Figure 5, go to the programmer's guide from page http://www.hipergate.org/docs/#user. This takes to the documentation at http://www.hipergate.org/docs/files/2.1.0/prog_guide-2.1.0-en.pdf.
Figure 5 corresponds to the model on page 22, and Figure 6 corresponds to the model on page 50. Description of model fields is also included in the same document.

Formatted: Bullets and Numbering

Appendix 2

Entity Relationship Model Notations

This database structure in Figure 1 is represented in a traditional entity relationship diagram. This diagram uses the crow's foot notation for Entity relationship diagrams. For notation details, the reader may refer to any of database design books. One of such books is [9]. Figure 7 summarizes the notation and could be found at [9], chapter 3, figure 3-2.

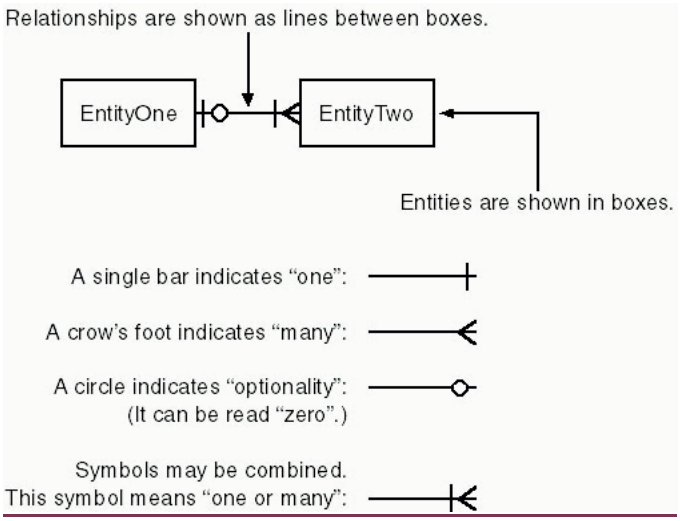

Figure 7: Entity Relationship Diagram Notation, Fig 3-2 [9]

References

1. Eduardo B. Fernandez, Yi Liu, and RouYi Pan, Patterns for Internet shops, in PLOP 2001.
2. E. Gamma, E. Helm, R. Johnson and J. Vlissides, Design Patterns –Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.
3. osCommerce project at http://www.oscommerce.com as accessed on July 2006.
4. ofbiz documentation at http://www.ofbiz.org/index.html as accessed on July 2006.
5. Hipergate documentation at http://www.hipergate.org as accessed on July 2006.
6. Dongkyu Kim, Sang-goo Lee, Jonghoon Chun, Sangwook Park, Jaeyoung Oh, CATALOG MANAGEMENT IN E-COMMERCE SYSTEMS, *proc. of Computer Science and Technology (CST 2003).*

7. K. Kim, et al, An Experimental Evaluation of Dynamic Electronic Catalog Models In Relational Database Systems, *Proc. of the Information Resources Management Association International Conf.,* 2002.
8. André Widhani, Stefan Böge, Andreas Bartelt, and Winfried Lamersdorf, Software Architecture and Patterns for Electronic Commerce Systems, Ninth Research Symposium on Emerging Electronic Markets 2002.
9. Rebecca M. Riordan, Designing Relational Database Systems, Microsoft press, 1999.

Formatted: Bullets and Numbering