Pattern: Hierarchical Star Schema

Problem

How can hierarchical relationships be traversed in a relational data model?

Context

The HiStar pattern applies in the following situation:

- the application data model is characterized by hierarchical relationships along several dimensions
- a distinguished entity is the focal point of the data model
- the data model is represented using relational tables
- the hierarchy must be traversed flexibly and efficiently.

Figure 1 presents an example of a typical HiStar candidate. An Engineering Information System, or EIS, tracks the construction of external products using internal processes. The key entities involved in construction are characterized by hierarchical relationships. The `DesignPart` serves as the central entity of the EIS to bridge the design and manufacturing activities.

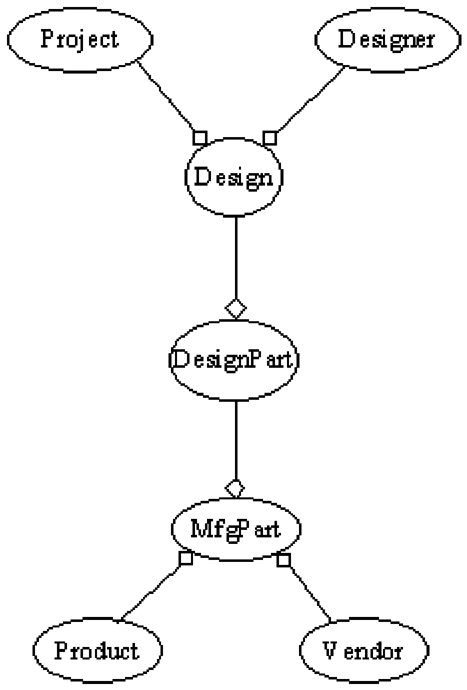

Figure 1. Engineering Information System Hierarchy

Although the hierarchy is well-defined, queries are dynamic. A query can start from any entity and traverse the hierarchy to discover information about related entities, e.g. "which vendors supply parts for the design?", or "which designer should be notified of a change to a manufacturing part?".

Forces

Supporting dynamic queries over a hierarchy using disjoint relational tables joined in an ad hoc manner complicates development and slows execution. The difficulties arise from the following forces:

1. Hierarchical relationships are not directly representable in the relational model. Entities are related in tables by foreign keys. Hierarchical relationships in a normalized schema are inferred from joins between all related entities in the hierarchical path.

2. Flexible query capability cannot prescribe a static set of traversal paths. If the topics of inquiry are not known beforehand, queries must be constructed dynamically involving a series of interrelated joins.

3. Programmatic navigation of hierarchy is cumbersome and limiting. Traversal from a parent entity to a child entity in the hierarchy involves an understanding of how navigation paths translate into joins. Dependence on these paths renders programs sensitive to schema change.

Solution

The combination of a relatively stable hierarchy and a distinguished central entity allows application of the HiStar pattern to simplify development and improve efficiency of dynamic queries. The solution is obtained as follows:

Step 1. Define the Hierarchical Relationships.

Step 2. Select a characteristic Anchor Entity Type.

Step 3. Organize the hierarchical dimensions in a Hub and Spoke Topology.

Each of these steps are described below.

1. Define the Hierarchical Relationships

This step defines an application data model consisting of significant domain entity types related in a hierarchical manner. The relevant entities to include are those entities that drive the business process represented by the model. For example, the Engineering Information System in Figure 1 contains two types of entities, classified according to their process focus. A *development entity* is an artifact used to determine product content. A *fabrication entity* directly relates to some aspect of the manufacturing process. The development entity types in Figure 1 include `Project`, `Designer`, `Design` and `DesignPart`. The remaining entity types--`MfgPart`, `Product and Vendor`--represent fabrication entities.

The hierarchy suggests natural traversal paths for navigation. For example, exploration of a project would proceed through a selected design to the parts within that design to the manufacturing characteristics of that part. Conversely, "where used" queries operate by working up the hierarchy. The use of a vendor is determined by examing the parts the vendor provides, what role those parts play in designs and which designers are responsible for the affected designs.

An entity type is typically associated with an *entity table* consisting of attributes that describe an enitity independent of its use in a hierarchical context. A `Product` entity table, for example, might include attributes that identify the product name, number, revision and price.

2. Select a Characteristic Anchor Entity Type

An *anchor entity type* ties together the development and fabrication processes. This step identifies a distinguished anchor entity type that will serve as the focal point for relating hierarchical entities. In the example, the `DesignPart` is the anchor entity type. The `DesignPart` serves as both a fundamental unit of the design activity as well as the basis for selecting a `MfgPart` that can be used in fabrication.

3. Organize Hierarchy into Hub and Spoke Topology

After the anchor entity type is chosen, the hierarchy can be reduced to a more tractable hub-and-spoke topology familiar from the Star Schema (cf. Related Patterns section). Each hierarchical path leading to or from the anchor entity type is flattened into a *map table* that records the occurence of hierarchical instances. A map table relates entities by foreign key values and is used, usually implicitly, in many relational schemes. Each record in a HiStar map table includes a reference to an anchor entity and one or more references to other entities in the hierarchy. The map tables for the EIS example are shown in Figure 2. The hierarchy is flattened using the map tables, as in Figure 3. The resulting tables denormalize the source tables. However, since the HiStar schema is only used for query purposes, the update anomalies resulting from the denormalization are not manifested.

Example

This example illustrates HiStar concepts for a simple Engineering Information System (EIS) domain. The basic hierarchies of interest are shown in Figure 1. The small rectangle denotes the "many" side of a one-to-many hierarchical relationship among entities. This example includes only one-to-many relationships; many-to-many relationships are considered in the Many-to-many Variation section below. Each entity type is assumed to have an associated entity table of the same name. For example, there is a `DesignPart` entity table that contains information about `DesignPart` entities.

In this domain, the `DesignPart` is the anchor entity type. The map tables are listed in Figure 2 below.

```
DsnPath(DesignPartId DesignName ProjectId DesignerEmpNo)
MfgPath(DesignPartId MfgPartId ProductId VendorId)
```

Path	**Map Table**
`Project Design DesignPart`	`DsnPath`
`Designer Design DesignPart`	`DsnPath`
`Design DesignPart`	`DsnPath`
`DesignPart MfgPart Product`	`MfgPath`
`DesignPart MfgPart Vendor`	`MfgPath`
`DesignPart MfgPart`	`MfgPath`

Figure 2. Relationship of Path to Map Table

A possible representation of the map table attributes is included. The key of the `DsnPath` map table (underlined) is the identifier of the anchor entity `DesignPart`. A `DsnPath` record contains a reference to all parents of the `DesignPart`. The key of the `MfgPath` map table includes the identifiers for the `DesignPart`, `MfgPart` and `Product`. On the assumption that there is only one `Vendor` per `MfgPart`, the `VendorId` attribute is fully functional dependent on `MfgPartId` alone. Hence, in this schema, the `MfgPath` table has been denormalized from third normal form to second normal form in order to optimize queries.

A different design, or evolving business requirements, could change the map table entry. This is a physical database design decision that is neither dictated nor precluded by the HiStar pattern. The use of an intermediary map table entry for navigational paths localizes schema dependency and reduces change impact. The only requirement for the map table is that each map table includes the anchor entity key. Given that feature, programmatic navigation references the map table entry for the given path, reducing the implicit dependence on a particular schema.

Note that a `DsnPath` record relates a single design part to a single design, project and designer. Support for a many-to-many relationship of design part to one of these parent entities is described in the Many-to-many relationships Variation below.

Reorganizing the schema around the map tables results in the hub-and-spoke topology of Figure 3.

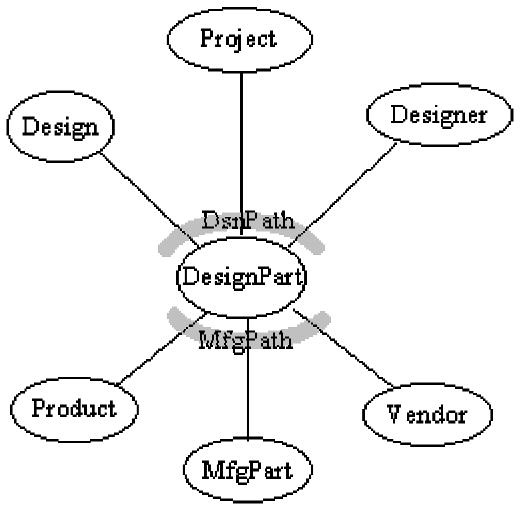

Figure 3. Hub-and-spoke arrangement

Resulting Context

The resulting schema supports hierarchical navigation along several dimensions. A navigational query can start from any supported entity. Navigation to another entity relates the corresponding map table records that share a common anchor entity. For example, navigation along the path `Project Design DesignPart` yields `DsnPath` records for `DesignParts` in the given `Project` and `Design`.

Hierarchy is traversed by referencing map table records that share a common anchor entity occurence. A query that seeks to obtain information about two entities in the hierarchy joins map tables if necessary to relate the entities, then joins entity tables to retrieve attributes about the related entites. For example, the query "which vendors supply parts for the design?" requires traversal from a `Design` through the common `DesignPart` to the `Vendor`. This entails three joins:
The entity table `Design` is joined to the map table `DsnPath` to select `DesignParts` for the given `Design` The map tables `DsnPath` and `MfgPath` are joined to determine `Vendors` for the `DesignParts` The entity table `Vendor` is joined to the map table `MfgPath` to get the vendor information.

In each case, the join is done on the common `DesignPart` foreign key. By contrast, navigation that does not follow the HiStar pattern must embed the semantics and physical representation of relationships that are implicit in the schema. The HiStar pattern makes the hierarchy explicit, localizes it to a map table entry for each path, and simplifies navigation by relating all entities to the anchor entity.

An incremental traversal proceeds similarly, exploring the hierarchy one level at a time. An incremental traversal starts with selection of a `Design`, shows `DesignParts` for the selected `Design`, allows selection of a `DesignPart` for further exploration, and finally discloses possible `Vendors` for the selected `DesignPart`. In all cases, joins are done by looking up the map table and joining on the `DesignPart`.

The navigation path can be *inverted* to start from the anchor entity and move outward. For example, the path `DesignPart Design Project` yields `DsnPath` records of a `Project` for the given `DesignPart` and `Design`. Intermediate nodes of the complete hierarchical path can be omitted to form an *elided* navigation path. The path `Designer DesignPart` reflects all `DsnPath` records for the given `Designer` without regard for the particular `Design`.

Rationale

- Programmatic navigation along a fixed hierarchy is unnecessary since any entity relates to any other entity via the map tables. For example, determining the possible vendors of a project can be done in a standard fashion by looking up the map tables, rather than a fixed solution embedded in program code.

- Since every map table includes an anchor entity foreign key, navigation from any entity to any other entity requires at most one join operation. Navigation is done through the map tables. If the two entities have the same map table, e.g. `Project` and `Designer`, then no join is required to relate the entities. Otherwise, the entities are related by joining the `MfgPath` and `MfgPath` tables on the common `DesignPart` key.

- Information about a given entity is obtained from its entity table regardless of the path used to reach the entity. There is a known table for the entity information and it is always obtainable by joining the entity table to its map table.

Known Uses

The HiStar Schema pattern has manifested itself in Product Data Management, Electronic Design Automation and Marketing Analysis domains. In the first two cases, a diversity of special-purpose programmatic navigation techniques led to a reconsideration of access profiles, construction of a common data model, emergence of a unifying anchor entity and restructuring around this entity. The HiStar Schema is particularly appropriate when both the development and fabrication activities are distinct and sizable. A product construction domain in which one of these activities dominates typically does not have organizational complexity sufficient to require flexible traceability through the development and fabrication process. The HiStar Schema best fits a domain that has a balance of development and fabrication process complexity, occupying the middle of the spectrum in Figure 4.

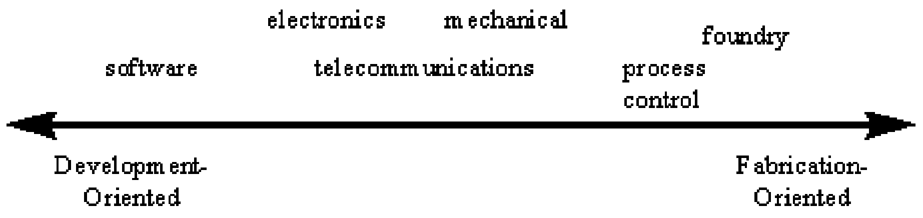

Figure 4. Process orientation by discipline

In the case of the Marketing Analysis application, customer information is organized hierarchically as in Figure 5. A customer belongs to a household, receives a catalog and places orders containing items. The data model is similar to the standard flat Star Schema shown in Figure 7, but the hierarchical organization is fundamental to the analysis and the Customer rather than a business activity is the unifying concept.

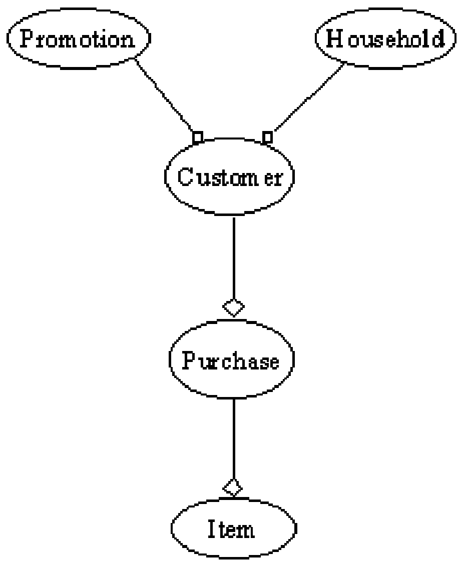

Figure 5. Marketing analysis hierarchy

The HiStar pattern may apply to other domains that require hierarchical navigation. The author welcomes accounts of additional known or potential uses of the HiStar pattern.

Related Patterns

The Star Schema pattern language [1] is the basis for the HiStar Schema pattern language. A star schema defines business entities that act as the *dimensions* of key business activities. Key business activity transactions are recorded in a *fact table*. Each dimension is represented in the fact table by a foreign key. The dimensions form a star pattern about a central fact table (Figure 6).

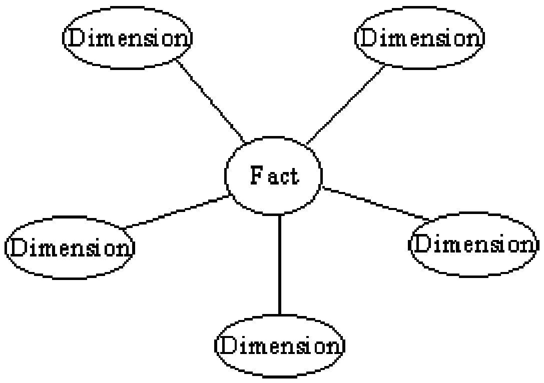

Figure 6. Star schema pattern

The resulting schema is the basis for designing a query-optimized database. Queries summarize activities along selected dimensions. An example of a star schema is shown in Figure 7.

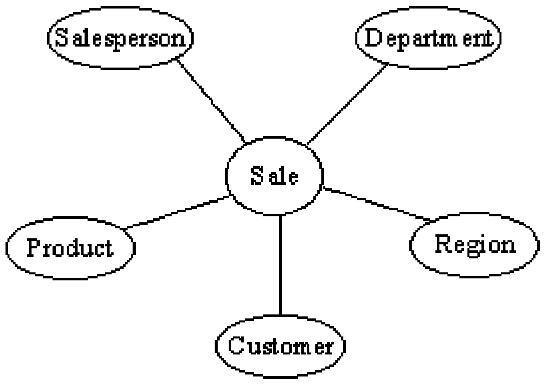

Figure 7. Star schema example

The schema represented here implicitly flattens any hierarchical relationships. For example, although a `Salesperson` may belong to a `Department`, and a `Customer` may reside in a `Region`, the `Sale` fact table flattens these relationships. The hierarchical paths are not relevant to the Star schema because analysis is focused on how the entities relate to the key activity, `Sale` in this case. The HiStar pattern generalizes the Star schema for applications which characteristically perform queries on the hierarchical relationships of business entities.

Variations

- Recursion
- Many-to-many relationships
- Secondary Attributes

Recursion

It is not unusual for a developmental entity type to hold a recursive hierarchical relationship to itself. For example, `DesignPart` might have a recursive relationship as in Figure 8. The design can now include a subassembly as well as atomic manufacturable parts. Each subassembly consists of other design parts but is captured as a design part in its own right, available for reuse in other designs.

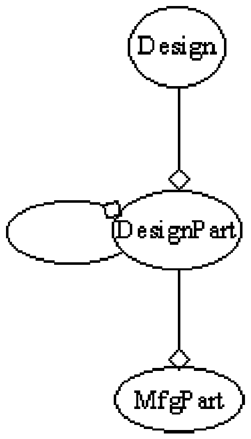

Figure 8. Recursive relationship example

For example, a compatible wheel-tire-hub assembly can be referenced as a single design part in bicycle construction, although the assembly is actually manufactured from other parts. Figure 9 shows a standard recursive scheme. The parent entity `P` contains a child entity `A`. `A` recursively contains children `B` and `C` of the same entity type. `B` recursively contains `D` and `E`. `C`, `D and E` are atomic child entities.

Parent-Child	Child-Child	Parent-Child	Child-Child	Parent-Child
`P A`	`A B`	`P A`	`A B`	`P C`
	`A C`		`A C`	`P D`
	`B D`		`B D`	`P E`
	`B E`		`B E`	
			`A D`	
			`A E`	
Recursive Child		**Expanded Child**		**Detail Parent-Child**

Figure 9. Eliminating recursion

Recursion can be handled as follows:

1. Expand the recursion into composite and detail entities. A composite recursive entity contains other entities of the same type; a detail recursive entity does not contain other entities of the same type.

2. Form the transitive closure of the recursion, DesignPart in the example as in Figure 9. The transitive closure includes records for all composite entities that contain detail entities, either directly or indirectly.

3. Reconstitute the parent-child map table to include only detail child entities. The new parent-child map eliminates the recursion and recasts the parent-child relationship into an association of parents to detail children only.

The context that results from this decomposition eliminates the recursion but loses information about intermediate composite recursive entities. However, the lost information is typically not important to queries using the HiStar pattern.

Many-to-many relationships

The data model considered so far only includes one-to-many hierarchical relationships, wherein the child entity is contained in exactly one parent entity. Hierarchy may also include many-to-many hierarchical relationships. In that situation, the map table can potentially hold several parent entities for a given child entity. This is the case, for example, if a `DesignPart` in the example could be reused by several `Designs` or if multiple `Designers` are responsible for a single `Design`.

Note that many-to-many relationships present a similar problem to the Star Schema as well. For example, if there is more than one `SalesPerson` for a given `Sale`, the configuration of Figure 7 must be modified to accomodate this relationship cardinality anomaly. Since the `Sale` fact table references the `SalesPerson` by a foreign key, there can be at most one `SalesPerson for` a given `Sale`.

Many-to-many cardinality is supported in hierarchical relationships by changing the relationship map table. Recall that a map table determines the hierarchical entity occurences for a given anchor entity. A `DsnPath` record in the simple example records the unique `Design`, `Project` and `Designer` for a given `DesignPart`. In the case of a many-to-many relationship of `Design` to `DesignPart`, the `DsnPath` map table still suffices to determine the hierarchical entity occurences, since there is only one `Project` and `Designer` for a given `Design`. However, multiple `Designers` for a single `Design` would necessitate separate map tables for the `Designer` hierarchy and the `Project` hierarchy, as shown in Figure 10.

```
PrjPath(DesignPartId DesignName ProjectId)
DsnrPath(DesignPartId DesignName DesignerEmpNo)
DsnPath(DesignPartId DesignName)
ProdPath(DesignPartId MfgPartId ProductId)
OrderPath(DesignPartId MfgPartId OrderNo)
VendorPath(DesignPartId MfgPartId VendorId)
MfgPartPath(DesignPartId MfgPartId)
```

Path	**Map Table**
`Project Design DesignPart`	`PrjPath`
`Designer Design DesignPart`	`DsnrPath`
`Design DesignPart`	`DsnPath`
`DesignPart MfgPart Product`	`ProdPath`
`DesignPart MfgPart Order`	`OrderPath`
`DesignPart MfgPart Vendor`	`VendorPath`
`DesignPart MfgPart`	`MfgPartPath`

Figure 10. Many-to-many map tables

Secondary Attributes

A secondary attribute in this context refers to an attribute that is dependent on two entities that do not have a direct hierarchical relationship in the schema. For example, the `Approval` relationship in Figure 11 indicates whether a `MfgPart` is approved for use in a `Project`. Such secondary attributes can convey important information for hierarchical query purposes.

A secondary attribute can be incorporated into the HiStar schema by a modest enhancement to the traversal mechanism. Recall that the attributes to be included may come from the map table for the path and the entity table of the last entity type in the path. Secondary attributes are obtained by augmenting these attributes with any secondary attribute records between the terminal entity in the path and any other entities in the path. The attribute set for the path `Project Design DesignPart MfgPart` is augmented to include the approval status for the `MfgPart` within a `Project`.

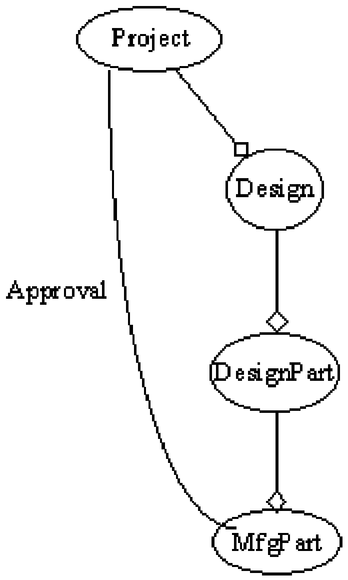

Figure 11. Secondary attribute example

References

[1] Peterson, Steve. Stars: A Pattern Language for Query Optimized Schema, *Proceedings of PLoP 94* (1994).

Author:
F. Nelson Loney
Method
loney@acm.org