Anchor Modeling

FLEXIBILITY
The environment surrounding a data warehouse is in constant change. Anchor modeling is built on this premise, such that a large change on the outside will result in a small change within.

INDEPENDENCE
The model itself is independent of business logic. Rules are descriptive rather than physical to increase the longevity of the data warehouse. You have the power to decide how the data should be interpreted.

SCOPING
This modular data warehouse modeling technique supports separation of concerns and simplifies project scoping. You can start small with prototyping and later grow into your enterprise data warehouse.

MODULARITY
Data from different functional units within a business are stored in the data warehouse as self-contained areas. They can be implemented at different times.

EXTENSION
Every change is implemented as an independent extension in the existing data warehouse model. This means that current applications will not be affected.

MAINTAINABILITY
Consistent use of a simple modeling technique like anchor modeling will yield conformity and increase maintainability across your data warehouses.

Where did it all come from?

Background

Anchor Modeling is built upon two techniques both discovered in the 1970's; the sixth normal form and entity relationships. In more recent years the sixth normal form has been discussed with respect to storing temporal data. Also, entity relationships has evolved into enhanced entity relationships, which adds semantic modeling concepts such as typing.

What building blocks are used?

Constituents

There are only four different types of tables used in anchor modeling. The three types: anchors, attributes, and ties are sufficient for all modeling needs, but for practical reasons a fourth type: knots, is added. It will allow for a simple physical implementation of the semantic concepts found in enhanced entity relationship modeling. We have introduced our own symbol for knots, but the others are commonly used and taken from entity relationship modeling.

Who holds the identities of entities?

Anchors

An anchor holds the identity of an entity in the data warehouse. This identity is always technical by nature and represented by a surrogate key, rather than the natural key.

■ An anchor must contain a surrogate key and should contain meta information, such as information relating an identity to the batch and source that generated it.

Where is actual data stored?

Attributes

Attributes always belong to an anchor. They hold actual attribute values that can be used to describe the entity whose identity is stored in the anchor. The value stored in an attribute can be of any data type. Note that the cardinality of an attribute may be less than that of the anchor.

○ An attribute must contain a foreign key referencing the corresponding surrogate key found in the anchor and exactly one attribute value. It should contain meta information similar to that found in the anchor. If, for a given identity, the attribute may change over time, it should also contain historization information. In the case of the attribute having a state or a type, it can also hold foreign keys of knots.

How do you relate entities?

Ties

Relationships between entities are modeled as ties between anchors. A tie will thereby relate identities to each other. The most common form is to relate two anchors, but there is no theoretical limit to how many anchors can be connected with a single tie.

◆ A tie must contain the foreign keys of the adjoining anchors. If, for the given entities, the relationship may change over time, it should also contain historization information. Further, it should also contain meta information and may have foreign keys to knots if the relationship has a type or state.

What simplifications can be made?

Knots

In order to simplify the modeling, a combination of an anchor and an attribute may be assembled into a knot. Knots are used for typing or representing states in the data warehouse. Note that since it contains both its identity and its value it may never change over time.

▢ A knot contains a surrogate key representing its identity as well as an attribute value. It should also contain meta information in the case that it is built from source data, but it may be left out if it is built by hand.

How do you name the tables?

Naming Conventions

A naming scheme based on prefixes can be used to give a good overview when looking at a database or model. It also avoids bad models since you will have trouble naming your tables if you are not designing them correctly.

`CA_Cat`
Anchors have a two letter prefix followed by an underscore and a descriptive camel cased name.

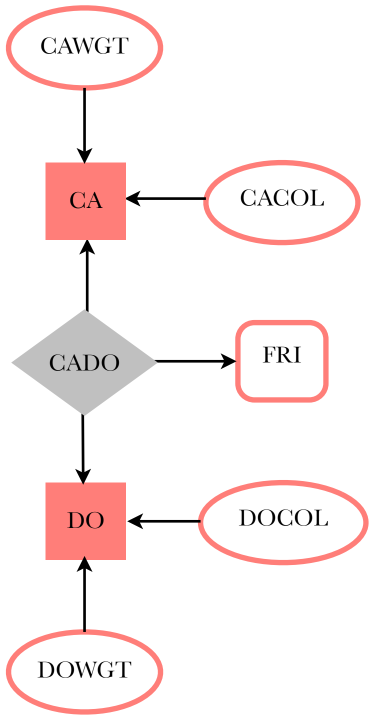
`CACOL_CatColor`
Attributes have a five letter prefix, where the first two letters indicate to which anchor it belongs.

`CADO_Cat_Dog`
Ties have a four, or 2*n*, letter prefix built up from the prefixes of the adjoining anchors. Note the extra underscore in the descriptive part to stress the fact that it is a tie.

`FRI_Friendliness`
Knots have a three letter prefix to separate them from the other kinds. Note that attributes or ties relating to knots do so without any change to their prefixes.

What does a model look like?
Example

In this simple example we will model a cat and a dog as separate anchors with a typed tie between them. Note that we can leave out the descriptive part in the logical model if we make sure that the prefixes are unique (recommended).



The above model contains information about cats and dogs in a many-to-many tie, their colors and weights as attributes as well as how friendly different individuals are with each other as a knot on the relation. Weight most likely needs to be historized, whereas color does not. If the friendliness may change over time, the relation itself must also be historized.

Could it have been different?
Generalization

We could also have made a generalized model based on an animal anchor, `AN`, with an animal type attribute, `ANTYP`. Then the tie for friendliness would refer the anchor to itself, `ANAN`. We would also have to describe the fact that there are colors only valid for cats, like tabby, in addition to the model. Business rules must in general be documented separately from the model and implementation.

How do you implement the model?
Implementation

Every object from the logical anchor model is implemented as its own table in the database. To ensure that duplicates never can enter the data warehouse each table must have a primary key (`p` below) which guarantees uniqueness. Historization information must therefore always be a part of it. Foreign keys can be declared to ensure integrity.

	CA_Cat
p	CA_ID
	_metadata

	FRI_Friendliness
p	FRI_ID
	FRI_Degree

	CACOL_CatColor
p	CA_ID
	CACOL_Color
	_metadata

	CADO_Cat_Dog
p	CA_ID
p	DO_ID
p	CADO_From
	FRI_ID
	_metadata

	CAWGT_CatWeight
p	CA_ID
p	CAWGT_From
	CAWGT_Weight
	_metadata

How do you load data?
Zero Update Strategy

We recommend using only selects and inserts when loading the warehouse and never update any rows. This will allow you to write a simple script using deletes to revert faulty data, since there will always be a one-to-one mapping between the data loaded in a batch and actual rows in the database. Streaming ETL tools can be used to fill many tables with just one scan of the source table.

Are there ways to speed queries up?
Indexing

The best query performance is achieved by creating clustered indexes over the primary keys. Since they are clustered they won't induce extra storage space, but work on the actual table data itself. The index should be arranged with the surrogate keys in ascending order and historization columns in descending order. That way, if you look at how data is physically structured on your storage media the latest version for any given key is always found first.

When do you create surrogate keys?
Identity Management

Proper identity management is key to success in a data warehouse. For anchor models there are two ways to achieve it. To connect the source with the surrogate you can either persistently store the natural key in the warehouse and generate new ones from there (late) or you can build the connection in your ETL process without the need for storing extra information (early).

How can I write simpler queries?
Collapsing Views

It is possible to create views that collapse anchors, attributes and ties into their corresponding third normal form, which you can query instead of directly accessing the anchor model. Most query optimizers will figure out which columns you are using and discard all other attribute tables, even though they are joined in the view. The commonly used ones are:

Historically correct view
This view will keep the historization information so that you can analyze changes over time.

Latest view
This view will find and show only the latest version of the attribute for any given anchor identity.

Point in time view
This view, or usually a table valued function, will take a point in time as an argument and return the latest view with respect to it.

Historization information is not propagated into the last two views. A left join from the anchor to the attribute with a subselect to find the latest (max) version normally gives the best performance.

What were the benefits again?

- ✦ Historization by design
  – *slowly changing dimensions to rapidly changing relations*
- ✦ Elimination of `NULL`
  – *one way referential integrity for missing attributes*
- ✦ Orphan handling
  – *early arriving facts can be added without existing parents*
- ✦ Separation of concerns
  – *access control, project scoping, and gradual extension*
- ✦ High performance querying
  – *narrow tables physically arranged for speed during scans*
- ✦ Simplicity in design and use
  – *naming conventions and collapsing views*