Event-Entity-Relationship Modeling In Data Warehouse Environments

Lars Bækgaard
The Århus School of Business
Fuglesangs Alle 4
DK-8210 Århus, Denmark
+45 8948 6695
lb@hha.dk

ABSTRACT

We use the event-entity-relationship model (EVER) to illustrate the use of entity-based modeling languages for conceptual schema design in data warehouse environments. EVER is a general-purpose information modeling language that supports the specification of both general schema structures and multi-dimensional schemes that are customized to serve specific information needs. EVER is based on an event concept that is very well suited for multi-dimensional modeling because measurement data often represent events in multi-dimensional databases.

Keywords

Data Warehousing, Event Modeling, Information Modeling, Multi-Dimensional Modeling, Star Schemes.

1. INTRODUCTION

A data warehouse environment supports data analysis within some domain of interest [1], [2], [3], [4], [5], [6].

We use the event-entity-relationship model (EVER) to illustrate the use of entity-based modeling languages for conceptual schema design in data warehouse environments.

Our work is based on the architecture in Figure 1. The data sources are used to populate the data warehouse in which data are stored in an integrated and flexible way in order to meet changing information needs. The purpose of an information warehouse is to provide efficient and effective access to relevant subsets of the data warehouse.

EVER supports schema definition at all levels of the data warehouse architecture. In information warehouses EVER supports specification of multi-dimensional star schemes that are customized to serve specific information needs.

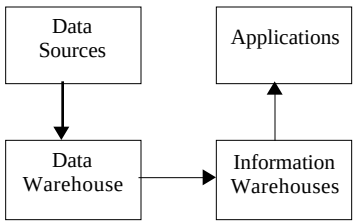

Figure 1 Data warehouse architecture.

EVER is very well suited for multi-dimensional modeling because events can be used to represent measurement facts and entities can be used to represent categorizing dimensions. We are not aware of other conceptual modeling languages that support event modeling in a way that is relevant and useful in data warehouse environments.

Other authors have presented multi-dimensional modeling languages that can be used to define star schemes in information warehouses [7], [8]. However, it is not clear if these languages are suited for schema design at all levels in the warehouse architecture. In [9] the entity-relationship model is extended to support the modeling of aggregates. The modeling approaches described in [7], [8], and [9] do not support event modeling.

The paper is organized as follows. In Section 2 we show how to use EVER to model information structures in terms of sets of events, entities, and relationships. In Section 3 we define the temporal semantics of EVER. In Section 4 we define the syntax and semantics of path expressions. In Section 5 we show how to define star schemes that are suitable for dimensional data analysis. In Section 6 we present a set of SQL-like queries on star schemes. In section 7 we conclude the paper and suggest directions for future research.

2. INFORMATION STRUCTURES

In this section we show how to define EVER schemes and we illustrate the notion of instance sets. EVER is an entity-based conceptual modeling language [10], [11], [12].

Often, measurement facts are categorized along multiple dimensions in multi-dimensional information models. The measurement facts are represented by events like “sale”, “order”, “request”, “deposit”, “pay”, and “ship”.

EVER supports modeling of information structures in terms of related entities and events.

An entity is an immutable representative of the uniqueness of a mutable phenomenon. An entity set is a named set of entities. Entity sets are symbolized by named rectangles.

An event is a unique and immutable representative of the occurrence of an atomic process phenomenon. An event set is a named set of events. Event sets are symbolized by named circles/ovals.

Values belong to domains like String, Integer, and TimePoint. A value set is a named and finite set of values from a specific domain. Value sets are symbolized by dotted rectangles.

A relationship is used to connect two elements. A relationship set is an unnamed, time-varying set of relationships. Unnamed lines symbolize relationship sets. Relationships between entities and values are used to model descriptive entity characteristics. Relationships between

events and values are used to model descriptive event characteristics.

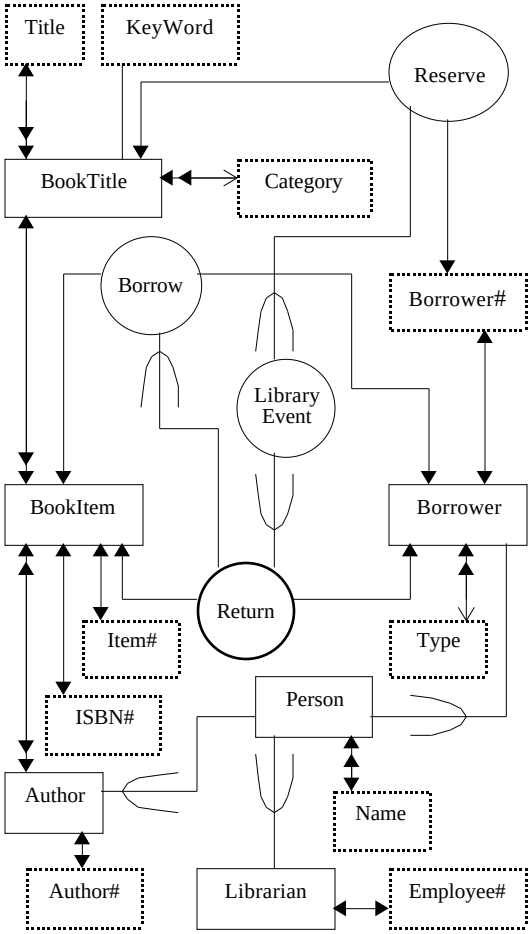

Figure 2 Library schema.

Figure 2 illustrates how EVER can be used to model a library data warehouse. A Librarian entity must be related to exactly one Employee# value. A BookItem entity can be related to any number of Borrow events and to any number of Return events. A BookTitle entity can be related to at most one Category element and to any number of KeyWord elements. A Reserve event must be related to exactly one BookItem entity and to exactly one Borrower entity. Author, Borrower, and Librarian must be subsets of Person. Borrow, Return and Reserve must be subsets of LibraryEvent.

We use the equivalent symbols Borrower<>Loan and Loan<>Borrower when we refer to the relationship set that relates Borrower and Loan.

The instance sets in Figure 3 and Figure 4 illustrate a partial population of the library schema in Figure 2. The entity e_3 represents a borrower whose number is 901. The entity have participated in a borrow event that is represented by the event ev_1.

An entity can be element in more than one entity set to reflect the fact that the modeled phenomenon can play more than one role. For example, in some situations a person plays a borrower role in a library. In other situations the same person plays a librarian role. As a Borrower element an entity represents the person playing a role as borrower. As a Librarian element the same entity represent the person playing a role as librarian.

The following rules define a set of meta-level constraints. (1) All entity sets are subsets of Entity. (2) All event sets are subsets of Event. (3) All value sets are subsets of Value. (4) All entity-entity relationship sets are subsets of Entity<>Entity. (5) All entity-value relationship sets are subsets of Entity<>Value. (6) All entity-event relationship sets are subsets of Entity<>Event. (7) All relationship sets are subsets of Element<>Element (=Relationship). (8) All sets are subsets of Element. (9) The sets Event, Entity, Element<>Element, and Value are mutually disjoint. The fact that these constraints must be satisfied can be utilized in query languages.

3. TEMPORAL DATA

EVER supports valid time, i.e., time in the modeled part of the world. EVER does not support transaction time, i.e., time within the modeled database [13]. However, in order to simplify matters we have assumed in this paper that valid time and transaction time are identical.

Informally, entities and relationships exist within time intervals whereas events occur at time points.

We use the domains TimePoint, TimeInterval, and TimeSet to model time values.

The domain TimePoint is an ordered set of values. Each TimePoint value represents a point in time. In this paper each TimePoint value represents one day. For example, May 4 1998 and April 23 1997 are TimePoint values. We use the symbol NONE to represent an undefined TimePoint value.

A TimeInterval value is a consecutive set of TimePoint values. A TimeInterval value is represented by an expression on the form $[t_1, t_2]$ where t_1 is the smallest TimePoint value and t_2 is the largest TimePoint value. For example, [May 1 1998, May 2 1998] and [May 4 1998, May 5 1998] are TimeInterval values.

A TimeInterval value is active if t_2=NONE. For example, [May 1 1998, NONE] and [May 4 1998, NONE] are active TimeInterval values. An active TimeInterval value represents the time interval from t_1 to the current time.

A TimeSet value is composed of a set of mutually disjoint TimePoint or TimeInterval values. For example, {April 30 1998, May 2 1998, [May 4 1998, NONE]} is a TimeSet value.

An entity exists as a member of a specific entity set in a set of time intervals and a relationship exists as a relationship member in a set of time intervals. This is modeled by the function EXIST that takes an element and a set name as arguments and returns a TimeSet value.

EXIST is interpreted as follows. If [t, NONE] EXIST(e(s)) then e is currently a member of s and has been a member of s since t. If [t, NONE] EXIST(e(s)) then e is not currently a member of s.

EXIST: Element(Set) → TimeSet

EXIST is updated whenever elements are inserted or deleted. Elements are inserted by means of TELL commands as illustrated by the following examples.

TELL e_4(Person)
TELL e_4(Borrower)<>904(Borrower#)

The first command inserts the element e_4(Person) into the entity set Person. The second command inserts the element e_4(Borrower)<>904(Borrower#) into the relationship set Borrower<>Borrower#.

Elements are deleted by means of UNTELL commands as illustrated by the following examples.

UNTELL e_1(Person)
UNTELL e_1(Person)<>"Gail"(Name)

These commands delete the elements e_1(Person) and e_1(Person)<>"Gail"(Name) from Person and Person<>Name, respectively.

The following ECA-rule (event-condition-action) defines the insertion semantics for EXIST. Initially, EXIST(e(s))=Ø for all combinations of e and s.

ON TELL e(s)
IF (s Entity s Relationship)
t: [t, NONE] EXIST(e(s))
THEN EXIST ← EXIST {(e(s), [EVAL(NOW), NONE])}

The rule is triggered when an entity or relationship is inserted into a set by a TELL command. If no dynamic TimeInterval value is related to e(s) in EXIST then e is not currently a member of s. In this situation the pair (e(s), [EVAL(NOW), NONE]) is added to EXIST. Otherwise, e is currently a member of s and EXIST is left unchanged. NOW is a function that returns the current time when it is evaluated.

The following ECA-rule defines the deletion semantics for EXIST.

ON UNTELL (e(s))
IF (s Entity s Relationship)
t: [t, NONE] EXIST(e(s))
THEN EXIST ← EXIST − {(e(s), [t, NONE])}
EXIST ← EXIST {(e(s), [t, EVAL(NOW)])}

The rule is triggered when an entity or relationship is removed from a set by an UNTELL command. If a dynamic TimeInterval value is related to e(s) in EXIST then e is currently a member of s. In this situation the dynamic TimeInterval value (e(s), [t, NONE]) is replaced by the TimeInterval value (e(s), [t, EVAL(NOW)]) in EXIST. Otherwise, e is not currently a member of s and EXIST is left unchanged.

We assume that the following subset constraint is satisfied. For a particular element the existence time corresponding to a subset membership must be contained in the existence time corresponding to a superset. If Borrower is defined as a subset of Person as in Figure 2 the following statement must be satisfied by any Borrower entity, e: EXIST(e(Borrower)) must be a subset of EXIST(e(Person)).

An event occurs at a specific time point. This is modeled by the function OCCUR. It takes an element and a set name as arguments and returns a time point. If OCCUR(e)=t then the event e occurred at the time t. If OCCUR(e)=NONE then the event e has not occurred yet.

OCCUR: Element → TimePoint

The following ECA-rule defines the insertion semantics for OCCUR. Initially, OCCUR returns NONE for all events.

ON TELL e(s)
IF s Event [e, NONE] OCCUR
THEN OCCUR ← OCCUR {[e, EVAL(NOW)]}

The rule is triggered when an event is inserted into a set by a TELL command. If e is related to NONE in OCCUR then e has not occurred at any point in time. In this situation the pair [e, EVAL(NOW)] is added to OCCUR. Otherwise, e has occurred at another point in time and OCCUR is left unchanged.

4. PATH EXPRESSIONS

In this section we define the notions of paths and path expressions.

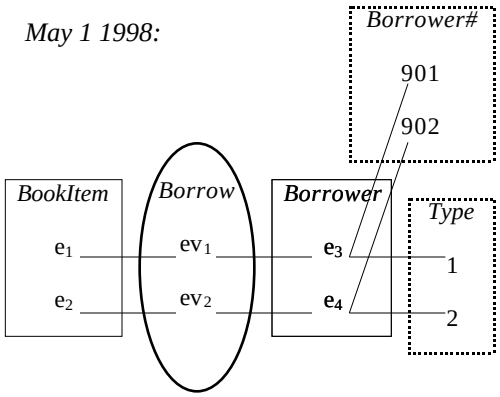

Figure 3 Instance sets.

Conventional path expressions evaluate to a specified set of terminal objects [14], [15], [16], [17], [18], [19]. Our path expressions evaluate to a set of paths that include the terminal objects as a special case.

The examples in the remaining part of the paper are based on the instance sets in Figure 3-Figure 4. On May 1, 1998 two Borrow events occurred. On May 4, 1998 one Borrow event and one Return event occurred.

Intuitively, a path is a sequence of named elements. Formally, we define a path as follows.

Definition. A *path* of length 1 has the form e(y) where e is an entity, event, or value and y is a name. A path of length n (n>1) is a sequence on the form $e_1(y_1)<>\ldots<>e_n(y_n)$ where each e_i is an entity, event, or value and each y_i is a name. We assume that $(y_i=y_j)$ $(e_i=e_j)$. The expression $p.y_i$ selects the element e_i from the path p.

A path expression defines a set of paths. We define a star-path expression as follows.

Definition. A *star-path expression* has the form $x_1/y_1<>$ $<>x_n/y_n$. Each x_i (i>1) is an entity or value set name and x_1 is an event set name. Each y_i represents a renaming of x_i. If some y_i is not specified explicitly x_i is used as default name. A star-path expression evaluates to a path set as follows.

EVAL(x/y) = {e(y) ½e(x) x}

EVAL(x_1/y_1<> <>x_n/y_n) = {$e_1(y_1)$<> <>$e_n(y_n)$½

(1) (j [1 n-1]:
OCCUR(e_1) EXIST($e_j(x_j)$<>$e_{j+1}(x_{j+1})$))

(2) (j [1 n]: e_j Entity
OCCUR(e_1) EXIST($e_j(x_j)$))

A star-path expression evaluates to a set of paths. A path belongs to the resulting path set if the following conditions are satisfied. (1) Each relationship in the path must exist as of the occurrence time of the event in the path. (2) Each entity in the path must exist as of the occurrence time of the event in the path.

May 4 1998:

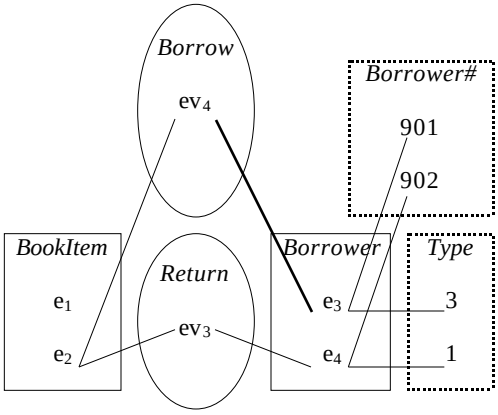

Figure 4 Instance sets.

A star-path can be used to represent one event and a set of related characteristics of the event. A set of star-paths in which a specific event occurs can be used to represent all characteristics of the event.

Definition. A *star* is defined by a set of star-paths that are based on the same event. If z is a variable that refers to a star the expression z.y will return the set of elements named y that occur in the set of star-paths.

When a star-path expression is evaluated the occurrence times of events determine the time slicing of the entities, values, and relationships in the qualifying paths. Consequently each event is related to its characteristics as of its occurrence time. The events are not attached to their characteristics as of current time.

This is illustrated by the following example.

EVAL(Borrow<>Borrower<>Type) →
{ev_1(Borrow)<>e_3(Borrower)<>1(Type),
ev_2(Borrow)<>e_4(Borrower)<>2(Type),
ev_4(Borrow)<>e_3(Borrower)<>3(Type)}

When the Borrower entity e_4 participated in the Borrow event ev_1 it was related to the type element 1. Later, when the same Borrower entity participated in the Borrow event ev_4 it was related to the Type element 3.

Path elements can be projected away by means of square brackets as illustrated by the following example.

EVAL(Borrow<>[Borrower]<>Type) →
{ev_1(Borrow)<>1(Type),
ev_2(Borrow)<>2(Type),
ev_4(Borrow)<>3(Type)}

In this example Borrower is used to define the qualifying paths but the Borrower entities are projected away in the resulting path set.

We use subset constraints to define downward and upward attribute expansion.

Definition. *Downward expansion* is defined by the symbol A↓ as follows.

(1) (A', B, a'<>b: A A' a'<>b A'<>B a' A)
(a'<>b A↓ <>B)
(2) EXIST(a'(A)<>b(B)) = EXIST(a'(A')<>b(B))

Downward expansion is used to extend the set of relationship sets of a subset and it can be used to simulate conventional inheritance. For example, the effect of downward expansion in the star-path expression Author↓ <>Name is that Author entities are related to Name elements via Person. The valid times of an element in Author↓ <>Name is identical to the element's valid times in Person<>Name.

Definition. *Upward expansion* is defined by the symbol A↑ as follows.

(1) (A', B, a'<>b: A' A a'<>b A'<>B)
(a'<>b A↑ <>B)
(2) EXIST(a'(A)<>b(B)) = EXIST(a'(A')<>b(B))

Upward expansion is used to extend the set of relationship sets of a superset. The effect of upward expansion in star-path expressions like Person↑ <>Author# is that Person entities are related to Author# elements via Author. The valid times of an element in Person↑ <>Author# is identical to the element's valid times in Author<>Author#.

5. STAR SCHEMES

In this section we show how EVER can be used to define conceptual star schemes for information warehouses. We assume that the library schema in Figure 2 defines a data warehouse.

Definition. A *star schema* is a named star-path expression set where all path expressions are based on the same event set.

The following star-path expression set can be used to define a star schema because the star-path expressions are based on the event set Borrow.

{Borrow<>BookItem<>BookTitle<>Category,
Borrow<>Borrower<>Type}

The following star-path expression set cannot be used to define a star schema because the star-path expressions are based on two different event sets.

{Borrow<>BookItem<>BookTitle<>Category,
Return<>Borrower<>Type}

The following star-path expression set can be used to define a star schema because the star-path expressions are based on the event set LibEvent.

{Borrow/LibEvent<>BookItem<>BookTitle<>Category,
Return/LibEvent<>Borrower<>Type}

A star schema is used to model a specific event type within a multi-dimensional space of entities and values.

Definition. A *multi-star schema* is a set of star schemes.

A multi-star schema is used to model a set of event types.

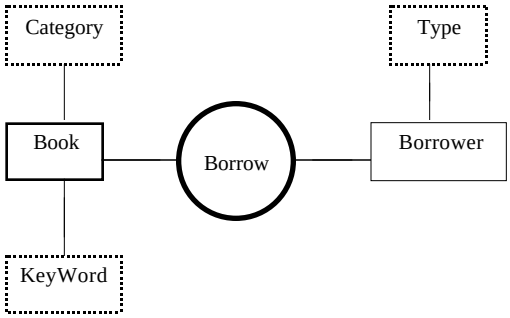

Figure 5 Star schema.

The EVER notation can be used to represent star schemes in a graphical way. As illustrated by the following examples fat circles/ovals emphasize the event sets.

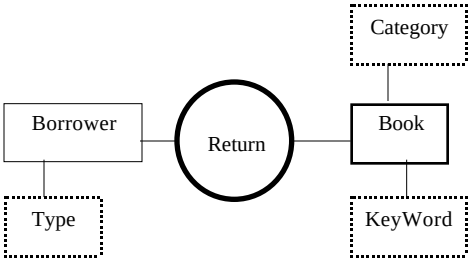

Figure 6 Star schema.

The star schema in Figure 5 is defined by the star-path expression set BorrowStar.

BorrowStar =
{Borrow<>BookItem/Book<>[BookTitle]<>Category,
Borrow<>BookItem/Book<>[BookTitle]<>KeyWord,
Borrow<>Borrower<>Type}

BorrowStar defines a heterogeneous star-path set that represents information about Borrow events. Information about a particular event is represented by the set of paths that involve the event.

The multi-star schema in Figure 5-Figure 6 is defined by the star-path expression sets BorrowStar and ReturnStar.

ReturnStar =
{Return<>BookItem/Book<>[BookTitle]<>Category,
Return<>BookItem/Book<>[BookTitle]<>KeyWord,
Return<>Borrower<>Type}

ReturnStar defines a heterogeneous star-path set that represents information about Return events. Information about a particular event is represented by the set of paths that involve the event.

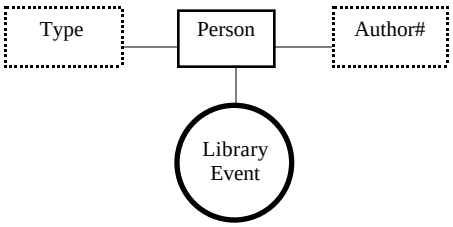

Figure 7 Star schema.

The star schema in Figure 7 is defined by the star-path expression set LibaryStar.

LibraryStar =
{LibraryEvent↑ <>Borrower/Person<>Type,
LibraryEvent↑ <>Person↑ <>Author#}

LibraryStar defines a heterogeneous star-path set that represents information about Borrow, Return, and Reserve events. Information about a particular event is represented by the set of paths that involve the event.

6. STAR QUERIES

In this section we illustrate how star schemes can be used as the basis of a query language. Our examples use an SQL-like syntax.

The following SQL-like expression illustrates how the star schema BorrowStar can be used for querying.

```
SELECT    COUNT(Borrow), Category
FROM      BorrowStar
```

The query processor iterates over the set of stars in BorrowStar. The number of Borrow events is counted for each different value of Category. If one or more events has no associated Category element these events are counted and related to a default NULL element.

The following SQL-like expression illustrates how the star schema BorrowStar can be used for querying.

```
SELECT    COUNT(Borrow), KeyWord
FROM      BorrowStar
```

The query processor iterates over the set of stars in BorrowStar. The number of Borrow events is counted for each different value of KeyWord. If one or more events has no associated KeyWord element these events are counted and related to a default NULL element.

The following SQL-like expression illustrates how the multi-star schema defined by BorrowStar and ReturnStar can be used for querying.

```
SELECT    COUNT(Borrow), Type
FROM      BorrowStar, ReturnStar
WHERE     SameLoan(Borrow, Return)
```

The query processor iterates over the subset of the Cartesian product of the sets of stars in BorrowStar and ReturnStar. The number of Borrow events is counted for each different value of Type. If one or more events has no associated Type element these events are counted and related to a default NULL element. SameLoan is a function that returns true if the Borrow event and the Return event (the parameters) were involved in the same loan.

The following SQL-like expression illustrates how the star schema LibraryStar can be used for querying.

```
SELECT    COUNT(LibraryEvent), Type
FROM      LibraryStar
WHERE     EXISTS Author#
```

The query processor iterates over the stars in LibraryStar. For each different value of Type the number of LibraryEvent events that are related to an Author# are counted. If one or more events has no associated Type element the subset of these events that are related to an Author# are counted and related to a default NULL element. EXISTS returns true if the current star contains at least one Author# value.

7. CONCLUSION

We have presented the entity-based model language EVER. It supports the modeling of information structures by means of events, entities, relationships, and values. We have illustrated how EVER can be used to define data warehouses and information warehouses.

We are currently implementing a prototype based on the EVER model and its query and data manipulation language. Future work includes design of a path expression language that supports specification of filtering predicates and aggregation functions.

8. REFERENCES

[1] Bischoff, J. and T. Alexander, Data Warehouse, Practical Advice from Experts. 1997: Prentice Hall.

[2] Calvanese, D., et al. A Principled Approach to Data Integration and Reconcilation in Data Warehousing. in International Workshop on Design and Management of Data Warehouses (DMDW'99). 1999. Heidelberg, Germany.

[3] Devlin, B., Data Warehouse. From Architecture to Implementation. 1997: Addison-Wesley.

[4] Mattison, R., Data Warehousing: Strategies, Technologies, and Techniques. 1996, Englewood Cliffs, New Jersey: Prentice-Hall.

[5] Poe, V., P. Klauer, and S. Brobst, Building a Data Warehouse for Decision Support. 2 ed. 1998, Englewood Cliffs, New Jersey: Prentice-Hall.

[6] Schouten, H. Analysis and Design of Data Warehouses. in International Workshop on Design and Management of Data Warehouses (DMDW'99). 1999. Heidelberg, Germany.

[7] Golfarelli, M. and S. Rizzi. A Methodological Framework for Data Warehouse Design. in DOLAP'98. International Workshop on Data Warehousing and OLAP. 1998. Washington, D.C., USA.

[8] Pedersen, T.B. and C.S. Jensen. Multidimensional Data Modeling for Complex Data. in Internationel Conderende on Data Engineering. 1999. Sydney, Australia: IEEE Computer Society.

[9] Franconi, E. and U. Sattler. A Data Warehouse Conceptual Data Model for Multi-Dimensional Aggregation. in International Workshop on Design and Management of Data Warehouses (DMDW'99). 1999. Heidelberg, Germany.

[10] Chen, P.P., The Entity-Relationship Model - Towards a Unified View of Data. ACM Transactions of Database Systems, 1976. **1**: p. 9-36.

[11] Shipman, D.W., The Functional Data Model and the Data Language DAPLEX. ACM Transactions on Database Systems, 1981. **6**(1): p. 140-173.

[12] Hammer, H. and D. McLeod, Database Description with SDM. A Semantic Database Model. ACM Transactions of Database Systems, 1981. **6**(3): p. 351-386.

[13] Snodgrass, R. and I. Ahn, Temporal Databases. Computer, 1986. **19**(9): p. 35-42.

[14] Bertino, E. and W. Kim, Indexing Techniques for Queries on Nested Objects. IEEE Transactions on Knowledge and Data Engineering, 1989. **1**(2): p. 96-214.

[15] Blakeley, J.A., OQL(C++): Extending C++ with an Object Query Capability, in Modern Database Systems. The Object Model, Interoperability, and Beyond, W. Kim, Editor. 1995, Addison-Wesley. p. 69-107.

[16] Ioannidis, Y.E. and Y. Lashkari. Incomplete Path Expressions and their Disambiguation. in SIGMOD'94. International Conference on Management of Data. 1994. Minneapolis, Minnesota, USA.

[17] Kemper, A. and G. Moerkotte, Physical Object Management, in Modern Database Systems. The Object Model, Interoperability, and Beyond, W. Kim, Editor. 1995, Addison-Wesley. p. 175-202.

[18] Stonebraker, M., J. Anton, and E. Hanson, Extending a Database System with Procedures. ACM Transactions on Database Systems, 1987. **12**(3): p. 350-376.

[19] Özsu, M.T. and J.A. Blakeley, Query Processing in Object-oriented Database Systems, in Modern Database Systems. The Object Model, Interoperability, and Beyond, W. Kim, Editor. 1995, Addison-Wesley. p. 146-174.*