Heft 100

Peter Loos

Representation of Data Structures Using the Entity Relationship Model and the Transformation in Relational Databases

January 1993

Representation of Data Structures Using the Entity Relationship Model and the Transformation in Relational Databases

Peter Loos
Institut für Wirtschaftsinformatik -IWi
Universität des Saarlandes, Postfach 151150, D-66041 Saarbrücken
phone +49/681/302-3106, fax +49/681/302-3696
e-mail: loos@iwi.uni-sb.de, http://www.iwi.uni-sb.de

Publications of the Institut für Wirtschaftsinformatik, paper 100, Saarbrücken, January 1993

Contents

1. Introduction

This paper is dedicated to support data modelling in the ESPRIT project CAPISCE (Computer Architecture for Production Information Systems in a Competitive Enviroment). The CAPISCE project aims at developing a system architecture and information tools to support and integrate company operations at strategic, tactical and operational levels in the process industries. A main task in this project is the development of general purpose data structures for the architecture mentioned above.

This paper should help to get a common understanding on data modelling methods to use for the practical work. It describes how to use the method Entity Relationship Model (ERM), gives comments on special modelling questions and describes how data structures can be transformed into a relational data base using the SQL language.

2. The ERM Concepts

The foundations of the Entity Relationship Model can be traced back to 1976 (*Chen 76*). It is one of the most popular methods for data modelling. However, the ERM has since then been extended and refined by several authors. A survey is given in (*Loos 92a*) and (*Hars et al. 91*). Below, the key concepts of ERM are introduced.

2.1 Basic Concepts

The ERM distinguishes between the basic concepts entities, attributes, and relationships.

- Entities are real or abstract things which are relevant to an enterprise, e.g., customers, articles, orders. Entites can be described by properties. If entities can be described by the same kind of properties, they are regarded as sets and referred to as entity types, their individual instances being the entities. Entity types are represented in the diagram of the ERM by using rectangles (cf. figure 1).

- Relationships are logical connections between two or more entities. Although entities can exist in their own right, relationships can only be defined in combination with the relevant entities. Therefore relationships can be regarded as aggregations of entities. A set of relationships connecting entities of the same entity type is called relationship type.

Between the same entity types several different relationship types can exist. A relationship type is graphically represented by a diamond, which is connected to the relevant entity types with single lines. A relationship type must have at least two connection lines.

- Attributes are the properties of entities or relationships, e.g. customer number, name or address. An attribute is assigned to a particular domain, e.g. a certain data type. Both entity types and relationship types can carry attributes. Attributes are usually listed in a circle or an oval which is connected to the relevant entity type or relationship type.
  Some attributes of an entity type or of a relationship type are grouped together forming a key. A key (or identifier or primary key) is the identifier for all entities, or relationships of a type. One key instance is assigned to at most one entity (relationship) of a type. Key attributes are underlined in an ERM.

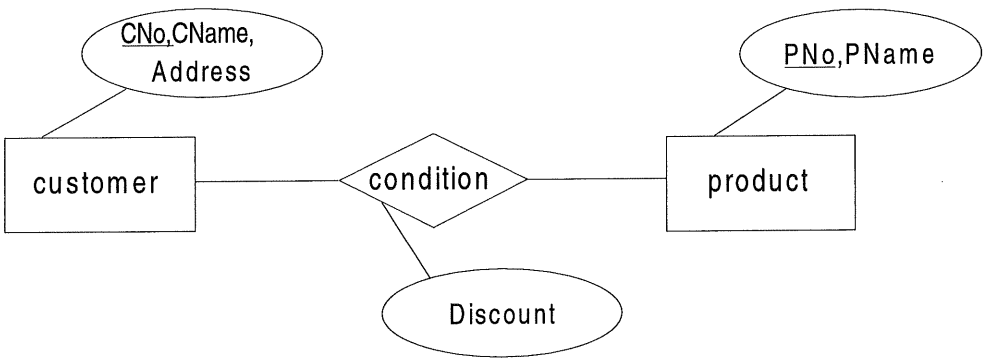

Fig. 1: Simple entity relationship diagram

Beside the aboved mentioned graphical representations sometimes other symbols are used. The method NIAM, for instance, (*Verheijen/VanBekkum 82*) uses circles for entity types (called non-lexical object types), dotted circles for attributes (called lexical object types) and rectangles for relationships (called facts), the method SAM* (*Su 85*) represents both entity types and relationship types by circles.

Figure 4 shows the graphical representations of the methods ERM (*Chen 76*) ARIS-ERM (*Scheer 92*) and PERM (*Loos 92a*), SERM (*Sinz 87*), NIAM (*Verheijen/VanBekkum 82*), the methods of the CASE tools ADW and Teamwork and further variations.

2.2 Specifying Relationships

When describing data structures one of the main tasks is to model the relations and connections between the entity types. Therefore relationship types can be further specified. The most common classification criteria are the degree of the relationship and its cardinality.

2.2.1 Degrees of Relationships

The degree of a relationship determines how many entities are aggregated within the relationship type, which is shown by the number of connection lines of this relationship type. All relationships of one relationship type have the same degree.

Some methods only allow to use relationships with two connection lines, so-called binary relationship models. Examples are the early versions of NIAM or the method used in the CASE tool ADW. So-called n-ary relationship models have no fixed limits of connection lines per relationship type, e.g. ARIS-ERM, PERM or ECRM (*Elmasri et al. 85*). Figure 2 a shows a ternary relationship *condition* where the condition depends on the customer, the product and the month.

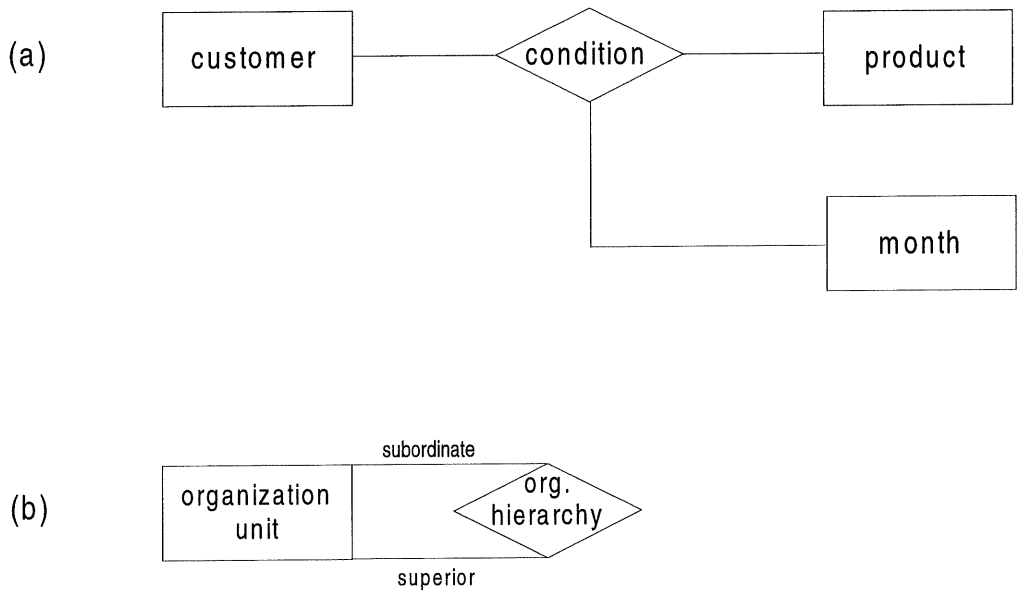

Fig. 2: Ternary relationship (a) and recursive binary relationship (b)

Although relationship types mostly connect to several different entity types, they can also connect more than once the same entity type. These relationships are called recursive

relationships. The significance of these relationships is to connect two or more entities of the same entity type, even the connection of an entity with itself is possible. A typical example of this kind of relationship is the organizational hierarchy between organizational units. To distinguish the connection lines between the same entity type and relationship type it is reasonable to assign role names (cf. figure 2 b).

2.2.2 Cardinalities of Relationships

Relationship types can be distinguished by the number of possible relationships a particular entity can have concerning the regarded relationship type. This range of numbers is called cardinality. In most methods the minimum and maximum number of relationships can be specified. Typically, a lower boundary of zero or one and an upper boundary of one or many are distinguished. If the lower boundary is zero, the relationship is optional.

Cardinalities are normally denoted at the connection lines. Thus, a relationship type has a number of cardinalities according to its degree. In the various methods cardinalities are expressed by numerous different variations of notations. One usual notation is the (min,max)-notation, in which the minimum and maximum values express the boundaries, e.g. (2,15) mean that each entity must have at least two relationships, but can have at most 15 relationships. While the (min,max)-notation allows any number as boundaries, the (1,c,m)-notation allows only the four boundaries mentioned above. Many CASE tools use the (1,c,m)-notation or variations of it. Figure 3 compares the (min,max)-notation with the (1,c,m)-notation and the variations used in the CASE tool Teamwork.

(min,max)-notation	(1,c,m)-notation	Teamwork-notation
0,1	c	0<=N<=1
1,1	1	1
0,n	cm	0<=N
1,n	m	N

Fig. 3: Comparison of (min,max)-notation and (1,c,m)-notation

In some variations of the (1,c,m)-notation graphical symbols are used, e.g. upper boundaries of "many" are depicted as "crowfeet", as half shadowed diamonds (*Teorey et al. 86*), as small

boxes (SERM) or as double arrows (CASE tool ADW). Figure 4 shows some examples of different notations.

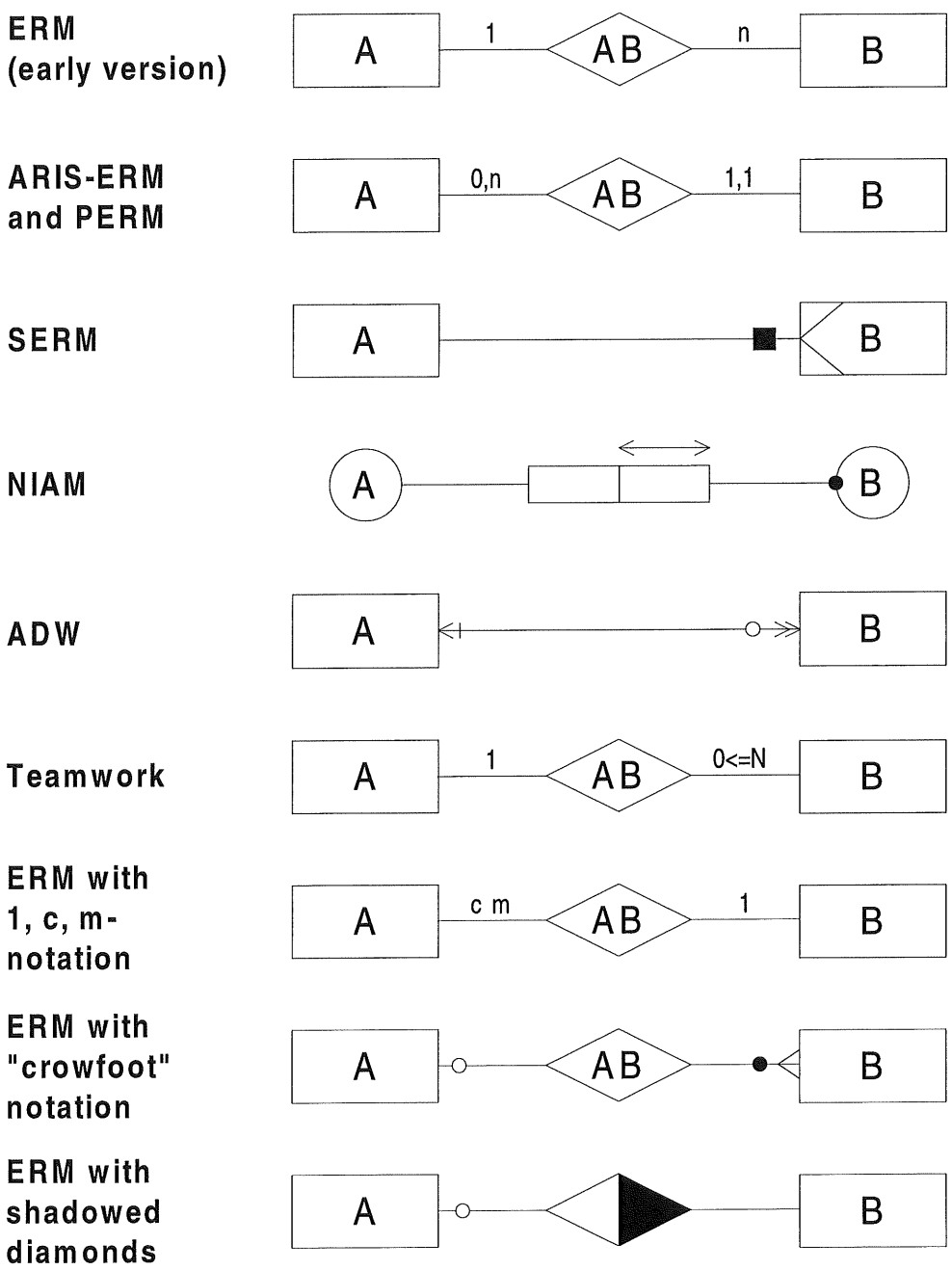

Fig. 4: Examples of notations of cardinality

Unfortunately, the side of the relationship type at which the cardinality is denoted is not consistent between the different notations (cf. figure 4, in which all notations are representing the same facts).

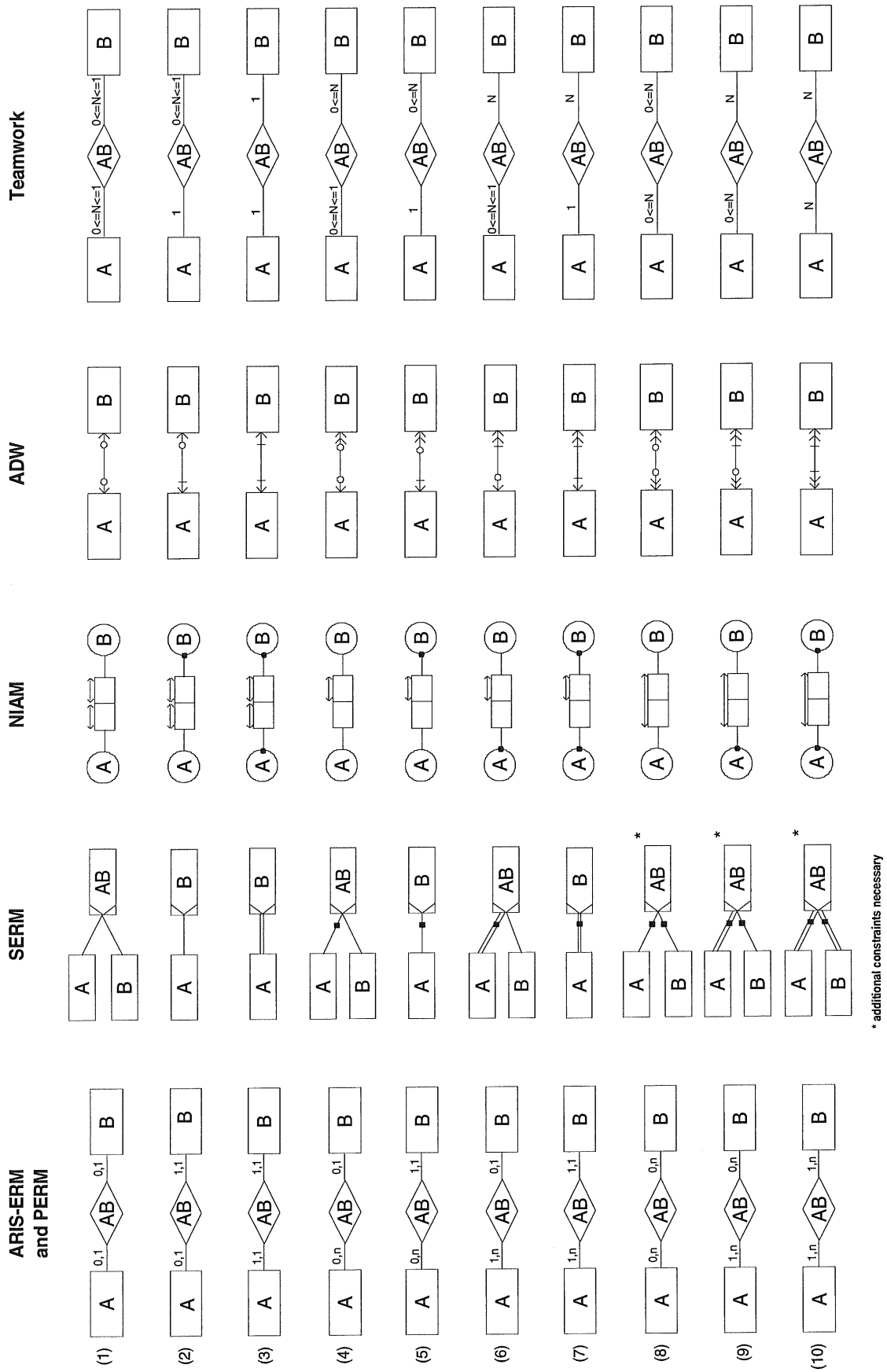

Fig. 5: Binary relationship types

While in the early version of ERM, in ADW or in Teamwork the cardinality has to be read as "an entity of A can have n instances of B assigned to", in ARIS-ERM, in PERM or in ERM with (1,c,m)-notation it has to be read "an entity of A can have n assignments concerning the relationship type AB". The second kind is more unequivocal, especially for n-ary relationships.

The binary relationship types can be differentiated into 10 distinctive kinds, when cardinalities with a lower boundary of zero or one and those with an upper boundary of one or "n" are considered[1]. Figure 5 gives an overview. These representations correspond to the (1,c,m)-notation.

As far as recursive binary relationships are concerned, only seven different types can be distinguished. Three out of the ten theoretical combinations cannot have valid instances with a limited number of entities, namely (0,1) and (1,1), (0,1) and (1,n) as well as (1,1) and (1,n). The possible combinations are shown in figure 6.

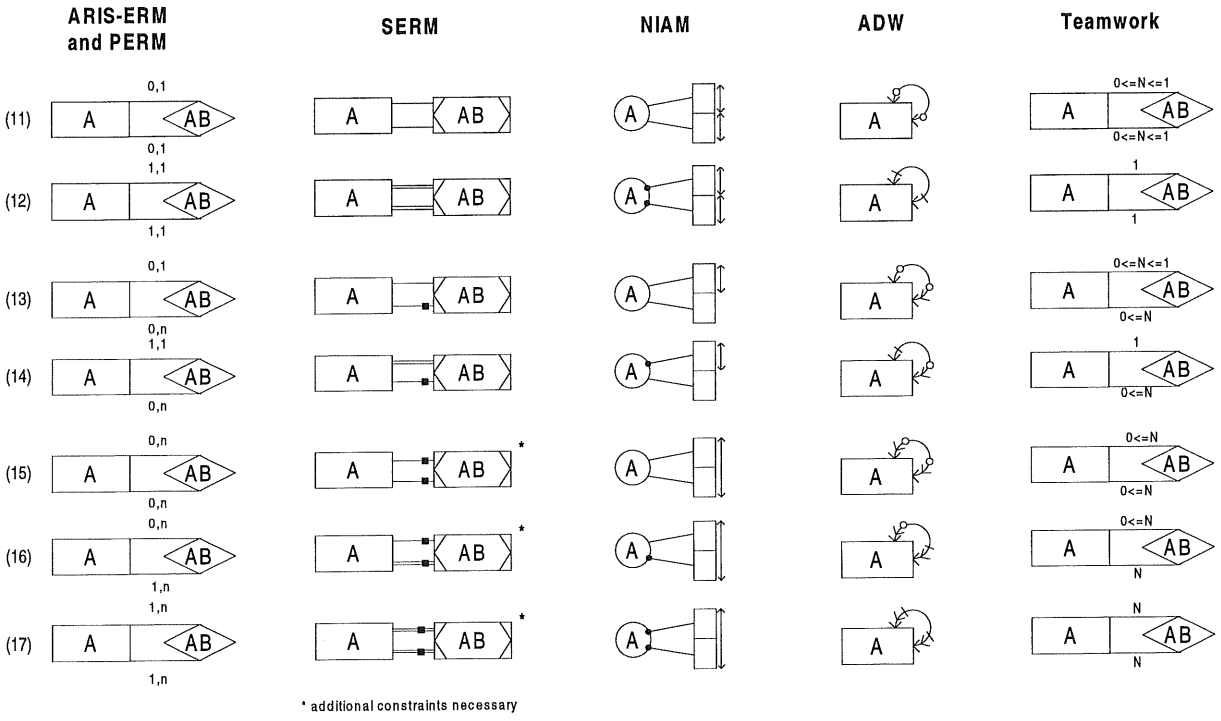

Fig. 6: Recursive binary relationship types

1) The possible number is a combination with repetition of 4 element of the order of 2, $Cr_4^{(2)} = 10$.

2.2.3 Properties of Relationships

As stated before, usually both entity types and relationship types methods can have attributes for their description. However, some methods do not allow attributes at relationship types, especially some binary relationship models, where relationship types have no own graphical representation, so that the two aggregated entity types are directly connected by a single line, e.g. the method of ADW (cf. figure 4). In that case, attributes concerning the relationships have to be assigned to one of the aggregated entity types or a new entity type has to be introduced (for more detailed information refer to chapter '4.1 Attributes of Relationships').

2.3 Reinterpretation of Relationships

Relationship types result from an aggregation of one or several entity types. In the subsequent process of modelling the relationship type can be used as a component for additional relationship types. In order to provide the reuse of relationship types in relationships their reinterpretation as an entity type has been introduced (*Schlageter/Stucky 83*) and included into the ERM-diagram as a diamond surrounded by a rectangle (*Webre 83*). Scheer stresses the necessity to reveal the modelling process behind the reinterpreted relationship type (*Scheer 89*). Therefore, the connecting lines coming from the aggregated entity types should touch the diamond, whereas the connecting lines coming from the reinterpreted relationship type should start from the edges of the rectangle (cf. figure 7 a). The same fact is represented by a rectangle also surrounding the entity types (*Briand et al. 88, Put 88*) as shown in figure 7 b. By circuling a fact NIAM uses the same concepts for reinterpretation as ARIS-ERM and PERM do (cf. figure 7 c). In the method for reinterpretation provided by Teamwork a new entity type has to be introduced (cf. figure 7 d).

2.4 Generalization

Generalization serves for uniting several sets of different classes (which can be described by some common attributes) into a superset. Smith/Smith defined the generalization as: "A generalization is an abstraction which enables a class of individual objects to be thought of generically as a single named object" (*Smith/Smith 77, p. 107*). An example for a generalization is the aggregation of the two entity types *customer* and *supplier* to the new entity type *business partner*. In that example, *business partner* is called the generic type or the super-class entity type, whereas the entity types *customer* and *supplier* are called sub-class entity types or subtypes.

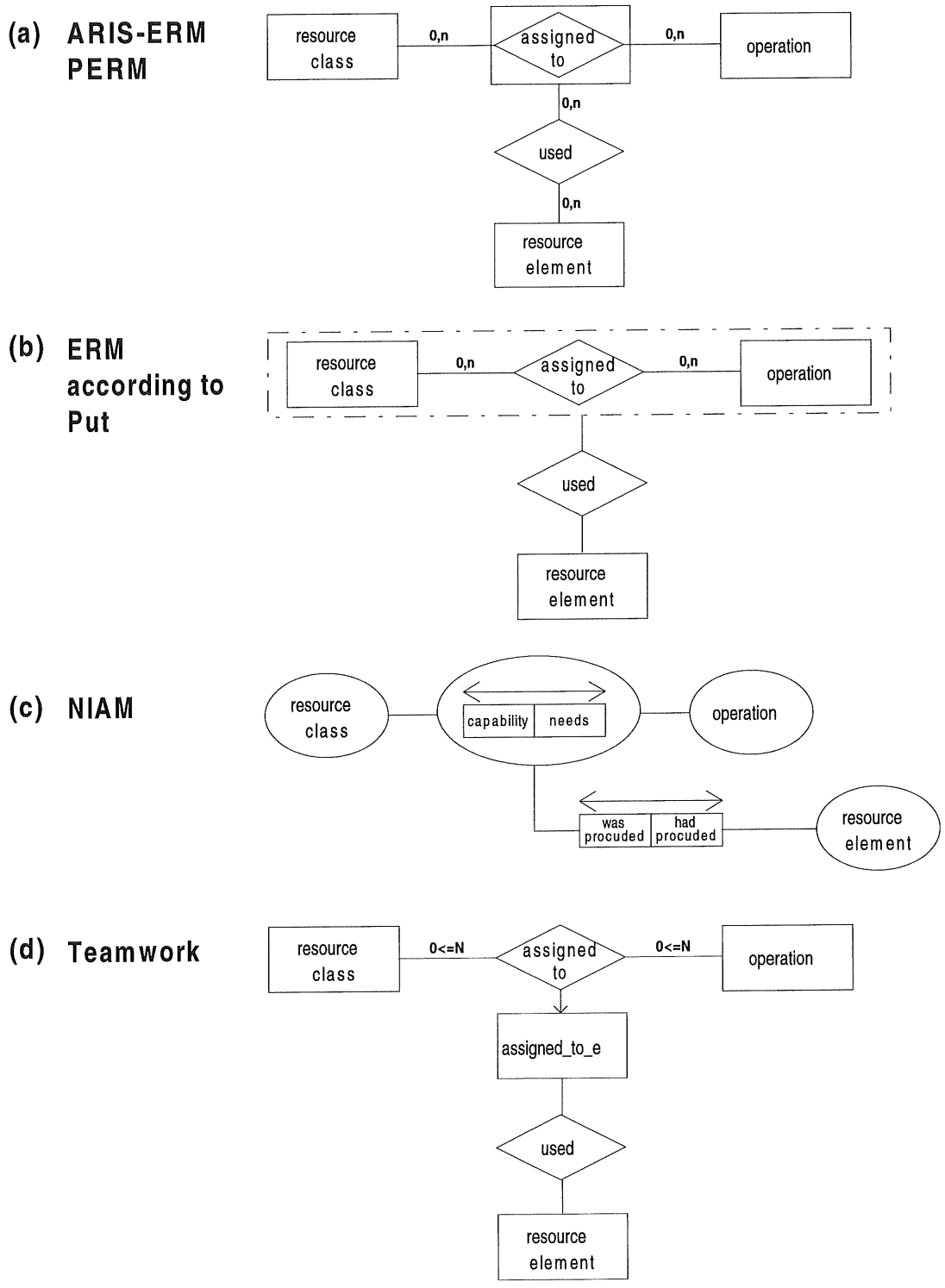

Fig. 7: Graphical representations from reinterpretations

Generalization is used for the purpose of:

- Common attributes: The sub-class entity type may have the same attributes. In this case these attributes can be defined in the super-class entity type. The sub-class entity types inherit all attributes of the super-class. In the above mentioned example this might be all attributes which are of interest for both customer and supplier, e.g. *company name*, *address*, *contact person*, etc., while supplier and customer specific attributes are assigned to the sub-classes.

- Common relationships: The sub-class entity types may have the same relationships to other entity types. In that case the relationships are assigned to the super-class entity type. All sub-class entity types inherit these relationships. In the example above the relationship to an entity type *bank account* should be assigned to *business partner*, because both customer and supplier have bank accounts.

Smith and Smith had already pointed out that the generalization can take different forms. Characteristics are the disjunction of the participating sets and the completeness of the generalization. Figure 8 shows the different characteristics.

characteristics of generalization	disjunctive $S_i \cap S_j = \emptyset \; \forall \; i,j \in (1,2..n) \wedge i \neq j$	non-disjunctive $S_i \cap S_j \neq \emptyset \; \exists \; i,j \in \{1,2..n\} \wedge i \neq j$	
		subset $S_i \subset S_j$	intersection $S_i \setminus S_j \neq \emptyset \wedge S_j \setminus S_i \neq \emptyset$
complete $G = S_1 \cup S_2 \cup \ldots \cup S_n$	(1)	(3a)	(3b)
non-complete $G \supset S_1 \cup S_2 \cup \ldots \cup S_n$	(2)	(4a)	(4b)

G: set of super-class, S: set of sub-class

Fig. 8: Characteristics of generalization

In disjunctive subsets of a generalization all "n" subsets have to be disjunctive when compared in pairs (mutually exclusive). If only one pair is non-disjunctive, then one set can be a subset of the other or be overlapping it partially. A subclass being a complete subset of another can be considered to be a generalization within the generalization. By means of this hierarchical approach the originally non-disjunctive generalization can be gradually transformed to be disjunctive in pairs. As a rule, all entities of the super-class of a complete generalization have

to occur in at least one of its subclasses, whereas this has not to be the case in a non-complete generalization. It has to be observed that every entity of a subtype exists as an entity of the super-class regardless of the kind of generalization.

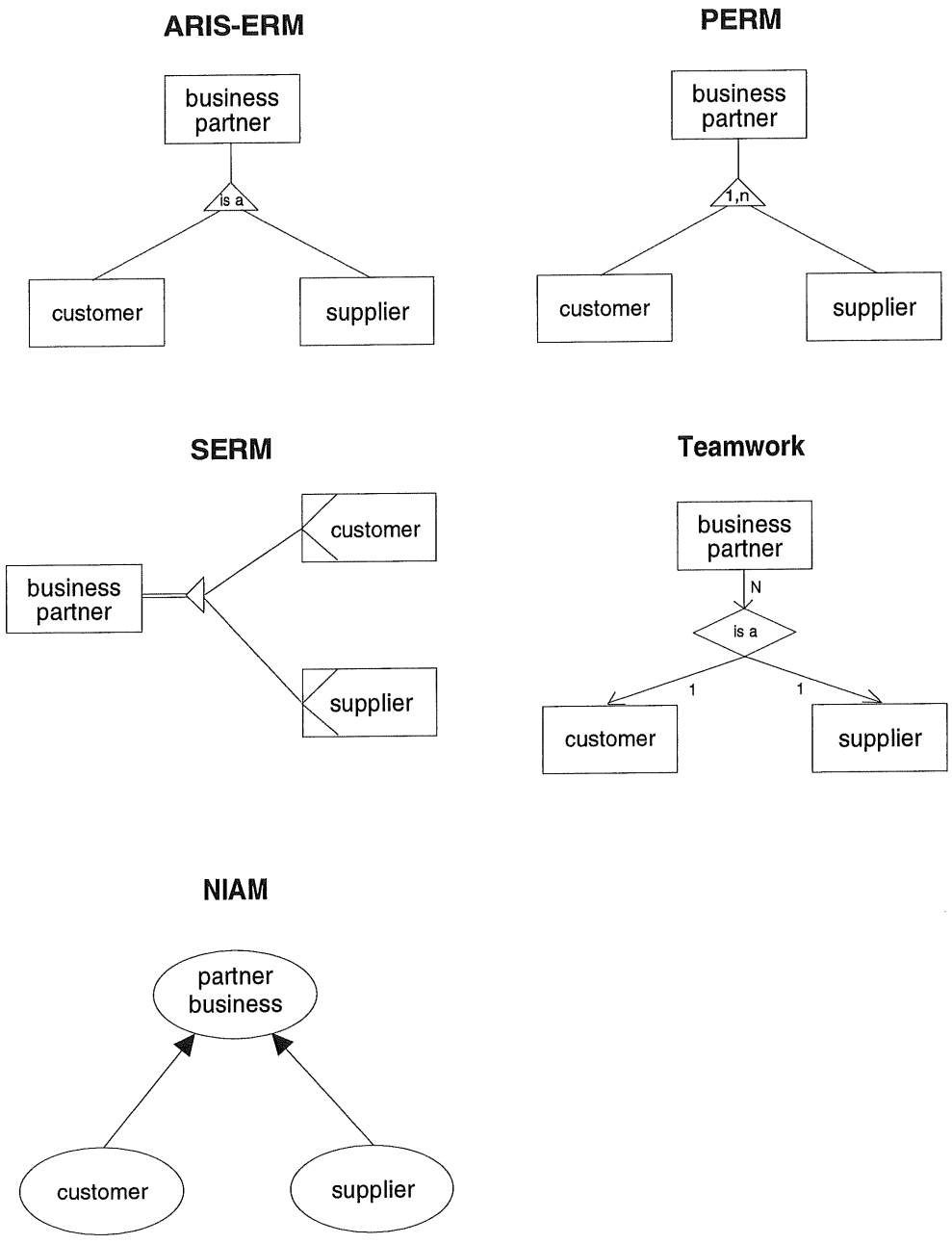

Fig. 9: Examples for notations of generalization

Generalizations are mostly represented in the ERM by a triangle connecting the generic type and the subtypes, some methods use diamonds, hexagons or arcs. Figure 9 shows the example of business partners mentioned above. In some methods, notations of the characteristics of generalizations are introduced. In PERM the specification is located within the triangle, using

a notation similar to the cardinality of relationships. The lower boundary expresses the completeness, where it has to be read as "an entity of the generic type must occur 0 (non-complete) or 1 (complete) times in a subtype". The upper boundary expresses the disjunctiveness, where it has to be read as: "an entity of the generic type must occur 1 time (disjunctive) or n times (non-disjunctive) in a subtype". This leads to the following notations (cf. figure 8):

- type (1) ⇒ (1,1)
- type (2) ⇒ (0,1)
- type (3) ⇒ (1,n)
- type (4) ⇒ (0,n)

In Teamwork an analogous notation can be used to denote the characteristics at the arc coming from the generic type (type (1): 1, type (2): 0<=N<=1, type (3): N, type (4): 0<=N). A triangle with a double line connection is used in SERM to represent a complete, non-disjunctive generalization.

The view of the abstraction from another perspective, i.e. from the generic entity type down to the subtypes, is called specialization (*Scheer 89*).

3. Transformation into SQL

3.1 Concepts of Representation within SQL-Databases

Relational database systems, data description languages and data manipulation languages which follow the ISO-SQL-standard[2] (*ISO 89* and *ISO 91*) offer five concepts to describe integrity constraints (*Loos 91*). Concepts having an impact on single tables are designated as intra-relational constraints, whereas concepts referring to several tables are called inter-relational constraints.

(1) Data Type Integrity
According to the schema definition, data type integrity means that an attribute can take only one value from the domain of a data type. These constraints refer to declarations of data types well-known from any higher programming language. Alphanumeric types

2) SQL from 1989 is currently the international standard. SQL2 is so far only a draft (*ISO 91*). The release to a international standard is soon expected.

(CHARACTER), exact numeric types (NUMERIC, DECIMAL, INTEGER) and approximate-numerical types (FLOAT, REAL, DOUBLE PRECISION) are provided by the ISO-standard according to *ISO 89*. Furthermore, SQL2 introduced data types like DATE, TIME, TIMESTAMP, INTERVAL, VARCHAR and BIT. Within some database management systems additional types like BOOLEAN, MONEY or BLOB (binary large oject) are implemented.

Example:

```
CREATE TABLE rel1 (field1 NUMERIC, field2 CHAR(20), ...)
```

The domain of a data field can be more restricted in SQL2. For that a DOMAIN can be defined which can be used like original data types in further table definitions.

Example:

```
CREATE DOMAIN field1_type as NUMERIC(5),
DEFAULT NULL,
CONSTRAINT field1_type_check (CHECK ( VALUE >= 10000))

CREATE TABLE rel1 (field1 field1_type, field2 CHAR(20), ...)
```

(2) NULL Values

As initial value or as a value for an unknown instance, NULL values (cf. *Wedekind 88*) in SQL-databases represent the characterization of an undefined value. NULL values differ from the numerical zero or empty strings. By means of the NOT NULL quotation within the schema definition, it can be excluded that a data record takes an undefined value regarding the characterized attribute. Since an undefined value can occur, a three-valued logic has to be introduced (cf. figure 10).

AND	t	n	f
t	t	n	f
n	n	n	f
f	f	f	f

OR	t	n	f
t	t	t	t
n	t	n	n
f	t	n	f

NOT	
t	f
n	n
f	t

t: true, n: undefined/NULL, f: false

Fig. 10: Three-valued logic

Example:

```
CREATE TABLE rel1 (field1 NUMERIC NOT NULL, ...)
```

(3) UNIQUE- and PRIMARY KEY-Constraint

With the help of UNIQUE-constraints an attribute or a combination of attributes can be defined as single-valued within a relation, i. e. each instance may exist only once within the relation. The UNIQUE-constraint implicitly contains the NOT NULL quotation, so that a defined instance of the attribute (or combination of attributes) is imperative. Due to that single-valuedness UNIQUE-attributes possess primary key properties; they are also called key candidates.

Several UNIQUE-constraints can be defined within a table. An attribute can also occur in several UNIQUE-attribute combinations. A UNIQUE-constraint can be defined as PRIMARY KEY. Unlike the UNIQUE-constraint a PRIMARY KEY can be defined only once in each table. In SQL2 only the PRIMARY KEY has the NOT NULL-implication (*Loos 92b*).

Example:

```
CREATE TABLE rel1 (field1 INT, field2 INT, field3 INT, field4 INT,...
PRIMARY KEY (field1), UNIQUE (field2, field3),
UNIQUE (field3, field4), ...)
```

(4) CHECK-Constraint

Additional integrity constraints can be introduced using the CHECK-statement. CHECK-constraints can be defined for all attributes of a table. In analogy to the SELECT-statement they may contain search arguments. Sub-queries and set-functions, however, are not permitted. Only SQL2 will also allow subqueries (*Loos 92b*). Thus, in SQL acccording to *ISO 89* the sphere of activity of the integrity constraint is restricted to the table in which it has been specified, as well as to the respective data record. Typical applications of the CHECK-constraint are the definition of the value area of an attribute or the comparison of attributes.

Example:

```
CREATE TABLE rel1 (field1 INT, field2 INT, field3 INT, field4 INT,...
CHECK (field2 BETWEEN 2000 AND 5000),
CHECK (field3 < field4) ...)
```

Independently from a CREATE TABLE statement in SQL2 CHECK-constraints can be defined with the CREATE ASSERTION statement.

Example:

```
CREATE ASSERTION check_1 CHECK (
  ( SELECT field1, field2 FROM rel1) IN
  ( SELECT field3, field4 FROM rel2) )
```

(5) FOREIGN KEY-Constraint

With the help of the FOREIGN KEY-statement referential integrity can be guaranteed. Here, the instance of an attribute or a combination of attributes is dependent on the existance of the same instance within another table. For example, in the machine master data record that cost unit can be noted, the machine is assigned to. But it is also demanded that the given cost unit is actually defined within the cost unit master data. However, by defining a FOREIGN KEY-statement only attributes or combinations of them from foreign tables can be referenced which have been defined as UNIQUE and whose data types correspond to those of the attributes of the referencing tables.

Example:

```
CREATE TABLE cost_unit (CostNo INT, ...
PRIMARY KEY (CostNo), ...)

CREATE TABLE Machine (MNo INT, Cost_unit INT, ...
PRIMARY KEY (MNo),
FOREIGN KEY (Cost_unit) REFERENCES Cost_unit (CostNo), ...)
```

While in SQL according to *ISO 89* the CHECK-constraint (concept 4) is an intra-relational constraint because of the subquery permission it can also be an inter-relational constraint in SQL2.

3.2 Entity types and Relationships

While treating the transformation into tables, the subject of segmentation of data structures which takes place in the course of normalization of the entity-relationship-model (*Codd 70; Date 81; Kent 83*) shall not be gone too deeply into. If only single-valued or scalar attributes are accepted in the process of modelling, then each object is in first normal form (1NF). Due to the attributation of object types their segmentation into smaller ones can be necessary in order to eliminate dependencies on partial keys (second normal form, 2NF) or transitive dependencies (third normal form, 3NF).

An entity type can directly be transformed into a table. Its columns represent the describing attributes of the entity type. The identifying attribute of the entity type forms the primary key of the table. In principle, each entity type is transformed into a table, although in particular cases several entity types can form a single table due to the kind of the linking relationship type (for transformation of ERM into relational data bases see also *Wong/Katz 80; Teorey et al. 86; Markowitz/Shoshani 89; Scheer 89; Batra et al. 90; Loos 92a*).

Relationship types are also transformed to tables. Depending on the kind of the relationship type, creation of own tables or the combination with a table of an entity type takes place. Consequently, tables can have the quality of a relationship type or of an entity type. In the course of the transformation the following rules have to be taken into account:

- The table receives the attribute(s) (or combination of attributes, respectively) as primary key which corresponds to the identifying attribute(s) of the relationship.

- All attributes which are inherited from tables of the linked entity types are foreign keys and have references to the primary keys of the tables by means of referential constraints.

- Different types (i.e. entity types and relationship types) may be comprised in a table if they bear the same primary key.

- In case of the inclusion of different types in a table all attributes of all types have to be included.

- When transforming a relationship type into an individual table, each row of the table has to reference all of the involved entity types. This is ensured by the implicit NOT NULL condition for the primary key or the UNIQUE condition for foreign keys in the table of the relationship type.

- If an entity type is transformed into a table and has connections to relationship types where the lower boundary of the cardinality is greater zero (non-optional relationship), then the primary key of the table is also the foreign key. Since a foreign key can only reference primary keys or UNIQUE attributes, the feasibility of the representation depends on the referenced table.

- If a table results from the integration of an entity type and a relationship type, and the relationship is optional (i.e. the lower boundary of the cardinality is equal zero), then the foreign key referencing the other table can carry the NULL value.

Consequence of the above mentioned rules is that relationship types resulting from binary relations can only be directly transformed into a table when the cardinalities of both connections bear an upper-boundary-value of greater than one. In all other cases the relationship type may be represented in the table of an entity type. Below, the different kinds of relationships (as shown in figure 5 and figure 6) are represented as tables[3]. If an entity type or relationship type is not represented by an own table, another table containing the describing attributes is mentioned.

(1)
```
CREATE TABLE A( a, ...,
PRIMARY KEY (a))

CREATE TABLE B( b, ...,
PRIMARY KEY (b))

CREATE TABLE AB( a, b UNIQUE, ...,
PRIMARY KEY (a), FOREIGN KEY (a) REFERENCES A(a),
FOREIGN KEY (b) REFERENCES B(b))
```

(2)
```
CREATE TABLE A( a, ...,
PRIMARY KEY (a))

CREATE TABLE B( b, a UNIQUE, ..., AB-attributes...,
PRIMARY KEY (b), FOREIGN KEY (a) REFERENCES A(a))
```

(3)
```
CREATE TABLE A( a, b UNIQUE, ..., B-attributes..., AB-attributes...,
PRIMARY KEY (a))
```

(4)
```
CREATE TABLE A( a, ...,
PRIMARY KEY (a))

CREATE TABLE B( b, ...,
PRIMARY KEY (b))

CREATE TABLE AB( b, a NOT NULL, ...,
PRIMARY KEY (b), FOREIGN KEY (a) REFERENCES A(a),
FOREIGN KEY (b) REFERENCES B(b))
```

or, especially if AB has no attributes:

```
CREATE TABLE A( a, ...,
PRIMARY KEY (a))

CREATE TABLE B( b, a, ..., AB-attributes...,
PRIMARY KEY (b), FOREIGN KEY (a) REFERENCES A(a))
```

---

3) The CREATE-statements confirm to ISO-SQL, but data types are not indicated.

(5)
```
CREATE TABLE A( a, ...,
PRIMARY KEY (a))

CREATE TABLE B( b, a NOT NULL, ..., AB-attributes...,
PRIMARY KEY (b), FOREIGN KEY (a) REFERENCES A(a))
```

(6) Exact transformation not possible, best transformed as in (4). Only in SQL2:

```
CREATE TABLE A( a, ...,
PRIMARY KEY (a)
CHECK a IN (SELECT a FROM AB))

CREATE TABLE B( b, ...,
PRIMARY KEY (b))

CREATE TABLE AB( b, a NOT NULL, ...,
PRIMARY KEY (b), FOREIGN KEY (a) REFERENCES A(a),
FOREIGN KEY (b) REFERENCES B(b))
```

(7) Exact transformation not possible, best transformed as in (5). Only in SQL2:

```
CREATE TABLE A( a, ...,
PRIMARY KEY (a)
CHECK a IN (SELECT a FROM B))

CREATE TABLE B( b, a NOT NULL, ..., AB-attributes...,
PRIMARY KEY (b), FOREIGN KEY (a) REFERENCES A(a))
```

(8)
```
CREATE TABLE A( a, ...,
PRIMARY KEY (a))

CREATE TABLE B( b, ...,
PRIMARY KEY (b))

CREATE TABLE AB( a, b, ...,
PRIMARY KEY (a,b), FOREIGN KEY (a) REFERENCES A(a),
FOREIGN KEY (b) REFERENCES B(b))
```

(9) Exact transformation not possible, best transformed as in (8). Only in SQL2:

```
CREATE TABLE A( a, ...,
PRIMARY KEY (a), CHECK a IN (SELECT a FROM AB))

CREATE TABLE B( b, ...,
PRIMARY KEY (b))

CREATE TABLE AB( a, b, ...,
PRIMARY KEY (a,b), FOREIGN KEY (a) REFERENCES A(a),
FOREIGN KEY (b) REFERENCES B(b))
```

(10) Exact transformation not possible, best transformed as in (8). Only in SQL2:

```
CREATE TABLE A( a, ...,
PRIMARY KEY (a), CHECK a IN (SELECT a FROM AB))

CREATE TABLE B( b, ...,
```

```
PRIMARY KEY (b), CHECK b IN (SELECT b FROM AB))

CREATE TABLE AB( a, b, ...,
PRIMARY KEY (a,b), FOREIGN KEY (a) REFERENCES A(a),
FOREIGN KEY (b) REFERENCES B(b))
```

(11)
```
CREATE TABLE A( a, ...,
PRIMARY KEY (a))

CREATE TABLE AB( a1, a2 UNIQUE, ...,
PRIMARY KEY (a1), FOREIGN KEY (a1) REFERENCES A(a),
FOREIGN KEY (a2) REFERENCES A(a))
```

Since NULL values in UNIQUE-attributes are allowed in SQL2, the following transformation is also possible, especially if AB has no attributes:

```
CREATE TABLE A( a1, a2 UNIQUE, ..., AB-attributes...,
PRIMARY KEY (a1), FOREIGN KEY (a2) REFERENCES A(a1))
```

(12)
```
CREATE TABLE A( a1, a2 UNIQUE, ..., AB-attributes...,
PRIMARY KEY (a1), FOREIGN KEY (a2) REFERENCES A(a1))
```

(13)
```
CREATE TABLE A( a, ...,
PRIMARY KEY (a))

CREATE TABLE AB( a1, a2, ...,
PRIMARY KEY (a1), FOREIGN KEY (a1) REFERENCES A(a),
FOREIGN KEY (a2) REFERENCES A(a))
```

or, especially if AB has no attributes:

```
CREATE TABLE A( a1, a2, ..., AB-attributes...,
PRIMARY KEY (a1), FOREIGN KEY (a2) REFERENCES A(a1))
```

(14)
```
CREATE TABLE A( a, ...
PRIMARY KEY (a), FOREIGN KEY (a) REFERENCES AB(a1))

CREATE TABLE AB( a1, a2, ...,
PRIMARY KEY (a1), FOREIGN KEY (a1) REFERENCES A(a),
FOREIGN KEY (a2) REFERENCES A(a))
```

(15)
```
CREATE TABLE A( a, ...,
PRIMARY KEY (a))

CREATE TABLE AB( a1, a2, ...,
PRIMARY KEY (a1,a2), FOREIGN KEY (a1) REFERENCES A(a),
FOREIGN KEY (a2) REFERENCES A(a))
```

(16) Exact transformation not possible, best transformed as in (15). Only in SQL2:

```
CREATE TABLE A( a, ...,
PRIMARY KEY (a), CHECK a IN (SELECT a FROM AB))

CREATE TABLE AB( a1, a2, ...,
PRIMARY KEY (a1,a2), FOREIGN KEY (a1) REFERENCES A(a),
FOREIGN KEY (a2) REFERENCES A(a))
```

(17) Exact transformation not possible, best transformed as in (15). Only in SQL2:

```
CREATE TABLE A( a, ...,
PRIMARY KEY (a), CHECK a IN (SELECT a FROM AB),
CHECK a IN (SELECT b FROM AB))

CREATE TABLE AB( a1, a2, ...,
PRIMARY KEY (a1,a2), FOREIGN KEY (a1) REFERENCES A(a),
FOREIGN KEY (a2) REFERENCES A(a))
```

In some recursive binary relationships the foreign key constraint has to point to its own table since both the relationship type and the entity type are represented in one table (cf. type (12) and (13)).

Types (6) and (7) cannot be transformed by using SQL according to *ISO 89* because the lower boundary values of one of the cardinalities require foreign key relations from the entity type tables to the tables of the unified entity types and relationship types. These foreign key relations cannot be represented since the corresponding attribute *a* in the tables B can exist more than once. The types (9), (10), (16), (17) are not transformable as a consequence of attribute *b* (and *a1*, respectively) which do not independently form a primary key for table AB but only in combination with the attribute *a* (and *a2*, respectively). Using SQL2 all types can be transformed.

Alternative representations of the types (4), (11) and (13) can be useful if the relationship type has no attributes. If the alternative representation is chosen anyway, despite of own attributes, it has to be guaranteed that within the table of the entity type the relationship attributes have NULL values for entities which form no relationships.

3.3 Generalization

Both the generic entity type and the entity types of the subtypes can be transformed into tables when dealing with generalization. The operator for the generalization (which is represented in ERM diagrams by a triangle) is not transformed into a table. Assuming that two entity types *NC-machine* and *bottleneck-machine* are generalized to the entity type *machine*, the following tables would result for the transformation:

- ```
  CREATE TABLE Machine( MNo, ...,
  PRIMARY KEY (MNo))
  ```

- ```
  CREATE TABLE NC-machine( MNo, ...,
  PRIMARY KEY (MNo),
  FOREIGN KEY (MNo) REFERENCES Machine(MNo))
  ```

- ```
  CREATE TABLE Bottleneck-machine( MNo, ...,
  PRIMARY KEY (MNo),
  FOREIGN KEY (MNo) REFERENCES Machine(MNo))
  ```

All tables bear the same primary key *MNo*. Furthermore, a referential constraint for the primary keys of the subtype tables to the primary key of the generic entity type table is established. In this way it can be ensured that every row of the subclass tables corresponds to an equivalent row in the generic structure. This link is also called semi-referential integrity (cf. *Steinbauer/Wedekind 85*).

The mentioned referential constraints can neither guarantee that for every row in the generic type table there exists a row in a subtype table, nor prevent that a row of a generic type table is contained in several subtype tables. Hence, this transformation represents a non-complete, non-disjunctive generalization of the type (4).

The complete generalization would require a referential constraint from the generic type to the subtypes whilst connecting the references to the subtypes by means of an exclusive OR. Such a referential constraint is not provided by the relational model.

The disjunctive generalization would require a "negative" reference of every subtype to all other subtypes which prevent the coexistance of equal rows in two subtype tables. That kind of constraint cannot be formulated in SQL according to *ISO 89*. Summing up, it may be said that SQL only allows the type (4) of generalization but not the types (1), (2) and (3) of figure 8.

Generalizations which cannot be represented directly in SQL according to *ISO 89* can be realized by means of auxiliary constructs:

- When dealing with disjunctive generalizations it is recommendable to include an attribute *subtype* into the generic type tables, which contains the name of the subtype table:

  ```
  CREATE TABLE Machine( MNo, subtype, ...,
  PRIMARY KEY (MNo),
  CHECK Subtype IN ('NC-machine','Bottleneck-machine'))
  ```

  The correct subtype can be determined if several rows exist in subtype tables accidently due to the missing integrity constraint. A CHECK-constraint ensures that the attribute *subtype* cannot contain anything else but the subtype names.
  ```

- In disjunctive, complete generalizations the attribute *subtype* can also be defined as NOT NULL:

```
CREATE TABLE Machine( MNo, subtype NOT NULL, ...,
PRIMARY KEY (MNo),
CHECK Subtype IN ('NC-machine','Bottleneck-machine'))
```

- If the generic type does not have any other attributes than the primary key, and if the primary key is only required as a foreign key by the subtype tables (i.e. the generic type does not have connections to relationship types), then the generic type does not have to be represented as an individual table. The complete set of entities (i.e. the set of rows of the subtype tables) can be gathered by an outer join (cf. *Codd 79*) which can be stored as a VIEW with the name of the generic type.

- The representation of a non-disjunctive, complete generalization can be supported by the inclusion of attributes, the task of which is to flag whether a row of the generic type table corresponds to a row in the subtype table. The attributes contain the names of the subtypes and are defined as logical variables by means of a CHECK constraint assigning the numerical value zero for "present" or one for "not present". An additional constraint guarantees the completeness in that the sum of all such defined attributes has to be greater or equal to one:

```
CREATE TABLE Machine( MNo, NC-machine NOT NULL,
                      Bottleneck-machine NOT NULL, ...,
PRIMARY KEY (MNo), CHECK NC-machine  IN (0,1),
CHECK Bottleneck-machine IN (0,1),
CHECK (NC-machine + Bottleneck-machine >= 1 ))
```

In SQL2, these types of generalizations can be represented by using the inter-relational capability of the CHECK constraint statement:

- Disjunctive generalizations are representable by "negative" references in each subtype:

```
CREATE TABLE NC-machine( MNo, ...,
PRIMARY KEY (MNo),
FOREIGN KEY (MNo) REFERENCES Machine (MNo),
CHECK (MNo NOT IN (SELECT MNo FROM Bottleneck-machine)))

CREATE TABLE Bottleneck-machine( MNo, ...,
PRIMARY KEY (MNo),
FOREIGN KEY (MNo) REFERENCES Machine (MNo),
CHECK (MNo NOT IN (SELECT MNo FROM NC-machine)))
```

- Complete generalizations are representable by "alternative" references in each subtype:

```
CREATE TABLE Machine( MNo, ...,
PRIMARY KEY (MNo),
CHECK (MNo IN (SELECT MNo FROM Bottleneck-machine) OR
       MNo IN (SELECT MNo FROM NC-machine)))
```

- A disjunctive, complete generalization (type (1)) can be represented by a combination of both.

4. Comments on the Application of Data Modelling

Basing on the above made explanations, some comments shall be made concerning frequently encountered problems and the general approach to data modelling.

4.1 Attributes of Relationships

As mentioned earlier (cf. chapter "2.2.3 Properties of Relationships"), not all methods for modelling allow for attributes in relationships. The product-customer example from figure 1 shows that the attribute *discount* can only be integrated within the relationship type *condition*. Since every entity of the two entity types can have many relationships and the attribute *discount* would therefore constitute a repeat group (assuming a relationship type (8)), its integration into one of the entity types is not possible. This kind of relationship can consequently not be modelled in the CASE-tool ADW. Such a kind of relationship has to be decomposed into one-to-many relationships (cf. chapter 4.2 "Modelling binary, many-to-many Relationships").

However, the dependency of attributes on relationships also arises in one-to-many relationships. The following example of a personnel-team relationship is to demonstrate this (cf. figure 11). One person participates in at most one team, whereas one team can consist of an arbitrary number of persons. This is realized by means of the relationship *works_in* (relationship type (4)). A person has attributes as *pid*, *name*, etc. The attributes *role* and *member_since* only make sense for persons who are assigned to a team. This would not be evident if the attributes were included in the entity type person. Only the inclusion in the relationship type *works_in* assures the validity of the membership status. A transformation to SQL produces the following tables:

```
CREATE TABLE Team( TId, ...,
PRIMARY KEY (TId))

CREATE TABLE Person( PId, Pname, ...,
PRIMARY KEY (PId))

CREATE TABLE works_in( PId, TId NOT NULL,
member_since NOT NULL, role, ...,
PRIMARY KEY (PId, TId),
FOREIGN KEY (PId) REFERENCES Person(PId),
FORGEIGN KEY (TId) REFERENCES Team(TId))
```

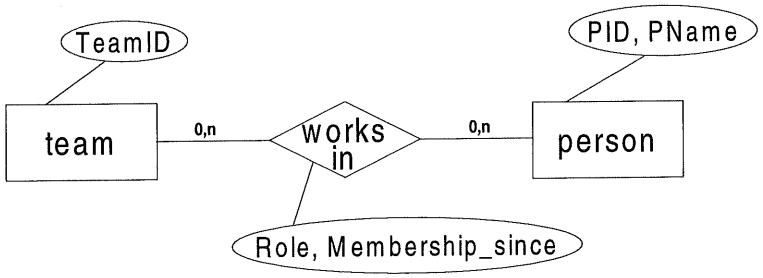

Fig. 11: Attributes depending on relationships

Only when an instance of an entity type is directly dependent on an instance of a relationship type (i.e. has a cardinality of (1,1), e.g. relationship types (2), (3), (5), (12) and (14)) the relationship attributes can be assigned to the entity type without any information loss.

4.2 Modelling Binary, Many-to-Many Relationships

The question whether to decompose binary many-to-many relationship (e.g. relationship types (8) to (10)) into two one-to-many relationships or not, is one often encountered (cf. chapter "4.1 Attributes of Relationships"). This arises from the fact that some methods (e.g. Bachmann notation) are incapable to model many-to-many relationships on the one hand, and from the property of SQL's foreign key references to represent only one-to-many relationships on the other hand. Decomposition of such relationships will cause a damage to semantic content if not combined with additional constraints.

Figure 12 a picks up the example of figure 1 adding three entities per entity type as well as three relationships corresponding to the respective cardinalities. The relationship provides the assignment of several products to a particular customer and vice versa, but a specific product can be assigned to a specific customer only once.

Figure 12 b shows a decomposition into two one-to-many relationships. An additional relationship of the type *condition* has been introduced at the instance level in order to demonstrate that a particular customer can be linked more than once to a specific product through several *conditions*. This does not obviously conform to the desired semantics.
This might be prevented by the formulation of further constraints like the adoption of primary keys[4].

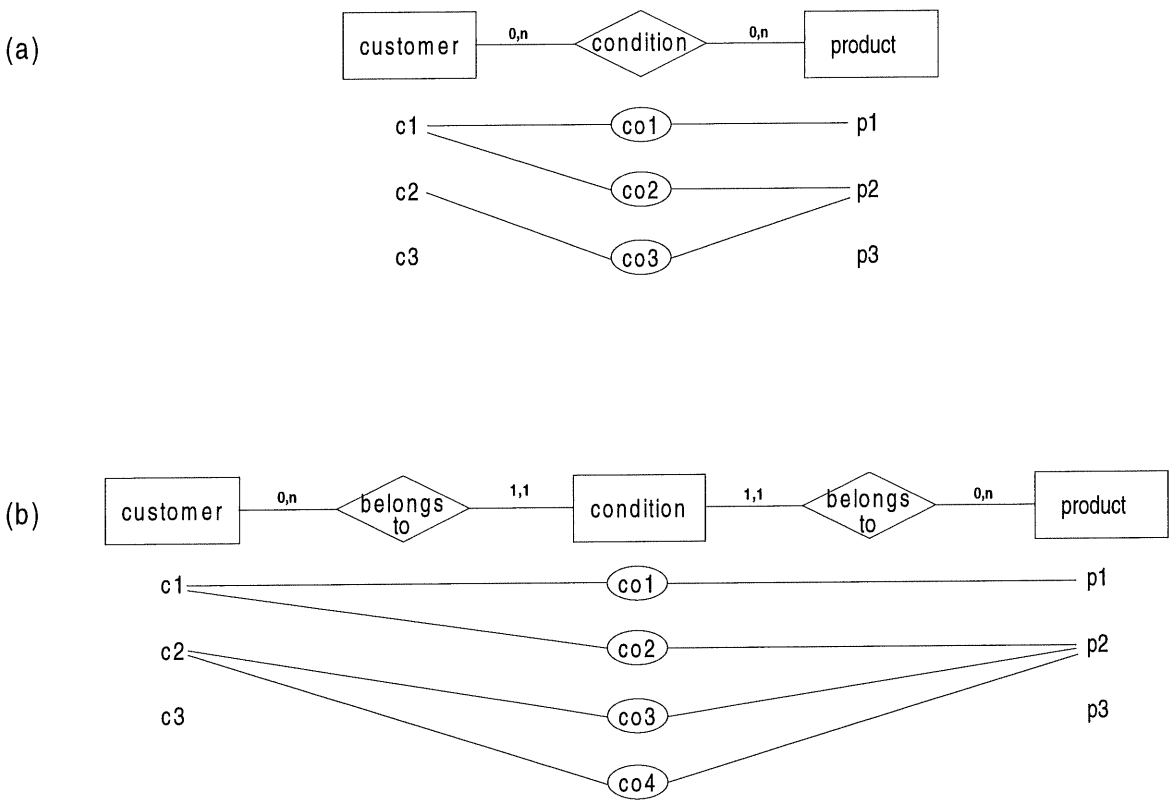

Fig. 12: Comparing many-to-many and one-to-many relationships

4.3 Sequencing Instances

Entity types, relationship types as well as tables represent sets of instances without any inherent fixed order. A sequence of instances has therefore to be declared explicitly. Figure 13 gives examples for the ordering of entities, both in ERM notation and at the level of instances. Part a depicts an entity type the entities of which form lists. Figure 13 b shows the hierarchy

4) The same applies to SERM concerning the relationship types (8) to (10) and (15) to (17) as mentioned in figure 0 and figure 0.

within an entity type, as it is usual within organizational units, for example (cf. Figure 2). A net-like order is represented in Figure 13 c. Bills of materials are a typical example. It has to be taken into account that recursive relationships which have an edge with the cardinality of (1,x) (e. g. types (12), (14), (16), (17)) cause recursions at the entity level (cf. Figure 13 d).

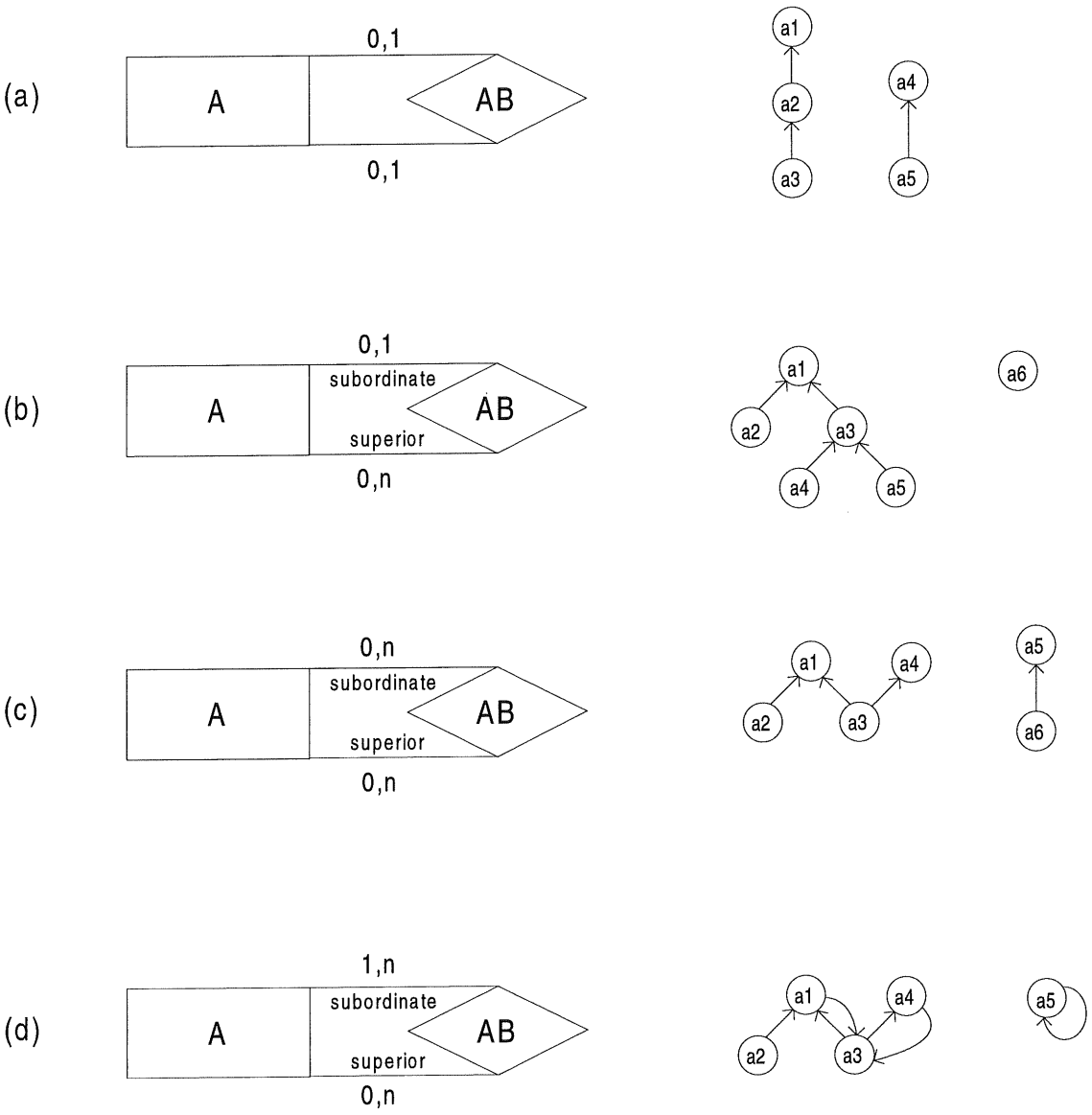

Fig. 13: Sequencing entities

Sequences within relationships refer to the corresponding relationships of one entity type's entity. These sequences can also be represented like before. Hereto, the relationships have to be reinterpreted in order to be able to form sequence relationships. Figure 14 a represents an

example of orders to which n-many articles can be assigned. For sorting the *order part* of an order the relationship *sequence* is introduced[5].

A simpler way to achieve these facts is to introduce the entity type position number (cf. figure 14 b). The resulting relationship *position* with its primary key *order_no* and *pos_no* forms a relationship with the type *article*. The entity type *pos_no* is virtual and does not have to be materialized as table. Unlike the previous solution a specific article can be assigned to a specific order several times.

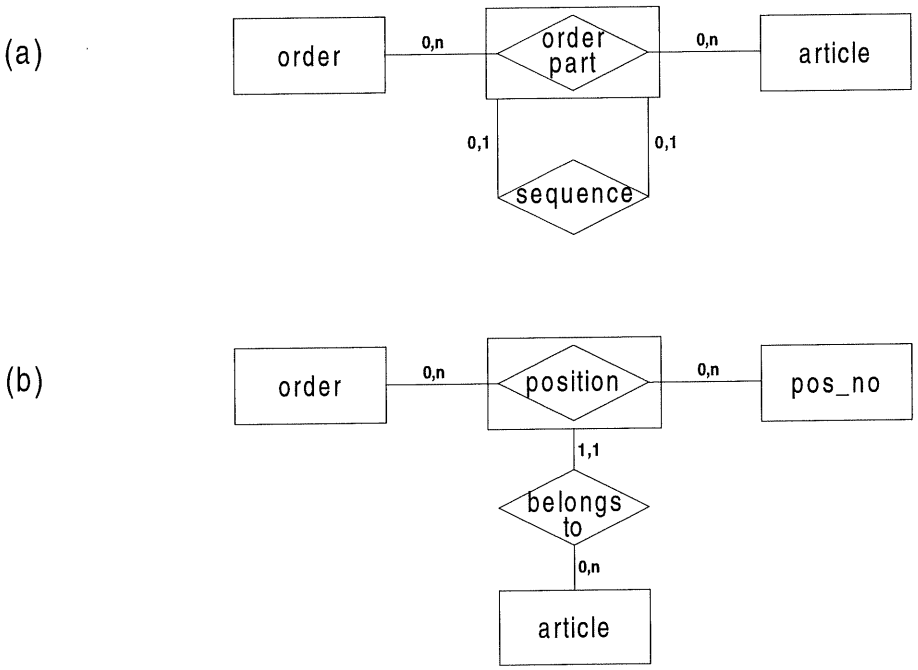

Fig. 14: Sequencing relationships

4.4 Temporal Aspects

On the one hand, the temporal dimension can be represented by attributes with the data types date or time. On the other hand, time can be integrated within the data model as an independent entity type, and serves for modelling order-related data. The entity type *time* is the basis for relationships and transmits the attribute *time_id* as a foreign key to the relationships. This might be especially helpful when relationships need a temporal dimension for their description. Figure 15 shows a schedule in which an operation of an equipment unit

5) By additional integrity constraints it has to be ensured that only order parts of the same order are connected.

is assigned to a specific time period. The entity type *time* has two edges to the relationship type *scheduled_on*, one for the start date, one for the end date. This data structures automatically ensures that an operation can only be planned for an equipment unit if start and end dates are simultaneously defined. A transformation into SQL can produce the following tables[6]:

```
CREATE TABLE scheduled_on (op_id, equi_unit,
start_date, end_date, ...
PRIMARY KEY (op_id),
FOREIGN KEY (op_id) REFERENCES operation (op_id),
FOREIGN KEY (equi_unit) REFERENCES equiment_unit (equi_unit),
CHECK (start_date <= end_date))
```

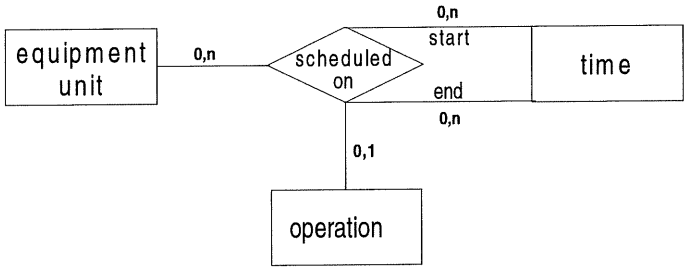

Fig. 15: Entity type time for temporal aspects

References

Batra et al. 90
Batra, D.; Hoffer, J. A.; Bostrom, R. P.: Comparing Representations with Relational and EER Models, in: Communications of the ACM, 33 (1990), S. 126 - 139.

Briand et al. 88
Briand, H.; Ducateau, C.; Hebrail, Y.; Herin-Aime, D.; Kouloumdjian, J.: From Minimal Cover to Entity-Relationship Diagram, in: March, S. T. (ed.), Entity-Relationship Approach (Proceedings of the Sixth International Conference on Entity-Relationship Approach, New York 1987), Amsterdam-New York-Oxford-Tokyo 1988, pp. 287 - 304.

Chen 76
Chen, P. P.: The Entity-Relationship Model - Towards a Unified View of Data, in: ACM Transactions on Database Systems, 1 (1976) 1, pp. 9 - 36.

6) To interpret cardinalities in n-ary relationships and to transfer them into SQL see *Loos 92a*.

Codd 70
Codd, E. F.: A Relational Model of Data for Large Shared Data Banks, in: Communications of the ACM, 13 (1970) 6, pp. 377 - 387.

Codd 79
Codd, E. F.: Extending a Database Relational Model to Capture More Meaning, in: ACM Transactions on Database Systems, 4 (1979), S. 397 - 434.

Date 81
Date, C. J.: An Introduction to Database Systems Reading, Vol. I, Massachusetts, 1981.

Elmasri et al. 85
Elmasri, R.; Weeldreyer, J.; Hevner, A.: The Category Concept: An Extension to the Entity-Relationship Approach, in: Data & Knowledge Engineering, 1 (1985), pp. 75 - 116.

Hars et al. 91
Hars, A.; Heib, R.; Kruse, C.; Michely, J.; Scheer, A.-W.: Concepts of Current Data Modelling Methodologies - Theoretical Foundations, in: Scheer, A.-W. (ed.), Veröffentlichungen des Institutes für Wirtschaftsinformatik, Heft 83, Saarbrücken 1991.

ISO 89
ISO/IEC 9075, JTC 1/SC 21/WG 3, 1989-04-01, Information processing system, Database language SQL with integrity enhancement, 1989.

ISO 91
ISO/IEC DIS 9075, JTC1/SC21/WG3, Database Language SQL2, 1991.

Kent 83
Kent, W.: A Simple Guide to Five Normal Forms in Relational Database Theory, in: Communications of the ACM, 26 (1983) 2, pp. 121 - 125.

Loos 91
Loos, P.: Repräsentation erweiterter Entity-Relationship-Strukturen im Relationenmodell, in: Himmelmann, G. W. et al. (ed.), Oracle in Theorie und Praxis (4. Deutsche Oracle Anwenderkonferenz, Hamburg, 17.-18. September 1991), Stuttgart 1991, pp. 113 - 132.

Loos 92a
Loos, P.: Datenstrukturierung in der Fertigung, München-Wien 1992.

Loos 92b
Loos, P.: Was bringt SQL2?, in: Scheer, A.-W. (ed.), Datenbanken 1992 - Praxis relationaler Datenbanken (Proceedings, Saarbrücken, 15.-16. June 1992), pp. 131 - 141.

Markowitz/Shoshani 89
Markowitz, V. M.; Shoshani, A.: On the Correctness of Representing Extented Entity-Relationship Structures in the Relational Model, in: ACM SIGMOD Record, 18 (1989) 2, S. 430 - 439.

Put 88
Put, F.: The ER Approach Extended with the Action Concept as a Conceptual Modelling Tool, in: Batini, C. (ed.), Proceedings of the Seventh International Conference on Entity-Relationship Approach, Rome 1988, pp. 283 - 300.

Scheer 89
Scheer, A.-W.: Enterprise-Wide Data Modelling - Information Systems in Industry, Berlin-Heidelberg-New York-Tokio 1989.

Scheer 92
Scheer, A.-W.: Architecture of Integrated Information Systems, Berlin-Heidelberg-New York-Tokio 1992.

Schlageter/Stucky 83
Schlageter, G.; Stucky, W.: Datenbanksysteme: Konzepte und Modelle, in: Leitfäden der angewandten Mathematik und Mechanik, Stuttgart 1983.

Sinz 87
Sinz, E. J.: Datenmodellierung betrieblicher Probleme und ihre Unterstützung durch ein wissensbasiertes Entwicklungssystem, Habilitation Thesis University Regensburg 1987.

Smith/Smith 77
Smith, J. M.; Smith, D. C.: Database Abstractions: Aggregation and Generalization, in: ACM Transaction on Database Systems, 2 (1977) 2, pp. 105 - 433.

Steinbauer/Wedekind 85
Steinbauer, D.; Wedekind, H.: Integritätsaspekte in Datenbanksystemen, in: Informatik Spektrum, 8 (1985), pp. 60 - 68.

Su 85
Su, S. Y. W.: Modeling Integrated Manufacturing Data Using SAM*, in: Blaser, A.; Pistor, P. (ed.), Datenbank-Systeme für Büro, Technik und Wissenschaft, Berlin-Heidelberg-New York-Tokio 1985, pp. 27 - 49.

Teorey et al. 86
Teorey, T. J.; Yang, D.; Fry, J. P.: A Logical Design Methodology for Relational Databases Using The Extended Entity-Relationship Model, in: ACM Computing Surveys, 18 (1986) 2, pp. 197 - 222.

Verheijen/VanBekkum 82
Verheijen, G. M. A.; van Bekkum, J.: NIAM: An Information Analysis Method, in: Olle, T. W. et al. (ed.), Information Systems Design Methodologies - A Comparative Review, Amsterdam-New York-Oxford 1982, pp. 537 - 589.

Webre 83
Webre, N.: An Extended Entity-Relationship Model and its Use on a Defense Project, in: Chen, P. P. (ed.), Entity-Relationship Approach to Information Modelling and Analysis (Preceedings of the Second International Conference on Entity-Realtionship Approach, Washington D. C. 1981), Amsterdam-New York-Oxford 1982, pp. 173-193.

Wedekind 88
Wedekind, H.: Nullwerte in Datenbanksystemen, in: Informatik Spektrum, 11 (1988), pp. 97 - 98.

Wong/Katz 80
Wong, E.; Katz, R. H.: Logical Design and Schema Conversion for Relational and DBTG Databases, in: Chen, P. P. (Hrsg.), Entity-Relationship Approach to System Analysis and Design (Proceedings of the International Conference On Entity-Relationship Approach to System Analysis and Design, Los Angeles 1979), Amsterdam-New York-Oxford 1980, S. 311 - 321.