Title	Basic Concepts in ER Modeling
Author	Murali Mani
Email	mmani@cs.wpi.edu
Address	Computer Science Department, Worcester Polytechnic University, 100 Institute Road, Worcester, MA 01605
Phone	(508) 831 6421
Fax	(508) 831 5776

Basic Concepts in ER Modeling

January 14, 2004

Database design process is typically done in different stages [2]. First, during *conceptual design phase*, the database designer represents the application requirements as a schema in a conceptual model. Examples of conceptual models include ER [3], UML [8], and ORM [7]. Then in the *logical design phase*, the conceptual schema is represented as a schema in a logical model such as relational model [4], object-relational model or XML. There is a third phase where the logical schema is translated to a physical schema. In this work, we will examine the basic features in ER conceptual model.

1 ER Conceptual Model

Entity Relationship (ER) model defined by Chen in the 1970s has been widely used as a conceptual model ever since [3]. A schema is represented in the ER model using a diagrammatic notation called ER diagram. In this section, we examine the features of ER model.

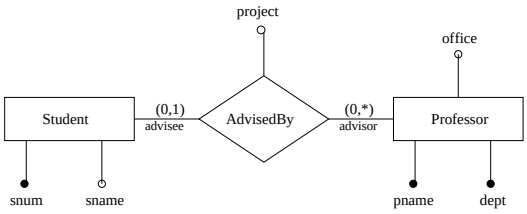
1.1 Basic Features of ER

In the ER model, we specify structures such as *entity types*, *relationship types*, and *attributes*. We denote an entity type by E_i, an entity instance (also called entity, for short) of an entity type E_i as e_{i_j}, and the set of entities of an entity type in a database instance as $I(E_i)$. Figure 1 shows an entity type $Student$; $I(Student) = \{s1, s2, s3\}$. A relationship type is denoted as R_i, and it represents an association between entity types. We denote a relationship instance (also called relationship) of relationship type R_i as r_{i_j}, and the set of relationships in a database instance as $I(R_i)$. Consider a relationship type R_i between entity types E_1, E_2, ..., E_n. $I(R_i)$ defines a n-ary relation between the sets $I(E_1)$, $I(E_2)$, ..., $I(E_n)$. A relationship $r \in I(R_i)$ that associates entities $e_1 \in I(E_1)$, $e_2 \in I(E_2)$, ..., $e_n \in I(E_n)$ is also represented as $(e_1, e_2, \ldots, e_n)$. Figure 1 shows a binary relationship type $AdvisedBy$ between entity types $Student$ and $Professor$. Figure 1 (c) shows $I(AdvisedBy) = \{r1, r2\}$. See that $r1 = (s1, p1)$, and $r2 = (s2, p1)$.

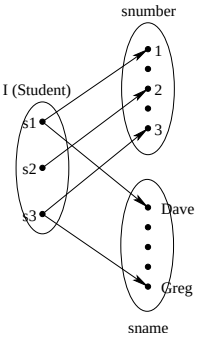
An entity type or a relationship type may also define attributes. Consider an attribute A of an entity type E (or relationship type R). A maps values of $I(E)$ (or $I(R)$) to "values" of A. In Figure 1, entity type $Student$ defines two attributes $snumber$ and $sname$; $snumber$ maps $s1 \rightarrow 1$, $s2 \rightarrow 2$, $s3 \rightarrow 3$; attribute $sname$ maps $s1 \rightarrow Dave$, and $s3 \rightarrow Greg$; note that $sname$ does not map $s2$ to any value. Also relationship type $AdvisedBy$ defines an attribute $project$.

We focus on two kinds of constraints that can be specified in the ER model: *key constraints* and *cardinality constraints*. Key constraints are specified for an entity type; we say the key for an entity type E_i is its attribute A_{i_j}, if for any database instance, A_{i_j} is a one-to-one function from $I(E_i)$ to values of A_{i_j}. The key for an entity type may be composite (that is, multiple attributes). The key for an entity type E_k is the attributes $(A_{k_1}, A_{k_2}, \ldots, A_{k_m})$, if for any database instance, we have a one-to-one function from $I(E_k)$ to the values in $A_{k_1} \times A_{k_2} \times \ldots \times A_{k_m}$. In Figure 1 (a), we define the key for $Student$ is $(snumber)$, and the key for $Professor$ is $(pname, dept)$.

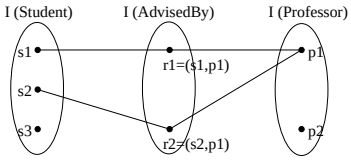
Cardinality constraints are specified for each entity type in a relationship type. We say that the cardinality constraints for an entity type E in a relationship type R is (min, max), if for any database instance, any $e \in I(E)$ must appear in at least min instances of $I(R)$, and no $e \in I(E)$ appears in



(a) Example ER schema



(b) Showing $I(Student)$ and the attributes



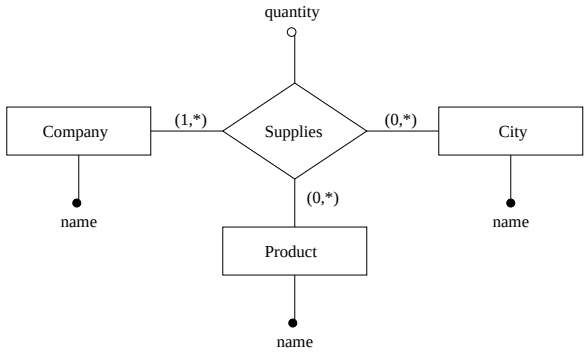
(c) Showing $I(AdvisedBy)$

Figure 1: Example ER schema and a corresponding database instance fragment.

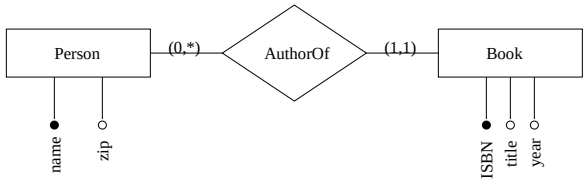
more than max instances of $I(R)$. We denote unbounded cardinality by $*$. Based on the cardinality constraints, binary relationship types can be called $1:1$, $1:n$, or $m:n$ [2].

ER also has the notion of *role*s. An entity type E in a relationship type R can be said to play a *role*, denoted l. For a database instance, we define $I(l) \subseteq I(E)$ as the set of entities that appear in R in the role l. In Figure 1, $Student$ plays the role of $advisee$, and $Professor$ the role of $advisor$ in $AdvisedBy$; $I(advisee) = \{s1, s2\}$, and $I(advisor) = \{p1\}$.

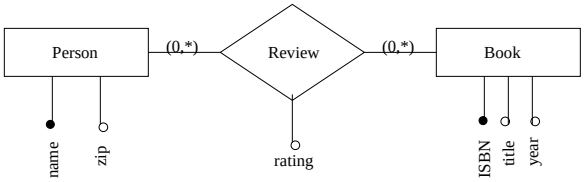
Figure 2 shows some typical ER relationship types. Note the recursive relationship type $Contains$ in Figure 2 (d). A recursive relationship type R between entity types $E_1, E_2, \ldots, E_n$ has $E_i = E_j$ for some $1 \leq i, j \leq n$. $Contains$ is a relationship type between $Part$ playing the role of $superPart$ and $Part$ playing the role of $subPart$.



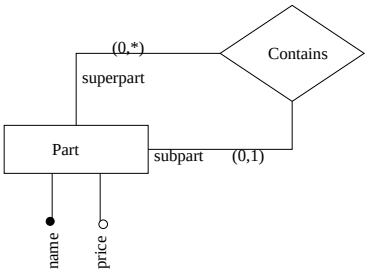
(a) Ternary relationship type



(b) Binary $1:n$ relationship type



(c) Binary $m:n$ relationship type



(d) Recursive relationship type

Figure 2: Example Relationship Types.

A simple translation algorithm to translate an ER schema [6] to a relational schema works as follows: create a separate relation for every entity type E, and every relationship type R. The relation for E contains attributes defined for E in the ER schema; the key for E in the ER schema is the key for this relation. The relation for R, where R is a relationship type between entity types $E_1, E_2, \ldots, E_n$ contains

Student

snumber	sname
1	Dave
2	null
3	Greg

Professor

pname	dept	office
John	CS	140 FL
Dave	Math	230 FL

AdvisedBy

snumber	pname	dept	project
1	John	CS	DB1
2	John	CS	DB2

Table 1: Relational Instance corresponding to ER in Figure 1

the key attribute(s) for every E_i, $1 \leq i \leq n$, and any attributes defined for R. If there is an E_i whose max cardinality in R is 1, the key for this relation is the same as the key for E_i; if there is no such E_i, the key for this relation is all the key attributes of every E_i. Also "appropriate" foreign keys are defined, but we omit their discussion here. This algorithm applied to Figure 1 gives the relations shown in Table 1. Also, there are techniques for decreasing the number of relations obtained [6].

1.2 ER model as the basis for studying application requirements

There are several good reasons for studying structural and constraint specifications requirements for a database application based on the ER model. First, ER model has always been considered a representative of real world database applications. Because of this, algorithms to translate an ER schema to schemas in different logical models have been studied extensively. A simple algorithm to translate an ER schema to relational schema as given in [6] gives the relations shown in Table 1.

The above relational instance also shows that ER model gives a clear interpretation of some basic concepts in database design such as nulls, compared to other models such as relational model. Another important concept in database design is normalization. Normalization can also be explained using ER model [3] as: if we assume any functional dependency $A \rightarrow B$ implies that there is an entity type with A as the key and attribute(s) B, then a "correct" ER schema and a "correct" translation algorithm will generate a relational schema guaranteed to have no redundancy (or according to [1] the entropy of any position in any instance is non-zero). In short, the resulting relational schema is in BCNF.

1.3 Extensions to the ER Model

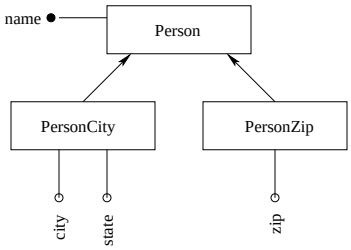
ER model was originally influenced by relational model, and lacks features necessary for applications. We therefore extend the ER model, and call the extended model the ERex model. The extensions are: (a) structural specification called *categories*, (b) constraint specifications called *coverage constraints*, and (c) order constraints.

1.3.1 Categories

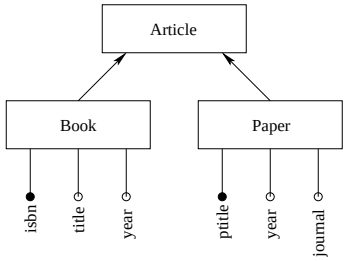
Categories is a kind of relationship type similar to ISA relationship types. Given entity types $E, E_1, E_2, \ldots E_n$, we can specify $E_1, E_2, \ldots E_n$ as categories of E, if in any database instance, $I(E_i) \subseteq I(E)$, for every $1 \leq i \leq n$. We represent this in an ERex schema with arrows from the E_i's to E. This is different from ISA relationship type, as well as from the ECR model [5]: (a) ISA requires that a key constraint be specified for E (b) ECR [5] requires $I(E) \subseteq I(E_1) \cup I(E_2) \cup \ldots \cup I(E_n)$, and allows $I(E_i) \not\subseteq I(E)$. Two examples of categories are shown in Figure 3.

1.3.2 Coverage Constraints

Coverage constraints are specified on entity types or roles. There are two kinds of coverage constraints (a) *total coverage*: Given entity types $E, E_1, E_2, \ldots, E_n$, where every E_i, $1 \leq i \leq n$ is a category of E, we specify that $E_1 \cup E_2 \cup \ldots E_n = E$ if for any database instance, $I(E_1) \cup I(E_2) \cup \ldots I(E_n) = I(E)$. We can specify total coverage on roles as: given entity type E, and roles $l_1, l_2, \ldots, l_n$, where every l_i, $1 \leq i \leq n$ is a role played by E in some relationship type, we specify $l_1 \cup l_2 \cup \ldots l_n = E$, if for any database instance,



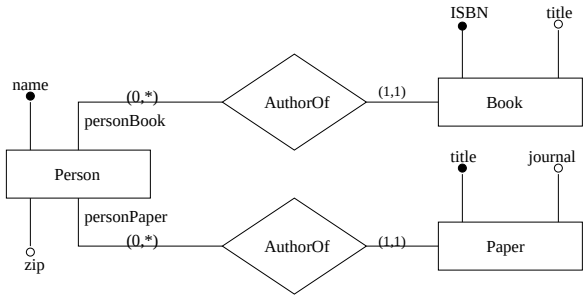
(a) $PersonCity$ and $PersonZip$ are categories of $Person$



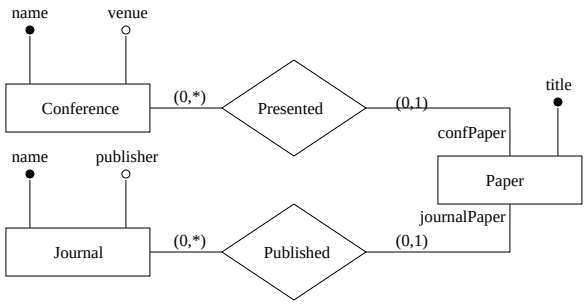
(b) $Book$ and $Paper$ are categories of $Article$

Figure 3: Example Categories.

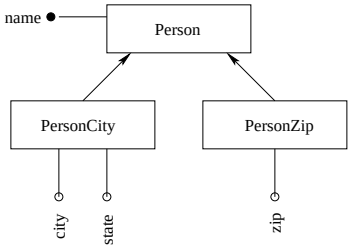
$I(l_1) \cup I(l_2) \cup \ldots \cup I(l_n) = I(E)$. (b) *exclusive coverage*: Given entity types E, E_1, E_2, where E_1, E_2 are categories of E, we specify $E_1 \cap E_2 = \phi$, if for any database instance, $I(E_1) \cap I(E_2) = \phi$. Similarly given entity type E, and roles l_1, l_2, where l_1, l_2 are roles played by E, we specify $l_1 \cap l_2 = \phi$, if for any database instance, $I(l_1) \cap I(l_2) = \phi$. Examples of coverage constraints are shown in Figure 4.



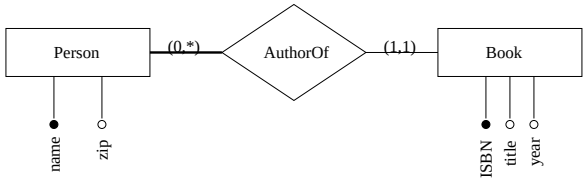
(a) Coverage Constraints: $\{personBook \cap personPaper = \phi, personBook \cup personPaper = Person\}$



(b) Coverage Constraints: $\{confPaper \cap journalPaper = \phi, confPaper \cup journalPaper = Paper\}$.



(c) Coverage Constraints: $\{PersonCity \cap PersonZip = \phi, PersonCity \cup PersonZip = Person\}$



(d) Relationship Type with Order Constraints. The books authored by a person are ordered.

Figure 4: (a) and (c) show coverage constraints on roles; (b) shows coverage constraints on entity types; (d) shows order constraints.

1.3.3 Order Constraints

Order constraints are specified for entity types in a relationship type. Consider relationship type R between entity types $E_1, E_2, \ldots E_n$. We specify that E_i, $1 \leq i \leq n$ is ordered in R, if in any database instance the relationship instances in $I(R)$ where any $e_i \in I(E_i)$ appear is ordered. We represent this constraint in an ERex schema with a thick line between E_i and R. See figure 4 (d).

References

[1] M. Arenas and L. Libkin. An information-theoretic approach to normal forms for relational and XML data. In *ACM PODS*, San Diego, CA, June. 2003.

[2] C. Batini, S. Ceri, and S. B. Navathe. *"Conceptual Database Design: An Entity-Relationship Approach"*. The Benjamin/Cummings Pub., 1992.

[3] P. P. Chen. "The Entity-Relationship Model". *ACM Trans. on Database Systems (TODS)*, 1:9–36, 1976.

[4] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. *Comm. ACM*, 13(6):377–387, 1970.

[5] R. Elmasri, J. Weeldreyer, and A. Hevner. "The Category Concept: An Extension to the Entity-Relationship Model". *J. Data & Knowledge Engineering (DKE)*, 1(1), May 1985.

[6] H. Garcia-Molina, J. D. Ullman, and J. Widom. *"Database Systems: The Complete Book"*. Prentice Hall, 2002.

[7] T. Halpin. *Informational Modeling and Relational Databases: From Conceptual Analysis to Logical Design.* Morgan Kaufmann Pub., 2001.

[8] OMG. Omg unified modeling language specification, version 1.5, Mar. 2003. http://www.uml.org/.