

---

# Object Role Modeling: An Overview

by Dr. Terry Halpin, BSc, DipEd, BA, MLitStud, PhD  
Director of Database Strategy, Visio Corporation

*In February 1998, Visio® Corporation acquired the assets and technology of InfoModelers, Inc. InfoModelers was the producer of state-of-the-art database design and access tools, including InfoModeler®, the premier data modeling tool based on the Object Role Modeling (ORM) methodology. The acquired InfoModeler technology will be integrated into future versions of Visio products, extending current capabilities to include full database design and re-engineering. Visio will continue to enhance database modeling through ORM. This overview presents database professionals and other Visio users with the opportunity to more closely examine ORM prior to the release of ORM-empowered Visio products.*

---

## Introduction

The quality of a database application depends critically on its design. To help ensure correctness, clarity, adaptability and productivity, information systems are best specified first at the conceptual level, using concepts and language that people can readily understand. The conceptual design may include data, process and behavioral perspectives, and the actual database management system (DBMS) used to implement the design can be based on one of many logical data models (relational, hierarchic, network, object-oriented, and so on). In this document, we focus on the data perspective, and assume the design is to be implemented in a relational database system.

Designing a database involves building a formal model of the application area or *universe of discourse (UoD)*. To do this properly requires a good understanding of the UoD and a means of specifying this understanding in a clear, unambiguous way. ORM simplifies the design process by using natural language— as well as intuitive diagrams that can be populated with examples— and by examining the information in terms of simple or *elementary facts*. By expressing the model in terms of natural concepts, like *objects* and *roles*, it provides a *conceptual* approach to modeling.

Early versions of ORM were developed in Europe in the mid-1970s (e.g., binary relationship modeling and NIAM). The version discussed here is based on the author's formalization of the method, and incorporates extensions and refinements arising from research conducted in Australia and the United States. The associated language FORML (Formal Object Role Modeling Language) is supported by InfoModeler, and will be extended in later Visio products.

Entity Relationship (ER) modeling provides another conceptual approach. Although ER models can be of use once the design process is finished, they are less suitable for formulating, transforming, or evolving a design. ER diagrams are further removed from

natural language, cannot be populated with fact instances, require complex design choices about attributes, lack the expressibility and simplicity of a role-based notation for constraints, hide information about the semantic domains that glue the model together, and lack adequate support for formal transformations. Many different ER notations exist that vary in the concepts they can express and the symbols used to express those concepts. For such reasons, ORM is preferred for conceptual modeling. Although the detailed picture provided by ORM diagrams is often desirable, for summary purposes it is useful to hide or compress the display of much of this detail. Though not discussed in this document, various abstraction mechanisms exist for doing this. If desired, ER diagrams can also be used to provide compact summaries, and are best developed as views of ORM diagrams. For example, IDEF1X diagrams can be generated automatically from ORM diagrams, and this is feasible for other ER variants, such as UML class diagrams.

This document attempts to convey the main ideas in ORM through a case study. First, we explain the steps used to develop a conceptual design. The conceptual design can be specified in either graphical or textual form. To help communicate the ideas, we deliberately make some mistakes, and later show how the design method helps to correct them. We also include a simple example to show how the conceptual design can be optimized for relational systems by applying a transformation. Next, we outline an algorithm for mapping this design to a normalized, relational database schema. With tool support, the conceptual design can be automatically mapped to a relational or object-relational schema for use in a variety of DBMSs. Finally, we give a brief sketch of how ORM can be used as a sound basis for conceptual queries, object-oriented modeling, and process/event modeling. A detailed discussion of ORM can be found in Halpin, T.A., *Conceptual Schema and Relational Database Design*, 2nd edn, Sydney: Prentice Hall Australia, 1995.

---

## The conceptual schema design procedure

The information systems life cycle typically involves the following stages:

- Feasibility study
- Requirements analysis
- Conceptual design of data and operations
- Logical design
- External design
- Prototyping
- Internal design and implementation
- Testing and validation
- Maintenance

The ORM *conceptual schema design procedure* (CSDP) focuses on the analysis and design of data. The conceptual schema specifies the information structure of the application: the

*types of fact* that are of interest, *constraints* on these facts, and perhaps *derivation rules* for deriving some facts from others.

With large-scale applications, the UoD is divided into convenient modules, the CSDP is applied to each, and the resulting subschemas are integrated into the global conceptual schema. The CSDP itself has seven steps (see Table 1). We now illustrate the basic working of this design procedure by means of a simple example.

**Table 1:** The conceptual schema design procedure (CSDP)

Step	Description
1	Transform familiar information examples into elementary facts, and apply quality checks
2	Draw a draft diagram of the fact types and apply a population check
3	Check for entity types that should be combined, and note any arithmetic derivations
4	Add uniqueness constraints, and check arity of fact types
5	Add mandatory role constraints, and check for logical derivations
6	Add any value, set comparison, and subtyping constraints
7	Add other constraints and perform final checks

## Step 1

The most important stage of the CSDP is *Step 1, transforming familiar information examples into elementary facts and applying quality checks*. Examples of the kinds of information required from the system are verbalized in natural language. Such examples are often available as output reports or input forms, or perhaps from a current documented version of the required system. If the examples are not available in these formats, the modeler can work with the client to produce examples of output reports that are expected from the system. To avoid misinterpretation, a UoD expert (a person familiar with the application) should perform or at least check the verbalization. UoD experts are also called “subject matter experts”. As an aid to the verbalization process, the speaker imagines he or she has to convey the information contained in the examples to a friend over the telephone.

For this document’s case study, we consider a fragment of an information system used by a university to maintain details about its academic staff and academic departments. One function of the system is to print an academic staff directory, as exemplified by the report extract shown in Table 2. Part of the modeling task is to clarify the meaning of terms used in such reports. The descriptive narrative provided here would normally be derived from a discussion with the UoD expert. The terms “empnr” and “extnr” abbreviate “employee number” and “extension number”. A phone extension may have access to local calls only (“LOC”), national calls (“NAT”), or international calls (“INT”). International access includes national access, which includes local access. In the few cases where different rooms or staff have the same extension, the access level is the same. An academic is either tenured or on contract. Tenure guarantees employment until retirement, while contracts have an expiration date.

**Table 2:** Extract from a directory of academic staff

Empnr	EmpName	Dept	Room	Phone Extnr	Phone Access	Tenure/Contract Expiration Date
715	Adams A	Computer Science	69-301	2345	LOC	1/31/95
720	Brown T	Biochemistry	62-406	9642	LOC	1/31/95
139	Cantor G	Mathematics	67-301	1221	INT	tenured
430	Codd EF	Computer Science	69-507	2911	INT	tenured
503	Hagar TA	Computer Science	69-507	2988	LOC	tenured
651	Jones E	Biochemistry	69-803	5003	LOC	12/31/96
770	Jones E	Mathematics	67-404	1946	LOC	12/31/95
112	Locke J	Philosophy	1-205	6600	INT	tenured
223	Mifune K	Elec. Engineering	50-215A	1111	LOC	tenured
951	Murphy B	Elec. Engineering	45-B19	2301	LOC	1/3/95
333	Russell B	Philosophy	1-206	6600	INT	tenured
654	Wirth N	Computer Science	69-603	4321	INT	tenured
...	...	...	...	...	...	...

The information contained in Table 2 must be stated in terms of *elementary facts*. Basically, an elementary fact asserts that a particular object has a property, or that one or more objects participate in a relationship, where that relationship cannot be expressed as a conjunction of simpler (or shorter) facts. For example, to say that Bill Clinton jogs and is the President of the United States is to assert two elementary facts.

Try to read the elementary facts expressed on row 1 of Table 2. As a first attempt, you might read the information as the six facts f1–f6. Each asserts a binary relationship between two objects. For discussion purposes the relationship type, or logical *predicate*, is shown in **bold** between the noun phrases that identify the objects. Object types are displayed here in *italic*. For compactness, some obvious abbreviations are used (“empnr”, “EmpName”, “Dept”, “extnr”); when read aloud you can expand these to “employee number”, “Employee name”, “Department”, and “extension number”.

- f1 The Academic with empnr 715 has EmpName ‘Adams A’.
- f2 The Academic with empnr 715 works for the Dept named ‘Computer Science’.
- f3 The Academic with empnr 715 occupies the Room with roomnr ‘69-301’.
- f4 The Academic with empnr 715 uses the Extension with extnr ‘2345’.
- f5 The Extension with extnr ‘2345’ provides the AccessLevel with code ‘LOC’.
- f6 The Academic with empnr 715 is contracted till the Date with mdy-code ‘01/31/95’.

Row 2 contains different instances of these six fact types. Row 3, because of its final column, provides an instance of a seventh fact type:

- f7 the Academic with empnr 139 is tenured.

This is called a unary fact– it specifies one property of an object. A logical predicate may be regarded as a sentence with one or more object holes in it– each hole is filled by a term or noun phrase that identifies an object. The number of object holes is called the arity of the predicate. Each of these holes determines a different role played in the predicate. For example, in f4 the academic plays the role of using, and the extension plays the role of being used. In f7 the academic plays the role of being tenured. On a diagram, each role is depicted as a separate box (see later).

Object Role Modeling is so called because it views the world in terms of objects playing roles. Facts are assertions that objects play roles. An n-ary fact has n roles. It is not necessary that roles be played by different objects. For example, consider the binary fact type: Person voted for Person. This has two roles (voting, and being voted for), but both could be played by the same object (e.g., Bill Clinton voted for Bill Clinton).

In FORML a predicate may have any arity (1, 2, 3 ..), but because the predicate is elementary, arities above 3 or 4 are rare. In typical applications, most predicates are binary. For these, we allow the inverse predicate to be stated as well, so that the fact may be read in both directions. For example, the inverse of f4 is:

f4' The Extension with extnr '2345' **is used by** the Academic with empnr 715.

To save writing, both the normal predicate and its inverse are included in the same declaration, with the inverse predicate preceded by a slash “/”. For example:

f4" The Academic with empnr **715 uses /is used by** the Extension with extnr '2345'.

Typically, predicate names are unique in the conceptual schema. In special cases, however (e.g., “has”), the same name may be used externally for different predicates; internally these are assigned different identifiers.

As a quality check at Step 1, we ensure that objects are well identified. Basic objects are either values or entities. Values are character strings or numbers: they are identified by constants (e.g., ‘Adams A’, 715). Entities are “real world” objects that are identified by a definite description (e.g., the Academic with empnr 715). In simple cases, such a description indicates the entity type (e.g., Academic), a value (e.g., 715), and a reference mode (e.g., empnr). A reference mode is the manner in which the value refers to the entity. Entities may be tangible objects (e.g., persons, rooms) or abstract objects (e.g., access levels). Composite reference schemes are possible (described later in this section).

Fact f1 involves a relationship between an entity (a person) and a value (a name is just a character string). Facts f2–f6 specify relationships between entities. Fact f7 states a property (or unary relationship) of an entity.

In setting out facts f1–f7, the employee number is unquoted while both extnr and roomnr are quoted. This indicates the designer treated empnr as a number, but considered extnr and roomnr as character strings. However, unless arithmetic operations are required for empnr, it may be quoted. Unless extnr and roomnr must permit non-

digits (e.g., hyphens or letters), or string operations are needed for them, they may be unquoted.

As a second quality check at Step 1, we use our familiarity with the UoD to see if some facts should be split or recombined (a formal check on this is applied later). For example, suppose facts f1 and f2 were verbalized as the single fact f8:

f8 The Academic with empnr 715 and empname 'Adams A' **works for** the Dept named 'Computer Science'.

The presence of the word “and” suggests that f8 may be split without information loss. The repetition of “Jones E” on different rows of Table 2 shows that academics cannot be identified just by their names. However the uniqueness of empnr in the sample population suggests that this suffices for reference. Because the “and-test” is only a heuristic, and sometimes a composite naming scheme is required for identification, the UoD expert is consulted to verify that empnr by itself is sufficient for identification. With this assurance obtained, f8 is now split into f1 and f2.

As an alternative to specifying complete facts one at a time, the reference schemes may be declared up front and then assumed in later facts. Simple reference schemes are declared by enclosing the reference mode in parentheses. Value types are followed by empty parentheses. For example, the object types and their identification schemes may be declared in this way:

```
Reference schemes: Academic (empnr);
                  AccessLevel (code);
                  Date (mdy)
                  Dept (name);
                  EmpName();
                  Extension (extnr);
                  Room (roomnr);
```

Then facts f1–f7 may be stated more briefly as follows. Here the names of object types begin with a capital letter:

```
f1 Academic 715 has EmpName 'Adams A'.
f2 Academic 715 works for Dept 'Computer Science'.
f3 Academic 715 occupies Room '69-301'.
f4 Academic 715 uses Extension '2345'.
f5 Extension '2345' provides AccessLevel 'LOC'.
f6 Academic 715 is contracted till Date '01/31/95'.
f7 Academic 139 is tenured.
```

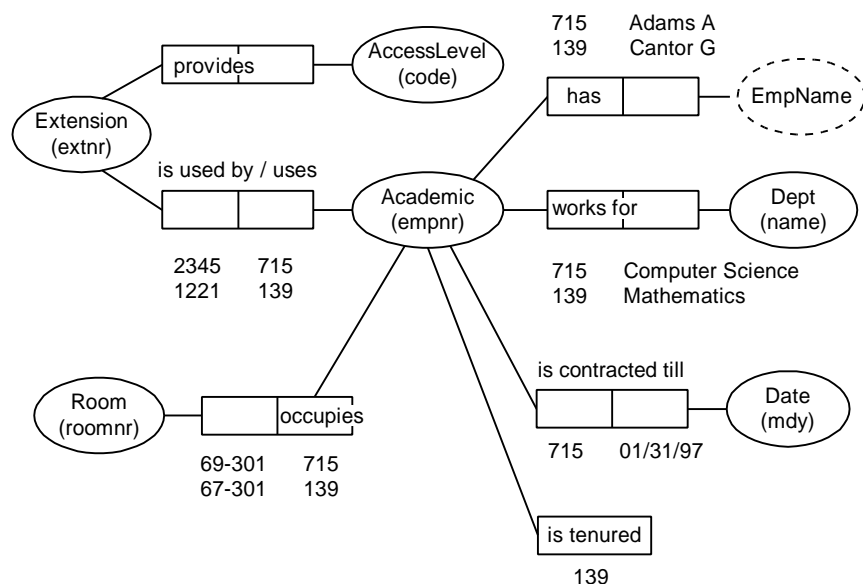
These facts are instances of the following fact types:

```
F1 Academic has EmpName
```

- F2 Academic works for Dept
- F3 Academic occupies Room
- F4 Academic uses Extension
- F5 Extension provides AccessLevel
- F6 Academic is contracted till Date
- F7 Academic is tenured

## Step 2

In *Step 2* of the CSDP, we *draw a draft diagram of the fact types and apply a population check* (see Figure 1). Entity types are depicted as named ellipses. Predicates are shown as named sequences of one or more role boxes, with the predicate name starting in or beside the first role box. Each predicate is ordered, from its first role box to the other end. An *n*-ary predicate has *n* role boxes. The inverse predicate names are omitted in this figure. Value types are displayed as named, broken ellipses. Lines connect object types to the roles they play. Reference modes are written in parentheses: this is an abbreviation for the explicit portrayal of reference types. For example, the notation “Academic (empnr)” indicates an injection (1:1-into mapping) from the entity type Academic to the value type empnr.



**Figure 1:** Draft diagram of fact types for Table 2 with sample population

In this example there are seven fact types. As a check, each is populated with at least one fact, shown as a row of entries in the associated fact table, using the data from rows 1 and 3 of Table 2.

The English sentences listed before as facts f1–f7, as well as other facts from row 3, can be read directly off this figure. Though useful for validating the model with the client and for understanding constraints, the sample population is not part of the conceptual schema diagram itself.

To help illustrate other aspects of the CSDP, we now widen our example. Suppose the information system is also required to assist in the production of departmental handbooks. Perhaps the task of schema design has been divided up, and another modeler works on the subschema relevant to department handbooks. Figure 2 shows an extract from a page of one such handbook. In this university academic staff are classified as professors, senior lecturers, or lecturers, and each professor holds a “chair” in a research area. To reduce the size of our problem, we have excluded many details that in practice would also be recorded (e.g., office phone and faxnr). To save space, details are shown here for only 4 of the 22 academics in that department. (The data is, of course, fictitious.)

**Department:** Computer Science

*Home phone of Dept head: 9765432*

*Chairs      Professors (5)*

Databases	Codd EF	BSc (UQ); PhD (UCLA) (Head of Dept)
Algorithms	Wirth N	BSc (UQ); MSc (ANU); DSc (MIT)

...

*Senior Lecturers (9)*

Hagar TA	BInfTech (UQ); PhD (UQ)
----------	-------------------------

...

*Lecturers (8)*

Adams A	MSc (OXON)
---------	------------

...

**Figure 2:** Extract from the Handbook of Computer Science Department

In verbalizing a report, at least one instance of each fact type should be stated. Let us suppose that the designer for this part of the application suggests the following fact set, after first declaring the following reference schemes: Dept (name); Professor (name); SeniorLecturer (name); Lecturer (name); Quantity (nr)+; Chair (name); Degree (code); University (code); Phonenr(). The “+” in “Quantity (nr)+” indicates that Quantity is referenced by a number, not a character string, and hence may be operated on by numeric operators such as “+”. For discussion purposes, the predicates are shown here in bold. In fact f13, a hyphen is used after “home” to bind the adjective to the following noun when constraints are verbalized (e.g., “has at most one home Phonenr” reads better than “has home at most one Phonenr”).

- f9    Dept ‘Computer Science’ **has professors in** Quantity 5.
- f10   Professor ‘Codd EF’ **holds** Chair ‘Databases’.
- f11   Professor ‘Codd EF’ **obtained** Degree ‘BSc’ **from** University ‘UQ’.
- f12   Professor ‘Codd EF’ **heads** Dept ‘Computer Science’.
- f13   Professor ‘Codd EF’ **has** home- Phonenr ‘965432’.
- f14   Dept ‘Computer Science’ **has senior lecturers in** Quantity 9.
- f15   SeniorLecturer ‘Hagar TA’ **obtained** Degree ‘BInfTech’ **from** University ‘UQ’.
- f16   Department ‘Computer Science’ **has** lecturers in Quantity 8.
- f17   Lecturer ‘Adams A’ **obtained** Degree ‘MSc’ **from** University ‘OXON’.



As a quality check for Step 1, we again consider whether entities are well identified. It appears from the handbook example that within a single department, academics may be identified by their name. Let us assume this is verified by the UoD expert. However, the complete application requires us to handle all departments in the same information system, and to integrate this subschema with the directory subschema considered earlier.

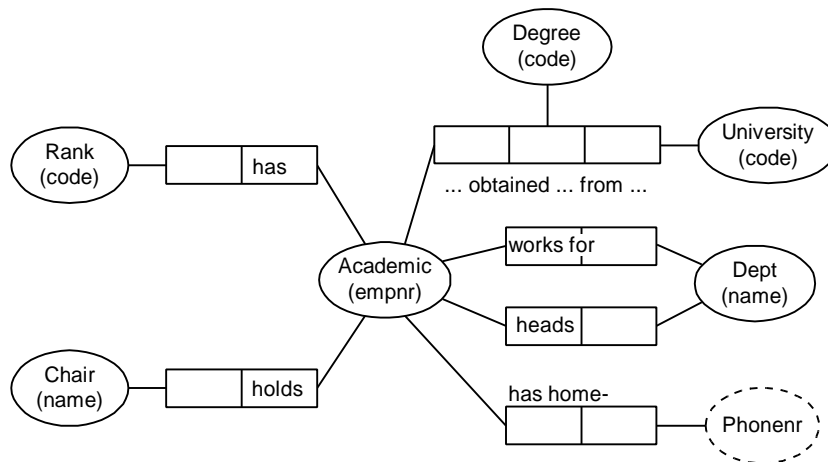
As a result, we must replace the academic naming convention used for the handbook example by the global scheme used earlier (i.e., empnr). Suppose that we can't see anything else wrong with facts f9–17, and proceed to expand the draft schema diagram to include this new information (this is left as an exercise for the reader).

### Step 3

At *Step 3* of the CSDP, we *check for entity types that should be combined and note any arithmetic derivations*. The first part of this step prompts us to look carefully at the fact types for f11, f15, and f17. Currently these are handled as three ternary fact types:

- Professor obtained Degree from University
- SeniorLecturer obtained Degree from University
- Lecturer obtained Degree from University

The common predicate suggests that the entity types Professor, SeniorLecturer, and Lecturer should be collapsed to the single entity type Academic, with this predicate now shown only once, as shown in Figure 3.



\* Dept *d* employs academics of Rank *r* in Quantity *q* iff *q* =  
**count each Academic who** has Rank *r* **and** works for Dept *d*

**Figure 3:** Extra fact types needed to capture the additional information in Figure 2

To preserve the original information about who is a professor, senior lecturer, or lecturer, we introduce the fact type: Academic has Rank. Let's use the codes "P", "SL" and

“L” for the ranks of professor, senior lecturer, and lecturer. For example, instead of fact f10 we now have:

f18 Academic 430 **has** EmpName ‘Codd EF’.  
f19 Academic 430 **has** Rank ‘P’.  
f20 Academic 430 **holds** Chair ‘Databases’.

Facts of the kind expressed in f9, f14, and f16 can now all be expressed in terms of the ternary fact type: Dept employs academics of Rank in Quantity. For example, f9 may be replaced by:

f9’ Dept ‘Computer Science’ **employs academics of** Rank ‘P’ **in** Quantity 5.

However, this does not tell us which professors work for the Computer Science department. Indeed, given that many departments exist, the verbalization in f9–f17 failed to capture the information about who worked for that department. This information is implicit in the listing of the academics in the Computer Science handbook. To capture this information in our application model, we introduce the following fact type: Academic works for Dept. For example, one fact of this kind is:

f21 Academic 430 **works for** Dept ‘Computer Science’

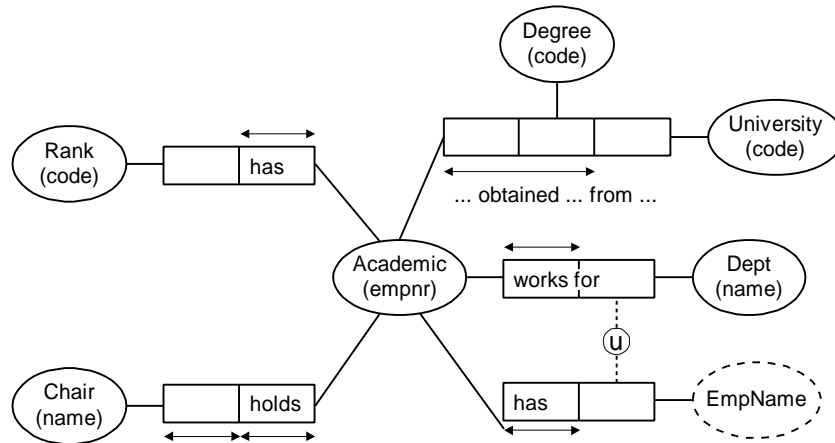
The second aspect of Step 3 is to see if some fact types may be derived from others by arithmetic. Because we now record the rank of academics as well as their departments, we can compute the number in each rank in each department simply by counting. So facts like f9’ are *derivable*. If desired, derived fact types may be included on a schema diagram if they are marked with an asterisk “\*” to indicate their derivability. To simplify the picture, it is usually better to omit derived predicates from the diagram. However, in all cases a derivation rule must be supplied. This may be written below the diagram (see Figure 3). Here “**iff**” abbreviates “if and only if”.

## Step 4

In *Step 4* of the CSDP, we *add uniqueness constraints and check the arity of the fact types*. Uniqueness constraints are used to assert that entries in one or more roles occur there *at most once*. A bar across  $n$  roles of a fact type ( $n > 0$ ) indicates that each corresponding  $n$ -tuple in the associated fact table is unique (no duplicates are allowed for that column combination). Arrow tips at the ends of the bar are needed if the roles are noncontiguous (otherwise arrow tips are optional). A uniqueness constraint spanning roles of different predicates is indicated by a circled “u”: this specifies that in the natural join of the predicates, the combination of connected roles is unique.

For example, a fragment of the conceptual schema under consideration is displayed in Figure 4. While these constraints are suggested by the original population, the UoD expert should normally be consulted to verify them. It is sometimes helpful to construct a test

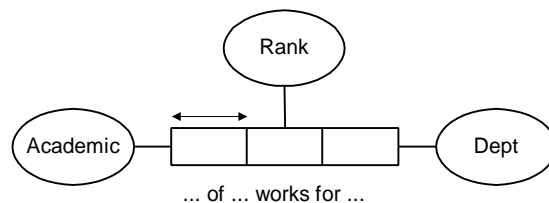
population for each fact type in this regard, though simple questions are usually more efficient.



**Figure 4:** Some of the fact types, with uniqueness constraints added

The internal uniqueness constraints on the binary fact types assert that each academic has at most one rank, holds at most one chair (and vice versa), works for at most one department, and has at most one employee name. The external uniqueness constraint stipulates that each department–empname combination applies to at most one academic (i.e., within the same department, academics have distinct names). The constraint on the ternary says that for each academic–degree pair, the award was obtained at only one university.

Once uniqueness constraints have been added, an arity check is performed. A sufficient but not necessary condition for an n-ary fact type to be split is that it has a uniqueness constraint that misses two roles. For example, suppose we tried to use the ternary in Figure 5. Because each academic has only one rank and works for only one department, the uniqueness constraint spans just the first role. This misses two roles of the ternary; so the fact type must be split on the source of the uniqueness constraint into the two binaries: Academic has Rank; Academic works for Dept.



**Figure 5:** This fact type splits because two roles are missed by the uniqueness factor

If a fact type is elementary, all its functional dependencies (FDs) are implied by uniqueness constraints. For example, each academic has only one rank (hence in the fact table for Academic-has-Rank, entries in the rank column are a function of entries in the

academic column). If in doubt, we check for FDs not so implied; if such an FD is found, the fact type is split on the source of the FD.

## Step 5

Step 5 of the CSDP is to *add mandatory role constraints and check for logical derivations*. A role is mandatory (or total) for an object type if and only if every object of that type that occurs in the database must be known to play that role. This is explicitly shown by a *mandatory role dot* where the role connects with its object type. If two or more roles are connected to the same mandatory role dot, this means the *disjunction* of the roles is mandatory (i.e., each object in the population of the object type must play at least one of these roles).

For example, Figure 6 adds mandatory role constraints to some of the fact types already discussed. These dots indicate that each academic has a rank and works for a department; moreover each academic *either* is contracted until some date *or* is tenured. Roles that are not mandatory are *optional*. The role of having a chair is optional. The roles of being contracted or being tenured are optional, too, but their disjunction is mandatory. If an object type plays only one fact role in the global schema, then by default this is mandatory, but a dot is not shown (e.g., the role played by Rank is mandatory by implication).

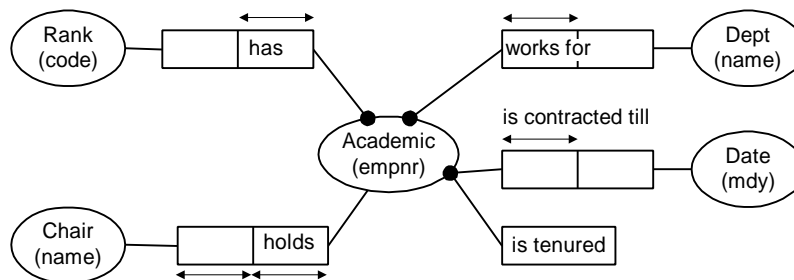


Figure 6: Some of the fact types, with mandatory role constraints added

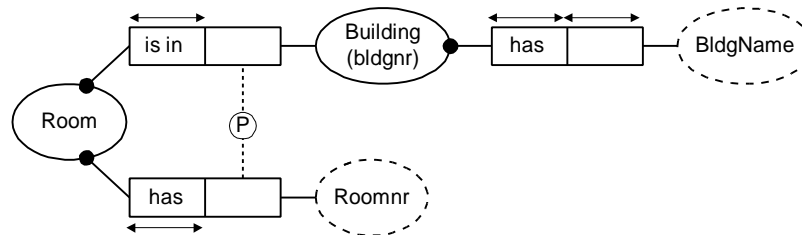
Now that we have discussed uniqueness and mandatory role constraints, reference schemes can be better understood. Simple reference schemes involve a mandatory 1:1 mapping from entity type to value type. For example, the notation “Rank (code)” abbreviates the binary reference type: Rank has Rankcode. If shown explicitly, both roles of this binary have a simple uniqueness constraint, and the reference role played by Rank has a mandatory role dot.

With composite reference, a combination of two or more values may be used to refer to an entity. For example, while *empnr* provides a simple primary identifier for *Academic*, the combination of *Dept* and *EmpName* provides a secondary identification scheme. Sometimes composite schemes are used for primary reference. For example, suppose that to help students find their way to lectures, departmental handbooks include a building directory, which lists the names as well as the numbers of buildings. A sample extract of such a directory is shown in Table 3.

**Table 3:** Extract from a directory of buildings

Bldgnr	BldgName
...	...
67	Priestly
68	Chemistry
69	Computer Science
...	...

Earlier we identified rooms by a single value. For example “69-301” was used to denote the room in building 69 that has room number “301”. Now that buildings are to be talked about in their own right, we should replace the simple reference scheme by a composite one that shows the full semantics (see Figure 7). Here Roomnr now means just the number (e.g., “301”) used to identify the room within its building. This is used in conjunction with the Buildingnr to identify the room within the whole university. To explicitly indicate that the external uniqueness constraint provides the *primary* reference for Room, the circled “u” may be replaced by a circled “P” (see Figure 7).



**Figure 7:** Room has a composite, primary reference scheme

Knowledge of uniqueness constraints and mandatory roles can assist in deciding when to *nest* a fact type. The ternary in Figure 4 could have been modeled by nesting as follows. First declare the binary: Academic obtained Degree. Now objectify relationship instances of this type by wrapping a *frame* (rounded rectangle) around the predicate, and adding a name (e.g., “DegreeAcquisition”). Now attach another binary predicate to this frame to connect it to University. This yields the nested version: DegreeAcquisition[Academic obtained Degree] was from University.

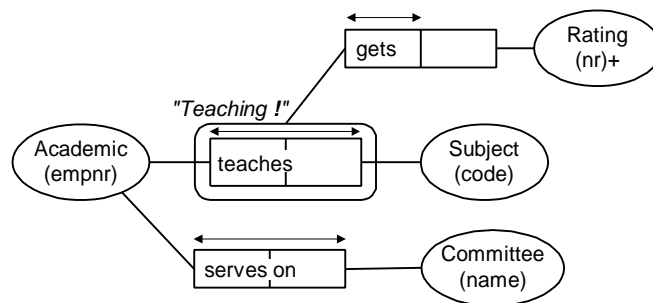
In this case, the objectified predicate plays only one role, and this role is mandatory. Whenever this happens we prefer the flattened version instead of the nested version, because it is more compact and natural, and it simplifies constraint expression. In all other cases, the nested version is preferred (i.e., choose nesting if the objectified predicate plays an optional role, or plays more than one role).

As an example, suppose the application also has to deal with reports about teaching commitments, an extract of which is shown in Table 4. Not all academics currently teach. If they do, their teaching in one or more subjects may be evaluated and given a rating. Some teachers serve on course curriculum committees.

**Table 4:** Extract of report on teaching commitments

Empnr	Emp. name	Subject	Rating	Committees
715	Adams A	CS100 CS101	5	
430	Codd EF			
654	Wirth N	CS300		BSc-Hons CAL Advisory

Here the new fact types may be schematized as shown in Figure 8. By default, an objectified predicate is fully spanned by a uniqueness constraint, to ensure elementarity (this is implicit in the frame notation, but may be shown explicitly as in the figure). Because not all  $\langle \text{Academic}, \text{Subject} \rangle$  pairs involved in Teaching have a rating, nesting is preferred. To flatten this we would need a binary for teaching subjects, and a ternary for rating the teaching of subjects, with a pair subset constraint (see later) between them.



**Figure 8:** Example of nesting

The nested object type Teaching plays only one role, and this role is optional. So instances of Teaching can exist independently without having to play a fact role. This makes teaching an *independent* object type. The independence of an object type is indicated by appending “!” to its name.

The second stage of Step 5 is to check for logical derivations (i.e., can some fact type be derived from others without the use of arithmetic?). One strategy here is to ask whether there are any relationships (especially functional relationships) that are of interest but which have been omitted so far.

Another strategy is to look for transitive patterns of functional dependencies. For example, if an academic has only one phone extension and an extension is in only one room, these fact types determine the room of the academic. However, for our application the same extension may be used in many rooms, so we discard this idea.

Suppose however that our client confirms that the rank of an academic determines the access level of his or her extension. For example, suppose a current business rule is that professors get international access while lecturers and senior lecturers get local access. This rule might change in time (e.g., senior lecturers might be arguing for national access).

To minimize later changes to the schema, we store the rule as data in a table (see Table 5). The rule can then be updated as required by an authorized user without having to recompile the schema.

**Table 5:** A functional connection between rank and access level

Rank	Access
P	INT
SL	LOC
L	LOC

Suppose we verbalize the fact type underlying Table 5 as: Rank ensures AccessLevel. The three fact instances listed in the table can be used to derive the access level of the hundreds of academic extensions, using the following derivation rule:

**define** Extension provides AccessLevel **as**  
 Extension is used by **an** Academic **who** has **a** Rank  
**that** ensures AccessLevel

Examination of the related portion of the schema indicates that this rule is safe only if each extension is used by only one academic, or at least only by academics of the same rank. Let us assume the first, stronger condition is verified by the client. In the case of the weaker condition, the constraint must be specified textually rather than on the diagram. At any rate, by adding the Rank ensures AccessLevel fact type and the above derivation rule, we can remove the Extension provides AccessLevel fact type from the diagram.

## Step 6

In *Step 6* of the CSDP, we add *any value, set comparison and subtyping constraints*. Value constraints specify a list of possible values for a value type. These usually take the form of an enumeration or range, and are displayed in braces beside the value type or its associated entity type. For example, Rankcode is restricted to {'P','SL','L'} and AccessLevelcode to {'INT','NAT','LOC'}. These are displayed in the final, global conceptual schema (Figure 10).

Set comparison constraints specify subset, equality, or exclusion constraints between compatible roles or between sequences of compatible roles. Roles are compatible if they are played by the same object type (or by object types with a common supertype, described later). In Figure 10, a pair-subset constraint (4) runs from the heads predicate to the works-for predicate, indicating that a person who heads a department must work for the same department.

An equality constraint (34) is equivalent to a pair of subset constraints (one in each direction). For example, in this application a person's home phone is recorded if and only if the person heads some department. This could be depicted by an equality constraint between the first roles of two fact types: Professor heads Dept; Professor has home-Phone.

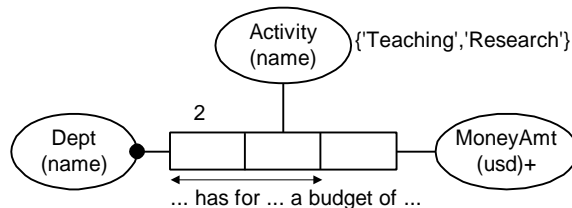
However we later choose another way of modeling this. The constraint that nobody can be tenured and contracted at the same time is shown by an exclusion constraint ( $\otimes$ ).

Subtyping is determined as follows. If an optional role is played only by some well-defined subtype, a subtype node is introduced with this role attached. Subtype definitions are written below the diagram and subtype links are shown as directed line segments from subtypes to supertypes. Figure 10 contains three subtypes: Teacher, Professor, and TeachingProf. In this university, each teacher is audited by another teacher (auditing involves observation and friendly feedback). Moreover, only professors may be department heads, and only teaching professors can serve on curriculum committees (not all universities work this way).

## Step 7

Step 7 of the CSDP adds other constraints and performs final checks. We briefly illustrate two other constraints. The audits fact type has both its roles played by the same object type (this is called a ring fact type). The  $^{\circ}ir$  notation beside it indicates the predicate is *irreflexive* (no teacher audits himself/herself).

Suppose we also need to record the teaching and research budgets of the departments. We might schematize this as in Figure 9. Here the “2” beside the role played by Dept is a *frequency constraint* indicating that each department that is included in the population of that role must appear there twice. In conjunction with the other constraints, this ensures that each department has both its teaching and research budgets recorded.



**Figure 9:** Each department has two budgets

The CSDP ends with some final checks to ensure that the schema is consistent with the original examples, avoids redundancy, and is complete. No changes are needed for our example. There is a minor derived redundancy, because if someone heads a department, we know from the subset constraint that this person works for that department; but this is innocuous. Other schematizations are possible (e.g., we can define works in and heads to be pair-exclusive, or use a unary is head instead of the binary heads) but we ignore these alternatives here.

Once the global schema is drafted, and the target DBMS decided, various optimizations can usually be performed to improve the efficiency of the logical schema that will result from the mapping. Assuming the conceptual schema is to be mapped to a relational database schema, the fact type in Figure 9 will map to a separate table all by itself (because of its composite uniqueness constraint). Because some other information about departments is mapped to another table, if we want to retrieve all the details about



departments in a single query we will have to perform a table join. Joins tend to slow things down. Moreover, we probably want to compute the total budget of a department, and with the current schema this will involve a self-join of the table, because the details of the two budgets are on separate rows. We can avoid all these problems by transforming the ternary fact type in Figure 9 into the following two binaries before we map: Dept has teaching budget of MoneyAmt; Dept has research budget of MoneyAmt. These binaries have simple keys, and will map to the “main” department table.

Another optimization may be performed that moves the home phone information to Dept instead of Professor, but the steps underlying this are a little advanced, so we ignore a detailed discussion of this move here. Figure 10 includes both these revisions to the conceptual schema. For a detailed discussion on conceptual schema optimization, see chapter 9 of *Conceptual Schema and Relational Database Design*.

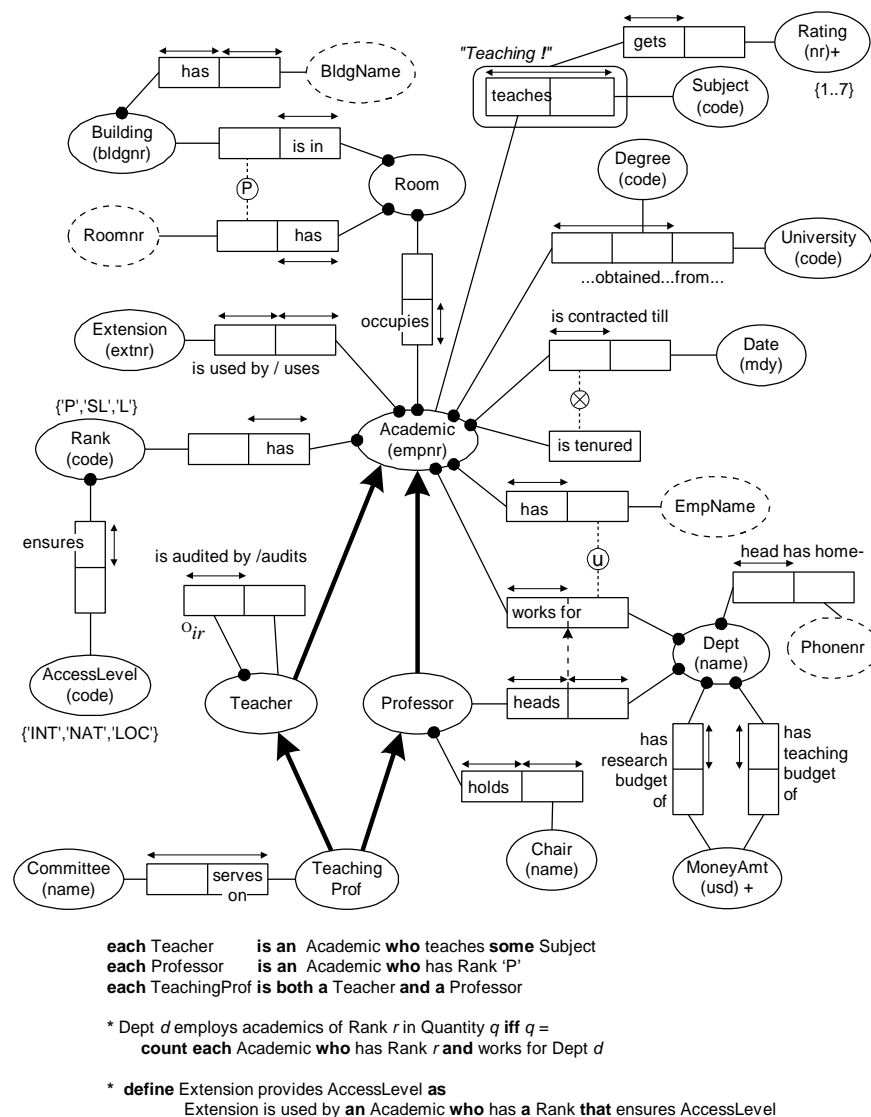


Figure 10: The final conceptual schema

---

## Relational mapping

Once the conceptual schema is specified, a simple algorithm is used to group these fact types into normalized tables. If the conceptual fact types are elementary (as they should be), then the mapping is guaranteed to be free of redundancy, because each fact type is grouped into only one table, and fact types that map to the same table all have uniqueness constraints based on the same attribute(s).

Before discussing the mapping, we define a few terms. A *simple key* may be thought of as a uniqueness constraint spanning exactly one role; a *composite key* is a uniqueness constraint spanning more than one role. A *compidot* (*compositely identified object type*) is either a nested object type (an objectified predicate) such as Teaching, or a co-referenced object type (its primary reference scheme is based on an external uniqueness constraint) such as Room.

The basic stages in the mapping algorithm are as follows.

1. Initially treat each compidot as an atomic “black box” by mentally erasing any predicates used in its identification, and absorb subtypes into their supertype.
2. Map each fact type with a composite key into a separate table, basing the primary key on this key.
3. Group fact types with simple keys attached to a common object type into the same table, basing the primary key on the identifier of this object type.
4. Unpack each mapped compidot into its component attributes.

With stage 3, a choice may arise with 1:1 binaries. If one role is optional and the other mandatory, then the fact type is grouped with the object type on the mandatory side. For example, the head-of-department fact type is grouped into the department table. Other refinements to the algorithm have been developed (e.g., other options for 1:1 cases and subtyping, mapping of independent object types, certain derived fact type cases, and partially null keys), but we do not consider these here.

Conceptual constraints and derivation rules are also mapped down. An exhaustive treatment of the mapping procedure is beyond the scope of this document. The conceptual schema under discussion maps to the relational schema shown in Figure 11. A generic notation (partly graphical) is used to specify the tables and constraints of resulting relational schema, and derivation rules are expressed as SQL views.

Keys are underlined. If alternate keys exist, the primary key is doubly-underlined. A mandatory role is captured by making its corresponding attribute mandatory in its table (not null is assumed by default), by marking as optional (in square brackets) all optional roles for the same object type that map to the same table, and by running an equality/subset constraint from those mandatory/optional roles that map to another table.

Most conceptual constraint notations map down with little change. Constraints on lists of role lists (e.g., subset, equality, exclusion) map to corresponding constraints on the

attributes to which they map. Equality constraints may be shown without arrowheads. Subtype constraints are typically stated as qualifications on square brackets or as qualifications on intertable subset constraints.

Conceptual object types are semantic domains: as current relational systems do not support this feature, domain names are usually omitted. Syntactic domains (data types) may be specified next to the column names if desired: if the reference mode has a “+”, the default data type is numeric, else the default is character string; the designer typically chooses more specific data subtypes as appropriate.

The  $\langle 2,1 \rangle$  in the pair-subset constraint indicates the source pair should be reversed before the comparison. In other words, the ordered pairs populating Department(headempnr, deptname) must also occur in the population of Academic(empnr, deptname).

Derived tables are shown below the base tables. The notation “ $R(..) ::=$ ” is short for “create view  $R(..)$  as select”. As with conceptual schemas, relational schemas may be displayed with levels of information hiding (e.g., for a brief overview some or all of the constraint layers may be suppressed).

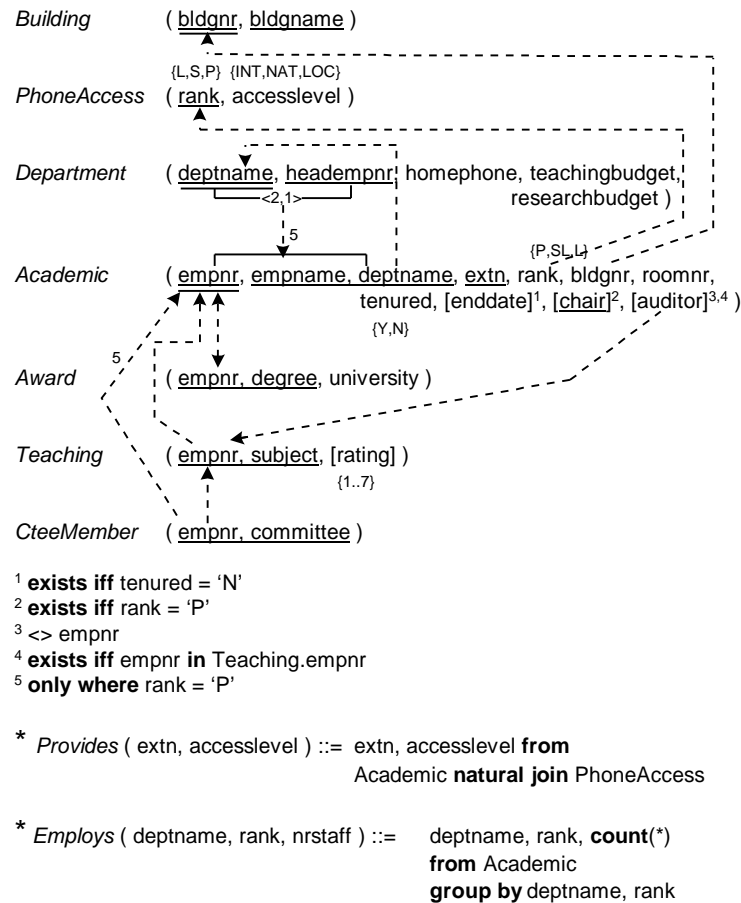
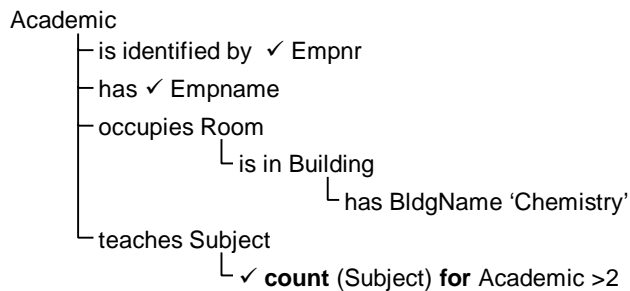


Figure 11: The relational schema mapped from Figure 10

---

## Conceptual queries

In addition to information modeling, ORM is also ideal for information querying. Using a language such as ConQuer (CONceptual QUERy language), an ORM model may be queried directly, avoiding the need to specify queries in terms of the underlying DBMS structures. As a simple example, consider the following English query on our academic database: list the empnr, empname and number of subjects taught for each academic who occupies a room in the Chemistry building and teaches more than two subjects. This can be expressed in ConQuer as follows:



A verbalization of the query is easily produced, and the SQL code is similar to the following:

```
select  X1.empnr, X1.empname, count(*)
from    Academic X1, Building X2, Teaching X3
where   X1.bldgnr = X2.bldgnr
        and X1.empnr = X3.empnr
        and X2.bldgname = 'Chemistry'
group by X1.empnr, X1.empname
having count(*) >2
```

As you can see, formulating queries in terms of objects and predicates is much easier than deciphering the semantics of the relational schema and coding in SQL or QBE.

---

## Object orientation

Although standard ORM includes some object-oriented (OO) features (e.g., inheritance), it differs from typical object-oriented approaches. This is actually a good thing, since OO approaches do not provide a clean conceptual path to information systems modeling. For example, OO methods are poor at providing structures that facilitate communication between modeler and client (e.g., consider fact instances and constraints), and they complicate the analysis phase with implementation details (e.g., immediate commitment on how facts are grouped into objects, and redundant specification).

As a simple example, consider two fact types from our academic UoD: Academic works for /employs Dept; Professor heads /is headed by Dept. Typically, OO approaches

store each of these twice, once for each object type. For example, the Academic “object” might include the single-valued attribute “deptWorkedFor” while the Dept object includes the set-valued attribute “academicEmployees”. The forward and inverse versions now need to be kept synchronized.

Consider the constraint that professors can head a department only if that is the department in which they work. In ORM this is specified declaratively, and checked with clients by populating the fact types. In OO this constraint is easy to miss, and if captured at all it is buried in procedural code. Worse still, because constraints, like facts, are supposed to reside inside objects, the modeler has to worry about where to put the constraint. Do we put it inside the Dept object, the Academic object, the Professor object, or all three? Clearly such decisions concern implementation details, and it is unwise to burden modelers with such decisions while they are trying to arrive at a conceptual picture of the UoD.

For such reasons, we recommend using ORM to develop the original conceptual model. After that, abstraction mechanisms may be applied to automatically generate an OO view of the ORM schema (in a similar way to generating an ER view). The conceptual schema may be implemented in a relational database, object-oriented database, or other database, by using a different mapping algorithm.

One OO aspect not covered here is the ability to encapsulate operations (or “methods”) inside objects. Standard ORM needs to be extended to cater for this as well as process/event modeling. We now briefly sketch how this can be done.

---

## Process/event modeling

Unlike data modeling approaches, which tend to fall within a few broad classes (e.g., ER, ORM), there are dozens of different approaches being used to model processes and events. Most of these can be used in conjunction with ORM. Visio® Professional already provides basic support for various approaches to process/event modeling (e.g., IDEF0), and more extensive support is currently under development.

Formal integration of the data/process/event models and cross-consistency checking requires that the relevant data model component is defined before its operations are bound to it. At the atomic level, processes can be translated into transactions comprised of addition/deletion of elementary facts. However a higher abstraction level is required for convenience. An algorithm can be used to identify major object types (e.g., Employee) on which not just attributes (as views of fact types) but operations (e.g., hire, promote, fire) can be defined. Basically we can work with an object-oriented view for this specification, while maintaining all the benefits of a flat ORM model underneath. Just as ER can be provided as an ORM view, techniques such as those within UML that apply operations to entity types can be supported as well.

---

## Further details

For a detailed treatment of Object Role Modeling, see Halpin, T.A. 1995, *Conceptual Schema and Relational Database Design*, 2nd edn, Sydney: Prentice Hall Australia, 1995 (available through Amazon.com). Another overview article by the author, titled “Business Rules and Object-Role Modeling,” appeared in the October 1996 issue of *Database Programming & Design*. For additional information on Object Role Modeling, see the Visio Web site ([www.visio.com](http://www.visio.com)).

---

*Copyright © 1997-98 Visio Corporation. Reproduced by permission. All rights reserved*