The StarView Intelligent Query Mechanism

R. D. Semmel
The Johns Hopkins University Applied Physics Laboratory
Laurel, MD 20723
rds@aplcen.apl.jhu.edu

D. P. Silberberg
The Space Telescope Science Institute
Baltimore, MD 21218
davids@stsci.edu

Abstract

The StarView interface is being developed to facilitate the retrieval of scientific and engineering data produced by the Hubble Space Telescope. While predefined screens in the interface can be used to specify many common requests, ad hoc requests require a dynamic query formulation capability. Unfortunately, logical level knowledge is too sparse to support this capability. In particular, essential formulation knowledge is lost when the domain of interest is mapped to a set of database relation schemas. Thus, a system known as QUICK has been developed that uses conceptual design knowledge to facilitate query formulation. By heuristically determining strongly associated objects at the conceptual level, QUICK is able to formulate semantically reasonable queries in response to high-level requests that specify only attributes of interest. Moreover, by exploiting constraint knowledge in the conceptual design, QUICK assures that queries are formulated quickly and will execute efficiently.

1. Introduction

The Space Telescope Data Archive and Distribution System (ST-DADS) is the repository for scientific and engineering data produced by the Hubble Space Telescope and ground system. Data sets recording the scientific results of six different astronomical instruments and more than twenty different sets of engineering data are archived onto optical disk at a rate of approximately one terabyte per year. Thousands of people with diverse interests, located around the world, are concerned with different aspects of the archived data. In particular, interests are shared among scientific investigators, guaranteed time observers, general observers, instrument calibration scientists, archival researchers, and project engineers.

ST-DADS is a computer cluster composed of archive, catalog, and host-computer subsystems. The archive subsystem is devoted to archiving, managing, and retrieving data sets. The catalog subsystem, or ST-DADS Catalog, is a descriptive database of the archived data. The host computers are devoted to managing user access to the ST-DADS system. StarView is the user interface software that facilitates access to ST-DADS. It is designed to operate on the ST-DADS host computers, allowing Telnet and dial-in access. It also can operate in client/server mode in which the client runs on a workstation and the server runs on the host computers.

Currently, the StarView user interface is a set of screens that allows users to browse the ST-DADS Catalog. Each screen is composed of a set of data fields appropriate for a specific search. Search qualifications are entered in screen fields and serve as selection criteria. When a request for data is submitted, StarView constructs the corresponding Structured Query Language (SQL) query and transmits it to the ST-DADS Catalog. The query is processed and the results are returned to the StarView screens; the returned records describe archived data sets. The user marks appropriate records

to indicate to StarView those data sets that are to be retrieved. StarView bundles the data set retrieval requests and transmits them to ST-DADS, which, in turn, returns the requested data via the Internet or other storage media.

As user interests vary significantly, it is not possible to predetermine the criteria for data selection. Furthermore, the ST-DADS Catalog currently consists of approximately 1500 attributes distributed among more than 40 relations, and grows at the rate of approximately 100 megabytes per year. As a result, developing an interface for ad hoc requests using traditional approaches has not been possible. Thus, while StarView provides more than forty screens with the most common sets of fields, these screens cannot be used to formulate queries in response to the many requests that have not been anticipated. Unfortunately, enabling users to build screens is not feasible, as typical users do not possess the expertise or knowledge of the database design to ensure that semantically meaningful results would be produced. Moreover, providing screens for the complete set of possible requests is not feasible because of the combinatorial nature associated with the many possible selection criteria involving multiple attributes distributed among multiple relations.

To simplify interface construction, a knowledge-based approach has been used that enables StarView to present a universal relation interface to users. This interface presents all database attributes as residing in a single relation, thus eliminating the need for explicit knowledge concerning the structural complexity of the underlying database design [Leymann 1989; Maier et al. 1984]. However, the interface must be "smart" about how various relations can be associated via relational joins, and it must ensure that semantically reasonable queries are generated in response to high-level requests that specify only attributes of interest.

StarView has been adapted to exploit the universal relation approach by allowing users to select attributes from an ad hoc query screen. Specifically, the complete list of ST-DADS attributes is displayed and the user selects those that correspond to his request. StarView then places the selected attributes on a temporary screen that is functionally equivalent to a defined screen. However, the temporary screen does not specify predefined joins. Instead, StarView passes the request to an intelligent query system known as QUICK (for "QUICK is a Universal Interface with Conceptual Knowledge") that formulates the query based on conceptual design knowledge and passes the query back to StarView for further processing.

In this paper, an overview of QUICK is presented. In the next section, semantic data modeling is discussed and the need for knowledge beyond the relational level is justified. In particular, an extended Entity-Relationship model is described that enhances the basic Entity-Relationship model with selected knowledge representation constructs, and a portion of the current design of ST-DADS using the extended constructs is shown. Then, in Section 3, the notion of contexts is introduced as a means for segmenting an EER conceptual schema into overlapping subgraphs of strongly associated objects that facilitate inference of valid relational joins. In Section 4, it is demonstrated how contexts can be used to automate query formulation and thus facilitate the construction of high-level and intelligent interfaces. Finally, the current status of development is described and some current research topics are discussed.

2. Semantic Data Modeling

Designing a database as complex as ST-DADS requires many person-years of effort. Unfortunately, the relational model, with its elegant, but simple, notions of relations and attributes, is too sparse a representation to use for direct modeling of complex domains. Instead, higher level representations typically are used that enable designers to communicate effectively with users as well as abstract an application domain to an appropriate level. These richer representations enable designers to model entities or objects in a world directly, and provide constructs for specifying explicit relationships or associations among entities [Hull and King 1987; Peckham and Maryanski 1988]. In contrast, the relational model does not distinguish between entities and relationships, requiring instead that each

abstraction be cast as a relation. Consequently, knowledge that was explicit at the conceptual level becomes implicit at the logical level. Moreover, knowledge that could be used to facilitate query formulation is lost as the world of interest is mapped to a set of relations (see Figure 1).

Exacerbating the loss of knowledge is the usual practice of relegating the conceptual schema to a minor role once the logical design has been completed. For instance, it is not uncommon for the conceptual schema to be used for initial documentation only. Once the logical schema has been created, changes often are made directly to it instead of to the conceptual schema, thus reducing the validity of the higher level representation. Moreover, users typically are not given access to the higher level conceptual schema, having to rely instead upon the sparse logical schema to formulate queries. Yet, the conceptual schema contains knowledge that can be used effectively for query formulation, and thus can serve as a knowledge base for an intelligent interface.

2.1. The Entity-Relationship Model

The Entity-Relationship (ER) model is the prominent semantic data model used for conceptual design. From a knowledge representation standpoint, the ER model resembles a restricted associationist scheme. Diagrammatically, entity types are represented by rectangles, and relationship types are represented by diamonds. As an example, consider Figure 2, where the entity type ARCHIVE-DATA-SET-ALL is related to the entity type OBSERVATION via the relationship type ADS-FROM-OBS. The ARCHIVE-DATA-SET-ALL entity type corresponds to one of the core relations in ST-DADS and contains general information about the class of data, generation time, and the name of the archive. Similarly, OBSERVATION corresponds to a core relation and contains information about the predicted and actual observations such as target position, magnitude, and duration of the exposure.

The two pairs of numbers associated with ADS-FROM-OBS indicate that there is a many-to-one relationship from ARCHIVE-DATA-SET-ALL to OBSERVATION. Specifically, a pair (MIN,MAX) indicates the minimum and maximum number of times a particular entity can participate in a set of relationship instances. Thus, the notation facilitates the specification of both cardinality ratio constraints (i.e., many-to-many, many-to-one, and one-to-one) and participation constraints (i.e., total and partial). Hence, an ARCHIVE-DATA-SET-ALL entity can be related to at most one OBSERVATION entity, but need not be related to any (i.e.,

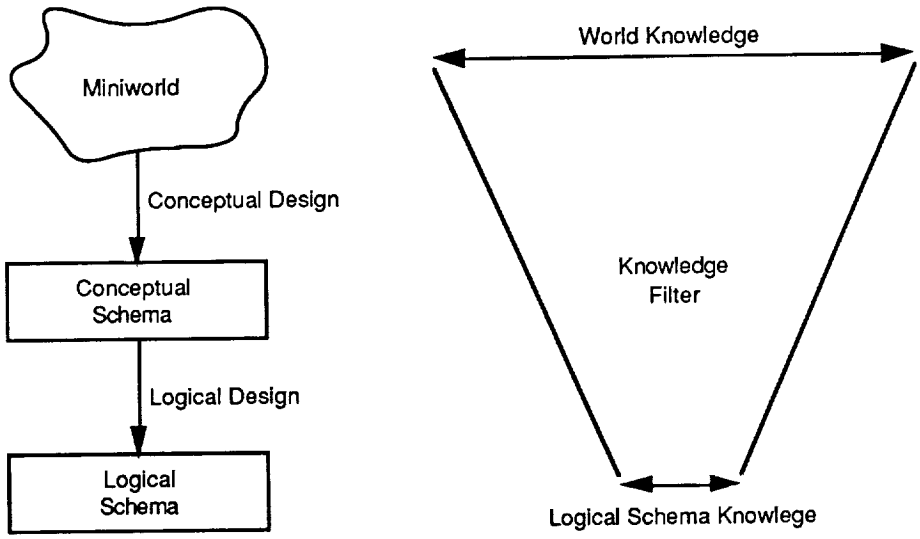

Figure 1. Knowledge lost during design.

participation is optional). On the other hand, an OBSERVATION entity must be associated with at least one ARCHIVE-DATA-SET-ALL entity (i.e., participation is total), and can be related to many.

As also shown in Figure 2, entity types (as well as some relationship types) have attributes, which are similar to the slots associated with frames, though attributes tend to have a restricted set of facets. Diagrammatically, an attribute is represented as a labeled oval connected via an edge to the ER object it characterizes. The set of attributes that can be used to uniquely identify a real-world object is known as the identifier, or key, and is shown underlined. (Note that only a few of the attributes associated with ARCHIVE-DATA-SET-ALL and OBSERVATION have been shown. Furthermore, the attribute names have been simplified to avoid confusion. As attributes do not play a central role in the discussion that follows, they will not be shown in subsequent diagrams.)

Given the simple ER specification of Figure 2, it is straightforward to create a corresponding set of relation schemas. First, entity types and their attributes are mapped to relation schemas. Then, the key attributes of the relation schema corresponding to the one-side entity type (i.e., OBSERVATION) are added as a foreign key to the relation schema corresponding to the many-side entity type (i.e., ARCHIVE-DATA-SET-ALL). The foreign key thus serves as the representation of the relationship type ARCHIVE-DATA-SET-ALL at the logical level.

The resultant relation schemas are as follows (note that logical level objects will use underscores instead of hyphens in names):

```
ARCHIVE_DATA_SET_ALL(Name,
   Archive_Class,Generation_Date,
   Access_Time,…,Update_Time,
   Program_ID,Obsnum,Obset_ID)

OBSERVATION(Program_ID,Obsnum,
   Obset_ID,Actual_Duration,…,
   Broad_Category)
```

As illustrated by the above relations, formulating even simple queries can be difficult at the logical level. For example, given a request for a list of PROGRAM-IDS, a formulator must know to use OBSERVATION instead of ARCHIVE_DATA_SET_ALL. While a heuristic could be used to favor the relation in which the attribute is an element of the key, the heuristic fails when more sophisticated knowledge representation constructs are employed. For example, a generalization lattice maps to a set of relation schemas, each of which has the same key. As a result, a request to list the key will be ambiguous at the logical level; however, as described below, this ambiguity is resolved easily at the conceptual level.

2.2. An Extended ER Model

The original ST-DADS conceptual schema used only the basic ER modeling constructs described above [Loral Aerosys 1992]. Though richer than the corresponding logical schema,

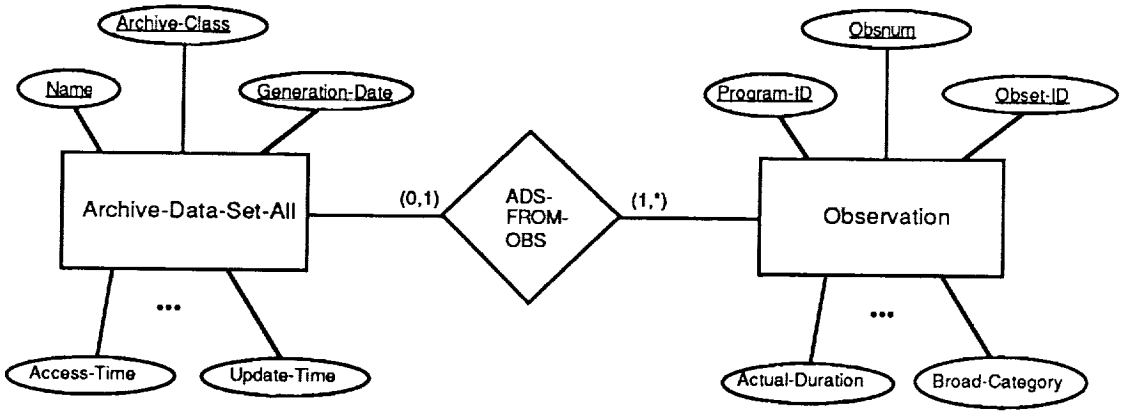

Figure 2. Simple ER objects.

the original conceptual schema did not accurately represent the ST-DADS domain. For, example the schema did not represent the fact that ARCHIVE-DATA-SET-ALL is a direct superclass of six subclasses. Instead, because generalization was not supported, simple relationships were used to associate the six subclasses to ARCHIVE-DATA-SET-ALL. However, this made it impossible to infer the hierarchical associations that actually existed. As a result, identifier attributes that should have been inherited from ARCHIVE-DATA-SET-ALL had to be explicitly replicated in the six subclasses, thus resulting in duplication of attributes and an inability to infer the primary ER object with which an attribute was associated. Furthermore, the hierarchical associations are disjoint, which means that an ARCHIVE-DATA-SET-ALL entity can participate as an entity in at most one subclass. Because the conceptual schema lacked this knowledge, a query formulator might incorrectly infer that the disjoint subclasses could be related via natural joins at the relational level.

As a result of the representational inadequacy of the basic ER model, an extended ER (EER) model has been adapted and augmented with constructs that are needed to model more complex domains [Batini et al. 1992; Elmasri and Navathe 1989; Teory 1986]. Figure 3 illustrates a portion (approximately half) of the current ST-DADS EER conceptual schema. Some of the more significant constructs are described below. In keeping with the basic ER model, entity types and basic relationship types are represented as in Figure 2.

In addition to basic entity types, weak entity types are supported in the QUICK EER model. (Weak entity types actually were defined in Chen's original model [Chen 1976]; however, many automated design tools do not support them.) Weak entity types lack a complete set of identifier attributes and, thus, a weak entity can be identified only with respect to an owner entity. For example, a DATA-SET-COMMENT entity, which provides a comment about a data set in the ST-DADS archive, can be identified only in the context of a specific ARCHIVE-DATA-SET-ALL entity. Thus, at the logical level, the key of the DATA_SET_COMMENT relation schema is the key of the ARCHIVE_DATA_SET_ALL relation schema concatenated with the partial identifier attributes from the DATA-SET-COMMENT entity type (i.e., USER-ID and COMMENT-TIME). Diagrammatically, a weak entity type is indicated by a double-edge rectangle connected to its identifying owner via an identifying relationship type, which is represented by a double-edge diamond.

Without the weak entity type abstraction, the identifier attributes of ARCHIVE-DATA-SET-ALL would have to be duplicated in the conceptual schema. With the abstraction, identifier attributes are associated only with their primary entity types. The remaining identifier attributes are, in a sense, inherited. (Note that attribute duplication cannot be avoided in the logical schema, as common attributes serve as associational links among relations.) Moreover, as illustrated in Figure 3, weak entity types can be defined in terms of other weak entity types. Thus, a set of DATA-SET-COMMENT-LINE entities makes sense only in the context of a particular DATA-SET-COMMENT entity. As before, identifier attributes are inherited, this time directly from DATA-SET-COMMENT and indirectly from ARCHIVE-DATA-SET-ALL.

Note that the weak entity type inheritance described above is different from the classical notion of inheritance in A.I. or object-oriented systems [Booch 1991; Rich and Knight 1991]. That is, a DATA-SET-COMMENT entity is not an ARCHIVE-DATA-SET-ALL entity; rather, a set of DATA-SET-COMMENT entities exists to support an ARCHIVE-DATA-SET-ALL entity, and thus is identified in the context of an ARCHIVE-DATA-SET-ALL entity. Consequently, only the identifier of the owner entity type is inherited as opposed to the complete set of attributes of the owner.

To deal with conventional inheritance, the generalization abstraction is provided, though the abstraction has been extended to deal with more complex types of inheritance as well. Diagrammatically, generalization is indicated by a labeled circle with an arrow directed toward the parent class. For example, Figure 3 illustrates that ARCHIVE-DATA-SET-ALL has six subclasses. The circle is labeled with a D,

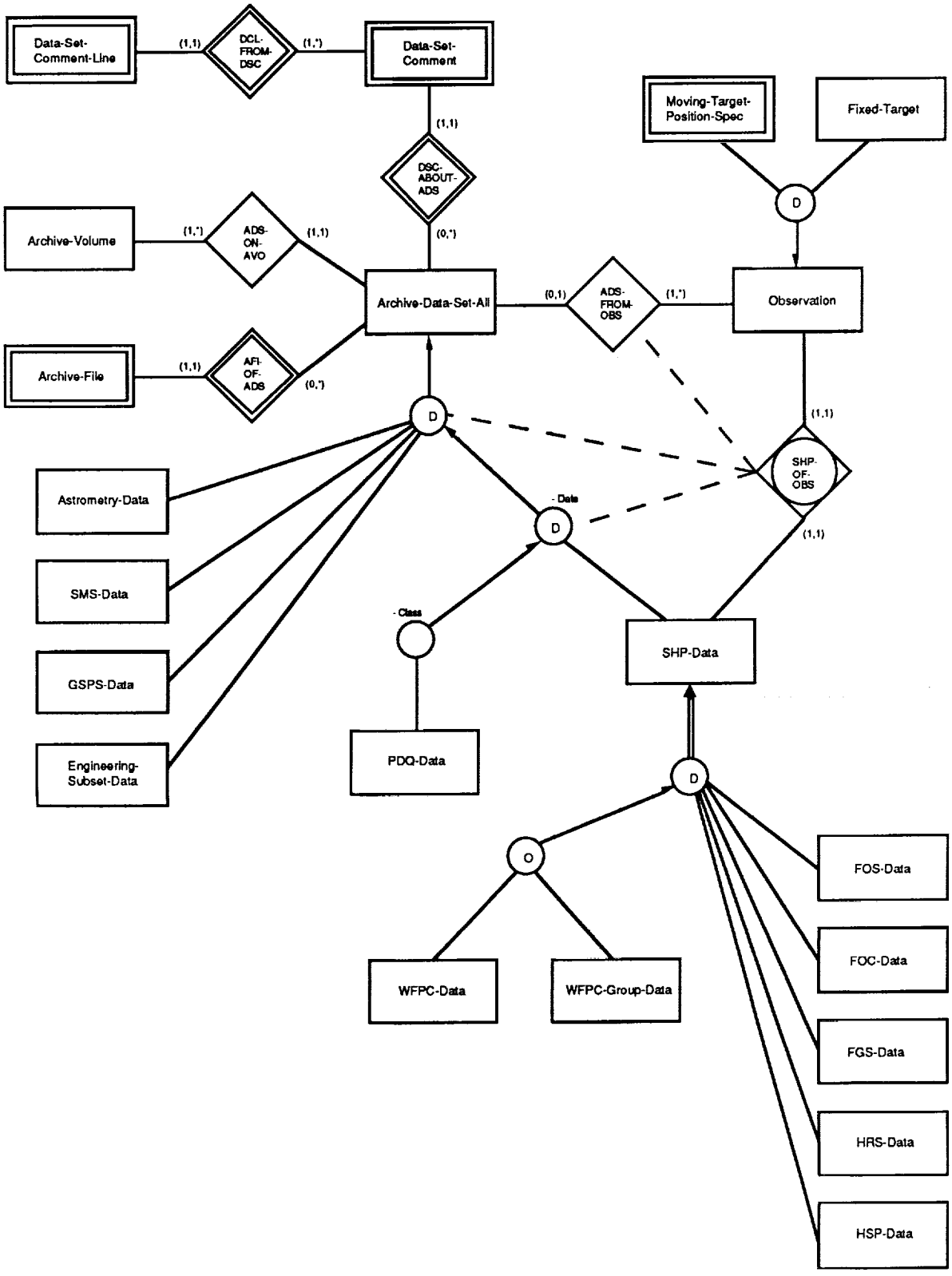

Figure 3. Portion of ST-DADS EER design.

C-4

indicating that the association is disjoint; therefore, an ARCHIVE-DATA-SET-ALL entity may participate in at most one subclass.

The four subclasses in the left portion of the diagram are direct subclasses, and thus inherit the complete set of attributes of ARCHIVE-DATA-SET-ALL. However, the two subclasses in the right portion of the diagram (i.e., SHP-DATA, a standard header packet containing telemetry values and spacecraft operation data, and PDQ-DATA, which contains product data quality data) are different. Consider, for example, the generalization type between ARCHIVE-DATA-SET-ALL and SHP-DATA. The identifiers are identical except that the DATE attribute exists in ARCHIVE-DATA-SET-ALL, but not in SHP-DATA. On cursory inspection, one might mistakenly classify ARCHIVE-DATA-SET-ALL as a weak entity type of SHP-DATA. However, the existence of an ARCHIVE-DATA-SET-ALL entity does not depend on an SHP-DATA entity. Rather, SHP-DATA is semantically a subclass of ARCHIVE-DATA-SET-ALL. In fact, the effect of removing DATE from the generalization type is to allow an SHP-DATA entity to be a child of multiple ARCHIVE-DATA-SET-ALL entities. The association of PDQ-DATA to ARCHIVE-DATA-SET-ALL is similar, except that only the NAME attribute is significant. Diagrammatically, such inheritance associations are represented as generalization types annotated with the identifier attributes that will not be inherited. If a circle is not labeled, as is the case with the PDQ-DATA generalization type, a simple subset association is indicated.

SHP-DATA has seven subclasses. However, in this case, each SHP-DATA entity must be associated with a subclass entity, as indicated by the double-edge arrow to SHP-DATA. This mandatory participation constraint stands in contrast to ARCHIVE-DATA-SET-ALL entities, which are not required to have corresponding subclass entities, as indicated by the single arrow to ARCHIVE-DATA-SET-ALL. The five subclasses on the right are direct subclasses, inheriting the identifier from SHP-DATA (and, therefore, from ARCHIVE-DATA-SET-ALL). Similarly, the two subclasses on the left (i.e., WFPC-DATA and WFPC-GROUP-DATA) correspond to an overlapping set of entities, as indicated by the o label on the generalization type circle. From a query formulation standpoint, this indicates that the relations corresponding to WFPC-DATA and WFPC-GROUP-DATA can be joined, though neither can be joined with relations corresponding to the other five subclasses of SHP-DATA. From a modeling standpoint, this representation was a concession resulting from the limitations of the underlying database management system. That is, conceptually, there is one WFPC entity type corresponding to the wide-field planetary camera; however, the huge amount of data produced by this camera requires that the relation corresponding to the conceptual WFPC entity set be split into two relations. To preserve mapping integrity from the conceptual level to the logical level, the conceptual notion of WFPC was split into two entity types at the EER level.

The disjoint association shown with OBSERVATION illustrates another facet of EER inheritance. Specifically, an OBSERVATION entity type can be classified as a fixed target or a moving target (or, strangely enough, as neither a fixed target nor a moving target as indicated by the single-edge arrow to OBSERVATION). The relationship of FIXED-TARGET to OBSERVATION is representative of the conventional inheritance association described previously. However, the relationship of MOVING-TARGET-POSITION-SPEC to OBSERVATION is different. In particular, a set of MOVING-TARGET-POSITION-SPEC entities corresponds to a single OBSERVATION, with the additional identifier attributes corresponding to date and time. Thus, a MOVING-TARGET-POSITION-SPEC is similar to a weak entity type, but, in this case, each MOVING-TARGET-POSITION-SPEC entity is an OBSERVATION entity. Furthermore, because of the disjoint generalization type, no MOVING-TARGET-POSITION-SPEC entity can exist as a FIXED-TARGET entity.

The final abstraction to be discussed is the optimization relationship type, as illustrated by SHP-OF-OBS in Figure 3. Optimization relationship types provide a means for representing efficiency decisions that have been made at the logical level to enhance

performance. Specifically, it sometimes is the case that joins of selected relations are needed on a regular basis. For example, in ST-DADS, OBSERVATION often is joined with ARCHIVE_DATA_SET_ALL, which in turn is joined with SHP_DATA. Often, ARCHIVE_DATA_SET_ALL is needed in the join only to associate OBSERVATION with SHP_DATA. As this sequence of joins is expensive, ST-DADS designers decided to improve performance by establishing a direct link from OBSERVATION to SHP-DATA. For requests involving only OBSERVATION and SHP-DATA, the direct link is used. If ARCHIVE-DATA-SET-ALL also were involved, then the longer join sequence would be used. Thus, SHP-OF-OBS serves as a direct link that informs a query formulator of the semantic equivalence of the path from OBSERVATION to SHP-DATA to the longer path from OBSERVATION through ARCHIVE-DATA-SET-ALL to SHP-DATA.

Note that if SHP-OF-OBS were represented as a simple relationship type, then a query formulator would have to infer that two different join paths existed that would result in semantically distinct results. On the other hand, specifying SHP-OF-OBS as an optimization relationship type ensures that the system can infer that the paths are equivalent, and thus serves to constrain the set of relations that will be used in the final query.

3. Contexts

From the portion of the ST-DADS EER conceptual schema shown in Figure 3, 21 relation schemas would be produced in the logical schema. However, even assuming that these 21 relation schemas constituted the complete logical schema would not enable a user employing only logical schema knowledge to query the ST-DADS system in a straightforward manner. For example, because of weak entity type and generalization type inheritance, the attribute NAME from ARCHIVE-DATA-SET-ALL appears in 17 relation schemas. Thus, if a request were made to list NAMES matching a specific pattern, a formulator would have to select some subset of the 17 relations. However, at the conceptual level, NAME occurs only in ARCHIVE-DATA-SET-ALL; consequently, a query formulator could infer immediately that only the relation corresponding to the entity type ARCHIVE-DATA-SET-ALL is needed.

In addition to attribute duplication, a formulator must be familiar with the rationale underlying the logical design to determine what is and, just as important, what is not a reasonable natural join. For example, from a purely syntactic standpoint, it would appear that any relation corresponding to a subclass of ARCHIVE-DATA-SET-ALL could be joined with any other subclass of ARCHIVE-DATA-SET-ALL because of the inherited, and, therefore, shared key attributes. However, as described in the previous section, these subclasses are disjoint and should not be joined. Once again, such knowledge is explicit in the conceptual schema.

While the conceptual schema is knowledge-rich, it is not necessarily appropriate for direct use as a database interface. For instance, the large number of attributes (i.e., several hundred) associated with many ST-DADS entity types precludes the use of graphical query languages that have been developed for various EER models [Czejdo et al. 1990; Zhang and Mendelzon 1983]. Moreover, the different views of the world held by different users, even at the conceptual level, makes it difficult to create a single, agreed-upon conceptual schema. Thus, it is better to shield the users from the conceptual schema as well as the logical schema.

By presenting a universal relation interface, users can conceive of the database as being structured as a single table of information. In turn, high-level requests must be mapped onto the underlying conceptual schema and subsequently translated into a query at the logical level. For this process to work, the underlying conceptual schema must satisfy two requirements:

1. It must be rich enough to support multiple views of the world.

2. It must map to the logical level in a straightforward way.

Given the many person-years of effort it takes to create a conceptual schema, it is reasonable to assume that Requirement 1 will be satisfied. If it is not, then, as with any knowledge acquisition task, additional effort must be expended in refining the schema. Requirement 2 is satisfied inherently by the QUICK EER model. Specifically, the abstractions in the QUICK EER model map directly to relations at the logical level in a straightforward manner. Furthermore, designers may employ various flags to control the mappings, thus ensuring that an appropriate logical schema will be produced from the conceptual schema. For example, the conceptual schema of Figure 3 was reverse-engineered from an existing logical schema. As the logical schema could not be changed, it was imperative that the conceptual schema map directly to it. By setting appropriate flags (e.g., allowing foreign key null values for selected relationship types), the desired mapping was realized without having to contrive the conceptual schema.

As there are certain join paths that are ruled out by the structures in a conceptual schema (e.g., by a disjoint generalization type), it makes sense to segment the schema into overlapping subgraphs of strongly associated objects. These subgraphs will be referred to as contexts [Semmel 1992], as they implicitly define what relations can be joined in a semantically reasonable way and, given a sufficiently rich conceptual schema, should correspond to classes of reasonable requests.

A context is maximal in the sense that no other object in the ER graph can be added to it without undermining the strong association criterion. However, determining what objects are strongly associated is not well-defined; instead, heuristics must be employed when determining strong association, as the existence of a path in the EER graph does not necessarily imply that a strong association exists among the objects in the path. Fortunately, there is a criterion for strong association at the logical level that can be abstracted to the conceptual level and used for automatic context generation. The criterion is based on a theorem in relational database theory that says if the intersection of the attributes of two relations multidetermines (or, by implication, functionally determines) one of the relations, then the two relations can be joined in a lossless manner [Korth and Silberschatz 1991].

At the conceptual level, the lossless join rule enables contexts to be defined inductively. Intuitively, a relationship type and its participating entity types form a context. This is justified by the fact that a relationship type conceptually contains the identifiers of each participating entity type; thus, a relationship type functionally determines each of its participating entity types. Similarly, an overlapping generalization type and its parent and children form a context, as each relation corresponding to an entity can be joined in a lossless manner. Finally, a disjoint generalization type with n children form n contexts, each consisting of the generalization type, the parent, and one of the n children.

Inductively, a context can be extended to include a relationship type and its participating entity types if a many-to-one or one-to-one cardinality ratio constraint exists from the context and no cycle is introduced in the extended context. This is justified by the fact that a localized functional dependency can be inferred from the entity type in the context to the other participating entity types of the relationship type. Similarly, a context can be extended to include a generalization type and one or more of its children. If the generalization type is disjoint, then for each child, a new context is formed consisting of the old context, the generalization type, and the child. If the generalization type is overlapping, then the original context is extended to include the generalization type and all of its children. As before, cycles are avoided, as joining a relation with itself at the logical level is superfluous.

The final way to extend a context is based on the notion of an articulation point. That is, if the removal of a relationship type disconnects the EER conceptual schema, then the relationship type and its participating entity types are included in the context. The justification for the disconnection rule is based on the conversion of an EER conceptual schema to a hypergraph and inferring multivalued dependencies when the hypergraph becomes

disconnected after removing a hyperedge corresponding to the relationship type [Ullman 1989].

Note that optimization relationship types are not considered when contexts are created. Rather, they are included as part of any context that has the bypassed association types specified by the optimization relationship type. Thus, in Figure 3, any context that included the three association types connected by dashed lines to SHP-OF-OBS would include SHP-OF-OBS as well. It is the role of the query formulator to use its knowledge of optimization relationship types to determine which set of joins is appropriate when the final query is constructed.

In the portion of the ST-DADS EER conceptual schema shown in Figure 3, there are 22 contexts. To see this, note that 11 contexts can be derived from the subclasses of ARCHIVE-DATA-SET-ALL. Four of these contexts are derived from the four subclasses on the left side of the diagram (i.e., ASTROMETRY-DATA, SMS-DATA, GSPS-DATA, and ENGINEERING-SUBSET-DATA). One of the contexts is derived from PDQ-DATA. Finally, six of the contexts are derived from SHP-DATA (i.e., one from the overlapping WFPC entity types and five from the remaining five subclasses of SHP-DATA). Then, for each of the 11 contexts, two new contexts are formed as the OBSERVATION generalization type is extended through to reach MOVING-TARGET-POSITION-SPEC and FIXED-TARGET. Thus, 22 contexts are produced. Of these 22 contexts, eight contain 15 EER objects, two contain 17 EER objects, ten contain 18 EER objects, and two contain 20 EER objects.

There are two more points worth mentioning with respect to contexts. First, in the worst case, the time it takes to generate contexts is an exponential function of the number of association types (i.e., relationship types and generalization types) in the EER conceptual schema. However, real-world designs tend to exhibit constraints that can be exploited to make automatic generation tractable (e.g., acyclic extensions can be initially pruned and then reintroduced once at the end of the context generation process). Exploiting these constraints for the complete ST-DADS EER conceptual schema enables contexts to be generated in less than one minute on a Lisp-based prototype of QUICK running on a Sun SPARCStation 2. As contexts need be generated only when the conceptual schema is created or modified, this performance is acceptable. The second point is that as contexts are generated heuristically, the contexts produced may not be consistent with designer expectations. In this case, the automatically generated set can be used as a starting point, and the set of contexts can be handcrafted. However, to this point in time, there has been no need for such handcrafting.

4. Automated Query Formulation

Given a set of contexts, query formulation is straightforward. First, the attributes in the high-level request are determined. Then, each context that covers the set of requested attributes is found. To ensure that needless joins are not performed, the found contexts are iteratively pruned of leaves until all leaves cover requested attributes. As a result of pruning, duplicate contexts may be introduced. As duplicate contexts are superfluous, they are eliminated. Then, the natural join orders of the remaining contexts are found. Finding these orders is straightforward, as the edges in the remaining context subgraphs identify valid natural join paths. The EER objects in each ordered context then are mapped to their underlying relation schemas. As some EER objects are represented by the same relation schema (e.g., in Figure 2, ADS-FROM-OBS and ARCHIVE-DATA-SET-ALL both map to the ARCHIVE_DATA_SET_ALL relation schema), duplicate relation schemas are eliminated. Then, subqueries are formulated and the union of the subqueries is returned as the final query.

Note that only when the conceptual schema contains cycles will more than one context be involved in the generation of the final query. This follows from the fact that only one path can exist between any two nodes in an acyclic graph, and, therefore, only one subtree will connect some set of nodes. The fact that there are multiple contexts does not affect this property, as context pruning will result in identical subtrees that, in turn, will be

eliminated. As optimization relationship types do not participate in the generation of contexts, no cycles exist in the portion of the ST-DADS conceptual schema shown in Figure 3.

To clarify the query formulation process, consider the following request:

> For all observations made by the faint object spectrograph between January 1, 1992 and February 1, 1992, display the target's description, right ascension, declination, proper motion and red shift, the instrument's detector and position angle of the aperture, and the relevant archived dataset names and comments about the datasets.

Recall that the StarView interface enables users to select attributes and qualify them on an ad hoc query screen. After the request specification is complete, it is translated into a form that QUICK can process. The language used by QUICK is referred to as USQL, as it resembles SQL, but assumes the existence of a universal relation. Thus, StarView passes the following USQL request to QUICK:

```
Select   target-descrip,
         ra,
         dec,
         ra-proper-motion,
         redshift,
         detector,
         pa-aper,
         data-set-name,
         line-text
Where    start-time >=
          "Jan 1, 1992"  And
         stop-time <=
          "Feb 1, 1992"  And
         instrume = "FOS"
```

QUICK begins processing by finding the applicable contexts. In this case, one context applies, as FOS-DATA is needed to cover DETECTOR, and FIXED-TARGET is needed to cover RA-PROPER-MOTION and REDSHIFT. As DATA-SET-COMMENT-LINE is needed to cover LINE-TEXT, only ARCHIVE-VOLUME, ADS-ON-AVO, ARCHIVE-FILE, and AFI-OF-ADS are pruned from the found context. Next, a natural join order is found for the conceptual schema objects of the pruned context:

```
(FIXED-TARGET,OBSERVATION-GT,
 OBSERVATION,ADS-FROM-OBS,
 ARCHIVE-DATA-SET-ALL,DSC-ABOUT-
 ADS,DATA-SET-COMMENT,DCL-FROM-
 DSC,DATA-SET-COMMENT-LINE,
 ARCHIVE-DATA-SET-ALL-GT,
 ARCHIVE-DATA-SET-ALL-SHP-GT,SHP-
 DATA,SHP-GT,FOS-DATA)
```

As will be discussed in more detail in the next example, QUICK semantically optimizes queries to ensure that the number of joins in the final query is minimized. For this example, QUICK recognizes that no attributes are requested from DATA-SET-COMMENT, which is used only to connect DATA-SET-COMMENT-LINE to ARCHIVE-DATA-SET-ALL. As DATA-SET-COMMENT-LINE inherits (via a weak entity type association) the identifier attributes of DATA-SET-COMMENT, the entity type DATA-SET-COMMENT and its identifying relationship type can be eliminated and replaced by a virtual link from ARCHIVE-DATA-SET-ALL to DCL-FROM-DSC. After mapping the remaining conceptual objects to the logical level, the following sequence of relation schemas is produced:

```
(FIXED_TARGET,OBSERVATION,
 ARCHIVE_DATA_SET_ALL,
 DATA_SET_COMMENT_LINE,
 SHP_DATA,FOS_DATA)
```

QUICK now exploits its knowledge of attribute mappings from the conceptual level to the logical level as well as its knowledge of natural join associations among relations to produce the final query. Specific to ST-DADS is the fact that designers provided distinct relation schema prefixes to all attributes, thus giving the impression that distinct conceptual attributes exist at the logical level. QUICK compensates for this by preserving attribute uniqueness at the conceptual level, recording prefix information, and inserting appropriate prefixes only when the final query is generated. However, QUICK also takes advantage of the fact that the logical-level attributes are syntactically unique to avoid relation

qualification. Here, then, is the final query generated in response to the request:

```
SELECT
        obs_target_descrip,
        obs_ra,
        obs_dec,
        fit_ra_proper_motion,
        fit_redshift,
        fos_detector,
        shp_pa_aper,
        ads_data_set_name,
        dcl_line_text
FROM
        fixed_target,
        observation,
        archive_data_set_all,
        data_set_comment_line,
        shp_data,
        fos_data
WHERE
        obs_start_time >=
          "Jan 1, 1992"  AND
        obs_stop_time <=
          "Feb 1, 1992"  AND
        shp_instrume =
          "FOS"  AND
        fit_program_id =
          obs_program_id  AND
        fit_obset_id =
          obs_obset_id  AND
        fit_obsnum =
          obs_obsnum  AND
        obs_program_id =
          ads_program_id  AND
        obs_obset_id =
          ads_obset_id  AND
        obs_obsnum =
          ads_obsnum  AND
        ads_archive_class =
          dcl_archive_class  AND
        ads_data_set_name =
          dcl_data_set_name  AND
        ads_generation_date =
          dcl_generation_date AND
        ads_archive_class =
          shp_archive_class  AND
        ads_data_set_name =
          shp_data_set_name  AND
        shp_archive_class =
          fos_archive_class  AND
        shp_data_set_name =
          fos_data_set_name
```

The complexity of the above query clearly demonstrates the need for an automated query formulation capability. Moreover, it takes only one to two seconds to generate the query, and the query is optimal with respect to the number of joins required. With relations as large as those in ST-DADS, join optimality is critical. However, achieving join optimality can be troublesome when code is generated from a high-level request and only logical level knowledge is available. Consequently, QUICK semantically optimizes final contexts via heuristic pruning to ensure that generated queries use the minimum number of relations and, thereby, require the minimum number of joins.

To gain an appreciation of QUICK's semantic query optimization capabilities, consider the following request:

```
List the data set comments for
astrometry data collected
during February 1993.
```

In USQL, the query can be expressed as follows:

```
Select  line-text,
        target-name,
        data-set-name
Where   generation-date >=
         "Feb 1, 1993"  And
        generation-date <
         "Mar 1, 1993"
```

As in the example above, first all contexts that cover the requested attributes are found. In this case, two contexts apply, both of which include ASTROMETRY-DATA, and one of which includes MOVING-TARGET-POSITION-SPEC and the other of which includes FIXED-TARGET. However, after pruning and duplicate elimination only one context remains:

```
(ASTROMETRY-DATA,ARCHIVE-DATA-
 SET-ALL-GT,ARCHIVE-DATA-SET-
 ALL,DSC-ABOUT-ADS,DATA-SET-
 COMMENT,DCL-FROM-DSC,DATA-SET-
 COMMENT)
```

At this point the conceptual schema objects are placed in natural join order and mapped to

relation schemas:

```
(ASTROMETRY_DATA, ARCHIVE_
 DATA_SET_ALL,DATA_SET_COMMENT,
 DATA_SET_COMMENT)
```

Without semantic query optimization, the following query is generated:

```
SELECT
        dcl_line_text,
        ast_target_name,
        ast_data_set_name
FROM
        astrometry_data,
        archive_data_set_all,
        data_set_comment,
        data_set_comment_line
WHERE
        ads_generation_date >=
         "Feb 1, 1993"  AND
        ads_generation_date <
         "Mar 1, 1993"  AND
        ast_data_set_name =
          ads_data_set_name  AND
        ast_archive_class =
          ads_archive_class  AND
        ast_generation_date =
          ads_generation_date AND
        ads_data_set_name =
          dsc_data_set_name  AND
        ads_archive_class =
          dsc_archive_class  AND
        ads_generation_date =
          dsc_generation_date AND
        dsc_data_set_name =
          dcl_data_set_name  AND
        dsc_archive_class =
          dcl_archive_class  AND
        dsc_generation_date =
          dcl_generation_date AND
        dsc_comment_time =
          dcl_comment_time  AND
        dsc_user_id =
          dcl_user_id
```

The above query produces a semantically valid result; however, the query can be improved. To see how, first note that the identifier of ARCHIVE-DATA-SET-ALL is inherited by ASTROMETRY-DATA, by DATA-SET-COMMENT, and (indirectly) by DATA-SET-COMMENT-LINE. Furthermore, the existence of an identifier value in any of the inheriting entity types implies that the value exists in an entity in ARCHIVE-DATA-SET-ALL. Similarly, the existence of an identifier value in DATA-SET-COMMENT implies that the value exists in an entity in DATA-SET-COMMENT-LINE. Thus, DATA-SET-COMMENT and ARCHIVE-DATA-SET-ALL serve only to associate ASTROMETRY-DATA with DATA-SET-COMMENT-LINE. That is, the conceptual level interpretation indicates that the join of the relations corresponding to ASTROMETRY-DATA and DATA-SET-COMMENT-LINE is equivalent to the join of the original four relations. Furthermore, as the attribute GENERATION-DATE was requested in the USQL WHERE clause, and is an identifier attribute of ARCHIVE-DATA-SET-ALL that is inherited, it can be derived from either ASTROMETRY-DATA or DATA-SET-COMMENT-LINE. Here, then, is the final query:

```
SELECT
        dcl_line_text,
        ast_target_name,
        ast_data_set_name
FROM
        astrometry_data,
        data_set_comment_line
WHERE
        ast_generation_date >=
         "Feb 1, 1993"  AND
        ast_generation_date <
         "Mar 1, 1993"  AND
        ast_data_set_name =
          dcl_data_set_name  AND
        ast_archive_class =
          dcl_archive_class  AND
        ast_generation_date =
          dcl_generation_date
```

There are several points worth noting about the above examples. First, the queries demonstrate the need for an interface that simplifies request specification. In this regard, USQL is a step in the right direction, but a higher level interface like StarView still is needed. Second, the use of conceptual knowledge ensures that semantically reasonable queries will be generated from high-level requests. Third, the queries are generated quickly, which facilitates user interaction with the interface; in fact, even with full optimization, queries typically are generated in less than two

seconds. Finally, the queries generally will execute as efficiently as handcrafted queries produced by experts.

5. Summary and Conclusions

The complexity of the ST-DADS logical schema prohibits most StarView users from generating semantically valid queries that correspond to ad hoc requests. However, by exploiting conceptual design knowledge, methods have been developed for automating query formulation and enabling a user to perceive the database as a universal relation. These methods have been incorporated in QUICK and rely on an extended ER model that enhances the current ST-DADS conceptual schema, thus ensuring that the conceptual schema will play a central role throughout the database life cycle.

By exploiting the notion of contexts, QUICK ensures that generated queries will be semantically reasonable. Moreover, the queries will be generated quickly and execute efficiently. The capabilities of QUICK also enable StarView designers to focus on interface issues instead of conceptual design issues. For example, user modeling can be addressed in the framework of a universal relation schema, thus simplifying the task of interface construction. Consequently, interface designers need not develop a deep understanding of the conceptual schema (which is described by a 900-page database design document [Loral Aerosys 1992]).

The first release of StarView is scheduled for May of 1993. Currently, QUICK is being integrated into StarView and will exist as a module in the May release. QUICK generates SQL queries for all of the screens provided by StarView as well as for the ad hoc field sets selected through the universal relation interface.

Current research efforts are focused in three areas. First, contexts are being extended to support arbitrary predicates. Such predicates could be used, for example, to restrict context access or to disambiguate among multiple contexts that apply to a request. Second, richer request structures are being explored. For example, to generate queries of arbitrary complexity (e.g., queries requiring the cartesian product of two relations), tuple variables are necessary. However, it is not clear how tuple variables can be presented to a user by the StarView interface in an intuitive manner. Furthermore, inferring tuple variables is a difficult problem for which no solution is apparent. Finally, some consideration is being given to alternative conceptual and logical models, which would facilitate the use of QUICK with object-oriented databases.

6. References

Batini, C., Ceri, S., and Navathe, S. B. 1992. *Conceptual Database Design: An Entity-Relationship Approach.* Benjamin/Cummings, Redwood City, CA.

Booch, G. 1991. *Object-Oriented Design with Applications.* Benjamin/Cummings, Redwood City, CA.

Chen, P. P. 1976. The Entity-Relationship Model - Toward a Unified View of Data. *ACM Transactions on Database Systems 1*, 1, 9-36.

Czejdo, B., Elmasri, R., Rusinkiewicz, M., and Embley, D. W. 1990. A Graphical Data Manipulation Language for an Extended Entity-Relationship Model. *Computer 23*, 3, 26-36.

Elmasri, R., and Navathe, S. B. 1989. *Fundamentals of Database Systems.* Addison-Wesley, Reading, MA.

Hull, R., and King, R. 1987. Semantic Database Modeling: Survey, Applications, and Research Issues. *ACM Computing Surveys 19*, 3, 201-260.

Korth, H. F., and Silberschatz, A. 1991. *Database System Concepts*, 2nd ed. McGraw-Hill, New York.

Leymann, F. 1989. A Survey of the Universal Relation Model. *Data & Knowledge Engineering 4*, 4, 305-320.

Loral Aerosys. 1992. *ST DADS Database Design Specification*, Build 2, Revision B.

Maier, D., Ullman, J. D., and Vardi, M. Y. 1984. On the Foundations of the Universal Relation Model. *ACM Transactions on Database Systems 9*, 2, 283-308.

Peckham, J., and Maryanski, F. 1988. Semantic Data Models. *ACM Computing Surveys 20*, 3, 153-189.

Rich, E., and Knight, K. 1991. *Artificial Intelligence*. McGraw Hill, New York.

Semmel, R. D. 1992. "Discovering Context in a Conceptual Schema," in *Proceedings of the First International Conference on Information and Knowledge Management* (Baltimore, Nov. 8-11), Yesha, Y., ed., International Society of Mini and Microcomputers - ISMM, pp. 222-230.

Teory, T. J., Yang, D., and Fry, J. P. 1986. A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model. *ACM Computing Surveys 18*, 2, 197-222.

Ullman, J. D. 1989. *Principles of Database and Knowledge-Base Systems*, Vol. 2. Computer Science Press, Rockville, MD.

Zhang, Z., and Mendelzon, A. O. 1983. A Graphical Query Language for Entity-Relationship Databases. In *Proceedings of the 3rd International Conference on Entity-Relationship Approach* (Anaheim, CA, Oct. 5-7), pp. 441-448.