A RAND NOTE

# Extending a Relational Database with Deferred Referential Integrity Checking and Intelligent Joins

Stephanie Cammarata, Prasadram Ramachandra, Darrell Shane

June 1989

RAND

The research described in this report was sponsored by the Defense Advanced Research Projects Agency under RAND's National Defense Research Institute, a Federally Funded Research and Development Center supported by the Office of the Secretary of Defense, Contract No. MDA903-85-C-0030.

This Note contains an offprint of RAND research originally published in a journal or book. The text is reproduced here, with permission of the original publisher.



Published by The RAND Corporation
1700 Main Street, P.O. Box 2138, Santa Monica, CA 90406-2138

A RAND NOTE

N-2990-DARPA

# Extending a Relational Database with Deferred Referential Integrity Checking and Intelligent Joins

Stephanie Cammarata, Prasadram Ramachandra,
Darrell Shane

June 1989

Prepared for
The Defense Advanced Research Projects Agency

RAND

# Extending a Relational Database with Deferred Referential Integrity Checking and Intelligent Joins[1]

Stephanie Cammarata (steph@rand.org)
Prasadram Ramachandra (ram@rand.org)
Darrell Shane (shane@rand.org)

The RAND Corporation
1700 Main Street
Santa Monica, CA 90406-2138
(213) 393-0411

## ABSTRACT

Interactive use of relational database management systems (DBMS) requires a user to be knowledgeable about the semantics of the application represented in the database. In many cases, however, users are not trained in the application field and are not DBMS experts. Two categories of functionality are problematic for such users: (1) updating a database without violating integrity constraints imposed by the domain and (2) using join operations to retrieve data from more than one relation. We have been conducting research to help an uninformed or casual user interact with a relational DBMS.

This paper describes two capabilities to aid an interactive database user who is neither an application specialist nor a DBMS expert. We have developed deferred Referential Integrity Checking (RIC) and Intelligent Join (IJ) which extend the operations of a relational DBMS. These facilities are made possible by explicit representation of database semantics combined with a relational schema. Deferred RIC is a static validation procedure that checks uniqueness of tuples, non-null keys, uniqueness of keys, and inclusion dependencies. IJ allows a user to identify only the "target" data which is to be retrieved without the need to additionally specify "join clauses". In this paper we present the motivation for these facilities, describe the features of each, and present examples of their use.

[1] This research was sponsored by the Defense Advanced Research Projects Agency under the auspices of RAND's National Defense Research Institute, a Federally Funded Research and Development Center sponsored by the Office of the Secretary of Defense. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official opinion of DARPA, the U.S. Government, or any person or agency connected with them.

Reprinted from *Proceedings of the 1989 ACM SIGMOD International Conference on the Management of Data*, Portland, Oregon, Vol. 18, No. 2, June 1989, pp. 88–97, James Clifford, Bruce Lindsay, David Maier (eds.), 

## 1. Introduction

With the advent of workstation environments, interactive software, and public domain databases, the use of DBMS is no longer limited to database administrators (DBAs), operations managers, and application programmers. Personnel in many different facets of a workplace are experimenting with DBMS for organizing, maintaining, and sharing information [McCa82]. In many cases, little or no database design is undertaken before a database is generated. Concerns for update anomalies and consistency maintenance, studied in theoretical discussions of relational database management, are rarely addressed in the practical data management activities of many organizations. Often, a novice user simply "relationalizes" a flat file into an intuitive set of tables. The resulting first normal form database implicitly relates tables through common attributes among the relations.

A complete representation of a user's application database should attempt to encode 1) the schema for every relation, 2) the data stored in the relations, and 3) the semantic relationships among relations. The first two categories are captured in every relational system. However, the semantics of the application is seldom expressed explicitly and is usually left to an individual user to interpret. Unfortunately, few DBMS tools and languages have facilities to store semantics and aid users in interpreting these semantics [Tou82, Blum87, Neuh88, Jone87]. Although common attributes between tables are based on underlying semantic relationships between relations, the relational model and its various implementations place no restrictions on the naming of attributes. Experienced users may establish their own conventions for relation and attribute names but no relational DBMS represents or enforces such conventions. Therefore, without an explicit conceptual model, it is difficult for users to access and validate the information they need [Curt81].

Developing an information model during database design is one means of expressing this information [Nava86]. However, commitment to such a formal effort is infrequent and the resulting model is usually a paper documentation aid, unavailable to interactive users. Another DBMS support tool, the data dictionary, interfaces a database to external applications by defining interface entities, application transactions, and generating reports, but is not suitable for a casual interactive user [Alle82, Dolk87]. Our approach recommends generating a knowledge base or information dictionary to capture previously implicit semantics of an existing relational schema and database. Research efforts toward integrating DBMS with expert systems have also adopted similar techniques [Reha85, Al-Z87, Schu88]. The universal relation model indirectly represents relational metadata by aiming to achieve complete access path independence [Maie82].

We have developed two capabilities, deferred Referential Integrity Checking (RIC) and Intelligent Join (IJ), which extend the operations of a relational database management system by utilizing explicit semantics and supplemental metadata combined with a relational schema. Deferred RIC is a database validation process that checks uniqueness of tuples, non-null keys, uniqueness of keys, and inclusion dependencies. This procedure can be invoked by the user or the system at specified intervals. IJ allows a user to identify only the "target" data which is to be retrieved without the need to additionally specify "join clauses". IJ subsequently navigates through the relations to generate the necessary join operations. These capabilities are supported by a metadata network constructed from both the relational schema and extended metadata stored in an information dictionary.

In the next section we present our previous work focussing on relational metadata and introduce an example database that we refer to throughout the remainder of the paper. Section 3 discusses a network representation for relational metadata which facilitates both Referential Integrity Checking and Intelligent Join algorithms. Discussion and examples of RIC are presented in section 4, and section 5 details IJ. We conclude with a discussion of the limitations, and suggestions for future work.

## 2. IID: An extended information dictionary

In previous work we have developed an Intelligent Information Dictionary (IID) to address the issues described above. IID serves as an interface between an interactive user and the query language of a relational DBMS [Camm88]. IID is implemented in Franzlisp Flavors running on a Sun Microsystems workstation. The dictionary communicates directly with the Ingres relational DBMS (also resident on a Sun machine) through the Lingres system, a Lisp to Ingres interface we previously implemented. Lingres provides the full functionality of the Quel query language, accessed from Lisp and Flavors.

### 2.1 IID functionality

IID aids a user in understanding the organization of a relational database by representing both the constructs of a relational database, and domain specific knowledge acquired from an application specialist. IID serves to augment a relational schema with supplemental metadata. By combining domain knowledge with knowledge of relational database concepts, IID supports interactive tools for browsing, customized data manipulation, and interactive value checking. Figure 1 shows the schema and extensional data of our test database, `atlas`. This geography database consists of seven relations, each with a primary key (which is underlined, "====", in figure 1). In many relational database applications, users are supplied with only the information shown in figure 1. The `atlas` database includes many of the typical anomalies found in first normal form databases.

In figure 2 we present portions of IID metadata, supplied by a domain expert, for the relations `animal` and `country` including a metadata description for the `atlas` database. In addition to metadata entries for relations, IID also represents column metadata such as value constraints, units information and conversion, and default values. IID is intended to serve the users' need for extended database capabilities, and not as a data model. However, because IID represents and uses the semantics of the database, considerable overlap exists between the capabilities of IID and ER (Entity-Relationship) or semantic modeling [Hull87]. In the future, we plan to develop a more complete modeling environment based on the existing IID framework.

One important component of an IID knowledge base is information about "interlinks". Interlinks represent the semantic information prescribing exactly how two or more relations are implicitly related. This information is generally referred to as integrity constraints, expressing structural conditions of a relational database. Knowledge of semantic interlink information and key attributes is essential for interactive users. However, without a facility like IID, there is no repository for this information. In figure 3, we show interlink metadata describing the relationships between the relations `weather`, `country`, and `vegetation`. Interlink identifiers correspond to the attribute "interlink-list" in figure 2. Specification of common columns or "join fields" are indicated in the "from-column-list" and "to-column-list". Information contained in interlinks combined with the attribute "key-list" found in the relation metadata make explicit the information needed by users for checking the referential consistency of a relational database and for manipulating the underlying data.

Until now, interlink information in IID was strictly passive. Facilities were available for a user to browse through an IID knowledge base to learn about the database. However, information about interlinks and keys did not actively contribute during the manipulation of the DBMS [Koss87, Gray88]. Our objective for the work described in this paper, was to build an operational extension to the capabilities of a relational DBMS which would automatically use this information to aid the user in retrieving data and checking for structural integrity.

### 2.2. Extended capabilities supported by IID

One important reason for representing relational metadata is to enable referential integrity checking (RIC). Validating referential integrity involves two categories of constraints: key constraints and referential constraints. A key constraint is implied by the existence of candidate keys and requires unique and non-null key values. Referential constraints are entailed by the relationship between a key in one relation and a foreign key in another. At any given time, the value of a foreign key in the first relation must be either null, or must be a key value in some tuple of the other relation.

Much research has addressed referential integrity, however, with the exception of Sybase, we know of no other commercial DBMS (excluding PC-based DBMS) which enforces referential integrity in real time during database manipulation [Casa88]. Our philosophy, however, is not to provide "immediate" referential checking. Instead, we are promoting "deferred" referential checking. In many cases, the benefits of immediate checking do not justify the excessive overhead [Lafu82, Hato88]. Deferred RIC can be initiated by the user or by the system at specified points in time. For example, many application databases are not designed using theoretical principles of relational normalization. Therefore, the consistency of such databses is questionable. Furthermore, users are not prevented from changing the value of a key attribute or violating inclusion dependencies. Deferred RIC fauna table

| country | animal | lat | long | distfeature |
|---|---|---|---|---|
| india | tiger | 60 | 75 | burning eyes |
| ussr | penguin | 70 | 100 | majestic |
| usa | dog | 40 | 100 | up-to-tricks |
| canada | penguin | 60 | 100 | sloppy |
| china | dog | 40 | 125 | listless |
| australia | kangaroo | 30 | 130 | jumps |
| | elephant | 60 | 75 | trunked |
| india | | 60 | 75 | |
| | | 60 | 75 | |
| india | dog | 60 | 75 | hooded |
| japan | tiger | 75 | 10 | burning-eyes |
| india | tiger | 60 | 75 | body-stripes |

country table

| country | latnorth | latsouth | longeast | longwest |
|---|---|---|---|---|
| india | 78 | 8 | 60 | 90 |
| ussr | 75 | 40 | 20 | 190 |
| usa | 50 | 30 | 125 | 65 |
| canada | 75 | 50 | 125 | 60 |
| china | 55 | 25 | 75 | 135 |
| australia | 10 | 40 | 115 | 155 |
| | 50 | 30 | 100 | 150 |
| india | 0 | 0 | 0 | 0 |

weather table

| latnorth | latsouth | longeast | longwest | zone | avgrain |
|---|---|---|---|---|---|
| 78 | 8 | 60 | 90 | tropic | 60 |
| 75 | 40 | 20 | 190 | tundra | 20 |
| 50 | 30 | 125 | 65 | temperate | 50 |
| 75 | 50 | 125 | 60 | tundra | 40 |
| 55 | 25 | 75 | 135 | tropic | 60 |
| 10 | 40 | 115 | 155 | temperate | 20 |
| 45 | 35 | 20 | 0 | mediterranean | 40 |
| 75 | 0 | 50 | 30 | | 50 |

natwildlife table

| country | natanimal | natbird |
|---|---|---|
| india | tiger | peacock |
| ussr | penguin | |
| usa | | eagle |
| australia | kangaroo | kiwi |
| china | | |
| canada | | |

economy table

| country | gnp | percapita |
|---|---|---|
| india | 210 | 300 |
| ussr | 2400 | 8370 |
| usa | 4200 | 17220 |
| canada | 450 | 17430 |
| china | 275 | 260 |
| australia | 231 | 12000 |
| usa | 4200 | 17220 |

animal table

| animal | distfeature | countryfound | maxspeed | anmtype | avgheight | avgweight | avglife |
|---|---|---|---|---|---|---|---|
| tiger | burning-eyes | india | 40 | carnivorous | 36 | 150 | [illegible] |
| elephant | trunk | china | 10 | herbivorous | 100 | 500 | [illegible] |
| kangaroo | jumps | australia | 25 | pouched | 84 | 180 | [illegible] |
| penguin | sloppy | ussr | 2 | polar | 36 | 100 | [illegible] |
| dog | listless | usa | 20 | omnivorous | 36 | 80 | [illegible] |

vegetation table

| zone | avgrain | maxtemp | mintemp | treetype | maincrop |
|---|---|---|---|---|---|
| temperate | 30 | 100 | -30 | deciduous | corn |
| tropic | 60 | 110 | 0 | evergreen | rice |
| tundra | 20 | 80 | -60 | coniferous | none |
| equatorial | 75 | 110 | 40 | evergreen | bamboo |

Figure 1: Atlas database with anomalies

```
Database-supplement: "ATLAS"
          name:             "ATLAS"
          description:      "atlas database contains an example of information about
                            different countries. This is used to illustrate the procedures
                            developed for Referential Integrity checking and Intelligent
                            Join operation."
          relation-list:    (WEATHER COUNTRY ECONOMY NATWILDLIFE ANIMAL VEGETATION FAUNA)
          interlink-list:   (exhibits/is-exhibited-by is-inhabited-by/lives-in
                            has/is-associated-with has-inhabitants/lives-at
                            supports/is-supported-by is-conducive-to/grows-in
                            is-represented-by/represents
                            has-characteristics/is-distinguished-by
                            has-same-distinguishing-features-as/
                            has-same-distinguishing-features-as)
          group-list:       (POSITION-VECTOR COUNTRY-NAME)
          ldatabase:        ATLAS

Relation-supplement: "COUNTRY"
          name:               "COUNTRY"
          name-explanation:   "This relation 'country' contains the position variables
                              for  every country ."
          description:        "Each record describes a country's approx. boundaries on
                              the four sides."
          column-list:        (COUNTRY LONGWEST LONGEAST LATSOUTH LATNORTH)
          key-list:           ((COUNTRY))
          group-list:         (POSITION-VECTOR COUNTRY-NAME)
          lrelation:          COUNTRY
          mydatabase:         ATLAS
          interlink-list:     (supports/is-supported-by has-inhabitants/lives-at
                              has/is-associated-with is-inhabited-by/lives-in
                              exhibits/is-exhibited-by)

Relation-supplement "ANIMAL"
          name:               "ANIMAL"
          name-explanation:   "animal is a relation containing all animals"
          description:        "Each record details an animal's characteristics"
          column-list:        (AVGLIFE AVGWEIGHT AVGHEIGHT ANMTYPE MAXSPEED COUNTRYFOUND
                              DISTFEATURE ANIMAL)
          key-list:           ((ANIMAL))
          group-list:         nil
          lrelation:          ANIMAL
          mydatabase:         ATLAS
          interlink-list:     (has-same-distinguishing-features-as/has-same-distinguishing
                              -features-as has-characteristics/is-distinguished-by
                              is-represented-by/represents is-inhabited-by/lives-in)
```

Figure 2: IID metadata

does not prevent these anomalies, however, it will subsequently detect the errors and notify the user of inconsistencies. Detailed discussion of RIC procedures and examples applying RIC to the `atlas` database are found in section 4.

Another compelling reason for needing to know interlink information focuses on the "select" or "retrieve" command. In both SQL and Quel, the user composes the desired selection using a qualification clause to effect an equijoin operation. In many cases, more than one equijoin over the relations is necessary to retrieve the desired information. For example, in the `atlas` database, suppose a user wishes to retrieve the `treetype` found in India. Ideally, the user would like to submit a query such as:

```
retrieve (country.country, vegetation.treetype)
       where country.country = "india"
```

However, this request requires a join across an intermediate relation, namely, `weather`. Therefore, to retrieve the desired data, the following query is necessary:

```
retrieve (country.country, vegetation.treetype)
        where country.country = "india" and
        country.latnorth = weather.latnorth and
        country.latsouth = weather.latsouth and
        country.longeast = weather.longeast and
        country.longwest = weather.longwest and
        weather.zone = vegetation.zone
```

In this example, the user must know implicit "interlink" information relating the relations `country` and `vegetation` and the user additionally needs to compose the query to join the relations `country`, `weather`, and `vegetation`. (In the remainder of this paper, use of the term "join" implies "equijoin".)

In effect, the user must navigate through the relations to establish the desired correspondences. It is interesting to note that when the relational model was introduced, declarative query languages were cited as an important benefit. Indeed, we have come a long way from tracing record pointers and maintaining currency indicators. Nevertheless, a notion of "navigation" still remains. Navigation in a relational context implies navigating through a "conceptual" model rather than a "physical" model. The Intelligent Join capability we have developed uses explicit metadata in IID to perform the

```
Interlink: "exhibits/is-exhibited-by"
   from-relation:     COUNTRY
   from-column-list: (LATNORTH LATSOUTH
                      LONGEAST LONGWEST)
   to-relation:    WEATHER
   to-column-list: (LATNORTH LATSOUTH
                    LONGEAST LONGWEST)
   cardinality: :ONE to 1.

   from-relation:     WEATHER
   from-column-list: (LATNORTH LATSOUTH
                      LONGEAST LONGWEST)
   to-relation:    COUNTRY
   to-column-list: (LATNORTH LATSOUTH
                    LONGEAST LONGWEST)
   cardinality: :ZERO-MANY to 1.


Interlink: "is-conducive-to/grows-in"
   from-relation:     WEATHER
   from-column-list: (ZONE)
   to-relation:    VEGETATION
   to-column-list: (ZONE)
   cardinality: :ONE to 1.

   from-relation:     VEGETATION
   from-column-list: (ZONE)
   to-relation:    WEATHER
   to-column-list: (ZONE)
   cardinality: :ZERO-MANY to 1.
```

Figure 3: IID interlinks

navigation and compose join clauses automatically for the user. In section 5 we provide a more detailed explanation and examples of IJ queries.

### 3. Metadata as a network structure

RIC and IJ capabilities require explicit representation of semantic metadata, which is traditionally recognized as implicit user knowledge. Metadata in IID, such as figures 2 and 3, is represented as object-oriented instances of classes such as "Relation-supplement", "Column-supplement", and "Interlink". However, to better facilitate navigation through a relational database, we augment our object-oriented IID representation with data structures corresponding to the network-like organization of a relational database. A network representation is generated directly from selective IID metadata. The network is constant throughout the life of the database (unless schema modifications are allowed); therefore, the overhead incurred by building the network is a one-time cost.

Given a relational schema, exactly one network can be produced which represents the database schema. The uniqueness of the network helps contribute to the computational efficiency when traversing and processing the network by eliminating the problem of recognizing and manipulating isomorphic networks. Although each relational schema maps to a single network, it is possible for more than one schema to map to a common network. This situation occurs when the databases have the same "meta-schema", that is, the same number of interlinks and the same kinds of constraints over those interlinks. In the remainder of this paper, references to a "relational schema" include supplemental metadata, such as keys and interlinks, and references to a "network" denote an IID metadata network.

#### 3.1. Building the metadata network

Nodes in the network represent relations, and links capture interlink information between the nodes. If $n$ interlinks are defined between two relations, then the corresponding nodes are connected by $n$ links. For example, in figure 2 there are nine interlinks declared in the `interlink-list` of the atlas `database-supplement`; therefore, nine links are represented in the corresponding network. Links not only represent an abstract relationship but also encode attributes which the two relations have in common. The network supports three different kinds of links depending on the characteristics of the common attributes in the relations denoted by the connected nodes. These characteristics distinguish between common attributes which are a key in *one, both,* or *neither* of the relations. We identify the corresponding link types as *singly directed, doubly directed,* or *undirected* links. Below we detail the specifications of each link type.

In a database with $k$ relations, $R_1, R_2, ..., R_k$, the corresponding metadata network has nodes $n_1, n_2, ..., n_k$ where the *ith* relation, $R_i$, is represented by $n_i$. An *undirected* link between two nodes, $n_i$ and $n_j$, indicates that there exists a set of attributes common to both relations, $R_i$ and $R_j$, which is a key in neither relation. A *doubly directed* link between $n_i$ and $n_j$ signifies that there exists a common set of attributes which is a key in both $R_i$ and $R_j$. A *singly directed* link emanating from $n_i$ indicates that there exists a set of common attributes between $R_i$ and $R_j$ that is a key in $R_j$. A singly directed link also implies that the common attributes are a foreign key in $R_i$. While there can be at most one undirected link between two nodes, there can exist zero or more singly and doubly directed links. Multiple singly directed links denote multiple foreign key/key relationships between two relations. More than one doubly directed link between two relations occurs only when there are multiple candidate keys in both relations.

The network corresponding to the `atlas` database is shown in figure 4, indicating nodes, links, and common attribute sets prescribed by the above rules. We have also underlined key attributes participating in each link. In this example, only one doubly directed link is found between any pair of nodes because we have defined only a single primary key for each relation in the `atlas` database. For some links the common attributes have the same name, for instance, `country` identifies the common attribute in both the `country` and `natwildlife` relations. However, attribute names are assigned for convenience and there is no restriction on naming conventions. For example, the common attributes between the relations `animal` and `natwildlife` are named `animal` and `natanimal` respectively. Therefore, the network must explicitly name the common attributes for both relations, and if there is more than one link between two nodes, proper correspondences must be maintained. It is also important to note that the contents of the database are not reflected in the network. Instead, the network captures the structure of the database among abstract entities such as relations and attributes. When it is necessary to refer to the contents of the database, for example, during RIC checking, we access the relations directly through Ingres.

So far, we have discussed only the foundations for the RIC and IJ capabilities. The network representation and encapsulation of a relational schema has proven to be an expressive and powerful formalism for explicit representation of relational metadata. In the subsequent sections, we present our methodology for integrity checking and intelligent joins within the framework of a metadata network.

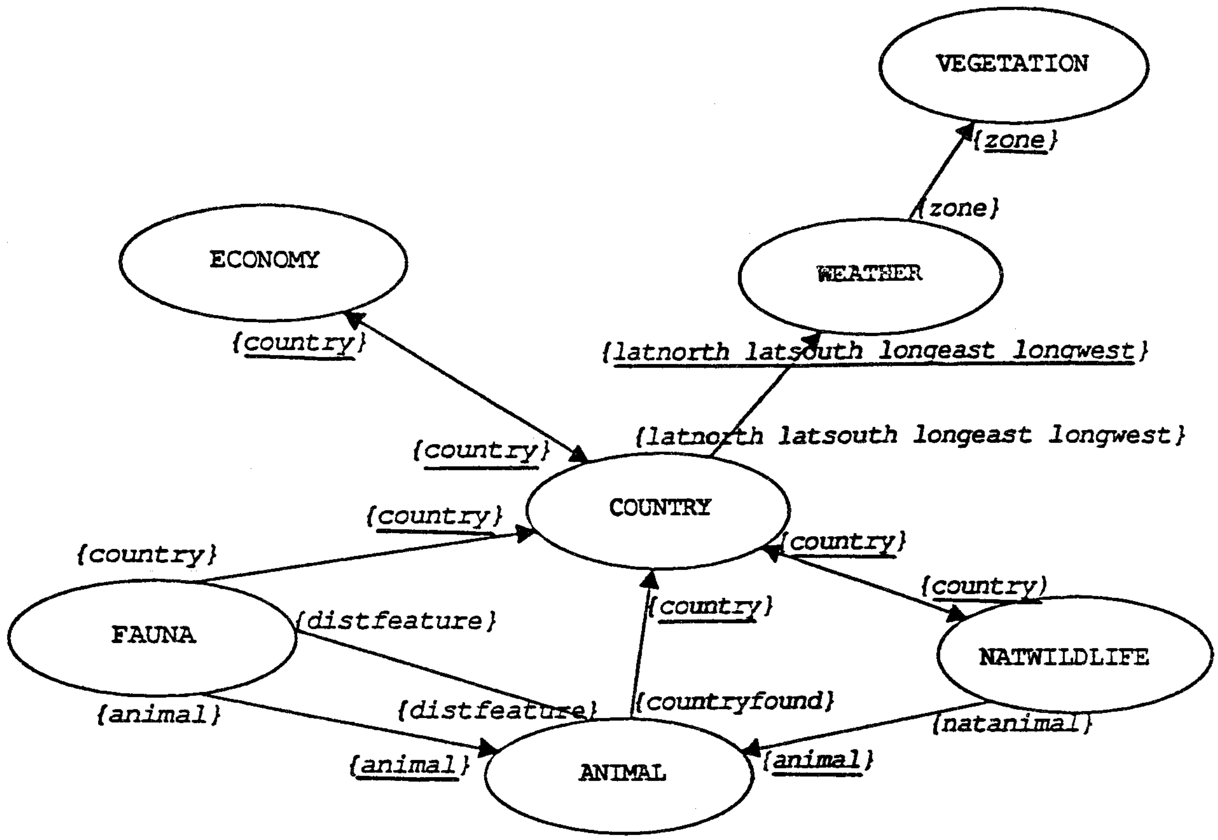


Figure 4: Metadata network with common attribute sets

## 4. Deferred referential integrity checking

This section describes work addressing one category of integrity constraints, namely, referential integrity. These constraints help describe the structural integrity of a relational schema and database. Unlike "value" constraints which encode information about allowable values and are enforced with predicates, referential integrity constraints focus on allowable mappings between relations. Although the particular values of an attribute are irrelevant, the use of those values for mapping across relations reflects the structural requirements of the database schema.

Two different approaches can be applied to enforce referential integrity. One method prevents integrity violations through the use of triggers and demons, and is intimately integrated into the data manipulation routines of the database [Ston85]. While this ensures a consistent database at all times, the associated overhead is usually high. The second option, which we have adopted, supports "deferred" referential integrity checking (RIC). In this approach, update anomalies are not blocked or prevented. Rather, deferred RIC performs a complete "sweep" of the database at regular intervals notifying the user of inconsistencies.

### 4.1. Background

We have observed the use of first normal form databases for simulation and modeling applications. In these domains, global databases are acquired as flat files from outside agencies. When corresponding Ingres databases are generated, they are already inconsistent and erroneous. Furthermore, users select and combine subsets of the global data to create their own local databases for their application models. Once their local databases are satisfactorily derived, few modifications are made to the data. In these simulation laboratories, deferred RIC is being used to validate entire global databases and also local derived databases. In addition, users can modify the integrity constraints of their local data by changing IID metadata. In this context, deferred RIC is an ideal solution for "scrubbing" inconsistent global databases and enforcing referential integrity within local versions.

Opponents of deferred validation argue that processing an entire database is expensive. Indeed, deferred checking is a dedicated, time-intensive process. However, in the scenario described above, validating the entire database is a task performed overnight when the database is idle. Local databases are usually small enough that they can be validated during a lunch hour! Although we have not yet performed any systematic studies or timings, our intuition suggests that many interactive database applications resemble the situations we have observed.

### 4.2. RIC algorithms

The current implementation to validate referential integrity entails three rules: The first rule requires non-null key values. The second restriction, also regarding key attributes, insists that key values be unique. The third rule addresses inclusion dependencies between foreign key and key attributes. However, before applying these rules to a database, deferred RIC first checks for duplicate tuples in each relation. Although the set-oriented theory underlying relational databases precludes duplicate "tuples", relational DBMS implementations do not enforce their uniqueness. The need for this type of validation check was motivated directly by the simulation and modeling applications discussed above, where duplicate tuples were responsible for extreme errors during selections which involved aggregation functions, such as, `count` and `sum`.

We have implemented RIC algorithms corresponding to the above four criteria. These procedures utilize information, such as keys and interlinks, encoded in the metadata network described in section 3. Because interlink mappings are identified by syntactic pattern matching across common attributes of relations, the RIC facility automatically generates the procedures necessary to validate referential integrity for any given relational schema.

In the following four subsections, we explain the method we have adopted for implementing each RIC rule. We also provide examples in each subsection demonstrating an application of the rule's procedure to the `atlas` database. The entire RIC process produces four error files corresponding to each RIC rule. The examples shown below are extracted from these output error files.

### 4.2.1. Unique tuples

The algorithm for validating uniqueness of tuples uses the Lingres (and corresponding Ingres) functions "count" and "countu[nique]". Both count and countu must return the same number of tuples to ensure uniqueness. We present portions of the error file, recording duplicate tuples in Figure 5. In the `atlas` database, there is only one occurrence of a duplicate tuple, found in the `economy` relation.

```
Checking for duplicate tuples in all relations

Relation COUNTRY:

 There are no duplicate tuples in COUNTRY.
Check completed


Relation ECONOMY:

There are duplicate tuples in ECONOMY.
Check completed
```

Figure 5: Checking for duplicate tuples

### 4.2.2. Non-null keys

To check for a null value among key attributes, the procedure constructs a retrieve command selecting those tuples with a null key value. If the retrieve is successful, that is, it returns at least one value, then this integrity constraint is violated in the selected tuple. Otherwise, referential integrity with respect to non-null keys is satisfied. In figure 6, we apply this constraint to the `fauna` relation. For relations with multi-attribute keys, such as `fauna`, this procedure identifies tuples where any of the key attributes are null.

### 4.2.3. Unique keys

Validating key uniqueness requires a retrieve command that selects all tuples where key attributes have the same value, and non-key attributes have different values. Selected tuples are flagged as corrupt. This procedure does not identify duplicate tuples because non-uniqueness of non-key attributes also holds. Therefore, for duplicate tuples the qualification is not fulfilled. Figure 7 shows uniqueness checking for the relations `vegetation` and `fauna`. No violations were identified in `vegetation` but two tuples were found in `fauna` with the same value for key attributes `animal` and `country`.

```
Checking for null values in key
        attributes of the relation FAUNA.

The key attributes are ("COUNTRY" "ANIMAL"):

LONG: 75
LAT: 60
DISTFEATURE: trunked
ANIMAL: elephant
COUNTRY:

LONG: 75
LAT: 60
DISTFEATURE:
ANIMAL:
COUNTRY: india

LONG: 75
LAT: 60
DISTFEATURE:
ANIMAL:
COUNTRY:

 End of null-key check.
```

Figure 6: Checking for null keys

```
Verifying Uniqueness of Keys for all relations

Checking for non-unique values of the key
   ("ZONE") in the relation VEGETATION
Pass completed for the key ("ZONE")

Checking for non-unique values of the key
   ("COUNTRY" "ANIMAL") in the relation FAUNA
LONG: 75
LAT: 60
DISTFEATURE: burning eyes
ANIMAL: tiger
COUNTRY: india

LONG: 75
LAT: 60
DISTFEATURE: body-stripes
ANIMAL: tiger
COUNTRY: india

Pass completed for the key ("COUNTRY" "ANIMAL")
```

Figure 7: Validating uniqueness of keys

### 4.2.4. Inclusion dependencies

Inclusion dependencies, unlike the previous three constraints, involve the comparison of attribute values between two relations. This constraint utilizes interlink relationships expressed between foreign keys and keys in the metadata network. The procedure verifies that the value of each non-null foreign key of one relation, is a key value in the other relation participating in the interlink. Our approach collects foreign key values by accessing attribute sets corresponding to singly directed links in the network. Membership of each foreign key in the corresponding set of key values is then verified. Verification of inclusion dependencies between relations `weather` and `vegetation` is demonstrated in figure 8. In this example, `zone` is a key in `vegetation` and a foreign key in `weather`. The procedure recognizes that the foreign key value "mediterranean", found in `weather`, is not a key value in `vegetation`.

```
Verifying Inclusion Dependency over all links

Inclusion Dependency Check for the path
 between the relations WEATHER and VEGETATION:

Relation WEATHER:

ZONE: mediterranean
ZONE: temperate
ZONE: tropic
ZONE: tundra


Relation VEGETATION:

ZONE: equatorial
ZONE: temperate
ZONE: tropic
ZONE: tundra

The following key values do not exist
   in the relation VEGETATION:
                  ((mediterranean)).
```

Figure 8: Verifying inclusion dependencies

## 5. Intelligent Join

The second relational DBMS extension we discuss in this paper is an Intelligent Join (IJ) capability. IJ eliminates the need for a user to compose join clauses when selecting data from more than one relation. IJ, like deferred RIC, is made possible by the explicit representation of IID metadata. This facility is particularly useful for non-professional database users where 1) the database has many relations; 2) mnemonics have not been used in attribute naming; or 3) typical database manipulation requires high inter-relation activity. IJ is a query pre-processor that generates required join clauses from an underspecified or incomplete retrieve command. With this capability, much of the burden of composing retrieve queries is shifted away from the user and onto the system.

Selecting data from more than one relation requires knowledge such as key and foreign key declarations, and relationships among tables in the database. Although this information is stored in an IID metadata network, the role of these semantics is more subtle and complex in IJ processing than in deferred RIC. During our IJ research, we identified two problematic issues which we have begun to address toward our objective of developing a fully general IJ capability. First, despite the availability of referential metadata, there exists multiple, semantically different sequences of join clauses, each fulfilling the user's *syntactic* retrieve request, but only one fulfilling the user's intended *semantics*. Second, if the user supplies partially complete join clauses, the solution which IJ generates must be compatible with and include the join specification provided by the user.

In the following two subsections we first present our analysis of the components of a Quel "retrieve" command and demonstrate the basic IJ functionality. (Although our examples use Quel syntax, analogous components and syntax are found in SQL.) Next, we detail the problems introduced above and describe how we have scoped the IJ task into manageable issues.

### 5.1. Interpreting a retrieve query

To help discuss the algorithms necessary for IJ, we have identified the components of a Quel retrieve statement as follows:

`retrieve` $t$ `where` $s_1$ `and` $s_2$ `and ...` $s_n$ `and` $j_1$ `and ...` $j_n$

where

$t$ is a *target attribute list* of the form: $(t_1, t_2, ..., t_n)$
and each $t_i$ is a legal Quel target item
representing a *projection*

$s_i$ is a legal Quel *selection clause* such as: *employee.age > 21*

$j_i$ is a legal Quel *join clause* of the form: $R_m.a_m = R_n.a_n$
such that $R_i$ is a relation (or range variable) in the
database and $a_i$ is an attribute in $R_i$

For example, in the Quel query introduced in section 2.2,

the target list is:
`(country.country, vegetation.treetype)`

the selection clause is:
`country.country = "india"`

the join clauses are:

```
country.latnorth = weather.latnorth
country.latsouth = weather.latsouth
country.longeast = weather.longeast
country.longwest = weather.longwest
weather.zone = vegetation.zone
```

The goal of IJ processing is to eliminate the need for the user to supply any join clauses. Conceptually, we transform retrieve queries into a sequence of joins followed by selections and projections. For instance, the algebraic representation of the above retrieve is the following (where subscripts indicate original database relations and superscripts denote derived relations):

$$R^1 = country \bowtie weather \bowtie vegetation$$

$$R^2 = \sigma_{country=india}(R^1)$$

$$R^3 = \pi_{country,treetype}(R^2)$$

The algebraic representation for the semantically identical, yet incomplete, query is:

$$R^1 = country \bowtie R_1 \bowtie R_2 \bowtie \cdots \bowtie vegetation$$

$$R^2 = \sigma_{country=india}(R^1)$$

$$R^3 = \pi_{country,treetype}(R^2)$$

In this paper, we limit our discussion to the problem of joining relations to produce $R^1$, e.g., determining a path through the network between nodes, `country` and `vegetation`. We define a path between nodes as a sequence of links that connects the nodes and has no cycles. Once IJ has determined the path, common attribute sets from the network are used to identify the join attributes. Figure 9 shows the two join clauses produced by IJ to join `country` and `vegetation`. The syntax produced in figure 9 is the prefix form of a Quel join clause where `(column` $R_i$ $a_i$`)` denotes the Quel syntax $R_i.a_i$. To compose a complete query, IJ builds a conjunction of the derived join clauses and the user-specified selection clauses.

Although this example shows path generation between only two relations, our implementation allows input of more than two relations.

```
-> (generate-join-clauses '(country vegetation))

(= (column WEATHER ZONE) (column VEGETATION ZONE))

(and
    (= (column WEATHER LATNORTH)
       (column COUNTRY LATNORTH))
    (= (column WEATHER LATSOUTH)
       (column COUNTRY LATSOUTH))
    (= (column WEATHER LONGEAST)
       (column COUNTRY LONGEAST))
    (= (column WEATHER LONGWEST)
       (column COUNTRY LONGWEST)))

(nil nil)
->
```

Figure 9: Join clauses for joining `country` and `weather`

### 5.2. IJ limitations

In the example presented above, there is only a single path between `country` and `vegetation` (in figure 4). However, between relations `fauna` and `natwildlife`, there are many possible paths. If more than one path exists, we have currently adopted the simplest solution, namely, choose the path with the least number of links. In cases where there exists only one minimal path, this approach is a reasonable decision criteria. However, when joining two nodes which have more than one link directly between them, such as figure 10, or where there are multiple minimal paths, it is necessary to resolve the ambiguity using more sophisticated rules. In these situations, the semantics of the join clause is radically different depending on the selected path. Consider the following example in figure 10, which shows a network corresponding to the following relational schema:

```
EMPLOYEE (ename, eaddress, eschool)
          =====

SCHOOL (sname, saddress, sprincipal)
        =====
```

The first query below requests the name, home address, school name, and school address of every employee. However, the second query requests, for every school, the school name, school address, name of the school's principal, and address of the school's principal.

```
Query 1:

retrieve (employee.ename, employee.eaddress,
          school.sname, school.saddress)
       where employee.eschool = school.sname

Query 2:

retrieve (school.sname, school.saddress,
          employee.ename, employee.eaddress)
       where school.sprincipal = employee.ename
```

The semantics (and resulting selection) of the two queries are very different depending on the attributes which are joined. However, the items in the target list are identical for both queries. Therefore, if only the target list of these queries were submitted to IJ, it is impossible to determine (without further information) which join clauses were intended by the user. Research is continuing on these issues.

We are striving for transparency of IJ processing from a user's point of view. This goal requires integration of IJ as part of a Lingres retrieve command. With full transparency, a user query would fit into one of four categories: 1) a complete query including all necessary join clauses; 2) a partially complete query containing some, but not all, required join clauses; 3) a query with no join clauses where the relations lie on a network path; or 4) a query (with or without join clauses) where the relations do not lie on a network path. IJ processing should initially determine if user-supplied join clauses produce a connected path. If a connected path is not reflected in the user's query, then user-supplied join clauses should help prune the set of potential paths. When a connected path between the necessary relations cannot be derived, a cross-product operation must be used to join two non-connected nodes. We have not yet studied the implications of manipulating disjoint paths in a network.

## 6. Concluding discussion

In this paper we have discussed a methodology for the representation of relational metadata and have demonstrated two relational DBMS capabilities, deferred Referential Integrity Checking and Intelligent Joins, made possible by a metadata network. In this concluding section we identify some limitations which we plan to address in future research.

The structure of our metadata network expresses interlink relationships between two relations but does not derive other more complex relationships. In the immediate future, we plan to strengthen IJ processing to compute these relationships dynamically. In the long term, however, we will be exploring the option of capturing and representing all relationships when the metadata network is generated. We expect the computational performance benefits to outweigh the costs of additional storage.

In the implementation of RIC we have combined the use of structural constraints with data values to perform referential integrity checking. Currently, the procedure to verify tuple uniqueness determines only the existence of duplicate tuples. It would be desirable to have an algorithm that could also help identify these duplicate tuples without having to incur the heavy cost associated with a naive implementation of it. In the short term, however, we plan to perform some benchmarks on the current RIC implementation. We will also be exploring the potential of representing and validating other types of constraints such as multi-valued dependencies.

In the previous section, we discussed some of the pending problems and limitations of IJ processing which we will be focusing on in the future. We have recognized IJ processing as a variation of graph traversal problems which are NP-complete. However, we assume that the number of relations in realistic situations will be small enough that our implementation of network traversal will be within the limits of our computational resources.

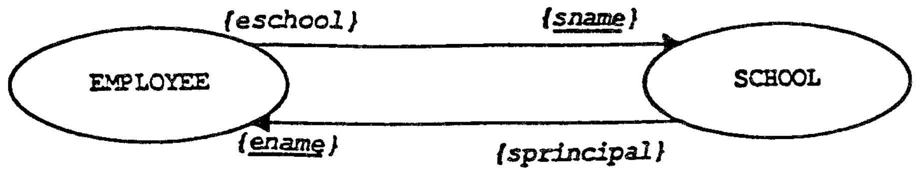


Figure 10: Network with potential for join ambiguity

References

[Al-Z87] Al-Zobaidie, A. and Grimson, J. B., "Expert systems and database systems: how can they serve each other?," *Expert Systems* 4(1), pp.30-37 (February 1987).

[Alle82] Allen, F. W., Loomis, M. E., and Manning, M. V., "The integrated dictionary/directory system," *ACM Computing Surveys* 14(2), pp.245-286 (June 1982).

[Blum87] Blum, B. I., Diamond, S. D., Hammond, M. G., Perkins, M. E., and Semmel, R. D., "A intelligent navigational assistant for a decision resource database," pp. 19-25 in *Proceedings of the third annual expert systems in government conference*, Washington, DC (October 1987).

[Camm88] Cammarata, S. J., "An intelligent information dictionary for the semantic manipulation of relational databases," in *Proceedings of the international conference extending database technology*, Venice, Italy (1988).

[Casa88] Casanova, M. A. and Tucherman, L., "Enforcing inclusion dependencies and referential integrity," pp. 38-49 in *Proceedings of the 14th international conference on very large data bases*, Los Angeles (1988).

[Curt81] Curtice, R.M., "Data dictionaries: An assessment of current practice and problems," pp. 564-570 in *Proceedings of 7th conference on very large data bases*, Cannes, France (September 1981).

[Dolk87] Dolk, D. and II, R. Kirsch, "A relational information resource dictionary system," *Communications of the ACM* 30(1), pp.48-61 (January 1987).

[Gray88] Gray, P., Storrs, G., and Boulay, J. du, "Knowledge representations for database metadata," *Artificial Intelligence Review* 2(1), pp.3-39 (1988).

[Hato88] Hatoun, T., *Deferred vs. immediate checking for consistency of databases*, Management Information Systems Week (June 6, 1988).

[Hull87] Hull, R. and King, R., "Semantic database modeling: Survey, applications, and research issues," *ACM Computing Surveys* 19(3), pp.201-260 (September 1987).

[Jone87] Jones, P. and Shave, M., "A language for simple interactive retrieval from a database system," *Data and Knowledge Engineering* 2(4), pp.303-321 (1987).

[Koss87] Kossman, R., "An active information resource dictionary," in *Proceedings of Ingres user association meetings*, San Francisco, CA (April 1987).

[Lafu82] Lafue, G., "Semantic integrity dependencies and delayed integrity checking," pp. 292-299 in *Proceedings of the 8th international conference on very large data bases*, Los Angeles (1982).

[Maie82] Maier, D., Ullman, J. D., and Vardi, M. Y., "Equivalence of universal relation definitions," Report No. STAN-CS-82-940, Computer Science Dept, Stanford University (1982).

[McCa82] McCarthy, J.L., "Metadata management for large statistical databases," in *Proceedings of 8th conference on very large databases* (September 1982).

[Nava86] Navathe, S. and Kerschberg, L., "Role of dictionaries in information resource management," *Information and Management* 10(1), pp.21-46 (January 1986).

[Neuh88] Neuhold, E. J. and Schrefl, M., "Dynamic derivation of personalized views," pp. 183-194 in *Proceedings of the 14th international conference on very large databases*, Los Angeles (1988).

[Reha85] Rehak, D. R. and Howard, H. C., "Interfacing expert systems with design databases in integrated CAD systems," *Computer Aided Design* (November 1985).

[Schu88] Schur, S., "Intelligent databases," *Database Programming and Design* 1(6), pp.46-53 (June 1988).

[Ston85] Stonebraker, M. and Rowe, L.A., "The design of POSTGRES," Memorandum No. UCB/ERL 85/95, University of California, Berkeley, Berkeley, CA (November 15, 1985).

[Tou82] Tou, F. N., Williams, M. D., Fikes, R., Henderson, A., and Malone, T., "Rabbit: An intelligent database assistant," pp. 314-318 in *Proceedings of the third annual national conference on artificial intelligence*, Pittsburg (1982).

RAND/N-2990-DARPA