GENERALIZED TUPLE SELECTION PREDICATES

James Clifford
Department of Information Systems
Leonard N. Stern School of Business
New York University
44 West 4th Street, Suite 9-170
New York, NY 10012-1126
(212) 998-0800

May 1987

Working Paper Series
STERN #IS-87-38

Abstract

Tuple selection predicates (such as CITY = "Brighton") provide the basis for both the Select and Join operators in the relational model (**RM**). Several proposed extensions to **RM** can be seen as relaxing the First Normal Form constraint of simple values to allow function-valued attributes. In order to take advantage of these complex values the class of allowable tuple selection predicates likewise needs to be extended. We propose a class of general tuple selection predicates based upon a general model allowing function values of any type constructed from some basic set of primitive domain types, and we investigate how various recent proposals fit into this general framework.

1. Introduction

In proposing the Relational Model of Data (**RM**) [Codd 70] as an alternative to the hierarchical and network "graph-based models," Codd cited a number of motivations. Chief among these was the provision of application independence from the details of the physical implementation. Unlike existing models, access paths were to be computed "on the fly" as it were, by means of the **contents** of the database itself, and not by means of pre-defined, physical links between records and files. The case for what has come to be known as **First Normal Form** (1NF), i.e., allowing only simple, unstructured domains for relation attributes, was made in that paper. Specifically it is argued that "[1NF] is not only an advantage for storage purposes but also for communication of bulk data between systems which use widely different representations of data." Clearly these arguments for 1NF are motivated by concerns of efficiency, not by concerns of data model expressivity.

It is interesting to note that the paper cited by Codd, [Childs 68], imposes no such restriction to simple domains on the "set-theoretic data structure" proposed. Moreover, a great deal of work on extensions to the relational model, motivated by concerns of modelling expressivity, have concluded that the 1NF restriction is too limiting. Some of this work has been very general, such as proposals to allow set-valued domains [JaeschkeSchek 82] and [Arisawa et al. 83], or to incorporate a temporal dimension into the model [CliffordCroker 87], or to include more complex semantic "objects" [HammerMcleod 78]. Other work has been motivated by particular application domains, e.g. CAD/CAM [Katz 86] and VLSI [BatoryKim 85], in some cases even looking back to the earlier "graph-based" models for some of the useful constructs there that were eliminated in **RM**. Thus it appears that the pendulum, which in the interests of data independence had swung in the direction of simplicity, is now moving back in the direction of complexity in order to meet the demands of more sophisticated applications and users.

In this paper we assume without argument the need for non-first normal form databases; the reader is referred to such works as [JaeschkeSchek 82] for some of the cogent arguments in support of relaxing the 1NF constraint. Within the framework of an extended RM allowing complex domains, then, we look at a model where domains are considered to be **functions**. We then propose an extension to the notion of tuple selection predicates as the basis for extending the relational algebra to cover these new data structures. We also show that in all cases these extended tuple selection predicates reduce to the simple predicates allowed in RM, in the sense described in [CliffordCroker86].

2. Tuple Selection Predicates

2.1. Preliminaries

Let $D = \{ D_1, D_2, ..., D_n\}$ be a set of **value domains**, where for each i, $D_i \neq \phi$. Each value domain D_i is analogous to the traditional notion of a domain in RM in that it is a set of atomic (non-decomposable) values.

Let $U = \{ A_1, A_2, ..., A_m \}$ be a (universal) set of **attributes**.

Let DOM : U --> D be a domain assignment function, which gives the domain of each attribute A in U, written DOM(A).

A **binary relation** R on sets A and B is a set of ordered pairs $R = \{ <a,b> \mid a \in A \wedge b \in B \}$. We sometimes write aRb when $<a,b> \in R$.

A **function** f from A to B, f: A --> B, is a binary relation such that whenever <a,b> and <a,c> are in f, b = c.

Given a function f, f: A --> B, we can be define certain auxiliary notions. The **domain of R**, which we will denote by $\mathcal{D}(R)$, is defined by: $\mathcal{D}(R) = \{ a \mid \exists b\ [aRb] \}$. The **range of R**, denoted $\mathcal{R}(R)$, is given by: $\mathcal{R}(R) = \{ b \mid \exists a\ [aRb] \}$. Finally, if S and A come from the same universe, the **restriction of**

f to S, denoted $f|_S$, is given by: $f|_S = \{<a,b> \mid <a,b> \in f \wedge a \in S\}$.

The essential component of a selection predicate is a comparison between two values; for this purpose we need to have comparison operators. Therefore let $\Theta_1 = \{=, \neq\}$, and $\Theta_2 = \{<, >, \leq, \geq\}$ be sets of binary relations on domains.

We assume for any domain D that all of the elements are Θ_1-comparable. On certain domains, which are said to be ordered, the comparators in Θ_2 are also defined.

2.2. The Selection Predicates of RM

For any attributes A_1 and $A_2 \in U$ that are Θ_1-comparable, any $\theta \in \Theta_1$, and any constants $a \in DOM(A)$, $B \in DOM(B)$, the following are tuple selection predicates:

- $A\ \theta\ B$
- $A\ \theta\ a$
- $b\ \theta\ B$

Moreover, if DOM(A) and DOM(B) are Θ_2-comparable, then the above predicates can also be formed with any $\theta \in \Theta_2$.

The intended semantics of tuple selection predicates is obvious. A tuple selection predicate P is simply a function from tuples to $\{0,1\}$: $P(t) = 0$ if P is false with respect to t, and $P(t) = 1$ if P is true with respect to t.

<u>**EXAMPLES:**</u> The following simple examples of Selection Predicates are familiar from RM:

1. SALARY <= 50000
2. NAME = "Jones"
3. MANAGER = CUSTOMER
4. 60 < STATUS

In the standard RM, since all values are **atomic** these are the only allowable predicates on tuples. These predicates can be looked at as "filters" on relations. If r is a relation and P a tuple selection predicate, then P(r) = {t in r | P(t) }; in other words, only the tuples which satisfy the "filter" P are in P(r).

It is important to emphasize this notion of a tuple selection predicate as a "filter;" objects either pass through the filter or they do not, but in passing through they remain **unchanged**. As we shall see certain proposed operations have hybridized a filtering operation with a restricting operation resulting in a lack of clarity.

Two basic operators in **RM** make use of tuple selection predicates, the Select (σ) operator and the Join (in several different forms).

<u>SELECT:</u> The relational algebraic Select operator with selection criteria P, denoted σ_P (r), is exactly P(r) as we have defined it.

<u>JOIN:</u> Although important at the "user" level in RM, the relational algebraic Join operator is not really "primitive." Cartesian Product is the basic operator for combining two relations, r χ s. The "Join" is simply a "filtering" (by means of a tuple selection predicate) of the results of the "complete" Cartesian Product to the subset satisfying the desired property. (With the so-called "Natural Join", redundant information in duplicate columns is discarded by means of a column projection.) Thus in all cases we have that r [P] s is just σ_P (r χ s)

Viewed in this light, then, it is obvious that the class of tuple selection predicates **P**, in effect, determines the power of both the Select and Join operations in the model. A major issue in an extended **RM** that allows complex value domains, then, is the definition of the allowable class **P** of tuple selection predicates, since these will serve as the basis for the **extended** Select and Join.

Given our assumption of function-values domains, the question in our case reduces to the following: Given two function f and g, what are the meaningful ways in which we can compare them?

3. Generalized Tuple Selection Predicates

We proceed to build a definition of the class *P* of generalized tuple selection predicates, proceeding from a definition of a hierarchy of function domain types built up recursively from the simple or primitive domains.

The set of ***Domain Types*** is the smallest set *DT* such that:

1. $D_1, D_2, \ldots D_n \in DT$; these are referred to as the **primitive domains**
2. $T = \{0,1\} \in DT$
3. If a, b $\in DT$, then $<a,b> \in DT$.

An expression of Domain Type a will denote an object of the indicated type. For example, primitive domains are analogous to the domains of simple values in *RM*. An expressions of Domain Type T will denote one of the Boolean values 0 or 1, and an expression of some constructed Domain Type $<a,b>$ will denote a **function** from something of type a to something of type b. Of particular interest (to set-valued extensions) are expressions of type $<a,T>$ which denote sets (actually, the characteristic functions of sets) of objects of type a.[1]

We assume that associated with each a $\in DT$ is a denumerable set of constants, and as usual the database schema defines a finite set of attributes $\{A_1, \ldots A_m\}$ each of which has an associated Domain of a given Domain Type DT(A).

Examples: We illustrate our definitions by examples drawn from the following simple environment. The primitive domains are the sets of values $D_1 = \{a, b, c, d, e\}$, $D_2 = \{5, 6, 7, 8\}$, and $T = \{0, 1\}$. The set of attributes is {A, B, C, D, E}, and the domains of each attribute are given by the function $DOM = \{<A,D_1>, <B,D_2>, <C,<D_1,D_2>>, <D,<D_1,T>>, <E,<D_1,<D_2,T>>>\}$. In this application, then, D_1 and D_2 are primitive domains and therefore Domain Types (by 1), T is a Domain Type (by 2), and $<D_1,D_2>$, $<D_1,T>$, $<D_2,T>$, and $<D_1,<D_2,T>>$ are Domain Types (by 3).

[1] e.g., the subset {a,c} of {a,b,c,d} can be represented by the characteristic function {<a,1>,<b,0>,<c,1>,<d,0>}.

The basic building blocks of $\boldsymbol{P}$ are the ***Terms***, each with an associated Domain Type, and are defined as follows:

1. Each constant c is a Term of Domain Type DT(c)
2. Each attribute A is a Term of Domain Type DT(A)
3. If X is a Term of Domain Type $<\alpha,\beta>$ and Y is a Term of Domain Type α, then X(Y) is a Term of Domain Type β
4. If X is a Term of Domain Type $<\alpha,\beta>$ then $\mathcal{D}$(X) is a Term of Domain Type $<\alpha,T>$
5. If X is a Term of Domain Type $<\alpha,\beta>$ then $\boldsymbol{R}$(X) is a Term of Domain Type $<\beta,T>$

The intended interpretation of each Term will not be given here in complete detail. In general, in any expression a constant denotes some individual value of the appropriate type, and an attribute denotes its current value in a given tuple. X(Y) denotes the application of the value of X (a function) to the argument Y. $\mathcal{D}$(X) and $\boldsymbol{R}$(X) denote the domain and range, respectively, of the function denoted by X.

<u>**Examples:**</u> In our example application, a and b are constant Terms of Domain Type D_1, {a,b,c} is a constant Term of Domain Type $<D_1,T>$, and {<a,5>, <b,5>, <c,5>, <d,5>, <e,5>} and {<a,0>, <b,0>, <c,1>, <d,0>, <e,1>} are constants of types $<D_1,D_2>$ and $<D_1,T>$ respectively (by 1). A is a Term of Domain Type D_1 and E a Term of Domain Type $<D_1,<D_2,T>>$ (by 2). C(A) is a Term of Domain Type D_2, D(A) and E(A)(B) Terms of Domain Type T, and E(A) a Term of Domain Type $<D_2,T>>$ (by 3). $\mathcal{D}$(C) is a Term of Domain type $<D_1,T>$ (by 4). Finally, $\boldsymbol{R}$(E) is a Term of Domain Type $<<D_2,T>,T>$ (by 5). With respect to a particular tuple t = <a, 6, {<a,5>, <b,5>, <c,5>, <d,5>, <e,5>}, {c, e}> on scheme (A B C D), the interpretation of the Term A is a and of the Term C is {<a,5>, <b,5>, <c,5>, <d,5>, <e,5>}; the Term C(A) denotes {<a,5>, <b,5>, <c,5>, <d,5>, <e,5>} (a) or 5; the term D(A) denotes {c, e} (a) or 0; and the Term $\mathcal{D}$(C) denotes {a, b, c, d, e}.

The class $\boldsymbol{P}$ of ***Tuple Selection Predicates*** can now be defined as follows:

1. If T and S are Terms of Domain Type α, for any α, then $T = S$, and $T \neq S$ are in $\boldsymbol{P}$.
2. For any primitive domain α, if α is ordered, and if T and S are Terms of Domain type α, then $T < S$, $T > S$, $T \leq S$, and $T \geq S$ are in $\boldsymbol{P}$.

3. If T and S are Terms of Domain type $<\alpha,\beta>$, then $T \subseteq S$, $T \subset S$, and $[T \cap S] \neq \emptyset$ are in ***P***.

4. If T is a Term of Domain Type α, and S is a Term of Domain Type $<\alpha,T>$, then $T \in S$ (or equivalently S(T)) is in ***P***.

<u>Examples:</u> D(A) = E(A)(B), A = a, B $\neq$ b, and D(A) are in ***P*** (by 1), A < b and C(A) > 6 are in ***P*** (by 2), D $\subset$ {a, b} and [D $\cap$ {a, b}] $\neq \emptyset$ are in ***P*** (by 3), and B $\in$ ***R***(C) is in ***P*** (by 4).

4. Example Models

In this Section we will show how the tuple selection predicates of four "relational" models fit into the general model of predicates which we have defined here. These results are summarized in Table 4-1.

	RM	*Set Values*	*Historical Model*	*Indexical Model*
Domains				
1. D_i	D_1 .. D_n	D_1 .. D_n	D_1 .. $D_{n'}$, *(values)*	D_1 .. D_n *(values)*
			TM[2] *(time)*	I_1 .. I_m *(indices)*
2. T	NO	YES *(implicit)*	*limited*	YES
3. $<a,b>$	NO	$<D_i,T>$ for each i	$<TM,D_i>$ for each i	$<a,b>$ for a an index, b any non-index
Terms				
1. c	YES	YES	YES	YES
2. A	YES	YES	YES	YES
3. A(B)	NO	A(c) *only*	A(s) *only*	YES
4. ***D***(A)	NO	NO	YES	YES
5. ***R***(A)	NO	NO	YES	YES
Predicates				
1. equality	YES	YES	YES	YES
2. ordering	YES	YES	YES	YES
3. subset	NO	YES	YES *(TM only)*	YES
4. membership	NO	YES	YES *(TM only)*	YES

Table 4-1: Comparison of Tuple Selection Predicates in Models

[2] We use TM for "Time", instead of T as in the original, in order not to confuse the reader with our T = {0,1}

4.1. Relational Model

Section 2.2 presented the definitions of the simple tuple selection predicates allowed in **RM**. As seen in Table 4-1, ***RM*** allows only simple domains, a Term is either a simple constant from some domain or the name of an attribute, and the only comparators available are those based on equality or ordering.

4.2. Set-Valued Relational Models

There have been many proposals for extending ***RM*** to allow not only simple values but also **sets** of these simple values as the domains of attributes in relations (for example, [JaeschkeSchek 82] and [Arisawa et al. 83]). While these proposals have not explicitly been cast in these terms, by considering the isomorphism between a set and its characteristic function we can view these set-valued models as allowing a specific type of **function** as an attribute domain. In particular, if D_i is some simple domain, then attributes in a set-valued RM could have either D_i or 2^{D_i} as their domain. No other domain types are allowed in these models.

If the set of primitive domains is $\{D_1, ..., D_n\}$, then the set of types is restricted to these and to those types of the form $<D_i,T>$ for each D_i. The Terms include the individual constants of type D_i (like "Noel Coward") or of type $<D_i,T>$ (i.e., sets like {"Lily Tomlin", "Noel Coward"}), the attributes, and (implicitly) Terms of function application like A(c) (generally written $c \in A$) for set membership. All of the predicates in ***P*** that can be constructed from these Terms are allowed in these models.

4.3. Historical Relational Models

While the HRDM proposed in [CliffordCroker 87] *explicitly* represents the temporal dimension added to ***RM*** as **functions** from the set of Times to a set of simple values, most of the other proposals for extending ***RM*** with a temporal dimension (for example, [CliffordWarren 83], [Snodgrass 84], [Ariav et al. 84], [Tansel 85], and [GadiaVaishnav 85]) can be mapped onto such a view and so can be said to *implicitly* makes such an extension. In these models, in addition to the basic set of value domains D_i, a distinguished domain of times, TM, is also included. Attributes may then draw their values either from some value domain D_i (in which case they are assumed to be **constant functions** over time) or from a domain TM^{D_i} (in which case their value is a function of time). (In [CliffordCroker 87] functions from the domain TM^{TM}, representing temporal-valued attributes which can vary over time, are identified as a special and interesting case that give rise to the additional special-case operators of "dynamic" Time-Slice and Join.)

If we call both the set of primitive domains $\{D_1, ..., D_n\}$ and the set of times TM *simple types*, then the set of types is restricted to these and to those types of the form $<TM,\alpha>$ for any simple type α. The Terms include the individual constants of type D_i, of type TM (like "1978"), or of type $<TM,T>$ (i.e., sets of times like {1978, 1979, 1980}); the attributes; Terms of function application like A(s) (where the argument to the function is always a Time); and Terms for the Domain of an attribute and the Range of a time-valued attribute (denoted "t.l" and "image of t(A)" respectively in [CliffordCroker 87].) All of the predicates in ***P*** that can be constructed from these Terms are allowed in these models, with the restriction that subset and set-membership predicates can only be formed with Terms of type TM and $<TM,T>$.

4.4. Indexical Relational Model

The Indexical Relational Model (***IRDM***) is an attempt to *generalize* the work on historical and set-valued extensions to ***RM*** to any number of dimensions, for the purposes of providing a common mechanism for representing multi-dimensional data within a common framework. Based on the field of indexical semantics, an indexical database views each value as a function from some set of viewpoints, or "indices" into a set of actual values. In [Clifford 87] we define a general indexical database model and show how several recent approaches to extending the relational model can be seen as special cases of our general model.

Among the kinds of information that we believe can be modelled uniformly in an indexical database model are the following, far from complete, list: opinions, expectations, judgements, personal observations, histories, predictions, expert advice, hypothesized scenarios, design versions, simulations, etc. While many, if not all, of these functions can be met by some combination of the DDL of a DBMS and a host programming language, the interactive query language mode is what has made DBMSs so attractive, by abstracting the general functions of database querying into a general purpose set of querying functions, allowing access to the contents of the database **without the need for programming**.

The inspiration for the Indexical Data Model is the field of intensional logic, an attempt to formalize the **pragmatic** component of linguistic theory. An intensional logic looks at the phenomenon of **context** as a major contributing component to defining the interpretation of a language. This notion of indexical is applied to provide a semantic theory to an extended relational model, in recognition of the need for potentially many **points of reference** in increasingly complex database applications. Moreover, following the guidelines proposed in [CliffordCroker 86], the model proposed is shown to be a consistent extension, not only of the Historical Relational Data Model (HRDM) [CliffordCroker 87], but also of the relational data model itself.

The basic idea of *IRDM* is to partition the set of primitive domains into two types, a set UD = { D_1, D_2, ..., D_n} serving as a set of **value domains**, and a set UI = { I_1, I_2, ..., I_m } as a set of **indices**. Each index represents a "contextual coordinate" [Lewis 76] which contributes to the context in which a particular fact is to be interpreted. The values of the attributes in a relation are drawn from some indexical domain which is a **function** from some number of application-dependent index domains to an underlying value domain. For example, a SALARY attribute might take values from a domain of functions from "recording time" to functions from "data time" to "salary values;" or a SUBPART attribute might have values which are functions from "date" to functions from "designer" to "part ID numbers." Operations such as Time-Slice, Temporal Join, version selection, etc., which have been proposed as special-purpose operators for particular extensions of *RM* are generalized to a set of extended relational operators on relations having indexical domains.

4.5. Hybrid Operators

A number of proposed extensions to RM, including [CliffordCroker 87] and [JaeschkeSchek 82], have exhibited Join-like operators which, like the "Natural" Join, are not equivalent to a Selection of the Cartesian Product. However, unlike the case of Natural Join, which is definable in terms of another primitive relational operator (viz. Project), both of these examples require an additional primitive operator for their definition.

In the case of [CliffordCroker 87], the operator in question is the Θ-JOIN in all of its varieties (including by extension the Natural Join). The general definition for this operation is is follows:

$$\begin{aligned} r1\ [A\ \Theta\ B]\ r2 = \{\ t \mid \exists\ t_{r1} \in r1\ \exists\ t_{r2} \in r2 \\ [t.1 = \{s \mid t_{r1}(A)(s)\ \Theta\ t_{r2}(B)(s)\ \} \wedge \\ t.v(R1 - A) = t_{r1}.v(R1 - A)|_{t.1} \wedge \\ t.v(R2 - B) = t_{r2}.v(R2 - B)|_{t.1} \wedge \\ t.v(A) = t_{r1}.v(A)|_{t.1} \wedge t.v(B) = t_{r2}.v(B)|_{t.1}\ ]\ \} \end{aligned}$$

In words what this definition says it that the result of joining r1 and r2 is a set of tuples t, where each t comes from two tuples t_{r1} and t_{r2} which stand in the relation Θ to one another, and is constructed from them by **restricting** both of them (as functions) to just those **shared points in time** when they are so related.

In [JaeschkeSchek 82] the operator in question is called the "Intersection Join" (*IJ*) and is defined as:

$$\begin{aligned} r1\ [R1.A\ \mathit{IJ}\ R2.B]\ r2 = \{\ t \mid \exists\ t_{r1} \in r1\ \exists\ t_{r2} \in r2 \\ [t(R1 - A) = t_{r1} \wedge t(R2 - B) = t_{r2} \wedge \\ t(A) = t_{r1}(A) \cap t_{r2}(B) \wedge t(A) \neq \emptyset\ ]\ \} \end{aligned}$$

In other words, the Intersection Join is a kind of Natural Join wherein each t in the result comes from two tuples t_{r1} and t_{r2} which share a common (non-empty) subset of values for their shared attribute(s); again the resulting tuple is constructed from them by only including these shared values (i.e. by **restriction**.)

We have already exploited the isomorphism between sets and their characteristic functions. In the same vein we can look at both of these independently defined operators as relying upon the same underlying "primitive" operation of function restriction. If we denote this operation by Υ, then provided A is a function-valued attribute and DT(S) = DT($\mathcal{D}$(A)) in the scheme of r, we can define the **restriction of r on attribute A to S** (or by extension, on a set of attributes X) as:

$$\Upsilon_{A,S}(r) = \{t \mid \exists\ t' \in r \wedge t(R\text{-}A) = t'(R\text{-}A) \wedge t(A) = t'(A)|_S\ \}.$$

In the historical case, then, the Θ-JOIN can be schematically re-defined as:

$$r1\ [A\ \Theta\ B]\ r2 = \Upsilon_{R1 \cup R2, DOM(A) \cap DOM(B)}\ \sigma_{|DOM(A) \cap DOM(B)| \neq \emptyset}\ [r1 \times r2]$$

Likewise the Intersection Join is given by:

$$r1\ [R1.A\ \mathit{IJ}\ R2.B]\ r2 = \Upsilon_{A,\ DOM(A) \cap DOM(B)}\ \sigma_{|DOM(A) \cap DOM(B)| \neq \emptyset}\ [r1 \times r2]$$

The similarity between these two operations, when decomposed into these primitive components, is obvious, and is compelling evidence for the need for the restriction operator Υ in such functionally extended models. (The Time-Slice operator in almost all of the historical relational models is just such an operator.)

5. Summary

Since its appearance in 1970, ***RM*** has served as the basis not only for a tremendous amount of research in the database community, but also for a growing number of commercially available DBMSs. In spite, or perhaps because, of this popularity, ***RM*** has also served as the springboard for numerous proposals to extend it in one direction or another. In this paper we have looked at a variety of different proposal for such extensions, all of which share (or can be viewed as sharing) the common thread of relaxing the restriction to simple values which has come to be known as First Normal Form. Moreover, we have shown how all such extensions can likewise be viewed as falling into a general class of extensions which allow the values of relational attributes to be **functions** in some function space.

We have focussed on the issue of the expressiveness of the tuple selection predicates, since the power of these filters provides the basis for the "extraction" operators Select and Join. We have defined a class of tuple selection predicates ***P*** which are an extension of the simple predicates allowed in ***RM***, take advantage of the complex values allowed in these extended models to provide for data extraction based on specified components of these complex values, and are seem to generalize particular operators defined in several different relational model extensions.

[HammerMcleod 78]
Hammer, M., and Mcleod, D.
The Semantic Data Model: A Modelling Mechanism for Data Base Applications.
In *ACM-SIGMOD International Conference on Management of Data*. Austin, 1978.

[JaeschkeSchek 82]
Jaeschke, G. and Schek, H.J.
Remarks on the Algebra of Non-First Normal Form Relations.
In *Proceedings of the 1st ACM SIGACT-SIGMOD Symp. on Principles of Database Systems*, pages 124-138. 1982.

[Katz 86] Katz, R. H.
Computer-Aided Design Databases.
In Ariav, G. and Clifford, J. (editors), *New Directions for Database Systems*, pages 110--123. Ablex Publishing Co., Norrwood, New Jersey, 1986.

[Lewis 76] Lewis, D.
General Semantics.
In Barbara H. Partee (editor), *Montague Grammar*, pages 1-50. Academic Press, Inc., New York, 1976.

[Snodgrass 84] Snodgrass, R.
The Temporal Query Language TQuel.
In *Proceedings of the 3rd ACM SIGACT-SIGMOD Symp. on Principles of Database Systems*, pages 204-212. Waterloo, Ontario, Canada, April, 1984.

[Tansel 85] Tansel, A. U.
An Extension of Relational Algebra to Handle Time in Relational Databases.
In *ACM-SIGMOD International Conference on Management of Data*. Austin, May, 1985.