WINDOWS ON THE WORLD

David Maier†
Oregon Graduate Center

David Rozenshtein†
David S. Warren†
State University of New York at Stony Brook

Summary

We discuss the philosophy, history and theory of window functions. Window functions (sometimes called *connections*) are a means to treat a relational database as a semantic whole, rather than as an arbitrary collection of relations. Simply stated, a window function maps a database state and a relation scheme to a relation over the scheme. Window functions are the basis for all existing universal scheme interfaces. We present an assumption inherent in universal scheme interfaces, the *unique role assumption*.

Window functions have evolved along two paths, giving rise to computational definitions and weak instance definitions. We examine several examples of each type of window function, with special attention to the association-object window function of PIQUE. We then look at properties we feel a reasonable window function should satisfy, notably the *containment condition* and *faithfulness*. We also define *implicit objects*, which are relation schemes that a window function treats in a special manner, and which are useful for describing the behavior of window functions.

1. The Why and What of Window Functions

As Maier, Vardi and Ullman <MUV1> note, the relational data model has gone far towards *physical* data independence, but has not achieved the goal of *logical* data independence. That is, users of relational systems are relieved of specifying access paths within the structure of a single relation, but they still must navigate between relations. *Universal scheme interfaces* are an attempt at logical data independence. In a universal scheme interface, all the semantics of the database is loaded onto the attributes. Queries are phrased in terms of attributes; the user need not know which attributes are in which relations. In a universal scheme interface, a database is presented as a semantic whole, accessible through its attributes alone. In the sequel, U will denote the universe of attributes in a database: the *universal scheme*.

There are several universal scheme systems extant and under development. The first were APPLE <CK> and that of Shenk and Pinkert <SP>. More recent systems are q <AK>, System/U <K, KU, U>, PIQUE <MRSSW, Ro1, St> and Parafrase <KMRS, KS>.

Query processing in a universal scheme system can be viewed as a two-stage procedure:

1. The set of attributes, call it X, appearing in the query is determined. Then, on the basis of the state of the database, a relation r over scheme X is generated. (If the query contains several variables, the attributes associated with each variable are used to compute separate relations.)

2. Further operations specified by the query are applied to $r(X)$ to generate the answer.

These stages are called *binding* and *evaluation*.

Example

Consider a simple database courses with the relations *taking*(*STUDENT COURSE*) and

†Work supported by NSF grant IST 81-04834.

teaching (*FACULTY COURSE*). In response to the query

retrieve *FACULTY* **where** *STUDENT* = Andrews,

the PIQUE system will construct a relation r on *FACULTY STUDENT*. Presumably, r will be

$$\pi_{FACULTY\ STUDENT}(taking \bowtie teaching).$$

In the evaluation stage of processing the query, PIQUE applies the selection $\sigma_{STUDENT = Andrews}$ to r to get the final answer to the query.

[]

The generation of a relation r on scheme X in the binding step is done, implicitly or explicitly, through the application of a ***window function***. For each subset $X \subseteq U$, there is a function $[X]$, called the ***window function, or simply window, on X, that maps database*** states to relations on X. While, properly, we should denote the value of $[X]$ on database state d as $[X](d)$, in most cases d will be understood, and we shall write simply $[X]$. The brackets themselves, $[\cdot]$, can be viewed as a functional that maps a subset of U to a function from database states to relations over that subset.

While the two steps do interact, they are loosely-coupled. Changes to the window functions can be made without changing the procedures in the evaluation step, although such changes will mean different answers for queries. A logical question at this point is why the same effect as window functions cannot be captured with virtual relation or view mechanisms, as in conventional relational systems? Unlike an arbitrary set of virtual relations, the window functions in a universal scheme system are meant to display some manner of semantic consistency. The choice of the term "window" is intended to convey the image of a consonant set of views into a single database world.

For a universal scheme interface to a database to be practicable, the database must at least satisfy the ***universal relation scheme assumption*** (URSA). URSA states that any attribute in U corresponds to the same class of entities wherever it appears. For example, *NUMBER* cannot refer to serial numbers of equipment in one place and social security numbers in another. Positing the existence of window functions makes an assumption stronger than URSA, which we shall call the ***unique role assumption*** (URA). URA requires that an attribute not only always represent the same class of entities, but also always represent the same role for that class. For example, *DATE* may not represent dates both in the role of birthdate and the role of appointment date. Put another way, URA means "the scheme determines the connection." For any set of attributes, there is at most one connection among them. In particular, no two database relations have the same scheme. If *DATE* played two roles, as above, there is no way to tell if [*EMPLOYEE DATE*] is asking for the connection between employees and birthdates or between employees and appointment dates.

It is unlikely that a relational database designed without URA in mind would satisfy that assumption. To get satisfaction, some attributes will probably have to be renamed, in order to distinguish the roles portrayed. It may not be apparent, after renaming, that different attributes represent the same class of entities, and the proliferation of attribute names can become unwieldy. We are currently considering ways to handle such problems by explicitly incorporating a generalization hierarchy as part of the database description <BK, Sc2, SS>.

Not all of the universal scheme systems mentioned above strictly enforce URA. Carlson and Kaplan, in their APPLE system <CK>, are trying to define window functions upon databases that do not necessarily satisfy URA, or even URSA. Trying to impose a universal scheme view after the fact upon a database that does not satisfy URA causes several complications. They must maintain explicit information on comparability of attributes. Their method for computing expressions for window functions can generate multiple connections between a pair of attributes, actually giving several window functions for the same scheme. Ultimately, the user must select among connections when several exist.

The query system q <AK> does not make any assumptions about the database. There is a "relfile" containing a list of schemes for stored and virtual relations, and procedures for computing virtual relations. To generate $[X]$, q sequentially scans the relfile for the first scheme of a stored or virtual relation containing X. While q's mechanism produces a single window function for any scheme, the views these windows present are not necessarily consistent with each other or with the database. In practice, the computations used to derive virtual relations generally are all joins, and the database does satisfy URA. One other problem with q is that the expressions for virtual relations must be given explicitly. Work is currently underway on methods to generate those expressions from dependency information about the database <Ho3>.

URA doesn't really say that there cannot be multiple connections among a set of attributes,

only that the system takes one of those connections as the most natural, and will make that connection automatically. Other connections must be made explicitly by the user.

2. Types of Window Functions

The window functions used in the universal scheme systems mentioned, and in various theoretical studies to be discussed, are not always expressed in the form given here. Quite often the definition of a window function is implicit within some computational method. The bracket notation used here follows the "output functions" of Maier <Ma>. In subsequent sections we shall be interested in properties of sets of window functions, one function for each subset of U. We call such a collection a *wall* of window functions, or simply a wall of windows. We sometimes let the brackets by themselves, $[\cdot]_s$, denote the wall of windows of type s: $\{[X]_s \mid X \subseteq U\}$.

Most window functions can be placed along one of two lines of development. There are window functions based on straight computational definitions, and there are those based on weak instances.

In the rest of this paper, we shall let R be the database scheme $\{R_1, R_2, \ldots, R_p\}$, and let $d = \{r_1(R_1), r_2(R_2), \ldots, r_p(R_p)\}$ be a database over R.

2.1. Computational Window Function Definitions

Most initial work on computational definitions for window functions made an assumption stronger than URA. That assumption is the *universal instance assumption* (UIA), which states that the database relations are all projections of a single relation I over U (I is a *universal instance*). That is, $r_i = \pi_{R_i}(I)$ for $1 \le i \le p$.

Under UIA, the window function $[X]$ is defined as $\pi_X(I)$. We shall distinguish this class of window functions as $[\cdot]_I$. The presumption is that I can be recovered from d as $r_1 \bowtie r_2 \bowtie \cdots \bowtie r_p$, or at least that $\pi_X(I)$ can be recovered, for certain X. A number of groups have studied the question of when all or part of I can be recovered from its projections <AC, BMSU, MMSU, Ri>. In short, the question they address is whether the dependencies that I must satisfy imply that $r_1, r_2, \ldots, r_p$ have a lossless join.

The work on UIA-based window functions concentrates on finding, for a given X, an alternative expression E such that $\pi_X(E) = \pi_X(I)$. Shenk and Pinkert <SP> were among the first to look at this question. They concentrated on *lossless subjoins*: a subset $\{s_1, s_2, \ldots, s_m\}$ of d such that $\pi_X(s_1 \bowtie s_2 \bowtie \cdots \bowtie s_m) = \pi_X(I)$. In their study, the only dependencies considered are FDs arising from keys, and the only joins are lossless joins of pairs of relations. The join of $r(R)$ and $s(S)$ may only be taken when $R \cap S \rightarrow R$ or $R \cap S \rightarrow S$. They attempt to find a lossless subjoin with the fewest relations by a dynamic programming method.

Lozinskii <Lo> also uses only key FDs and pairwise lossless joins. However, he is looking for lossless joins with a single *source* relation. The join expression starts off with the source relation, and all subsequent relations are joined with it, so there is only a single intermediate result. For a given X, he is interested in finding all the lossless subjoins for computing $[X]$, so that he may pick the best according to a given cost function. Honeyman <Ho1> is also looking for a sequence of joins emanating from a single source, but he uses an arbitrary set of FDs (as long as they are embedded in the database scheme), and allows projections of relations into the join expression. (He terms such joins *extension joins*.)

UIA is well known to have its shortcomings. It is a hard condition to test in general, and it is not realistic in many applications. Several researchers have defined window functions where UIA is used to determine the lossless subjoins, but it is not expected to actually hold for an arbitrary database state.

Osborn <Os> defined $[X]$ as the union of X-projections of all lossless subjoins covering X. Her join expressions also start with a source relation that is augmented through pairwise lossless joins. Her dependency information is solely key FDs. We denote her wall of window functions by $[\cdot]_{LJ}$.

The designers of System/U <KU, U>, and Fagin, Mendelzon and Ullman <FMU> advocate a UIA approach, with $[X]$ being $\pi_X(r_1 \bowtie r_2 \bowtie \cdots \bowtie r_p)$. In System/U, to avoid problems where relations do not join completely, the join is minimized under weak equivalence. That is, the expression above is pruned under the assumption that the relations do join completely. The technique used is tableau minimization <ASU1, ASU2>, which has the advantage that dependency information can be brought in to help reduce the number of joins. We shall let $TM(E)$ denote the tableau minimization of an expression E, and let $[\cdot]_{SU}$ represent the System/U window functions; so

$$[X]_{SU} = TM(\pi_X(r_1 \bowtie r_2 \bowtie \cdots \bowtie r_p)).$$

One problem with this approach is that there can be several alternatives for $TM(E)$. Those alternatives are equivalent if UIA holds, but not for arbitrary states of the database. In cases where *TM* yields multiple expressions equivalent to the original, the union of these expressions is used.

Example

Consider a database with relations $r(ACD)$, $r(BCD)$, and $r(DE)$. (Since schemes are unique in a URA database, we may call all relations r.) For the window $[CE]_{SU}$, the expression can be minimized to $\pi_{CE}(r(ACD) \bowtie r(DE))$ or to $\pi_{CE}(r(BCD) \bowtie r(DE))$. The union of these two expressions is used to compute $[CE]_{SU}$.

[]

The System/U approach gives reasonable window functions when the database scheme **R** is *acyclic*. (See Beeri et al. <BFMMUY, BFMY> for material on acyclic database schemes.) Maier and Ullman <MU> note that in the presence of cycles, the definition above for $[\cdot]_{SU}$ may not represent a natural connection, or is likely not to be the particular connection a user had in mind.

Example

Consider a database on attributes B (bank), L (loan), A (account), and C (customer), with relations $r(BL)$, $r(LC)$, $r(BA)$, and $r(AC)$. The database contains information about loans and accounts at banks, and about which customers took out the loans and own the accounts. In System/U, $[BC]_{SU}$ will be computed as

$$\pi_{BC}(r(BL) \bowtie r(LC) \bowtie r(BA) \bowtie r(AC)).$$

This expression gives all customers who have both a loan and an account at a bank, which is not a very natural meaning for the bank-customer connection.

[]

Using the ideas of Sciore <Sc>, Maier and Ullman define *maximal objects* to break up cycles in the database scheme. A maximal object is a subset of U. Let **M** be a set of maximal objects. To compute $[X]$, for each $W \in \mathbf{M}$ such that $X \subseteq W$, they form the join of all relations whose schemes are contained in W. The join for each applicable maximal object is then pruned by tableau minimization, and projected onto X. The window $[X]$ is finally obtained as the union of all these expressions. We denote this wall of window functions by $[\cdot]_{\mathbf{M}}$, for a set **M** of maximal objects.

Example

Returning to the last example, let $\mathbf{M} = \{BLC, BAC\}$. This set of maximal objects says that there are two ways to connect banks and customers, through loans and through accounts. Without maximal objects, however, there is only a single connection, through loans and accounts simultaneously. Here, $[BC]_{\mathbf{M}} =$

$$\pi_{BC}(r(BL) \bowtie r(LC)) \cup \pi_{BC}(r(BA) \bowtie r(AC)).$$

[]

Our view in developing window functions for the PIQUE query language is that data dependencies by themselves are not sufficient for inferring the desired connections among attributes. We have defined window functions based on a set **A** of *associations* and a set **O** of *objects* <MW>.

Associations are sets of attributes that represent permissible units of update. An association is a possible scheme for a tuple entered into the database. We let $r(R)$ denote all the tuples in the database whose scheme is the association R. The reader will not be far from right to assume that the set **A** of associations is the database scheme, although, in practice, heterogeneous tuples may be stored in a single relation through the use of placeholders. We depart from usual practice in that we allow subassociations of associations. The term association is used instead of scheme to emphasize this departure. We may have tuples over both associations R and S in the database, where R is a proper subset of S. As we have noted elsewhere <MR>, under UIA, there is not much sense in having relations over both R and S, as $r(R)$ will be equal to $\pi_R(r(S))$. Without UIA, relations over both schemes do make sense, although URA does dictate certain restrictions, as we shall see in Section 3.

Objects are also sets of attributes, and represent units of retrieval. Objects dictate which joins will be used to construct window functions. For each object $W \in \mathbf{O}$, we assume that W is the union of some associations in **A**, and define a relation on W, denoted $r'(W)$, by joining on all associations contained in W:

$$r'(W) = \bowtie_{R \in \mathbf{A},\, R \subseteq W} r(R).$$

We then define window functions from these object relations by projecting the appropriate object relations.

$$[X]_{\mathbf{A},\mathbf{O}} = \bigcup_{W \in \mathbf{O},\, W \supseteq X} \pi_X(r'(W)).$$

Example

Consider a database where $U = \{S$ (student), C (course), F (faculty)$\}$, $\mathbf{A} = \{SC, CF\}$, meaning a student takes a course and a faculty member teaches a course. Let $\mathbf{O} = \{SCF\}$. For this choice of **A** and **O**, the connection between student and faculty is computed as $\pi_{SF}(r(SC) \bowtie r(CF))$ If we add SCF as an association, then we explicitly store from which faculty member a student is taking a course. Now, the connection between student and faculty will be $\pi_{SF}r(SCF)$.

[]

We shall argue in Section 3 for the desirability of the set **O** being closed under nonempty intersection.

2.2. Weak Instance Window Functions

The second main path along which window functions have evolved is weak instance definitions. Weak instances, and their cousins, representative instances, were first introduced as a means for discussing global satisfaction of a set of dependencies by a database <Ho, Gr, Va>, and for inferring missing information in a database state <Ma, Wa>. They have also been used recently to study the equivalence of database schemes <Me>. We show now how weak instances are used to define window functions.

A relation $I(U)$ is a ***containing instance*** for database $\boldsymbol{d}$ if

$$\pi_{R_i}(I) \supseteq r_i \text{ for } 1 \le i \le p.$$

For a set of dependencies **C**, $I(U)$ is a ***weak instance under*** **C** for $\boldsymbol{d}$ if $I \in SAT(\mathbf{C})$ and I is a containing instance of $\boldsymbol{d}$. We abbreviate "weak instance under **C**" to **C**-WI. A database state need not always have a weak instance.

A window function on X can be defined as all those X-components of tuples that appear in every weak instance of $\boldsymbol{d}$:

$$[X](\boldsymbol{d}) = \bigcap_{I \text{ a } \mathbf{C}\text{-WI}} \pi_X(I)$$

Since different choices for **C** give different window functions, we distinguish the wall of window functions for **C** as $[\cdot]_{\mathbf{C}}$.

Of course, this definition for $[X]_{\mathbf{C}}$ does not give an effective method of computation. If the dependencies in **C** can be used in a chase computation <ABU, MMS>, then ***representative instances*** can be used to compute $[X]_{\mathbf{C}}$. A representative instance for a database $\boldsymbol{d}$ is formed in two stages. First, all of the relations in $\boldsymbol{d}$ are padded out to have scheme U, using distinct null values. Let $PAD(\boldsymbol{d})$ denote the union of all these padded relations. Second, the chase procedure for dependencies in **C** is applied to $PAD(\boldsymbol{d})$ to equate nulls and generate new tuples. The result of the second stage is the representative instance for $\boldsymbol{d}$ under **C**, which we denote $RI_{\mathbf{C}}(\boldsymbol{d})$. To summarize:

$$RI_{\mathbf{C}}(\boldsymbol{d}) = chase_{\mathbf{C}}(PAD(\boldsymbol{d})).$$

It is possible that a contradiction to an equality-generating dependency is encountered during the chase, in which case we define the representative instance to be the empty relation on U.

Maier, Ullman and Vardi <MUV1> show that for a large class of dependencies

$$[X]_{\mathbf{C}} = \pi{\downarrow}_X(RI_{\mathbf{C}}(\boldsymbol{d})),$$

where $\pi{\downarrow}_X$ is ***X-total projection***: the X-component of all tuples that have no nulls in the X-columns.

Sagiv <Sa1> considered $[X]_F$, where F is a set of FDs expressed by keys. He gave a condition on database states, the ***modified foreign key constraint***, that, together with local satisfaction of F, guarantees that a database state will have at least one F-WI. Sagiv later defined the ***uniqueness condition*** on FDs and database schemes, which ensures that any locally satisfying database state has a weak instance <Sa2>. His uniqueness condition is a characterization of independence for database schemes under key FDs. (A database scheme **R** is ***independent*** relative to a set **C** of dependencies if any database $\boldsymbol{d}$ on **R** that locally satisfies **C** also globally satisfies **C**, i.e., has a **C**-WI.) We shall denote the wall of windows based on representative instances under FDs that satisfy Sagiv's uniqueness condition by $[\cdot]_{\mathbf{K}}$, where **K** is the set of key FDs.

Yannakakis <Ya> looked at $[X]_{\mathbf{C}}$, where **C** is a single join dependency (JD) corresponding to the database scheme **R**. In other words, $\mathbf{C} = \{*[R_1, R_2, \ldots, R_p]\}$. We denote this class of window functions by $[\cdot]_{*\mathbf{R}}$.

Although representative instances give a means to compute $[X]_{\mathbf{C}}$, the method is not very manageable, especially when the database is large and **C** contains tuple-generating dependencies. Sagiv <Sa2> showed that $[X]_{\mathbf{K}}$ can be computed as the union of projections of ***extension*** joins, a particularly efficient type of join <Ho>. Yannakakis <Ya> showed that $[X]_{*\mathbf{R}}$ can be computed efficiently when **R** is acyclic. Maier, Ullman and Vardi <MUV1> give conditions for $[X]_{\mathbf{C}}$ to be first-order (computable with algebraic operations), although they are not particulary concerned with the efficiency of computation. We give here a generalization of Sagiv's result, where the FDs need

not be keys, but can be any independent set of FDs embedded in **R**. An FD-join is similar to an extension join. Given a set F of FDs, and relations $r(R)$ and $s(S)$, $r \bowtie \pi_{XY}(s)$ is an *FD-join* under F if $X \rightarrow Y$ is implied by F and $X \subseteq R$.

Theorem 1

Let F be an independent set of FDs that can be embedded in **R**. For any $X \subseteq U$, $[X]_F$ can be computed as the union of projections of FD-joins.

Sketch of Proof (With thanks to Jeff Ullman.)

If, when doing the chase to compute a representative instance, nulls are never equated, then the non-null portion of any row in the representative instance shows up in some FD-join.

Call **R** "bad" if there is a set X, such that $[\cdot]_F$ cannot be computed from FD-joins. We want to show that if **R** is "bad," then **R** is not independent relative to F. Let d be a database state that shows **R** is "bad" for X. We generate a database state d' from d that also shows that **R** is "bad," but where we need not equate nulls to compute the representative instance. In computing $RI_F(d)$, every time we use $X \rightarrow Y$ to equate nulls, we instead try to equate both nulls to a value. We look for a tuple t in a relation $r(R)$, where $R \supseteq XY$, to supply a value. If no appropriate tuple is present, the independence of R allows us to add such a tuple, and still have a satisfying state. However, we must be able to compute $\pi_X(RI_F(d'))$ with FD-joins, meaning **R** must really have been "good."

[]

Maier, Ullman and Vardi <MUV2> suggest a departure from the two-step paradigm for universal scheme query processing that can be used with representative instances. Rather than apply a query to the intersection of projections of weak instances, apply it to the projections individually, and then intersect the results. For the alternative model to be attractive, there must be an effective method for evaluating queries, of course.

3. Properties and Theory of Window Functions

Here we look at several properties of window functions, and see which window functions defined so far have those properties. We feel that the first two properties given, the containment condition and faithfulness, are minimum conditions for a reasonable window function.

3.1. The Containment Condition

By URA, a set of attributes uniquely determines a connection among the attributes themselves, and it is this connection that a window function is supposed to transmit. If a set of attributes X is a subset of Y, then whatever the connection among the attributes of X is, it must be an aspect of the connection among the attributes of Y.

Example

Under URA, it is permissible for [*STUDENT COURSE FACULTY*] to mean that a student takes a course from a faculty member and for [*STUDENT COURSE*] to mean that a student takes a course. URA would not be satisfied, however, if the meaning of [*STUDENT COURSE FACULTY*] were changed to a student is a TA for a course under a faculty member.

[]

For a wall of window functions to be consistent, whenever $X \subseteq Y$ and t is a tuple in $[Y]$, $t(X)$ should be in $[X]$. Stated another way, $[X] \supseteq \pi_X([Y])$. This inequality is the ***containment condition***. It is similar to Sciore's notion of ***downward closure*** <Sc1>.

Lemma 1

$[\cdot]_I$ satisfies the containment condition.

Proof

For the universal instance window, we have a stronger condition, namely $[X]_I = \pi_X([Y]_I)$ for $X \subseteq Y$.

[]

Lemma 2

Let $\{s_1(S_1), s_2(S_2), \ldots, s_m(S_m)\}$ be a set of relations where $S_1 S_2 \cdots S_m \supseteq X$. Let $\{q_1, q_2, \ldots, q_n\}$ be a set of relations that contains $s_1, s_2, \ldots, s_m$. Then

$$\pi_X(s_1 \bowtie s_2 \bowtie \cdots \bowtie s_m) \supseteq \pi_X(q_1 \bowtie q_2 \bowtie \cdots \bowtie q_n).$$

[]

Corollary

$[\cdot]_{LJ}$, $[\cdot]_{SU}$, $[\cdot]_{\mathbf{M}}$, and $[\cdot]_{\mathbf{A}\,\mathbf{O}}$ satisfy the containment condition.

[]

Corollary

$[W]_{\mathbf{A}\,\mathbf{O}} = r'(W)$ for $W \in \mathbf{O}$.

[]

Lemma 3

Any weak instance wall of window functions satisfies the containment condition.

Proof

Let $X \subseteq Y$ and let **C** be the set of dependencies for the wall of windows.

$$[X]_{\mathbf{C}} = \bigcap_{I \text{ a C-WI}} \pi_X(I) = \bigcap_{I \text{ a C-WI}} \pi_X(\pi_Y(I)) \supseteq$$

$$\pi_X(\bigcap_{I \text{ a C-WI}} \pi_Y(I)) = \pi_X([Y]_{\mathbf{C}}).$$

[]

3.2. Faithfulness

The principle for the next condition is "What you see is what you've got." The containment condition requires that the set of views given by the windows in a wall be consistent with each other. The views given by a wall should also be consistent with the contents of the database. A wall of window functions is ***faithful*** if for any relation scheme $R \in \mathbf{R}$, for all states of the database, the relation on R in the database agrees with the window on R, or, in other words, $r(R) = [R]$.

The wall of windows defined by $[\cdot]_{LJ}$ is not necessarily faithful. Consider the database scheme $\mathbf{R} = \{AB, BC, AC\}$, with B as a key of BC. The window $[AC]$ will not necessarily agree with $r(AC)$, since $[AC]$ will contain tuples from $\pi_{AC}(AB \bowtie BC)$.

Lemma 4

$[\cdot]_{SU}$ and $[\cdot]_{\mathbf{R}}$ are faithful.

Proof

Under tableau minimization, $\pi_R(r_1 \bowtie r_2 \bowtie \cdots \bowtie r_p)$ will always be reduced to r_i, where r_i is the relation with scheme R.

[]

Theorem 2

$[\cdot]_{\mathbf{A},\mathbf{O}}$ is faithful if and only if $\mathbf{A} \subseteq \mathbf{O}$ and the relations on associations in **A** satisfy the containment condition.

Proof

(In the proof, $[\cdot]$ will mean $[\cdot]_{\mathbf{A},\mathbf{O}}$)

(only if) We show the contrapositive. Let R be an association in **A** that is not in **O**. Consider a state of the database where $r(R) \neq \phi$, $r(S) = \phi$, and $S \neq R$. If **O** has no object containing R, then $[R] = \phi$, and the window it not faithful to $r(R)$. If $W \in \mathbf{O}$ and $W \supseteq R$, the join used to form $r'(W)$ must include at least one relation apart from $r(R)$. Hence $r'(W) = \phi$. This equality holds for any object containing R, so $[R] = \phi \neq r(R)$.

Now suppose $\mathbf{A} \subseteq \mathbf{O}$, but the database relations do not satisfy the containment condition. Let R and S be associations in **A** such that $R \subseteq S$, but $r(R) \not\supseteq \pi_R(r(S))$. It follows that $r(R) \bowtie r(S) \neq r(S)$ (thus, the right side properly contains the left). For any object W, $W \supseteq S$, relations $r(R)$ and $r(S)$ will enter the join for $r'(W)$, so $\pi_S(r'(W)) \subseteq r(R) \bowtie r(S)$. Hence, $[S]$ is contained in $r(R) \bowtie r(S)$, and is not faithful to $r(S)$.

(if) If $\mathbf{A} \subseteq \mathbf{O}$, and the containment condition holds for the database relations, it is not hard to show that $r(R) = r'(R)$ for any $R \in \mathbf{A}$. It follows that $[R] \supseteq r(R)$. By Lemma 2, for any object W containing R, $\pi_R(r'(W)) \subseteq r'(R)$, so $[R] \subseteq r(R)$, and we have the desired equality.

[]

Henceforth, when dealing with association-object window functions, we shall assume that $\mathbf{A} \subseteq \mathbf{O}$ and that the relations on associations in **A** satisfy the containment condition.

Weak instance window functions are not necessarily faithful. For FDs, Mendelzon <Me> shows that for a database state d, there is a complete state d' with the same set of weak instances. A ***complete*** state essentially is one that is the projection of its representative instance. Weak instance window functions are faithful on complete database states. We also have the following two theorems about particular weak instance window functions.

Theorem 3

$[\cdot]_{\mathbf{K}}$ is faithful if every relation scheme in **R** has a nontrivial key.

Sketch of Proof

Consider a locally satisfying (hence globally satisfying) database state d. Let $R \in \mathbf{R}$ be a relation scheme, such that $\pi\downarrow_X(RI_{\mathbf{K}}(d))$ contains a tuple t not in $r(R)$. Let K be a nontrivial key for R. Modify d to d' by adding t' to $r(R)$, where $t'(K) = t(K)$, but $t'(R - K) \neq t(R - K)$. Since r(R) does not already have a tuple that agrees with t on K, d' is locally satisfying. However, d' is not globally satisfying, since t will still show up as part of a row of $RI_{\mathbf{K}}(d')$ and contradict t'.

[]

The windows in $[\cdot]_{\mathbf{K}}$ can be unfaithful if **R** has relations with only trivial keys. Let $\mathbf{R} = \{AB, AC, BD, CD\}$, with key FDs $A \rightarrow C$ and $B \rightarrow D$. Then $\pi_{CD}(r(AB) \bowtie r(AC) \bowtie r(BD))$ can add

tuples to $[CD]_{\mathbf{K}}$ that are not in $r(CD)$. Note that a relation scheme formed by synthesis <Be> will have no trivial keys. Theorem 3 also holds if the only scheme with a trivial key is a universal key <BDB>.

Theorem 4

$[\cdot]_{*\mathbf{R}}$ is faithful.

Proof

Look at $R \in \mathbf{R}$ and consider the computation of $RI_{*\mathbf{R}}(d)$. The JD $*\mathbf{R}$ is the only dependency used in chasing $PAD(d)$ when forming the representative instance for d. We can show by induction that at each stage of the computation of the representative instance, if any row w is non-null on R, then $w(R) \in r(R)$. Also, for any tuple $t \in r(R)$, there will always be a row w with $w(R) = t$. We conclude that $[R]_{*\mathbf{R}} = r(R)$.

[]

3.3. Integrity of Objects

The purpose of the next condition is to prevent a little knowledge from being a dangerous thing. The condition is stated in terms of objects, so it applies only to association-object window functions. In this subsection, $[\cdot]$ will mean $[\cdot]_{\mathbf{A},\mathbf{O}}$. In Section 3.4, we show how objects can be defined on any wall of window functions, so we shall be able to apply the condition to any window function.

The idea behind integrity of objects is that if someone knows the semantics of all the associations within an object W, then he should be able to deduce the meaning of the connection on any subset of W. Formally, for $W \in \mathbf{O}$, let

$$a(W) = \{R \mid R \in \mathbf{A}, R \subseteq W\}.$$

Object W is *integral* relative to $[\cdot]$ if, for any subset X of W, $[X]$ can be computed from $\{r(R) \mid R \in a(W)\}$.

Example

As an example of where integrity of objects fails, consider $U = \{P$ (painting), O (owner), A (artist), T (telephone)$\}$, $\mathbf{A} = \{PO, PA, OT, AT\}$, and $\mathbf{O} = \mathbf{A} \cup \{POT, PAT\}$. We are storing information on owners and artists of paintings, and telephones of owners and artists, and making connections on owners and artists. In the object POT, the connection from painting to telephone is via owner. However, the object PAT can also add tuples to $[PT]$, so $[PT]$ cannot be computed from relations in $a(POT)$ alone. The danger here is that if a user knows that the database has information about paintings, owners and telephones, but does not know about artists, his assumption as to the meaning of $[PT]$ will be incorrect. The window $[PT]$ is really the combination of two different connections.

[]

The next theorem shows that integrity of objects is equivalent to the objects being closed under nonempty intersection. This closure property has a computational advantage. It implies that for any X, there is a unique minimal object W containing X. That is, for any other object V that contains X, $V \supseteq W$. Thus, $[X]$ can be computed as $\pi_X(r'(W))$, since for any object $V \supseteq W$,

$$\pi_X(r'(V)) \subseteq \pi_X(r'(W)).$$

No unions need be taken to compute $[X]$.

Theorem 5

All objects in $\mathbf{O}$ are integral if and only if $\mathbf{O}$ is closed under nonempty intersection.

Proof

(if) By the remarks above, if W is an object and $W \supseteq X$, there is a minimal object W', $W \supseteq W' \supseteq X$, such that $[X] = \pi_X(r'(W'))$. The object relation $r'(W')$ depends only on relations for $a(W')$, which is a subset of $a(W)$. Hence, W is integral.

(only if) Let X be the intersection of objects V and W, where X is not itself an object. Assume that no objects smaller than V and W have intersection X. There must be some association R in $a(W)$, such that R is not a subset of V, so R is not in $a(V)$. By considering states of the database that differ by $r(R)$ being empty or nonempty, it is possible to induce changes in $[X]$ that do not depend on relations for $a(V)$. Therefore, V is not integral.

[]

3.4. Implicit Objects

While objects were used in the definition of only one of the window functions, we can pick out sets of attributes that behave as objects relative to other window functions. V is an *implicit object* for a wall of windows $[\cdot]$ if there is some state of the database where the inclusion

$$[V] \supseteq \bigcup_{W \supset V} (\pi_V[W])$$

is strict. (The inclusion always holds if the wall satisfies the containment condition.) That is, $[V]$ can contain a tuple that is not in the projection of any window on a scheme larger than V.

It is not hard to show for $[\cdot]_{A,0}$ that the implicit objects are precisely **0**. For the two specific weak instance window functions we covered, we can characterize the implicit objects as follows.

Theorem 6

V is an implicit object for $[\cdot]_K$ if *V* is the union of relation schemes that have a lossless extension join under **K**.

[]

Theorem 7

V is an implicit object for $[\cdot]_{*R}$ if *V* is the scheme of an embedded join dependency ***S** implied by ***R**, where $S \subseteq R$.

Proof

The result follows from two facts.

First, if $RI_{*R}(d)$ contains a row that is non-null exactly on *V*, then there is an embedded JD implied by ***R** with scheme *Y*. (Lemma 5.1 of Yannakakis <Ya>.)

Second, if *Y* is the scheme of an embedded JD implied by ***R**, then it is possible to find a database state *d*, such that $RI_{*R}(d)$ contains a row defined exactly on *V* and no rows that are non-null on more than *V*.

[]

Both theorems imply that all relation schemes are implicit objects. Using hypergraph notation <BFMY>, we can describe the implicit objects for $[\cdot]_{*R}$. The JD ***R** implies the embedded JD ***S**, $S \subseteq R$, if and only if **S** is closed, connected, and whenever it contains two edges of a block of **R**, it contains all the edges in the block.

The definition of an object being integral can be extended to any wall of windows by phrasing it in terms of implicit objects and defining $\alpha(W)$ in terms of **R**.

Lemma 5

$[\cdot]_K$ does not guarantee integrity of objects.

Proof

Let $R = \{ABC, BCD, ADE\}$ with keys *AB*, *BC* and *AD*. (This example is due to Sagiv <Sa1>.) The expression for $[AD]_K$ is

$$\pi_{AD}(r(ADE)) \cup \pi_{AD}(r(ABC) \bowtie r(BCD)).$$

ADE is an implicit object containing *AD*, but $[AD]_K$ depends on more than $r(ADE)$.

[]

Lemma 6

$[\cdot]_{*R}$ guarantees integrity of objects.

Proof

If *V* and *W* are schemes of embedded JDs implied by ***R**, then there is an embedded JD on scheme $V \cap W$. Thus, implicit objects for $[\cdot]_{*R}$ are closed under intersection, and Theorem 4 applies.

[]

4. Current Work

We are now adding more semantic information than simply associations and objects to our data model, and considering ways of using the information to extend the PIQUE window function. Such extensions require connections to be made by means other than just natural joins. A simple extension is to aggregate attributes; for example, "date" might be composed of "month," "day," and "year." A more complex extension is the inclusion of a role or "ISA" hierarchy on attributes. Such a semantic structure is essential to recover information on relatability of attributes that is lost when attributes are renamed in order to satisfy URSA. We extend the set of object relations by making connections with equijoins on compatible attributes in the hierarchy <Ro2>. There are difficulties with this approach, as care must be taken that object relation formed with equijoins not clash with one formed with natural joins. These extensions to the model are leading us to believe that relational algebra may not be a sufficiently powerful system with which to compute window functions. Therefore, we are investigating logic programming languages as an alternative.

5. References

<AK> A.V. Aho, B.W. Kernighan. research!user/ava/q/README, 1980.

<ABU> A.V. Aho, C. Beeri, J.D. Ullman. The theory of joins in relational databases, *ACM TODS* 4(3), September 1979, 297-314.

<ASU1> A.V. Aho, Y. Sagiv, J.D. Ullman. Equivalence of relational expressions, *SIAM J. on Computing* 8(2), May 1979, 218-246.

<ASU2> A.V. Aho, Y. Sagiv, J.D. Ullman. Efficient optimization of a class of relational expressions, *ACM TODS* 4(4), December 1979, 435-454.

<AC> A.K. Arora, C.R. Carlson. The information preserving properties of relational database transformations, VLDB IV, October 1978, 352-359.

<BFMMUY> C. Beeri, R. Fagin, A.O. Mendelzon, D. Maier, J.D. Ullman, M. Yannakakis. Properties of acyclic database schemes, Thirteenth ACM Symp. on Theory of Computing, May 1981, 355-362.

<BFMY> C. Beeri, R. Fagin, D. Maier, M. Yannakakis. On the desirable properties of acyclic database schemes, to appear *JACM*.

<BK> C. Beeri, H.F. Korth. Compatible attributes in a universal relation, ACM Symp. on Principles of Database Systems, March 1982, 55-62.

<BMSU> C. Beeri, A.O. Mendelzon, Y. Sagiv, J.D. Ullman. Equivalence of relational database schemes, *SIAM J. on Computing* 10(2), May 1981, 352-370.

<Be> P.A. Bernstein. Synthesizing third normal form relations from functional dependencies, *ACM TODS* 1(4), December 1976, 277-298.

<BDB> J. Biskup, U. Dayal, P.A. Bernstein. Synthesizing independent database schemas, 1979 ACM SIGMOD Conf., May-June 1979, 143-152.

<CK> C.R. Carlson, R.S. Kaplan. A generalized access path model and its application to a relational database system, 1976 ACM SIGMOD Conf., June 1976, 143-154.

<FMU> R. Fagin, A.O. Mendelzon, J.D. Ullman. A simplified universal relation assumption and its properties, *ACM TODS* 7(3), September 1982, 343-360.

<Gr> M.H. Graham. On the universal relation, CSRG Report, Univ. of Toronto, December 1979.

<Ho1> P. Honeyman. Extension Joins, VLDB VI, October 1980, 239-244.

<Ho2> P. Honeyman. Testing satisfaction of functional dependencies, *JACM* 29(3), July 1982, 668-677.

<Ho3> P. Honeyman. Finding lossless joins, unwritten manuscript, 1982.

<Ko> System/U: a progress report, XP2 Workshop on Relational Database Theory, June 1981.

<KU> H.F. Korth, J.D. Ullman. System/U: A database system based on the universal relation assumption, XP1 Workshop on Relational Database Theory, June-July 1980.

<KMRS> S.M. Kuck, D.A. McNabb, S.V. Rice, Y. Sagiv. The Parafrase database user's manual, Computer Science Technical Report 80-1046, Univ. of Illinois, December 1980.

<KS> S.M. Kuck, Y. Sagiv. A universal relation database system implemented via the network model, ACM Symp. on Principles of Database Systems, March 1982, 147-157.

<Lo> E.L. Lozinskii. Construction of relations in relational databases, *ACM TODS* 5(2), June 1980, 208-224.

<Ma> D. Maier. Discarding the universal instance assumption: preliminary results, XP1 Workshop on Relational Database Theory, June-July 1980.

<MMSU> D. Maier, A.O. Mendelzon, F. Sadri, J.D. Ullman. Adequacy of decompositions of relational databases, *J. of Computer and System Sciences* 17(2), December 1980, 368-379.

<MMS> D. Maier, A.O. Mendelzon, Y. Sagiv. Testing implications of data dependencies, *ACM TODS* 4(4), December 1979, 455-469.

<MR> D. Maier, D. Rozenshtein. The case for subscheme relations, unpublished manuscript, 1982.

<MRSSW> D. Maier, D. Rozenshtein, S.C. Salveter, J.Stein, D.S. Warren. Toward logical data independence: A relational query language without relations, 1982 ACM SIGMOD Conf., June 1982, 51-60.

<MU> D. Maier, J.D. Ullman. Maximal objects and the semantics of universal relation databases, to appear *ACM TODS*.

<MUV1> D. Maier, J.D. Ullman, M.Y. Vardi. The equivalence of universal relation definitions, unpublished manuscript, 1982.

<MUV2> D. Maier, J.D. Ullman, M.Y. Vardi. Beyond universal instances, manuscript in preparation, 1982.

<MW> D. Maier, D.S. Warren. Specifying connections for a universal relation scheme database, 1982 ACM SIGMOD Conf., June 1982, 1-7.

<Me> A.O. Mendelzon. Database states and their tableaux, XP2 Workshop on Relational Database Theory, June 1981.

<Os> S.L. Osborn. Towards a universal relation interface, VLDB V, October 1979, 52-60.

<Ri> J. Rissanen. Independent components of relations, *ACM TODS* 2(4), December 1977, 317-325.

<Ro1> D. Rozenshtein. Query and authorization of updates in the association-object data model, Computer Science Technical Report 82-040, SUNY at Stony Brook, May 1982.

<Ro2> D. Rozenshtein. Query and role playing in the association-object data model, Doctoral dissertation in preparation, SUNY at Stony Brook, 1983.

<Sa1> Y. Sagiv. Can we use the universal instance assumption without using nulls?, 1981 ACM SIGMOD Conf., April-May 1981, 108-120.

<Sa2> Y. Sagiv. A characterization of globally consistent databases and their correct access paths, Computer Science Technical Report, Univ. of Illinois, July 1981.

<Sc1> E. Sciore. The universal instance and database design, Doctoral dissertation, Princeton Univ., October 1980.

<Sc2> Improving semantic specification in a relational database, 1979 ACM SIGMOD Conf., May-June 1979, 170-178.

<SP> K.L. Schenk, J.R. Pinkert. An algorithm for servicing multi-relational queries, 1979 ACM SIGMOD Conf., August 1977, 10-20.

<SS> J.M. Smith, D.C.P. Smith. Database abstractions: Aggregation and generalization, *ACM TODS* 2(2), June 1977, 105-133.

<St> J. Stein, Data definition and update in the association-object data model, Doctoral dissertation in preparation, SUNY at Stony Brook, 1983.

<Ul> J.D. Ullman. The U.R. strikes back. ACM Symp. on Principles of Database Systems, March 1982, 10-22.

<Va> Y. Vassiliou. Functional dependencies and incomplete information, VLDB VI, October 1980, 260-269.

<Wa> A. Walker. A universal table relational data base model with blank entries, unpublished manuscript, 1979.

<Ya> M. Yannakakis. Algorithms for acyclic database schemes, VLDB VII, September 1981, 82-94.