An Extension of the Relational Data Model to Incorporate Ordered Domains

WILFRED NG
The Hong Kong University of Science and Technology

We extend the relational data model to incorporate partial orderings into data domains, which we call the ordered relational model. Within the extended model, we define the Partially Ordered Relational Algebra (the PORA) by allowing the ordering predicate $\sqsubseteq$ to be used in formulae of the selection operator (σ). The PORA expresses exactly the set of all possible relations which are invariant under order-preserving automorphism of databases. This result characterises the expressiveness of the PORA and justifies the development of Ordered SQL (OSQL) as a query language for ordered databases. OSQL provides users with the capability of capturing the semantics of ordered data in many advanced applications, such as those having temporal or incomplete information. Ordered Functional Dependencies (OFDs) on ordered databases are studied, based on two possible extensions of domain orderings: (1) pointwise-ordering and (2) lexicographical ordering. We present a sound and complete axiom system for OFDs in the first case and establish a set of sound and complete chase rules for OFDs in the second. Our results suggest that the implication problems for both cases of OFDs are decidable and that the enforcement of OFDs in ordered relations are practically feasible. In a wider perspective, the proposed model explores an important area of object-relational databases, since ordered domains can be viewed as a general kind of data type.

Categories and Subject Descriptors: H.2.1 [**Database Management**]: Logical Design—*data models*; H.2.3 [**Database Management**]: Languages—*query languages*; H.2.4 [**Database Management**]: Systems—*relational databases*

General Terms: Design, Languages, Theory

Additional Key Words and Phrases: Partially ordered domains, ordered relational model, ordered relations, pointwise-ordering, lexicographical ordering, mixed ordering, ordered functional dependencies, implication problem, axiom system, chase rules, order-preserving database automorphism, valuation mapping, tableaux, partially ordered relational algebra, ordered SQL, language expressiveness, non-uniform completeness

Address: Department of Computer Science, The Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong; email: wilfred@cs.ust.hk

1. INTRODUCTION

The relational data model has been introduced by Codd [Codd 1970] over the last two decades, resulting in the development of relational DBMSs. Let us begin by reviewing the three main advantages offered by this model.

—From the point of view of *usability*, the model is natural and has a simple interpretation in terms of real world concepts. The essential data structure of the model is a relation, which can be visualised in a tabular format. Due to this simplicity, relational databases have gained acceptance from a broad range of users.

—From the point of view of *applicability*, the model is flexible and general, and can be easily adapted to many applications, especially business-oriented ones such as accounting and payroll processing. Thus, the model has the advantage that it has gained popularity and credibility in a variety of application areas.

—From the point of view of *formalism*, the model is elegant enough to support extensive research and analysis. Since the framework of the model is based on well-established set-theoretic formalism, it facilitates better theoretical research in many fundamental issues arising from database query languages and dependency theory, which have had a major impact on DBMS development.

In this paper, we propose an extension of the relational data model, in which we strive for a balance between maintaining the mentioned desirabilities of the conventional data model and searching for a new data model to facilitate the better use of database technology into new application domains. On the one hand, the extension we propose is as minimal as possible, in the sense that we preserve the formal basis of the relational model. On the other hand, our extended model is fundamental enough to unify significant classes of different specialised applications, and to provide a sound basis for the investigation of new possible applications.

An important notion introduced in our extended model is that, given a data domain, apart from the *standard domain orderings* such as numerical orderings and alphabetical orderings, a user can also declare new *semantic orderings* to override the standard domain orderings. The *system orderings* may or may not follow the domain ordering because different DBMSs have their own storage and retrieval strategy. The choice of ordering at this level depends entirely on the implementation of the system. Using the three DBMS levels of the conventional model [ANSI/X3/SPARC 1975; Ullman 1988], we show in Figure 1 the differences between the various notions of orderings introduced so far. We emphasise that within the context of the ordered relational model the external level may provide a number of semantic orderings which corresponds to different database applications or user groups.

The following example illustrates the use of semantic orderings in three domains.

A Motivating Example of Ordered Domains

Example 1. In Figure 2(a) we have the semantic domain EMP_RANK, consisting of three employee ranks describing a simplified post hierarchy in a company. The semantics are that two Vice-Presidents of Marketing and Development (VPM and VPD) are the subordinates of the Chief Executive Officer (CEO). In Figure 2(b) we

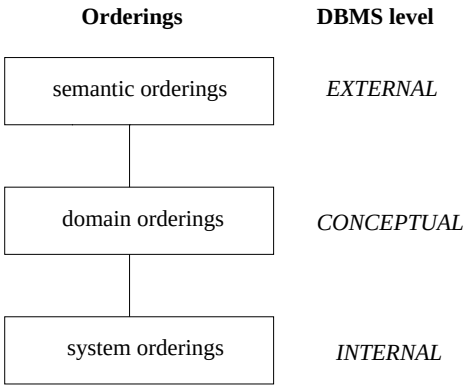

Fig. 1. Orderings at different DBMS levels

have the semantic domain SALARY_REVISED_DATE consisting of four usual dates which simply follow the chronological ordering provided by a DBMS. Note that in this case we use the standard domain ordering for Gregorian time system (day-month-year). In case of other specialised time systems we may need to consider lexicographical orderings over a list of user-defined time domains. The idea of using a list of integer domains to capture the semantics of non-standard time data has been proposed by Lorentzos [Lorentzos 1992]. Such an approach to handling time data can be easily adapted in the framework of our ordered relational model. In Figure 2(c) we have the semantic domain INCOMPLETE which captures the semantics of different null values in a database. These null values model various types of incomplete information as follows: the known data value "programmer" is *more informative* than the null symbol UNK (data exists but is UNKnown), and the null symbols UNK and DNE (data Does Not Exist) are *more informative* than the null symbol NI (No Information to decide whether it is the case of UNK or DNE).

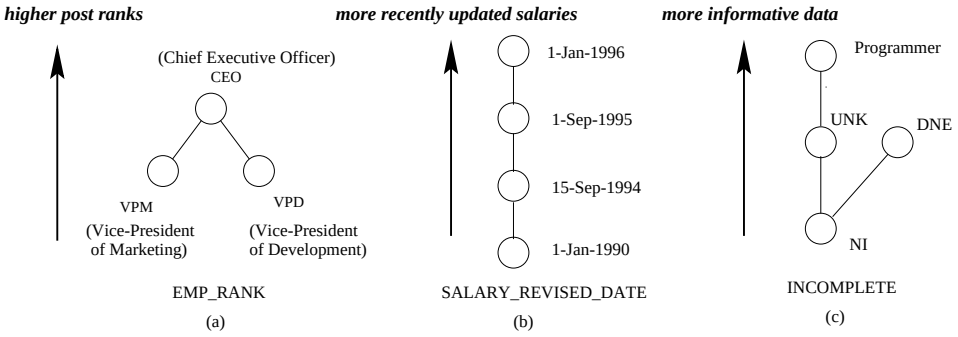

Fig. 2. Using ordered domains to capture the semantics in various information

We note that *object-oriented* methodology has the benefit of using the notion of partial ordering to form class hierarchies [Rumbaugh 1988]. For example, the "kinds

of" relationship, which represents the fact that one object class (super class) is a generalisation of another class (subclass), is clearly a kind of partial ordering. The "part of" relationship, which represents the fact that one object class (component class) is an aggregation of another class (assembly class), also satisfies the criteria of partial ordering. In this context, we use partial ordering to capture the semantics of the association between data elements in a given domain, which are different from the class relationships used in modelling objects association in the following two perspectives. First, we do not consider object features, such as an *object identity* and *associated operations*, and therefore we still assume atomic domains. Second, the class relationships in the object-oriented approach are commonly used to indicate various kinds of semantic connection between classes [Blaha and Premerlani 1998]. However, we are more concerned with the reasoning of ordering information properties with respect to an application, such as "higher position in a post hierarchy" or "more informative data" as shown in Figure 2.

The Scope of Our Investigation

Specifically, we study the effects on the following three components of the conventional model of incorporating partial orderings, which are based on set-theoretic formalism:

Structural. We impose a partial ordering on all the data domains of attributes; by this we mean that a partial ordering is an integral part of an ordered domain[1]. There follows an induced lexicographical ordering on tuples as the relation schema is assumed to be linearly ordered. This serves as a minimal extended model which incorporates orderings.

Operational. We extend the relational algebra to the Partially Ordered Relational Algebra (which we call the PORA) by allowing the use of the ordering predicate, $\sqsubseteq$, in the language. The formalism of the PORA provides the basis to extend SQL to Ordered SQL (OSQL), which is a query language for ordered databases. As a simple illustration of the usefulness of the PORA, consider the semantic domain given in Figure 2(a), which consists of three post names where 'VPM' $\sqsubseteq$ 'CEO' and 'VPD' $\sqsubseteq$ 'CEO'. Suppose we would like to find the names of all the subordinates of the CEO. This query can be formulated in the PORA as $\pi_{NAME}(\sigma_{POST \sqsubseteq 'CEO'}(\text{STAFF}))$, where STAFF is a relation over $\{NAME, POST\}$. We note that such semantics cannot easily be captured without imposing an order on the underlying domain.

Constraints. We consider Ordered Functional Dependencies (which we call OFDs) which are generalised forms of Functional Dependencies (FDs) [Ullman 1988; Atzeni and De Antonellis 1993)]. OFDs can capture a monotonicity property between two sets of values projected onto some attributes in a relation. We find that monotonicity properties arise naturally in many applications, especially in those that consist of temporal data. Two simple examples are that an OFD can capture the constraint that the salary of an employee increases every year, and that in a bank account the chronological ordering of dates increases as does the numerical ordering of cheque numbers.

[1]In linguistic custom we shall often say "let X be a partially ordered set," when what we really mean "let X be the domain of a partial order" and the precise meaning is an ordered pair $\langle X, \sqsubseteq \rangle$.

Related Research Work

There is strong evidence in recent research that ordering is inherent to the underlying structure of data in many database applications [Buneman et al. 1991; Maier and Vance 1993; Read 1995; Libkin 1996; Raymond 1996; Wijsen 1998]. For instance, Raymond [Raymond 1996] illustrates the potential of using partial orderings with many application examples such as textual and software information. Read [Read 1995] shows that multi-resolution domains have very strong connections with the notion of approximation such as incompleteness or impreciseness of data. A multi-resolution set is defined as a partially ordered set that has a unique minimal element and some maximal elements, and in addition, there is an associated truth function mapping each element onto Boolean values in order to determine its resolution level. Buneman [Buneman et al. 1991] and his colleagues [Jung et al. 1991; Libkin 1996] have extensively studied various kinds of orderings on powerdomains considered to be useful in incomplete information. There has also been a fair amount of research to extend the relational data model to include *lists* or *sequences* as data types [Ginsburg and Tanaka 1986; Guting et al. 1989; Seshadri et al. 1996]. A list can arrange objects in some pre-defined order and thus a non-repeating list of real world objects can be regarded as a linearly ordered domain.

Ginsburg [Ginsburg and Hull 1983] introduced the term *order dependencies* and examined the issue of the extension of functional dependencies to incorporate information involving partial order. The central notion of order dependencies is similar to that of our definition of ordered functional dependencies arising from pointwise-orderings (POFDs), except that the involved domain orderings in order dependencies are classified into total order, empty order and general partial-order. They also exhibit a sound and complete set of inference rules for order dependencies but the implication problem is shown to be co-NP complete [Garey and Johnson 1979]. Compared with order dependencies, POFDs do not take into account such refinement on an ordered domain. However, the implication problem for POFDs is found to be linear. Importantly, we investigate the new notion of OFDs arising from lexicographical orderings (LOFDs), and further clarify the issues arising from the semantics of POFDs, LOFDs and other variant forms of OFDs in the context of ordered relations.

The rest of the paper is organised as follows. In Section 2 we clarify the notion of order and formally extend the relational data model to include partial orderings into the structure of the model. In Section 3 we investigate the expressive power of the PORA and show that it is complete, in the sense that it satisfies a generalised form of Paredaens' and Bancilhon's Theorem. We describe OSQL and demonstrate its capabilities of capturing semantics in three advanced applications. In Section 4 we extend the notion of FDs in the context of ordered databases, and study their implication problems by using the axiom system approach and the chase rule approach. In Section 5 we conclude our work with some final remarks and discuss future work resulting from the ordered relational model.

2. THE ORDERED RELATIONAL MODEL

In this section we clarify the notions of order [Halmos 1974; Gratzer 1978], formally define the ordered relational model, and compare the features of ordered relations

with conventional relations.

2.1 Extensions of Partial Orderings

Let S and T be sets, then $\mid S \mid$ denotes the *cardinality* of S, $S \subseteq T$ denotes *set inclusion*, $S \subset T$ denotes *proper set inclusion* and $\mathcal{P}(S)$ denotes the *finite powerset* of S. We denote the k term Cartesian product $S \times S \cdots \times S$ by S^k, and the singleton $\{A\}$ simply by A when no ambiguity arises. We assume the usual definition of a *partial ordering* $\sqsubseteq$ on the set S: a binary relation on S satisfying the conditions of *reflexivity*, *anti-symmetry* and *transitivity* [Gratzer 1978]. In the special cases when $\sqsubseteq$ is a *linear ordering*, we denote it by $\leq$. At the other extreme, when each element is only comparable with itself, S is completely unordered, i.e., $\sqsubseteq$ is just the equality predicate $=$. We denote that x and y are incomparable by $x \parallel y$, and that $x \sqsubseteq y$ but $x \neq y$ by $x \sqsubset y$. Note that for any elements x and y in S, if $x \neq y$, then exactly one of the following holds: $x \sqsubseteq y$, $y \sqsubseteq x$, or $x \parallel y$. This implies that $x \parallel y$ if and only if $x \not\sqsubseteq y$ and $y \not\sqsubseteq x$.

A *partially ordered set* (or simply an ordered set) is a structure $\langle S, \sqsubseteq \rangle$. It consists of a set S which is partially ordered by the relation $\sqsubseteq$. In particular, the structure $\langle S, \leq \rangle$ is called a *linearly ordered set* and the structure $\langle S, = \rangle$ is called an *unordered set*. From now on the term *ordered* will mean partially ordered, unless stated explicitly otherwise. Furthermore, when two ordered sets $\langle T, \sqsubseteq_T \rangle$ and $\langle S, \sqsubseteq_S \rangle$, where $T \subseteq S$, satisfy $a_1 \sqsubseteq_T a_2$ if and only if $a_1 \sqsubseteq_S a_2$ for all $a_1, a_2 \in T$, we call $\mathcal{T}$ a *subordering* of $\mathcal{S}$. In this case we may write $\langle T, \sqsubseteq_T \rangle$ as $\langle T, \sqsubseteq_S \rangle$.

We now define *pointwise-ordering* and *lexicographical ordering*, which are two important kinds of domain orderings on the Cartesian product of ordered sets.

Definition 1. (*Pointwise-Ordering and Lexicographical Ordering*) Let $t_1, t_2 \in S$. A *pointwise-ordering* on S, denoted by $\sqsubseteq_S^p$, is defined as follows: $t_1 \sqsubseteq_S^p t_2$, if, for all $1 \leq i \leq n$, $t_1[i] \sqsubseteq_{D_i} t_2[i]$. A *lexicographical ordering* on S, denoted by $\sqsubseteq_S^l$, is defined as follows: $t_1 \sqsubseteq_S^l t_2$, if either (1) there exists k with $1 \leq k \leq n$ such that $t_1[k] \sqsubset_{D_k} t_2[k]$, and for all $1 \leq i < k$, $t_1[i] = t_2[i]$, or (2) for all $1 \leq i \leq n$, $t_1[i] = t_2[i]$.

A pointwise-ordering occurs naturally in incomplete information. The standard way of dealing with incomplete information is by using the so-called *null values* which are commonly interpreted as "value at present UNKnown" [Codd 1986]. Other interpretations of null values are DNE and NI [Zaniolo 1984], whose relationship has been discussed in Example 1. For a simple illustration of using pointwise-ordering in an incomplete relation we just assume that a domain of constants, denoted as Dom, contains a distinguished symbol UNK, and define a partial ordering in Dom as follows, for all $x, y \in Dom$, $x \sqsubseteq y$ if $x = y$ or $x =$ UNK. Then we can extend $\sqsubseteq$ to be a pointwise-ordering in a relation r over $\{A, B\}$ as follows, for all $t_1, t_2 \in r$, $t_1 \sqsubseteq^p t_2$ if $t_1[A] \sqsubseteq t_2[A]$ and $t_1[B] \sqsubseteq t_2[B]$. This extension naturally captures the meaning of t_1 being *less informative* than t_2, or alternatively t_2 being *more informative* than t_1. We emphasise that in this approach the "informativeness" between tuples is being compared rather than comparing data values, and thus the correct interpretation of two UNK symbols should be "equally informative". Actually, the relationship between incompleteness and orderings is commonly used to study issues concerning incomplete information [Zaniolo 1984;

Libkin 1996; Levene and Loizou 1997].

We can construct the lexicographical ordering on alphabets, which we commonly call a *dictionary ordering* or an *alphabetical ordering*, since it resembles the ordering of words in a dictionary. The ordering of the domain $DATE$, called *chronological ordering*, can be viewed as the lexicographical ordering of the domains $YEAR$, $MONTH$ and DAY, if (1) the domain $MONTH$ has the ordering as $\{Jan \leq Feb \leq \cdots \leq Dec\}$ and (2) the Cartesian product of the domains is taken in the following order: $YEAR \times MONTH \times DAY$.

2.2 Orderings in Databases

Ordering is a fundamental property of almost all primitive data types and is inherent to the underlying structure of data in many database applications [Maier and Vance 1993; Read 1995; Libkin 1996; Raymond 1996; Ng and Levene 1997b]. However, all relational database systems support only the following three kinds of *standard domain orderings* considered to be essential in practical utilisation: (1) the *alphabetical ordering* over the domain of strings, (2) the *numerical ordering* over the domain of numbers, and (3) the *chronological ordering* over the domain of dates [Date 1990]. The limited support of domain orderings results in a loss of the semantics of data. We call ordering semantics in the context of a specific application *semantic ordering* and incorporate this notion into the relational data model. The relationship between various notions of ordering has been shown in Figure 1.

We let D be a countably infinite set of constant values and $\sqsubseteq_D$ be an ordering on D. For the sake of simplicity in presenting our results, we assume that all attributes share the same ordered domain, though in practice it may need more than one ordering imposed over D. Note that our results obtained in the subsequent sections can be generalised in a straightforward manner to the situation where several partial orderings are defined on D, thus in this sense our assumption is no loss in generality.

We now give the definition of ordered databases.

Definition 2. (*Attributes and Ordered Domains*) We assume a countably infinite linearly ordered set of *attribute names*, $\langle U, \leq_U \rangle$[2]. For all attributes $A \in U$, the domain of A is $\langle D, \sqsubseteq_D \rangle$. We call $\sqsubseteq_D$ the *domain ordering* of D.

Definition 3. (*Relation Schema and Database Schema*) A *relation schema* (or simply a schema) R, is a subset of U consisting of a finite set of attributes $\{A_1,\ldots,A_m\}$ for some $m \geq 1$. A *database schema* is a finite set $\mathbf{R} = \{R_1, \ldots, R_n\}$ of relation schemas, for some $n \geq 1$.

Definition 4. (*Tuple and Tuple Projection*) Let $X = \{A_1, \ldots, A_m\}$ be a finite subset of U. A *tuple* t over X is a member of D^m. We let $t[A_i]$ denote the ith coordinate of t. The *projection* of a tuple t onto a set of attributes $Y = \{A_{i_1}, \ldots, A_{i_k}\}$, where $1 \leq i_1 < \cdots < i_k \leq m$, is the tuple $t[Y] = \langle t[A_{i_1}], \ldots, t[A_{i_k}] \rangle$.

Definition 5. (*Ordered Relation and Ordered Database*) An *ordered relation* (or simply a relation) r defined over a schema R is a finite set of tuples over R. An *ordered database* (or simply a database) over $\mathbf{R} = \{R_1, \ldots, R_n\}$ is a finite set $d =$

[2]We should assume an ordering between attributes since the order of attributes in a relation schema affects the order of tuples in an ordered relation.

$\{r_1, \ldots, r_n\}$ such that each r_i is a relation over R_i. We call r and d an *unordered relation* and an *unordered database*, respectively, if the underlying domain $\langle D, \sqsubseteq_D \rangle$ is unordered, i.e., it is $\langle D, = \rangle$. Similarly, we call r and d a *linearly ordered relation* and a *linearly ordered database*, respectively, if the underlying domain is linearly ordered.

We can view a conventional database as a special case of ordered databases with unordered domains. Two important properties of conventional relations are preserved in ordered relations as follows: all domain elements are atomic and no duplicate tuples are allowed. However, it is important to note that there are two essential differences between ordered and conventional relations: first, in an ordered relation tuples are ordered according to the extension of the domain ordering but in a conventional relation tuples are unordered, and second, the attributes in the schema of an ordered relation are assumed to be linearly ordered.

3. ORDERED RELATIONAL ALGEBRA AND ORDERED SQL

In this section we extend the relational algebra to the Partially Ordered Relational Algebra (the PORA) by allowing the ordering predicate, $\sqsubseteq$, to be used in the formulae of the *selection* operator (σ). We apply Paredaens and Bancilhon's Theorem to examine the expressiveness of the PORA, and show that the PORA expresses exactly the set of all possible relations which are invariant under order-preserving automorphism of databases. The extension is consistent with the two important extreme cases of unordered and linearly ordered domains. We also investigate the three hierarchies of: (1) computable queries, (2) query languages and (3) partially ordered domains, and show that there is a one-to-one correspondence between them.

The PORA provides the formal basis to develop OSQL, which is an extension of SQL for the ordered relational model. Queries in OSQL are formulated in essentially the same way as in standard SQL. Let us first demonstrate this mode of querying with the following example showing how OSQL simplifies the specification of certain queries.

Example 2.

(1) Obtain the months having the three lowest amount of rainfall according to a rainfall record.
(Q_1) *SELECT* MONTH, RAINFALL *FROM* RAIN_RECORD_TABLE
WITHIN RAINFALL_ORDER *WHERE* TUPLE(1..3).

(2) Obtain the names of all staff who are more senior than the Vice-President of Development (VPD).
(Q_2) *SELECT* NAME *FROM* STAFF
WHERE POST > 'VPD' *WITHIN* RANK_ORDER.

The meaning of the above statements is quite easy to understand and the syntax is similar to standard SQL. The *tuple level set* (1..3) of the built-in predicate *TUPLE* in the query (Q_1) means that the first to third tuples of the RAIN_RECORD_TABLE, which are sorted according to the order of RAINFALL amount, are returned as output. The keyword *WITHIN* in the query (Q_2) specifies that the comparison POST > 'VPD', which is interpreted according to the semantic ordering of RANK_ORDER.

Throughout this section we let *id* be the identity mapping on any set. We use the term *active domain*, $adom(d)$, to represent the set containing those values that appear in a database instance d. Thus, $\langle adom(d), \sqsubseteq \rangle$ is a subordering of the underlying domain of d (recall the meaning of subordering discussed in Section 2.1).

Definition 6. (*Active Domain*) The *active domain* of a relation r over R, denoted as $adom(r)$, is defined by $adom(r) = \{v \mid \exists A \in R, \exists t \in r \text{ such that } t[A] = v\}$. The *active domain* of a database instance $d = \{r_1, \ldots, r_n\}$ over $\mathbf{R}$ is defined by $adom(d) = \bigcup_{i=1}^{n} adom(r_i)$.

3.1 Query Language: the PORA

The PORA is essentially the classical relational algebra with the ordering predicate added to deal with ordered domains. This language consists of a collection of six operators, each of which maps a set of relations to a relation.

Definition 7. (*Partially Ordered Relational Algebra*) The PORA is a collection of the following six operators: *union* ($\cup$), *Cartesian product* ($\times$), *difference* ($-$), *projection* (π_X), where $X \subseteq U$ is a finite set of attributes, *renaming* ($\rho_{X \to Y}$), where $X \to Y$ is a bijective function from a finite set of attributes $X \subseteq U$ to a finite set of attributes $Y \subseteq U$, and lastly extended *selection* (σ_F), where the *selection formula* F is restricted to be one of the forms: $A = B$, $A \neq B$, $A \sqsubseteq B$ or $A \not\sqsubseteq B$, where $A \in U$, and either $B \in U$ or B is a constant.

The six operators given in Definition 7 are the standard ones (see [Atzeni and De Antonellis 1993)] for their formal definitions and semantics), and the meaning of σ over the formula $A \sqsubseteq B$ is also as expected, i.e., given a relation r, $\sigma_{A \sqsubseteq B}(r) = \{t \in r \mid t[A] \sqsubseteq t[B]\}$. In order to avoid mismatching in domain orderings, we choose to interpret the *union compatibility* as follows: the union is applicable only to two relations with the same schema, and the orderings of the domains of the corresponding attributes are the same. (Recall that domain ordering is assumed to be an integral part of a domain in Definition 2.)

Definition 8. (*PORA Expressions and their Equivalences*) A *PORA expression* is a well-formed expression using PORA operators whose operands are relation schemas. E_{PORA} is the set of all PORA expressions. The *answer* $e(d)$ to an expression $e \in E_{PORA}$ with respect to a database d over $\mathbf{R}$ is obtained by substituting the relation r_i for every occurrence of R_i in e, for each i, and computing the result by invoking the operators present in e. The answer is undefined if some operand R of an expression is not in $\mathbf{R}$. Two sets of expressions E_1 and E_2 are *equivalent*, $E_1 \equiv E_2$, if for every $e_1 \in E_1$ there is some $e_2 \in E_2$ such that $e_1(d) = e_2(d)$ for all $d \in DB(\mathbf{R})$, and vice versa.

We observe that $\sigma_{=}$, $\sigma_{\neq}$ and $\sigma_{\not\sqsubseteq}$ are not primitive[3], since for any relation r they can be simulated as follows, $\sigma_{A=B}(r) \equiv \sigma_{A \sqsubseteq B}(\sigma_{B \sqsubseteq A}(r))$, $\sigma_{A \neq B}(r) \equiv r - \sigma_{A=B}(r)$, and $\sigma_{A \not\sqsubseteq B}(r) \equiv r - \sigma_{A \sqsubseteq B}(r)$. In the extreme case of an unordered domain $\sigma_{\sqsubseteq}$ becomes $\sigma_{=}$, i.e., for any relation r, $\sigma_{A \sqsubseteq B}(r) \equiv \sigma_{A=B}(r)$. Therefore, our definition of the PORA is consistent with the standard relational algebra used in [Paredaens

[3]It is also interesting to note that $\sigma_{\|}$ is not primitive and can be simulated by $\sigma_{A \| B}(r) \equiv \sigma_{A \not\sqsubseteq B}(\sigma_{B \not\sqsubseteq A}(r))$.

1978]. Let UORA = $\{\rho, -, \times, \cup, \pi, \sigma_{=}, \sigma_{\neq}\}$ be the unordered relational algebra and LORA = $\{\rho, -, \times, \cup, \pi, \sigma_{\leq}, \sigma_{\not\leq}\}$ be the linearly ordered relational algebra for a given linear ordering of D. We formalise our observations as follows:

PROPOSITION 1. *Let $\langle D, \sqsubseteq_D \rangle$ be the underlying domain. Then*

(1) $E_{PORA} \equiv E_{UORA}$ if $\langle D, \sqsubseteq_D \rangle$ is unordered, and
(2) $E_{PORA} \equiv E_{LORA}$ if $\langle D, \sqsubseteq_D \rangle$ is linearly ordered.

Note that those relations which can be generated by E_{PORA} involve only relations in d and contain values solely in $adom(d)$. We denote by $e_{ad}(d)$ the PORA expression that generates $adom(d)$. The following proposition will be repeatedly used in many formal proofs in Sections 3.2 and 3.3.

PROPOSITION 2. *Let $e_{ad}(d) = \bigcup_{i,j} \pi_{A_j}(r_i)$, $\forall A_j \in R_i$ where $R_i \in \mathbf{R}$, and $\forall r_i \in d$. Then $adom(d) = e_{ad}(d)$.*

The *possible information* of d is the countably infinite set of all relations that can be derived from the $adom(d)$.

Definition 9. (*Possible Information*) The *possible information* of d, denoted by $Poss(d)$, is defined by $Poss(d) = \bigcup_{i=0}^{\infty} \mathcal{P}(adom(d)^i)$.

3.2 Expressiveness of the PORA

We first discuss the concept of order-preserving database automorphism and then examine the expressive power of the PORA by Paredaens' and Bancilhon's Theorem [Paredaens 1978; Bancilhon 1978]. The notion of automorphism [Atzeni and De Antonellis 1993)] is generalised to the context of ordered databases.

Definition 10. (*Ordering Automorphism*) Let $\langle S, \sqsubseteq \rangle$ be an ordered set. The function $f : S \longrightarrow S$ is an *ordering automorphism*, if f is bijective and satisfies the condition that $a_1 \sqsubseteq a_2$ if and only if $f(a_1) \sqsubseteq f(a_2)$. If the set $\{a \in S \mid f(a) \neq a\}$ is finite, then we call f a *finite* ordering automorphism. We denote the set of all finite ordering automorphisms of an ordered set $\langle S, \sqsubseteq \rangle$ by $Aut(S, \sqsubseteq)$, or simply $Aut(S)$.

We now define an order-preserving automorphism of a database. Informally, this is a permutation of the values in the active domain of a database instance that does not alter the database and also preserves the ordering of the active domain.

Definition 11. (*Order-Preserving Database Automorphism*) Let h be an ordering automorphism of $\langle adom(d), \sqsubseteq \rangle$. Then h is extended to tuples t, relations r and databases d as follows: $h(t) = \langle h(a_1), \ldots, h(a_m) \rangle$ where $t = \langle a_1, \ldots, a_m \rangle$, $h(r) = \{h(t_1), \ldots, h(t_k)\}$ where $r = \{t_1, \ldots, t_k\}$, and $h(d) = \{h(r_1), \ldots, h(r_n)\}$ where $d = \{r_1, \ldots, r_n\}$. We call h an *order-preserving database automorphism*, if its extension to d satisfies the condition that $h(r_i) = r_i$ for $1 \leq i \leq n$. We simply write this condition as $h(d) = d$ if no ambiguity arises. The set of all order-preserving database automorphisms of database d is denoted by $Aut(\sqsubseteq, d)$, or simply $Aut(d)$ when $\sqsubseteq$ is clear from the context.

It follows from Definition 11 that, for all partial orderings $\sqsubseteq$, $id \in Aut(\sqsubseteq, d) \subseteq Aut(=, d)$. It also follows that $Aut(\sqsubseteq, d) = Aut(=, d) \cap Aut(adom(d), \sqsubseteq)$. The following example should help to clarify the meaning of $Aut(d)$.

Example 3. Let d contain just a single relation having 4 tuples, $r = \{\langle x, z\rangle, \langle y, z\rangle, \langle x, w\rangle, \langle y, w\rangle\}$, and let $\langle adom(d), \sqsubseteq\rangle = \langle\{w, x, y, z\}, \{x \sqsubseteq y, x \sqsubseteq z, x \sqsubseteq w\}\rangle$. We define functions: h_1 by $h_1(x) = y$, $h_1(y) = x$, $h_1(z) = z$ and $h_1(w) = w$; h_2 by $h_2(x) = x$, $h_2(y) = z$, $h_2(z) = y$ and $h_2(w) = w$; and h_3 by $h_3(x) = x$, $h_3(y) = y$, $h_3(z) = w$ and $h_3(w) = z$. Then $h_1 \notin Aut(d)$ because, although it preserves the database instance, it does not preserve the ordering; and $h_2 \notin Aut(d)$ because, although it preserves the ordering, it does not preserve the database instance; however, $h_3 \in Aut(d)$ because it preserves both the ordering and the database instance.

We now present our result of the generalisation of Paredaens' and Bancilhon's Theorem. The underlying principle in our approach is to view an ordered database as an unordered database together with a binary relation s representing $\langle adom(d), \sqsubseteq \rangle$. An ordered relation r derived from d is regarded as an unordered relation $r \times s$. We assume from now on that (1) $adom(r) \subseteq adom(d)$ (we note that this is equivalent to assuming $r \in Poss(d)$), and (2) no constant is allowed to be used in selection formulas. We need the following technical lemmas to establish our main theorem. The proofs of the next two lemmas follow from the definition of order-preserving automorphism.

LEMMA 1. *Let $d = \{r_1, \ldots, r_n\}$ be a database over $\{R_1, \ldots, R_n\}$ and S be a schema having two attributes, s be the unordered relation over S given by $s = \{\langle a, b\rangle \mid a \sqsubseteq b$ and $a, b \in adom(d)\}$, and let $d' = \{r_1, \ldots, r_n, s\}$ be considered as an unordered database over $\{R_1, \ldots, R_n, S\}$. Then $Aut(=, d') = Aut(\sqsubseteq, d)$.*

For a relation r, we define $Aut(r) = Aut(\{r, adom(d)\})$, where $adom(d)$ is regarded as a unary relation and d is understood from context.

LEMMA 2. *Let r be a relation over R and $r' = r \times s$ be considered as an unordered relation over RS, where s is defined as in Lemma 1. Then $Aut(=, r') = Aut(\sqsubseteq, r)$.*

Defining d' and r' as in the above two lemmas, the following result may be proved using induction on the number of relational operators together with some algebraic manipulation.

LEMMA 3. *Let d be a database over $\mathbf{R}$ and r a relation over R. Then $e'(d') = r'$ for some $e' \in E_{UORA}$ if and only if $e(d) = r$ for some $e \in E_{PORA}$.*

Using our notation, we can state Paredaens' and Bancilhon's Theorem as follows:

LEMMA 4. *Let d be an unordered database. Then $e(d) = r$ for some $e \in E_{UORA}$ if and only if $Aut(=, d) \subseteq Aut(=, r)$.*

We now show that this can be generalised to ordered databases.

THEOREM 1. *Let d be an ordered database over $\mathbf{R}$ and r an ordered relation over R. Then $e(d) = r$ for some $e \in E_{PORA}$ if and only if $Aut(\sqsubseteq, d) \subseteq Aut(\sqsubseteq, r)$.*

PROOF. By Lemma 1, $Aut(\sqsubseteq, d) = Aut(=, d')$ and by Lemma 2, $Aut(\sqsubseteq, r) = Aut(=, r')$. So $Aut(\sqsubseteq, d) \subseteq Aut(\sqsubseteq, r)$ if and only if $Aut(=, d') \subseteq Aut(=, r')$. By Lemma 4, $Aut(=, d') \subseteq Aut(=, r')$ if and only if $e'(d') = r'$ for some $e' \in E_{UORA}$. The result then follows by Lemma 3 that $Aut(\sqsubseteq, d) \subseteq Aut(\sqsubseteq, r)$ if and only if $e(d) = r$ for some $e \in E_{PORA}$. □

We note that Theorem 1 can be straightforwardly extended to data domains having any specified binary predicate, but in this case $\sigma_=$ may be primitive. Our result can also be easily generalised to the case of $C \neq \emptyset$, where C is the set of constants involved in queries, by replacing $Aut(d)$ in Theorem 1 by the so-called *C-fixed* $Aut(d)$ (see Section 2.3 in [Atzeni and De Antonellis 1993)]), which is defined as $\{h \in Aut(d) \mid h \text{ is an identity on } C\}$.

The following corollary is an interesting result that follows from Theorem 1. Informally, in the case of linearly ordered domains, the LORA expresses exactly the countably infinite set of all possible relations generated by the active domain of a given database. This follows immediately from the fact that, for any linear ordering $\leq$, $Aut(d) = \{id\}$.

COROLLARY 1. *Let d be a linearly ordered database. Then, for all $r \in Poss(d)$, $e(d) = r$ for some $e \in E_{LORA}$.*

For any database d, it follows from the above discussion that the PORA applied to d has no more expressive power than the UORA applied to $d \cup \{s\}$, where s is the partial order relation on $adom(d)$ as defined in Lemma 1. However, in most cases it is much more economical and efficient to compare domain elements computationally than storing and using the relation s (e.g., for a numerical or lexicographical ordering). Even for a partial order with no apparent computational structure (e.g., a post rank hierarchy), it is more economical to store the Hasse diagram [Gratzer 1978] rather than its transitive closure.

3.3 Hierarchy of Computable Queries with Ordered Domains

We now investigate the relationship between computable queries, ordered domains and partially ordered relational algebras. We first define a hierarchy for each of them and then show that there exists a one-to-one correspondence between them.

We will use index subscripts to denote different orderings over D, i.e., $\mathcal{D}_i = \langle D, \sqsubseteq_i \rangle$ where i is a positive integer. Similarly, we also write $Aut(\mathcal{D}_i)$ and $PORA_i$. The semantics of "more ordered" domains can be defined in terms of ordering automorphisms of the subsets of domains.

Definition 12. (*More Ordered Domain*) A domain $\mathcal{D}_2$ is said to be *more ordered* than another domain $\mathcal{D}_1$, denoted by $\mathcal{D}_1 \preceq \mathcal{D}_2$, if, for all $T \subseteq D$, $Aut(T, \sqsubseteq_2) \subseteq Aut(T, \sqsubseteq_1)$.

The informal reason for allowing $T \subseteq D$ in the above definition is that we take into account the fact that an active domain of a database can be defined on any subset of D. As a consequence of the definition, $Aut(d)$ is not affected by the automorphisms induced from outside the active domain. The following simple example illustrates this point.

Example 4. In Figure 3 we use Hasse diagrams to represent ordered domains. It is easy to see that, for all $T \subseteq D = \{a, b, c\}$, $Aut(T, \sqsubseteq_3) \subseteq Aut(T, \sqsubseteq_2) \subseteq Aut(T, \sqsubseteq_1)$ and thus the relationship $\mathcal{D}_1 \preceq \mathcal{D}_2 \preceq \mathcal{D}_3$ can be captured by Definition 12 in a natural manner.

Now we consider the expressiveness of the PORA for different orderings. Let the set of relations generated from the information contained in a given database d,

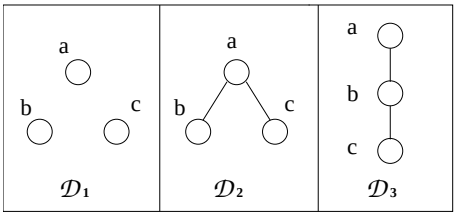

Fig. 3. Hasse diagrams of ordered domains

denoted by $Gen(\sqsubseteq_i, d)$, be defined as $\{r \mid r = e(d) \text{ for some } e \in E_{PORA_i}\}$.

Definition 13. (*More Powerful Relational Algebra*) A relational algebra $PORA_2$ is *more powerful* than another $PORA_1$, denoted by $PORA_1 \preceq PORA_2$, if, for all databases d, $Gen(\sqsubseteq_1, d) \subseteq Gen(\sqsubseteq_2, d)$.

If $PORA_2$ is a more powerful language than $PORA_1$, then we can retrieve more relations from a given database instance using $PORA_2$. We still need to extend the notion of computable query to ordered databases, but we take a different approach from [Chandra and Harel 1980]. The motivation for our definition is to include those queries which are meaningful with respect to the ordered domain concerned. The criterion for being meaningful over an ordered database d is that the query must be invariant under all order-preserving database automorphisms over d.

Let $\chi = \bigcup_{i=0}^{\infty} \mathcal{P}(D^i)$ and $DB(\mathbf{R})$ be the countably infinite set of all databases defined over a database schema $\mathbf{R}$. (Recall that D is assumed to be a common domain.)

Definition 14. (*Meaningful Computable Query*) A *meaningful computable query* with respect to a given domain $\mathcal{D}_i$, denoted by δ, is a partial recursive function from $DB(\mathbf{R})$ to χ such that for all $d \in DB(\mathbf{R})$,

(1) if $\delta(d)$ is defined, then $\delta(d) \in Poss(d)$, and
(2) for all $h \in Aut(\sqsubseteq_i, d)$, $h(\delta(d)) = \delta(d)$.

We denote the set of all meaningful computable queries by Q_i.

Note that our definition of a meaningful computable query is the same as the conventional one if we restrict ourselves to unordered domains. Now we state two technical lemmas and then present our main theorem. The first lemma follows using Theorem 1 and Lemma 2. It can be regarded as a generalisation of Lemma 2 to databases. The second lemma is useful when we compare different ordered databases. Basically it allows us to consider ordering automorphisms on the underlying domain instead of automorphisms on databases.

LEMMA 5. *Let $d = \{r_1, \ldots, r_n\}$ be a database over $\{R_1, \ldots, R_n\}$, s be the unordered relation over S, where s is defined as in Lemma 1, and let $r = r_1 \times \cdots \times r_n \times s$, considered as an unordered relation over $R_1 \cdots R_n S$. Then $Aut(\sqsubseteq, d) = Aut(=, r)$.*

LEMMA 6. *$\mathcal{D}_1 \preceq \mathcal{D}_2$ if and only if $Aut(\sqsubseteq_2, d) \subseteq Aut(\sqsubseteq_1, d)$ for all databases d over $\mathbf{R}$.*

PROOF.
IF: Consider any $h \in Aut(T, \sqsubseteq_2)$ with $T \subseteq D$. Let $X = \{a \in T \mid a \neq h(a)\}$ and, since h is a finite automorphism, suppose $X = \{a_1, \ldots, a_k\}$. Define a database d over $\mathbf{R}$ as follows, for all $r \in d$, r consists of exactly k tuples $\{t_1, \ldots, t_k\}$, where $t_i = \langle a_i, \ldots, a_i \rangle$ for $1 \leq i \leq k$. Obviously, we have that $h \in Aut(\sqsubseteq_2, d)$. By hypothesis, this implies $h \in Aut(\sqsubseteq_1, d)$ and thus $h \in Aut(T, \sqsubseteq_1)$.

ONLY IF: This follows easily by using the fact that $Aut(\sqsubseteq_i, d) = Aut(=, d) \cap Aut(T, \sqsubseteq_i)$ for any database d, where $T = adom(d)$. □

We now present our main result stating the association between domains, queries and languages. This allows us to establish hierarchies for these entities.

THEOREM 2.

(1) $\mathcal{D}_1 \preceq \mathcal{D}_2$ if and only if $Q_1 \subseteq Q_2$,
(2) $\mathcal{D}_1 \preceq \mathcal{D}_2$ if and only if $PORA_1 \preceq PORA_2$.

PROOF.
(1) *IF:* Assume $\mathcal{D}_1 \not\preceq \mathcal{D}_2$. By Lemma 6, this implies that there exists a database d' such that $h_2 \notin Aut(\sqsubseteq_1, d')$ for some $h_2 \in Aut(\sqsubseteq_2, d')$. Let $d' = \{r'_1, \ldots, r'_n\}$. We now construct a query that is in Q_1 but not in Q_2. We substitute d' for d and r' for r in Lemma 5. Thus, for all $h \in Aut(\sqsubseteq_1, d')$, we have $h(r') = r'$. On the other hand, $h_2(r') \neq r'$ since $h_2 \notin Aut(\sqsubseteq_1, d')$. We define a query δ as follows: $\delta(d) = r'$ when $d = d'$ and $\delta(d)$ is equal to the empty set otherwise. By part (2) of Definition 14, $\delta \in Q_1$ but $\delta \notin Q_2$.

ONLY IF: Let $\delta \in Q_1$ and $d \in DB(\mathbf{R})$. From Definition 14, $\delta(d) \in Poss(d)$ and, for all $h \in Aut(\sqsubseteq_1, d)$, $h(\delta(d)) = \delta(h(d))$. By the assumption $\mathcal{D}_1 \preceq \mathcal{D}_2$ and Lemma 6, $Aut(\sqsubseteq_2, d) \subseteq Aut(\sqsubseteq_1, d)$. Therefore, for all $h \in Aut(\sqsubseteq_2, d)$, $h(\delta(d)) = \delta(d)$ and thus $\delta \in Q_2$.

(2) *IF:* Assume $\mathcal{D}_1 \not\preceq \mathcal{D}_2$. By Lemma 6, there exists a database $d' = \{r_1, \ldots, r_n\}$ such that $Aut(\sqsubseteq_2, d') \not\subseteq Aut(\sqsubseteq_1, d')$. It suffices to exhibit a database d and a relation r such that $r \in Gen(\sqsubseteq_1, d)$ but $r \notin Gen(\sqsubseteq_2, d)$. We let $d = d'$ and $r = r_1 \times \cdots \times r_n \times s$ and $s = \{\langle a, b \rangle \mid a \sqsubseteq_1 b \text{ and } a, b \in adom(d')\}$. Clearly, s can be derived from d by some $e \in PORA_1$ and thus $r \in Gen(\sqsubseteq_1, d)$. It remains to show $r \notin Gen(\sqsubseteq_2, d)$. Suppose $r \in Gen(\sqsubseteq_2, d)$. By Theorem 1, $Aut(\sqsubseteq_2, d') \subseteq Aut(\sqsubseteq_2, r)$, so $Aut(\sqsubseteq_2, d') \subseteq Aut(=, r)$. By Lemma 5, it follows that $Aut(\sqsubseteq_2, d') \subseteq Aut(\sqsubseteq_1, d')$, which leads to a contradiction.

ONLY IF: Let $r \in Gen(\sqsubseteq_1, d)$. We need to show that $r \in Gen(\sqsubseteq_2, d)$. By Theorem 1, $Aut(\sqsubseteq_1, d) \subseteq Aut(\sqsubseteq_1, r)$. Thus $Aut(adom(d), \sqsubseteq_2) \cap Aut(\sqsubseteq_1, d) \subseteq Aut(adom(d), \sqsubseteq_2) \cap Aut(\sqsubseteq_1, r)$. Moreover, we have $Aut(\sqsubseteq_1, d) = Aut(adom(d), \sqsubseteq_1) \cap Aut(=, d)$ and $Aut(\sqsubseteq_1, r) = Aut(adom(d), \sqsubseteq_1) \cap Aut(=, r)$. So it follows that $Aut(adom(d), \sqsubseteq_2) \cap Aut(adom(d), \sqsubseteq_1) \cap Aut(=, d) \subseteq Aut(adom(d), \sqsubseteq_2) \cap Aut(adom(d), \sqsubseteq_1) \cap Aut(=, r)$. By the assumption of $\mathcal{D}_1 \preceq \mathcal{D}_2$ and by Definition 12, we have $Aut(adom(d), \sqsubseteq_2) \subseteq Aut(adom(d), \sqsubseteq_1)$. It follows that $Aut(adom(d), \sqsubseteq_2) \cap Aut(=, d) \subseteq Aut(adom(d), \sqsubseteq_2) \cap Aut(=, r)$. Hence, we have $Aut(\sqsubseteq_2, d) \subseteq Aut(\sqsubseteq_2, r)$. By Theorem 1 again, we have $r \in Gen(\sqsubseteq_2, d)$. □

The following corollary states that there is a correspondence between the set of

meaningful computable queries and the partially ordered relational algebra. Informally, the relational algebra $PORA_i$ non-uniformly expresses the set of queries Q_i, i.e., the expression $e \in E_{PORA_i}$ depends on the database instance as well as the query $\delta \in Q_i$. Therefore, in this sense we can say that the language $PORA_i$ is *non-uniformly complete*.

COROLLARY 2. *$Q_1 \subseteq Q_2$ if and only if $PORA_1 \preceq PORA_2$.*

We present the diagram in Figure 4, which summarises the relationship between the hierarchies of (1) meaningful computable queries, (2) partially ordered domains, and (3) partially ordered relational algebras. The implications of this result are that if the underlying data domains of an ordered database have more inherent structure, then a wider scope of queries is possible. In other words, the ordered relational model can provide more expressive query languages than those of the conventional one, and in this sense we can say that more meaningful queries are possible with respect to an ordered relational database. We also remark that the main results in Theorems 1 and 2 can be extended to the situation where several partial ordering relations are defined on D. However, under this circumstance we need to generalise the notion of an order-preserving database automorphism as follows: let $\Omega_i = \{\sqsubseteq_1, \ldots, \sqsubseteq_{n_i}\}$ be a given set of orderings on D, where n_i is a positive integer, then $Aut(\Omega_i, d)$ is defined as $Aut(\sqsubseteq_1, d) \cap \cdots \cap Aut(\sqsubseteq_{n_i}, d)$.

Queries	$Q_= \subseteq$	$\cdots$	$\subseteq Q_i \subseteq$	$\cdots$	$\subseteq Q_\leq$
	$\updownarrow$		$\updownarrow$		$\updownarrow$
Domains	$\langle D, = \rangle \preceq$	$\cdots$	$\preceq \langle D, \sqsubseteq_i \rangle \preceq$	$\cdots$	$\preceq \langle D, \leq \rangle$
	$\updownarrow$		$\updownarrow$		$\updownarrow$
Algebras	$PORA_= \preceq$	$\cdots$	$\preceq PORA_i \preceq$	$\cdots$	$\preceq PORA_\leq$

Fig. 4. A correspondence between hierarchies of queries, domains and languages

3.4 Query Language: OSQL

Ordered SQL (OSQL) is an extension of SQL for the ordered relational model. There are three main features in OSQL. First, we implement a new built-in predicate called *TUPLE* which uses a *tuple level set* as the parameter. A *tuple level set* is basically the collection of different levels of an *internal hierarchy* of a relation. Second, the *WHERE* clause is extended to implement the ordering predicate in the PORA and thus it allows us to compare attributes according to semantic orderings. Finally, the *ORDER BY* clause is also extended as follows: the tuples in an output relation generated by an OSQL expression can be ordered according to semantic orderings, in addition to the usual system orderings.

We now discuss the underlying idea of the extension of an internal hierarchy, which can be viewed as a generalisation of the position of a tuple in a linearly ordered relation. We denote by $part(r)$ a *partition* of a relation r, which is a set of pairwise disjoint non-empty subsets of r such that $\bigcup_{T \in part(r)} T = r$, and call an

element $T \in part(r)$ a *tuple level* of r. An *internal hierarchy* of r is essentially a linearly ordered partition induced by $\sqsubseteq_r^l$.

Definition 15. (*Internal Hierarchy of a Relation*) An *internal hierarchy* of a relation r is a linearly ordered set $\langle part(r), \leq \rangle$, such that

(1) $\forall T \in part(r)$, $\forall t_1, t_2 \in T$, either $t_1 = t_2$ or $t_1 \parallel t_2$ (i.e., T is unordered).
(2) $\forall T_i, T_j \in part(r)$, $T_i < T_j \Rightarrow \forall t_1 \in T_i, \forall t_2 \in T_j$, $t_2 \not\sqsubseteq_r^l t_1$.
(3) $\forall T_i, T_j \in part(r)$, $T_i < T_j \Rightarrow \exists t_1 \in T_i, \exists t_2 \in T_j$ such that $t_1 \sqsubset_r^l t_2$.

A tuple $u \in s$ is said to be *minimal*, where $s \subseteq r$, if, for any $t \in s$, $t \sqsubseteq_r^l u$ implies that $t = u$. We remark that s may have more than one minimal tuple. In one special case of linearly ordered relations, s has a unique minimal tuple. In another special case of an unordered relation, all tuples in s are minimal.

Example 5. Consider a unary relation $r = \{a, b, c, d, e\}$ (5 tuples), where $a \sqsubseteq_r c$, $b \sqsubseteq_r c$, $c \sqsubseteq_r e$ and $d \sqsubseteq_r e$. We now show two possible internal hierarchies $part(r)$ $= \{T_1, T_2, T_3\}$ given in Figure 5, in which a tuple is represented by a node.

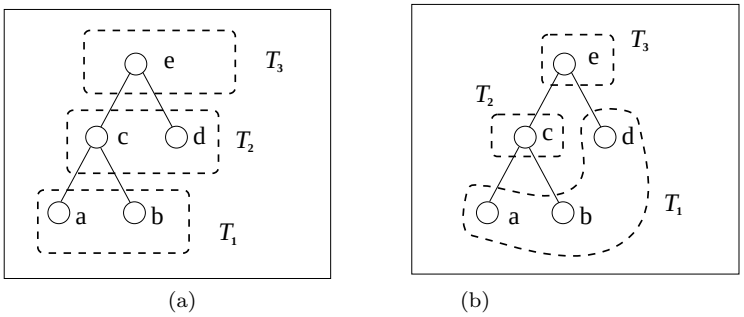

Fig. 5. Two possible internal hierarchies for a relation r

The following lemma shows that by successively collecting the sets of minimal tuples in the subsets of a relation, we can construct an internal hierarchy as illustrated in Figure 5(b). We remark that this method of constructing an internal hierarchy is essentially a matter of convention and is one of the simplest choices. Later, in Lemma 9 we will further show that our choice has the desirability that each level can be obtained by PORA expressions. It is interesting to note that the internal hierarchy constructed by collecting sets of *maximal* tuples also gives rise to the same number of tuple levels as with the method used in Lemma 7 (i.e. Figure 5(a) is generated); in this case we then need to generalise the parameter r_i in order to handle the disconnected components arising from Algorithm 1. We also observe that, in general, the total number of tuple levels in an internal hierarchy of relation r is greater than or equal to the number of nodes in the longest *chain* of the Hasse Diagram corresponding to r.

LEMMA 7. *Every relation contains an internal hierarchy.*

PROOF. We let r be a given relation and use the following algorithm to generate a partition.

ALGORITHM 1.

1. **begin**
2. $r_0 = r$ *and* $T_0 = \emptyset$*;*
3. **do until** $r_{i-1} = \emptyset$
4. T_i *is the set of minimal tuples of* $r_i = r_{i-1} - T_{i-1}$*;*
5. **return** *Result* $= \{T_1, \ldots, T_l\}$*;*
6. **end**.

It is trivial that Algorithm 1 will terminate for a finite relation r. Let the last tuple level generated by the algorithm be T_l and $\{T_1 < T_2 < \cdots < T_l\}$ be a collection of subsets obtained by the above algorithm, where the linear ordering on this partition is according to the order of generation of T_i in the steps 3 and 4. It is easy to see that it is a partition of r and that for all $t_1, t_2 \in T_i$, if t_1 and t_2 are distinct, then we have $t_1 \parallel t_2$, since they are both the minimal tuples of r_i. Thus, it satisfies part (1) in Definition 15. Assume to the contrary that $\exists t_1 \in T_i, \exists t_2 \in T_j$ such that $t_2 \sqsubseteq_r^l t_1$ and $T_i < T_j$. Then it follows that $t_2 = t_1$, since t_1 is a minimal tuple and is less than t_2. However, this is impossible because T_i and T_j are disjoint. Hence part (2) is also satisfied. Finally, part (3) can be established by noting that T_i is the set of all minimal elements of some superset of T_j. It follows that for any element $t_2 \in T_j$, there is an element $t_1 \in T_i$ such that $t_1 \sqsubset_r^l t_2$. □

The next lemma is immediately followed by the definition of Algorithm 1.

LEMMA 8. *The internal hierarchy generated by Algorithm 1 is unique.*

PROOF. This can be easily established by using induction on T_i and the fact that T_i is the unique set of all minimal tuples of r_i. □

The following lemma shows that the tuple levels of the internal hierarchy generated by Algorithm 1 can be expressed by the PORA for a relation r. We use the notation $\sigma_{X\tilde{\sqsubset}Y}$ to represent the PORA expression that compares the projections of a tuple onto X and Y according to the lexicographical ordering.

LEMMA 9. *Any tuple level of the internal hierarchy generated by Algorithm 1 can be expressed by the PORA.*

PROOF. Let $\sigma_{X\tilde{\sqsubset}Y}(r)$ be a shorthand notation to represent the PORA expression $\sigma_{A_1 \sqsubset B_1}(r) \cup (\sigma_{A_1=B_1}(\sigma_{A_2 \sqsubset B_2}(r))) \cup \cdots \cup (\sigma_{A_1=B_1} \cdots (\sigma_{A_{n-1}=B_{n-1}}(\sigma_{A_n \sqsubset B_n}(r))) \cdots)$, where $X = \langle A_1, \ldots, A_n \rangle$ and $Y = \langle B_1, \ldots, B_n \rangle$. We can generate T_i, where $1 \le i \le n$, recursively as follows.

$i = 1$: $T_1 = s \cup (r - w)$, where
$s = \rho_{R_1 \to R}(\pi_{R_1}(\sigma_{R_1 \tilde{\sqsubset} R_2}(r \times r))) - \rho_{R_2 \to R}(\pi_{R_2}(\sigma_{R_1 \tilde{\sqsubset} R_2}(r \times r)))$, and
$w = \rho_{R_1 \to R}(\pi_{R_1}(\sigma_{R_1 \tilde{\sqsubset} R_2}(r \times r))) \cup \rho_{R_2 \to R}(\pi_{R_2}(\sigma_{R_1 \tilde{\sqsubset} R_2}(r \times r)))$.

$i > 1$: $T_i = s \cup (r_i - w)$, where $r_i = (\cdots((r - T_1) - T_2) \cdots - T_{i-1})$,
$s = \rho_{R_1 \to R}(\pi_{R_1}(\sigma_{R_1 \tilde{\sqsubset} R_2}(r_i \times r_i))) - \rho_{R_2 \to R}(\pi_{R_2}(\sigma_{R_1 \tilde{\sqsubset} R_2}(r_i \times r_i)))$, and
$w = \rho_{R_1 \to R}(\pi_{R_1}(\sigma_{R_1 \tilde{\sqsubset} R_2}(r_i \times r_i))) \cup \rho_{R_2 \to R}(\pi_{R_2}(\sigma_{R_1 \tilde{\sqsubset} R_2}(r_i \times r_i)))$. □

Lemmas 7, 8 and 9 have practical significance as they indicate that a unique internal hierarchy can be generated by collecting the minimal tuples of a relation (or its subset) and, in addition, using the PORA we can express a tuple level of

such a hierarchy for a given relation. The concept of a tuple level is very natural and easy to understand. In the special case of linearly ordered relations, T_i is the singleton containing the ith tuple. Thus, our choice of the *SELECT* statement in OSQL to include the *TUPLE* predicate can be justified by the formalism of an internal hierarchy.

We now describe the extensions of Ordered SQL (OSQL) to the Data Manipulation Language (DML) and the Data Definition Language (DDL): the *SELECT* statement of the DML and the *CREATE* statement of the DDL; the full reference of the syntax of OSQL in Backus-Naur Form (BNF) can be consulted in Appendix A.

1. The DML of OSQL

SELECT ⟨ lists of attributes ⟩
FROM ⟨ lists of ordered relations ⟩
WHERE {⟨ comparison expressions ⟩ | *TUPLE* ⟨ sets of tuple levels ⟩}
ORDER BY ⟨ lists of attributes ⟩

An *attribute list* above is a list of attributes similar to the usual one, such as using the symbol "$*$" to represent the projection on all attributes, when used after the *SELECT* keyword. However, an attribute can be associated with a semantic ordering by using the syntax *attribute name WITHIN order name* in the *comparison expression* or in the *ORDER BY clause*. The purpose of declaring a *WITHIN* clause is to override the system ordering with the semantic ordering specified by a *domain order name*. When the *WITHIN* clause is missing, then the system ordering will be assumed and in this case OSQL is just equivalent to conventional SQL.

The *TUPLE* predicate, an optional condition following the *WHERE* keyword, is used with the parameter *tuple levels*. Tuple levels are represented as a set of positive numbers, with the usual numerical ordering, which can be written in some short forms (see Appendix A2). As a set of tuples in a linearly ordered relation $r = \{t_1, \ldots, t_n\}$ is isomorphic to a set of linearly ordered tuples, we interpret each number i in a tuple level set as an index to the position of the (only) tuple t_i, where $i = 1, \ldots, n$ and $t_1 < \cdots < t_n$. An interesting situation to consider is when the output of a relation is partially ordered as a tree, having levels $\{l_1, \ldots, l_m\}$. In such a case we choose to interpret each number j in a tuple level set as an index to a corresponding tree level l_j, where $j = 1, \ldots, m$ and $l_1 < \cdots < l_m$. Hence, using the predicate $TUPLE(j)$ we can retrieve (all) the tuples in a specified level l_j. We remark that by choosing the optional keyword *DESC* the tuple order can be reversed and thus the usual $TUPLE(LAST - j)$ is equivalent to $TUPLE(j + 1)$ in the reversed case.

Example 6. We use the post hierarchy of an organisation as shown in Figure 6 to clarify the semantics of the *TUPLE* predicate. We can see from this figure that if a user specifies the first tuple level by $TUPLE(1)$ following the *WHERE* keywords, the system returns 'VPD' and 'VPM'. If a user specifies $TUPLE(LAST)$ or $TUPLE(2)$ instead, the system just returns 'CEO'. Note that without the *TUPLE* predicate in the *WHERE* clause the system simply returns all the post names.

A *comparison expression* following the *WHERE* keyword has the usual compara-

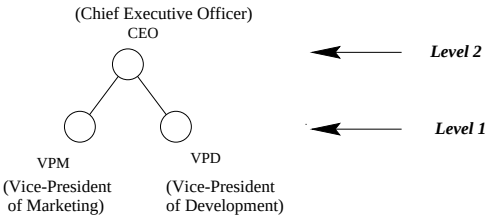

Fig. 6. Relationship between post ranks in an organisation

tors $<, >, <=, >=$ whose meaning is extended to include semantic comparison as we have mentioned earlier. (Recall also that incomparsion can be simulated by these comparators.) A typical form of a semantic comparison expression is

$$\langle \text{ attribute } \rangle \ \langle \text{ comparator } \rangle \ \langle \text{ attribute } \rangle \ \textit{WITHIN} \ \langle \text{ semantic order } \rangle.$$

Without the optional *WITHIN* clause, the comparison is just the conventional one and is based on the relevant system ordering.

Example 7. Let us examine the following OSQL statements:

(Q_3) *SELECT* POST, AGE *FROM* STAFF.
(Q_4) *SELECT* AGE, POST *FROM* STAFF.
(Q_5) *SELECT* POST, AGE *FROM* STAFF
ORDER BY (POST *WITHIN* RANK_ORDER).

Note that when the *ORDER BY* clause is not used in an OSQL expression, the ordering of tuples in an output relation then depends on two factors: first, on the ordering of domains of individual attributes, and second, on the order of the attributes in an attribute list. The attribute list of the query (Q_3) is (POST, AGE), and thus tuples in the output answer are ordered by POST first and only then by AGE (see Figure 7(a)). Therefore the ordering of tuples is, in general, different to that of query (Q_4), whose list is specified as (AGE, POST), since the output of (Q_4) is ordered by AGE first, and then by POST (see Figure 7(b)). It will also be different from that of (Q_5) which uses the *ORDER BY* clause with the attribute (POST *WITHIN* RANK_ORDER) (see Figure 7(c)), where the semantic ordering of POST is given by RANK_ORDER as shown in Figure 6. Note also that 'VPD' and 'VPM' are incomparable according to RANK_ORDER and thus they are alphabetically ordered in the output result.

POST	AGE
CEO	48
VPD	48
VPM	47

(a)

AGE	POST
47	VPM
48	CEO
48	VPD

(b)

POST	AGE
VPD	48
VPM	47
CEO	48

(c)

Fig. 7. An employee relation STAFF with different orderings

2. The DDL of OSQL

CREATE DOMAIN ORDER ⟨ domain order names ⟩ ⟨ data types ⟩
AS ⟨ ordering specifications ⟩.

The statement declares a semantic ordering by specifying its *domain order name* and *data type*. Following the *AS* keywords is a specification of the ordering of a semantic domain. The basic syntax of the *ordering-specification* is: (⟨ data-pair ⟩, ⟨ data-pair ⟩, ...) where *data-pair* is of the form, *data-item* B < *data-item* A, if and only if *data-item* A is greater than *data-item* B in the semantic domain.

Example 8. The definition of the semantic ordering on the domain shown in Figure 2(a) can be written as follows:

(Q_6) *CREATE DOMAIN ORDER* RANK_ORDER *CHAR*(3)
AS ('VPM'<'CEO', 'VPD'<'CEO').

For a large and complex semantic ordering, this syntax may be tedious. Thus OSQL provides two useful short forms to make the task of specifying semantic orderings easier. First, we allow the use of set notation, {}, to represent a set of data items with common predecessor (or successor). For instance, the statement (Q_6) can be rewritten as follows:

(Q_7) *CREATE DOMAIN ORDER* RANK_ORDER *CHAR*(3)
AS ({'VPD','VPM'}<'CEO').

Second, we allow the use of the keyword *OTHER* for those data items not mentioned explicitly, with two options *OTHER SYO* and *OTHER UNO*, meaning that those data values not mentioned are treated as *SYstem Ordered* or *UNOrdered*. Note that by default we assume other data items are unordered unless there is an explicit declaration that orders these items.

We now introduce another *CREATE* statement used for declaring a *tuple order* on a set of tuples, which is an extension of the corresponding list of domain orders.

CREATE TUPLE ORDER ⟨ tuple order names ⟩ ⟨ lists of data types ⟩
AS ⟨ lists of domain order names ⟩.

The first part of the statement is similar to the *CREATE DOMAIN ORDER* statement except that it needs to specify a *list of data types* and a *list of domain order names* in defining a tuple order. Note that a *tuple order* is a lexicographical ordering on the Cartesian product of the existing domains. This statement affords us the ability to create a semantic ordering on tuples, based on established semantic orderings. When a tuple order is stated after a relation name in the *FROM* clause, the *TUPLE* predicate then returns the tuples of the specified levels for the relation, according to the tuple order. This should not be confused with the usage of the *ORDER BY* clause, which is only used to control the ordering of an output relation for a query.

Example 9. Let us first define a tuple order, which is formed by the semantic ordering RANK_ORDER, as shown in (Q_6), and the system ordering *SYO*, for the relation STAFF in Figure 7, as follows.

(Q_8) *CREATE TUPLE ORDER* STAFF_ORDER (*CHAR*(3), *NUM*(3)) *AS* (RANK_ORDER, *SYO*).

We compare the following OSQL statements, which refer to the post hierarchy in Figure 7, in order to illustrate one usage of a tuple order:

(Q_9) *SELECT* POST, AGE *FROM* STAFF
WHERE TUPLE(1).

(Q_{10}) *SELECT* POST, AGE *FROM* STAFF *WITHIN* STAFF_ORDER
WHERE TUPLE(1).

The statement (Q_9) returns only $\langle 'CEO', 48\rangle$ according to the usual system ordering on tuples, whereas the statement (Q_{10}) returns $\langle 'VPD', 48\rangle$ and $\langle 'VPM', 47\rangle$ according to the semantic ordering specified by STAFF_ORDER on the relation STAFF.

There is still another important usage of a tuple order in a comparison expression following the *WHERE* keyword. It allows more than one attribute to be used on the two sides of a comparator and then compares the attributes according to the semantic ordering specified by a given tuple order. We will illustrate the use of this powerful feature when discussing the use of OSQL in temporal information (see Example 10 in Section 3.5).

3.5 Using OSQL in Advanced Applications

We now show by the following running example how OSQL can be applied to solve various problems that arise in relational DBMSs involving the applications having tree-structured information [Biskup 1990], temporal information [Tansel et al. 1993], and incomplete information [Codd 1986] under the unifying framework of the ordered relational model.

Example 10. Let us consider the relation EMP_DETAIL shown in Figure 8. The semantics of the attributes are self-explanatory or will be further clarified whenever necessary.

NAME	POST	SALARY	PRE_WORK	MONTH	YEAR
Ethan	VPM	42K	UNK	Dec	1994
Mark	CEO	40K	NI	Sep	1990
Mark	CEO	48K	NI	Sep	1996
Nadav	VPD	45K	Programmer	Jan	1995

Fig. 8. An employee relation EMP_DETAIL

—Tree-structured Information:
Suppose that we have the domain order RANK_ORDER being declared by the statement (Q_6) (or equivalently (Q_7)) to describe the hierarchy of the employees in EMP_DETAIL. We can formulate the query for finding the employee information, ordered by the post ranks, as follows:
(Q_{11}) *SELECT* $*$ *FROM* EMP_DETAIL
ORDER BY (POST *WITHIN* RANK_ORDER).

We can also formulate a more complex query for finding the name of the common bosses of Nadav and Ethan as follows:

(Q_{12}) *SELECT* E1.NAME *FROM* EMP_DETAIL E1, EMP_DETAIL E2
WHERE (E1.POST > E2.POST *WITHIN* RANK_ORDER)
AND (E2.NAME = 'Ethan' *OR* E2.NAME = 'Nadav').

We remark that the keyword *WITHIN* specifies that the comparison expression E1.POST > E2.POST is interpreted according to RANK_ORDER.

—Temporal Information:

We assume that MONTH and YEAR are time attributes whose values are timestamps of the tuples in the relation EMP_DETAIL (for simplicity in presentation, we assume that the timestamping denotes *valid time* [Tansel et al. 1993]). We need the tuple order YEAR_MONTH to capture the semantics of the time order of year-month, which is defined by the following two statements ((Q_{13}) and (Q_{14})):

(Q_{13}) *CREATE DOMAIN ORDER* MONTH_ORDER *CHAR*(3) *AS*
($'Jan' < 'Feb'$, $'Feb' < 'Mar'$, $'Mar' < 'Apr'$, $'Apr' < 'May'$, $'May' < 'Jun'$, $'Jun' < 'Jul'$, $'Jul' < 'Aug'$, $'Aug' < 'Sep'$, $'Sep' < 'Oct'$, $'Oct' < 'Nov'$, $'Nov' < 'Dec'$).

(Q_{14}) *CREATE TUPLE ORDER* YEAR_MONTH
(*NUM*(4), *CHAR*(3)) *AS* (*SYO*, MONTH_ORDER).

For instance, we can see that Mark had a salary of 40K in Sep-1990 and his salary increased in Sep-1996. Note that we do not record Mark's salary if there had been no change since the month it was last updated. We can use the keyword *LAST* as a parameter in the *TUPLE* predicate to find the last time the tuple was updated, since the tuple order YEAR_MONTH for the projection over the attribute list (YEAR, MONTH) is a linear ordering. With the following query, we show how to find Mark's salary in Dec-1993 as follows:

(Q_{15}) *SELECT* YEAR, MONTH, SALARY *FROM* EMP_DETAIL
WHERE NAME = 'Mark' *AND TUPLE*(*LAST*)
AND ((YEAR, MONTH) <= (1993, 'Dec') *WITHIN* YEAR_MONTH).

—Incomplete Information:

Suppose we have the domain order INCOMPLETE as in Figure 2(c) to capture the semantics of different null values, which is defined for the attribute PRE_WORK, meaning the PREvious WORKing experience of an employee, as follows:

(Q_{16}) *CREATE DOMAIN ORDER* INCOMPLETE *CHAR*(10) *AS*
('NI'<'DNE','NI'<'UNK'< *OTHER*).

We can now formulate the query which finds the NAME and PRE_WORK of those employees whose previous work is more informative than NI, as follows:

(Q_{17}) *SELECT* NAME, PRE_WORK *FROM* EMP_DETAIL
WHERE (PRE_WORK > 'NI' *WITHIN* INCOMPLETE).

The OSQL statements in (Q_{13}) to (Q_{17}) reveal the potential of using OSQL to support the above-mentioned three advanced applications under the unifying framework of the ordered relation model. We emphasise that our design of OSQL adheres to the principle of upward compatibility, in the sense that the syntax of OSQL is equivalent to that of standard SQL when those OSQL facilities related to

semantic orderings are not used in formulating expressions. The overall change to OSQL is kept at a minimal level and thus the current users will be able to adapt the language in a short time. We summarise the advantages of OSQL and compare them against the features of conventional SQL in the table given in Figure 9.

Conventional SQL	**OSQL**
No semantic comparison is possible in the *WHERE* clause. Comparison is limited to only a few kinds of standard system orderings provided by a DBMS.	Semantic comparison is allowed to be written as a conditional expression in the *WHERE* clause in a statement. It is easy to compare attributes according to the semantic orderings defined by users.
No simple and general way of obtaining the *nth* tuple in a relation, where n is a natural number.	The *nth* tuple can be conveniently retrieved by the built-in *TUPLE* predicate, based on the formal notion of an internal hierarchy.
The *ORDER BY* clause uses only system orderings.	The *ORDER BY* clause can use system orderings or semantic orderings.
Only pointwise-ordering can be defined when comparison involves more than one attribute. This is achieved by using the logical connective *AND* between simple comparison expressions in the *WHERE* clause.	By using the *CREATE TUPLE ORDER* statement we could first define a lexicographical ordering on a set of tuples and then use it as a conditional expression to compare attribute lists in the *WHERE* clause.

Fig. 9. Comparison between conventional SQL and OSQL

4. FUNCTIONAL DEPENDENCIES FOR THE ORDERED RELATIONAL MODEL

In this section we formalise the notions of FDs being satisfied in ordered relations and call them OFDs. OFDs are classified according to whether we use pointwise-orderings (POFDs) or lexicographical orderings (LOFDs) in their definitions. We show that the axiom system comprising the inference rules for POFDs, which is a superset of Armstrong's axiom system for FDs, is sound and complete. We extend the chase rules for the case of LOFDs and the notion of *tableaux* for LOFDs, and show that the chase is sound and complete for LOFDs.

Throughout this section we refer to a sequence of attributes as a shorthand for a sequence of distinct attributes and use the common notation for both sequences and sets when no ambiguity arises. We denote the fact that two sequences have the same elements by $X \sim Y$. The difference between two sequences of attributes, denoted as $X - Y$, is defined by the sequence resulting from removing all the common attributes in X and Y from X while maintaining the original order of the remaining attributes in X. We also denote by XY the *concatenation* of two sequences X and Y, if X and Y are *disjoint*, otherwise XY is defined by $X(Y - X)$.

4.1 Ordered Functional Dependencies (OFDs)

Bearing in mind that the implication problem is an important issue arising in developing the theory of data dependencies, we first formalise the notions of *logical implication* and *an axiom system*.

Definition 16. (*Logical Implication and Axiom System*) A set of data dependencies F *logically implies* a data dependency f over R, written F $\models f$, whenever for all relations r over R, if, for all $f' \in$ F, $r \models f'$ holds, then $r \models f$ also holds. An *axiom system* $\mathcal{A}$ for F is a set of inference rules (or simply rules) that can be used to *derive* data dependencies from F over R. We say that f is *derivable* from F by $\mathcal{A}$, if there is a finite sequence of data dependencies over R, whose last element is f, and where each data dependency in the sequence is either in F or follows from a finite number of previous data dependencies in the sequence by one of the inference rules. We denote by F $\vdash f$ the fact that f is *derivable* from F by a specified axiom system.

Definition 16 will be used in different contexts of OFDs: when discussing POFDs in Section 4.2, we will use F $\models f$ to mean that a set of POFDs F logically implies a POFD f, and when discussing LOFDs in Section 4.3, we will also use F $\models f$ to mean that a set of LOFDs F logically implies an LOFD f.

We assume that readers have knowledge of Armstrong's axiom system for FDs [Armstrong 1974; Ullman 1988], which provides a set of inference rules that can infer new FDs from given ones. It is well-known that Armstrong's axiom system is sound and complete for FDs being satisfied in conventional relations. The semantics of FDs in the context of ordered databases is straightforward, that is, an FD in conventional relational databases can be viewed as a special case of an OFD when a database is unordered. Armstrong's system can be directly carried over to ordered relations, since we assume that the equality predicate still applies to ordered domains.

The semantics of an OFD with two or more attributes on either the left or right hand side is defined according to pointwise-orderings and lexicographical orderings on the Cartesian product of the underlying domains of the attributes in the OFD, which gives rise to POFDs and LOFDs, whose short forms are written as POFDs ($X \hookrightarrow Y$) and LOFDs ($X \rightsquigarrow Y$), respectively.

To illustrate the usage of OFDs, we show in Figure 10 a relation called SALARY_RECORD over the set of attributes {NAME, POST, YEARS, SALARY}. The semantics of SALARY_RECORD are: an employee with a NAME and a given POST, who has been working in a company for some YEARS, has the present SALARY.

NAME	POST	YEARS	SALARY
Mark	Senior Programmer	15	35K
Nadav	Junior Programmer	7	25K
Ethan	Junior Programmer	6	22K

Fig. 10. An employee relation SALARY_RECORD

We assume that there is a semantic ordering in POST as represented by the following domain {'Junior Programmer' < 'Senior Programmer'}. The relation SALARY_RECORD given in Figure 10 then satisfies the POFD, {POST, YEARS} $\hookrightarrow$ SALARY, which states that the SALARY of an employee is greater than that of other employees with junior titles and less experience in the company, and the

LOFD, {POST, YEARS} $\rightsquigarrow$ SALARY, which states that SALARY of an employee is greater than that of other employees with junior titles, or with the same title but less experience in the company. Note that the semantics of the POFD and the LOFD mentioned above are different. For instance, in the first case, an employee has a higher salary only if he or she has both a senior post and more experience than another, whereas in the second case, he or she has to have a more senior post than another. If Mark leaves his post, Ethan replaces him and his record is updated to $\langle Ethan, SeniorProgrammer, 6, 26K\rangle$ (i.e., updating the third tuple), then this updating violates neither the POFD nor the LOFD. However, if his record is updated to $\langle Ethan, SeniorProgrammer, 6, 24K\rangle$, then it violates the LOFD, since Ethan now has a more senior title but a lower salary than Nadav. But the POFD still holds in this updating, since Nadav still has more experience than Ethan. The appropriateness of the choice of the POFD or the LOFD in this case depends entirely on the semantics of the promotion policy adopted by the company.

We observe that the notions of POFDs and LOFDs are incomparable. A relation satisfying the POFD $X \hookrightarrow Y$ may not necessarily satisfy the LOFD $X \rightsquigarrow Y$ and conversely, a relation satisfying the LOFD $X \rightsquigarrow Y$ may not necessarily satisfy the POFD $X \hookrightarrow Y$. The following example helps to illustrate this point.

Example 11. Consider the relations r_1 and r_2 over $R = \{A, B, C\}$ shown in Figure 11. It is trivial that in (a) $r_1 \models A \rightsquigarrow BC$ but $r_1 \not\models A \hookrightarrow BC$. On the other hand, in (b) $r_2 \models AB \hookrightarrow C$ but $r_2 \not\models AB \rightsquigarrow C$.

$r_1 =$

A	B	C
1	3	6
2	4	5

(a)

$r_2 =$

A	B	C
1	4	6
2	3	5

(b)

Fig. 11. Relations r_1 and r_2 showing that POFDs and LOFDs are incomparable.

4.2 OFDs Arising from Pointwise-Orderings

We give the definition of a POFD as follows:

Definition 17. (*Ordered Functional Dependency Arising from Pointwise-Orderings*) An *ordered functional dependency arising from pointwise-orderings* (or simply a POFD) over a relation schema R, is a statement of the form $X \hookrightarrow Y$, where $X, Y \subseteq R$ are sequences of attributes. A POFD, $X \hookrightarrow Y$, is *satisfied* in a relation r over R, denoted by $r \models X \hookrightarrow Y$, if, for all $t_1, t_2 \in r$, $t_1[X] \sqsubseteq^p_X t_2[X]$ implies that $t_1[Y] \sqsubseteq^p_Y t_2[Y]$. The POFD $X \hookrightarrow Y$ is said to be *standard* if $X \neq \emptyset$.

Hereinafter we will assume that all POFDs are standard. We next give a set of inference rules for POFDs, and show that Armstrong's axiom system carries over to ordered relations with respect to POFDs.

Definition 18. (*Inference Rules for POFDs*) Let X, Y, Z, W be subsets of R, and F be a set of POFDs over R. The *inference rules for POFDs* are defined as follows:

(POFD1) *Reflexivity*: if $Y \subseteq X$, then $\mathrm{F} \vdash X \hookrightarrow Y$.
(POFD2) *Augmentation*: if $\mathrm{F} \vdash X \hookrightarrow Y$ and $Z \subseteq R$, then $\mathrm{F} \vdash XZ \hookrightarrow YZ$.

(POFD3) *Transitivity*: if $\mathrm{F} \vdash X \hookrightarrow Y$ and $\mathrm{F} \vdash Y \hookrightarrow Z$, then $\mathrm{F} \vdash X \hookrightarrow Z$.
(POFD4) *Permutation*: if $\mathrm{F} \vdash X \hookrightarrow Y$, $W \sim X$ and $Z \sim Y$, then $\mathrm{F} \vdash W \hookrightarrow Z$.

We remark that POFD4 is needed because we are dealing with sequences of attributes rather than the usual sets of attributes in FDs. The following lemma can be readily proved by induction on the number of steps in the inference of $X \hookrightarrow Y$ from a set of POFDs.

LEMMA 10. *Let F be a set of POFDs, $f = X \hookrightarrow Y$ be a POFD and $f^* = X \rightarrow Y$ be an FD corresponding to f. We define $F^* = \{f^* \mid f \in F\}$. Then f^* is derivable from F^* using Armstrong's axiom if and only if $F \vdash f$.*

The above lemma is useful because it suggests that we can apply existing algorithms for FDs to determine whether a POFD f can be inferred from a given set of POFDs using the inference rules from POFD1 to POFD4. For example, Beeri and Bernstein's algorithm [Beeri and Bernstein 1979] can be used to compute the closure of a set of attributes with respect to a set of POFDs. We now state the theorem that the axiom system in Definition 18 is sound and complete for POFDs, holding in ordered relations. The underlying idea in this proof is standard and similar to Theorem 7.1 in [Ullman 1988]. We also need to assume the unique name axiom (c.f. Section 9.3 in [Atzeni and De Antonellis 1993)]) and that each domain has at least two distinct and comparable elements, say, $0 < 1$.

THEOREM 3. *The axiom system comprising POFD1 to POFD4 is sound and complete for POFDs.*

PROOF. We define the notion of the closure of a set of attributes X^+ in the context of POFDs, which is given by $X^+ = \{A \mid \mathrm{F} \vdash \mathrm{X} \hookrightarrow \mathrm{A}\}$. It is straightforward to show that the inference rules from POFD1 to POFD4 are sound. The completeness can be proved by exhibiting a counter-example relation to show that if $\mathrm{F} \nvdash X \hookrightarrow Y$, then $\mathrm{F} \not\models X \hookrightarrow Y$. □

4.3 OFDs Arising from Lexicographical Orderings

We give the definition of an LOFD as follows:

Definition 19. (*Ordered Functional Dependency Arising from Lexicographical Orderings*) An *ordered functional dependency arising from lexicographical orderings* (or simply an LOFD) over a relation schema R, is a statement of the form $X \rightsquigarrow Y$, $X, Y \subseteq R$ are sequences of attributes. An LOFD, $X \rightsquigarrow Y$, is *satisfied* in a relation r over R, denoted by $r \models X \rightsquigarrow Y$, if, for all $t_1, t_2 \in r$, $t_1[X] \sqsubseteq^l_X t_2[X]$ implies that $t_1[Y] \sqsubseteq^l_Y t_2[Y]$.

Similar to POFDs, we assume that all LOFDs are standard.

The chase is a fundamental theorem proving tool in relational database theory. The main uses of the chase have been to test implications of data dependencies [Maier et al. 1979] and to test the consistency of a relational database with respect to a set of data dependencies [Grahne 1984; Levene and Loizou 1996]. We now extend the classical chase defined over conventional relations with respect to FDs [Maier et al. 1979; Atzeni and De Antonellis 1993)] to ordered relations with respect to LOFDs. The extended chase will be used as a sound and complete inference tool

for LOFDs in Theorem 4. We need the notion of *linear extension*, and the *equate* and *swap* operations to manipulate values in ordered domains before presenting our chase rules. A linear extension of a partial ordering $\sqsubseteq_D$ is defined as a linear ordering $\leq_D$ satisfying the condition that, for all $a, b \in D$, if $a \sqsubseteq_D b$, then $a \leq_D b$. It is clear that if $\sqsubseteq_D$ is a linear ordering, then its linear extension is unique and just equal to itself. We now assume in the following definition that there is a fixed linear extension being imposed on the involved domain.

Definition 20. (*Equate and Swap Operations*) We denote $min(a, b)$ and $max(a, b)$ the minimum and the maximum of the values a and b in a domain according to the linear extension of the domain ordering. For any two distinct tuples $t_1, t_2 \in r$ over R and some $A \in R$, the *equate* of t_1 and t_2 on A, denoted as *equate*($t_1[A]$, $t_2[A]$), is defined by replacing both $t_1[A]$ and $t_2[A]$ by $min(t_1[A], t_2[A])$; the *swap* of t_1 and t_2 on A, denoted as *swap*($t_1[A]$, $t_2[A]$), is defined by replacing $t_1[A]$ by $min(t_1[A], t_2[A])$ and $t_2[A]$ by $max(t_1[A], t_2[A])$, respectively.

Note that in the above definition the imposed linear extension guarantees that the equate and swap operations are applicable to all elements in a given ordered domain. From now on, we assume the usual numerical ordering for integers unless explicitly stated otherwise. We demonstrate how to use the equate and swap operations with the following example:

Example 12. Consider a relation r shown in Figure 12(a), which consists of two tuples $t_1 = \langle 2 \rangle$ and $t_2 = \langle 1 \rangle$, respectively. We apply the equate operation of t_1 and t_2 on A, resulting in the relation shown in Figure 12(b). We apply the swap operation of t_1 and t_2 on A, resulting in the relation shown in Figure 12(c). Note that, in this example, if the data elements 1 and 2 are incomparable but their imposed linear extension is $1 \leq 2$, the equate and swap operations are still applicable, leading to the same result.

	A
t_1	2
t_2	1

(a) $r = \{t_1, t_2\}$

	A
t_1	1
t_2	1

(b) $equate(t_1[A], t_2[A])$

	A
t_1	1
t_2	2

(c) $swap(t_1[A], t_2[A])$

Fig. 12. An example of using the equate and swap operations

We now give the chase rules, which are applied to two tuples in a relation with respect to a set of LOFDs.

Definition 21. (*Chase Rules for LOFDs*) Let t_1 and t_2 be two tuples in r such that $t_1[X] \sqsubseteq^l_X t_2[X]$ but $t_1[Y] \not\sqsubseteq^l_Y t_2[Y]$, A be the first attribute in X such that $t_1[A] \neq t_2[A]$, if such an attribute exists, and B be the first attribute in Y such that $t_1[B] \neq t_2[B]$, then the *chase rules* for the LOFD $X \rightsquigarrow Y$, are defined by the following two rules:

Equate rule: if $t_1[X] = t_2[X]$ but $t_1[B] \neq t_2[B]$, or if $t_1[A] \not\parallel t_2[A]$ but $t_1[B] \parallel t_2[B]$, then *equate*($t_1[B], t_2[B]$);

Swap rule: if $t_1[A] \sqsubset t_2[A]$ but $t_2[B] \sqsubset t_1[B]$, then *swap*($t_1[B], t_2[B]$), or if $t_2[A] \sqsubset t_1[A]$ but $t_1[B] \sqsubset t_2[B]$, then *swap*($t_1[A], t_2[A]$).

The said chase rules cater for all the possible cases when there are two tuples in a relation violating $X \rightsquigarrow Y$. The equate rule is needed in Definition 21 because the two scenarios of violation stated in this rule require equating elements, rather than swapping elements, in order to fix the inconsistency. In applying the chase rules we also need a fixed ordering on the tuples t_1 and t_2. If we choose different orderings on t_1 and t_2 in different applications of the rules, then the chase procedure may result in a non-terminating process. We further clarify this point by the following example:

Example 13. Let F = $\{A \rightsquigarrow B, C \rightsquigarrow B\}$ and the tuples $t_p = \langle 1, 4, 6\rangle$ and $t_q = \langle 2, 3, 5\rangle$ as shown in Figure 13(a). First we let $t_1 = t_p$ and $t_2 = t_q$, then apply the swap rule with respect to $A \rightsquigarrow B$, obtaining the result shown in Figure 13(b). Now we let $t_1 = t_q$ and $t_2 = t_p$ (i.e., reverse the ordering of t_p and t_q), then apply the swap rule with respect to $C \rightsquigarrow B$, obtaining the result as shown in Figure 13(c), which is the beginning relation that we have shown in Figure 13(a).

	A	B	C
t_p (as t_1)	1	4	6
t_q (as t_2)	2	3	5

(a) before the chase

	A	B	C
t_p (as t_2)	1	*3*	6
t_q (as t_1)	2	*4*	5

(b) chase for $A \rightsquigarrow B$ on (a)

	A	B	C
t_p	1	*4*	6
t_q	2	*3*	5

(c) chase for $C \rightsquigarrow B$ on (b)

Fig. 13. An example showing that the chase procedure never terminates

Fortunately, this undesirable property can be removed if we impose a fixed linear ordering on r and assign t_1 to be the smaller tuple and t_2 to be the larger tuple with respect to this ordering. We will show in Lemma 11 that under such a condition the chase procedure always terminates. Therefore, in Example 13, if we assume that the ordering of t_p and t_q is fixed as given in Figure 13(a) *throughout the chase procedure*, then the process terminates. It can be checked that the final relation is obtained as shown in Figure 14.

	A	B	C
t_p (as t_1)	1	*3*	*5*
t_q (as t_2)	2	*4*	*6*

Fig. 14. The chase procedure terminates in Example 13 with a fixed ordering

Let $r = \{t_1, \ldots, t_n\}$ be a relation over R and F be a set of LOFDs with $\mid R \mid = m$. We now give the pseudo-code of an algorithm designated $CHASE$(r,F), which applies the chase rules given in Definition 21 to R as long as possible and returns the resulting relation r over R, also denoted as $CHASE$(r,F).

ALGORITHM 2. *(CHASE(r,F))*

1. **begin**
2. *Result* := $r = \langle t_1, \ldots, t_n\rangle$;
3. *Tmp:*= $\emptyset$;

4. **while** *Tmp* $\neq$ *Result* **do**
5. *Tmp := Result;*
6. **if** $\exists X \rightsquigarrow Y \in F$, $\exists\ t_p, t_q \in$ *Result such that* $t_p[X] \sqsubseteq^l_X t_q[X]$ *but* $t_p[Y] \not\sqsubseteq^l_Y t_q[Y]$ **then**
7. *Apply the appropriate chase rule to Result with* $t_1 = t_{min(p,q)}$ *and* $t_2 = t_{max(p,q)}$*;*
8. **end while**
9. **return** *Result;*
10. **end**.

LEMMA 11. *CHASE(r,F) in Algorithm 2 terminates and satisfies F.*

PROOF. Let P_j with $1 \leq j \leq m$ be the sequence $\langle a_{1j}, \ldots, a_{nj}\rangle$, where $a_{ij} = t_i[A_j]$ (i.e., $\pi_{A_j}(Result) = \{a_{1j}, \ldots, a_{nj}\}$), a_j^{min} be the minimum value in P_j according to the linear extension of the domain ordering, and P_j^{min} be the sequence $\langle a_j^{min}, \ldots, a_j^{min}\rangle$ (a sequence of n identical values). Suppose that an application of a chase rule changes P_j to $P'_j = \langle a'_{1j}, \ldots, a'_{nj}\rangle$. Since the chase rules neither change the value a_j^{min} nor introduce any new values into the variable *Result*, P_j^{min} is unchanged throughout the process of the chase. In order to prove that $CHASE(r,\mathrm{F})$ terminates, it suffices to show that $P_j^{min} \leq^l P'_j <^l P_j$. There are two cases to consider.

In the first case the change to P_j is due to an application of the equate rule. Then by Algorithm 2 we have $a_{pj} \neq a_{qj}$. It follows that $a'_{pj} = min(a_{pj}, a_{qj})$, $a'_{qj} = min(a_{pj}, a_{qj})$ and $a'_{ij} = a_{ij}$ for $i \notin \{p, q\}$. Thus, $P'_j <^l P_j$.

In the second case the change to P_j is due to an application of the swap rule. Without loss of generality, we assume $p < q$. Then by Algorithm 2 $a_{qj} < a_{pj}$. It follows that $a'_{pj} = min(a_{pj}, a_{qj})$, $a'_{qj} = max(a_{pj}, a_{qj})$ and $a'_{ij} = a_{ij}$ for $i \notin \{p, q\}$. Thus, $P'_j <^l P_j$.

It is also trivial that in both cases $P_j^{min} \leq^l P'_j$, since the minimum of any two values in P_j is greater than or equal to the minimum of all values in P_j.

Due to the above consideration, it follows that $CHASE(r,\mathrm{F})$ satisfies F, otherwise we can apply one of the chase rules given in Definition 21 to $CHASE(r,\mathrm{F})$, thus leading to a contradiction, since $CHASE(r,\mathrm{F})$ has not yet terminated. □

LEMMA 12. *CHASE(r,F) in Algorithm 2 can be computed in time polynomial in the sizes of r and F.*

PROOF. By Definition 21, we observe that lines 6 to 7 in Algorithm 2 can be executed at most $O(m)$ times for an LOFD in F, where m is the number of distinct symbols in r. Thus, there is at most $O(m)$ application of chase rules to r. So each execution of the while loop beginning in line 4 and ending at line 8 can be computed in polynomial time in the sizes of r and F. □

Example 14. Let $\mathrm{F} = \{A \rightsquigarrow B, B \rightsquigarrow C\}$ and a relation r consist of three tuples $t_1 = \langle 3, 3, 2\rangle$, $t_2 = \langle 2, 2, 1\rangle$ and $t_3 = \langle 3, 1, 3\rangle$, as shown in Figure 15(a). First, we carry out the chase rules to eliminate the violation of $A \rightsquigarrow B$ as follows, apply the chase rule $swap(t_2[B], t_3[B])$ since $t_2[A] \sqsubset t_3[A]$ but $t_3[B] \sqsubset t_2[B]$, and then apply the chase rule $equate(t_1[B], t_3[B])$ since $t_1[A] = t_3[A]$ but $t_1[B] \neq t_3[B]$. We obtain the intermediate result as shown in Figure 15(b), which satisfies $A \rightsquigarrow B$.

Second, we carry out the chase rules to eliminate the violation of $B \rightsquigarrow C$ as follows, apply the chase rule $equate(t_1[C], t_3[C])$ since $t_1[B] = t_3[B]$ but $t_1[C] \neq t_3[C]$. The chase procedure now terminates and the final result $CHASE(r,\mathrm{F})$ is given in Figure 15(c), which satisfies F. Note that without the equate rule we are not able to continue the chase to reach the consistent relation in Figure 15(c), since the swap rule cannot remove the inconsistency arising from t_1 and t_3 in Figure 15(b).

	A	B	C
t_1	3	3	2
t_2	2	2	1
t_3	3	1	3

(a) r prior to the chase

	A	B	C
t_1	3	*2*	2
t_2	2	*1*	1
t_3	3	*2*	3

(b) chase for $A \rightsquigarrow B$ on (a)

	A	B	C
t_1	3	2	2
t_2	2	1	1
t_3	3	2	*2*

(c) chase for $B \rightsquigarrow C$ on (b)

Fig. 15. An example of obtaining $CHASE(r,\mathrm{F})$

We note that the result of the chase is not necessarily unique. For instance, in the above example we can apply $equate(t_1[B], t_3[B])$ first to eliminate the violation of $A \rightsquigarrow B$, resulting in at least two '1's under the column of attribute B, and leading to a final result different from that given in Figure 15(c). Although the final result of the chase may not be unique, we still can apply it in tackling the implication problem of LOFDs. This point is illustrated by the results shown in the next theorem and Theorem 5.

THEOREM 4. *Let r be a relation over R and F be a set of LOFDs over R. Then $r \models F$ if and only if $r = CHASE(r,F)$.*

PROOF.
IF: Assume to the contrary that $r \not\models \mathrm{F}$. So there exists an LOFD, $X \rightsquigarrow Y \in \mathrm{F}$ such that $r \not\models X \rightsquigarrow Y$. It follows that there must be two rows, $t_1, t_2 \in r$, such that $t_1[X] \sqsubseteq^l_X t_2[X]$ but $t_1[Y] \not\sqsubseteq^l_Y t_2[Y]$. Thus, the chase rule for $X \rightsquigarrow Y$ can be applied to r, resulting in a different relation. This leads to a contradiction, since we have $r \neq CHASE(r,\mathrm{F})$.

ONLY IF: It follows from Definition 21 that a chase rule for F can be carried out only if r violates some LOFD in F. □

Lemma 11 and Theorem 4 are fundamental because they allow the chase procedure to be employed to test the satisfaction of r with respect to a set of F in a finite number of steps; many similar results for different kinds of data dependencies such as FDs, INDs (INclusion Dependencies) and JDs (Join Dependencies) can be found in [Maier et al. 1979], [Johnson and Klug 1984] and [Mannila and Raiha 1988].

4.4 A Proof Procedure for LOFDs

In order to provide a proof procedure for LOFDs, we now define the notion of *ordered variables*. Such variables afford us the ability to infer orderings between attribute values and to establish a set of *templates for relations*, which are essentially the same concept as the tableaux used in [Maier et al. 1979], [Atzeni and De Antonellis 1993)], and [Levene and Loizou 1997].

Definition 22. (*Ordered Variables and Variable Domain*) The *variable domain* of a relation schema R with $\mid R \mid = m$, denoted by $vdom(R)$, is the finite set $\{l_1, \ldots, l_m, h_1, \ldots, h_m, w_1, \ldots, w_m\}$. The variables l_i, h_i and w_i, where $1 \leq i \leq m$, are called *low ordered variables*, *high ordered variables* and *incomparable ordered variables*, respectively. We call them collectively *ordered variables*, whose ordering is given by $l_i \sqsubset h_i$.

We now construct a set of relations defined over variable domains with respect to a given LOFD, which basically enumerate all the possible cases for two tuples violating the LOFD.

Definition 23. (*Template Relations for an LOFD*) Let f be the LOFD $X \rightsquigarrow Y$ over R with $\mid X \mid = n$ and $\mid R \mid = m$. We use two shorthand symbols u_i and v_i to represent one of the following four cases: (1) $u_i = l_i$ and $v_i = l_i$, (2) $u_i = l_i$ and $v_i = h_i$, (3) $u_i = h_i$ and $v_i = l_i$, or (4) $u_i = l_i$ and $v_i = w_i$. A *template relation* (or simply a template) with respect to f, denoted as r_f, is a relation consisting of two tuples, t_1 and t_2, whose underlying domain is $vdom(R)$, such that it is equal to either T_0 or T_k as shown in Figure 16, where $Pre(X) = \langle x_1, \ldots, x_k \rangle$ for $1 \leq k \leq n$.

$T_0 =$

	X	$R - X$
t_1	$l_1 \cdots l_n$	$u_{n+1} \cdots u_m$
t_2	$l_1 \cdots l_n$	$v_{n+1} \cdots v_m$

$T_k =$

	$x_1 \cdots x_{k-1}$	x_k	$R - Pre(X)$
t_1	$l_1 \cdots l_{k-1}$	l_k	$u_{k+1} \cdots u_m$
t_2	$l_1 \cdots l_{k-1}$	h_k	$v_{k+1} \cdots v_m$

Fig. 16. Template relations for an LOFD

We remark that in Definition 23 the symbols u_i and v_i represent four possible combinations choosing from l_i, h_i and w_i. Therefore, it is easy to verify that there are 4^{m-n} templates defined by T_0 and 4^{m-k} templates defined by T_k for each k. Altogether, there are $4^{m-n} + (4^{m-n} + \cdots + 4^{m-1}) = 4^{m-n} + \frac{4^m - 4^{m-n}}{4-1} = \frac{2^{2m} + 2^{2m-2n+1}}{3}$ templates. Note that there are some redundant templates in both T_0 and T_k, if we take into account the fact that there are two possible orderings for t_1 and t_2. However, this does not affect the order of the upper bound of the number of templates, which is shown to be $O(4^m)$.

We apply the chase rules to a template relation using the ordering defined on a variable domain $vdom(R)$. The following proposition gives the result corresponding to Theorem 4.

PROPOSITION 3. *Let r_f be a template relation over R and F be a set of LOFDs over R. Then $r_f \models F$ if and only if $r_f = CHASE(r_f, F)$.*

A template relation can be viewed as a relation instance consisting of two tuples by mapping ordered variables to values in D.

Definition 24. (*Valuation Mapping*) Let $R = \{A_1, \ldots, A_m\}$ and $vdom(R) = \{l_1, \ldots, l_m, h_1, \ldots, h_m, w_1, \ldots, w_m\}$. A *valuation mapping* ρ is a mapping from $vdom(R)$ to D such that $\rho(l_i) \sqsubset \rho(h_i)$ and $\rho(l_i) \parallel \rho(w_i)$ for all $1 \leq i \leq m$. We extend ρ to a tuple t by $\rho(t) = \langle \rho(t[A_1]), \ldots, \rho(t[A_m]) \rangle$. We also extend ρ to a template relation r_f by $\rho(r_f) = \{\rho(t_1), \rho(t_2)\}$.

The next proposition states that if there is a valuation mapping relating a template relation to a relation having two tuples, then they satisfy the same set of LOFDs.

PROPOSITION 4. *Let $\rho(r_f) = r$, where r is a relation over R having two tuples. Then $r_f \models X \rightsquigarrow Y$ if and only if $r \models X \rightsquigarrow Y$.*

PROOF. The result immediately follows Definition 24, since the ordering of data values in the ith column of r corresponds to the ordering of the ordered variables l_i, h_i and w_i for all $1 \leq i \leq \mid R \mid$. □

The following example shows how to apply a valuation mapping to a template relation.

Example 15. Consider the template relation r_f over $\{A, B, C, D\}$ with respect to the LOFD f, $A \rightsquigarrow BCD$, which is shown in Figure 17(a) in which we assume that $a \parallel b$. We define the valuation mapping ρ by $\rho(l_1) = 1$, $\rho(l_2) = 2$, $\rho(h_2) = 3$, $\rho(l_3) = 4$, $\rho(h_3) = 5$, $\rho(l_4) = a$ and $\rho(w_4) = b$. (The mapping of other ordered variables is immaterial.) We then have $\rho(r_f)$ shown in Figure 17(b). Note that in this example r_f is one of the templates defined by T_0 in Definition 23.

$r_f =$

A	B	C	D
l_1	l_2	h_3	l_4
l_1	h_2	l_3	w_4

(a)

$\rho(r_f) =$

A	B	C	D
1	2	5	a
1	3	4	b

(b)

Fig. 17. An example showing the application of a valuation mapping

We now extend the notion of tableaux for an LOFD f to be a set of templates. The tableaux in our case is different from that for FDs, which just requires a single template for FDs (c.f. Theorem 4.2 in [Atzeni and De Antonellis 1993)]). We define *tableaux* T_f to be the set of all template relations given in Definition 23.

Definition 25. (*Satisfaction and a Valuation Mapping of Tableaux*) The chase of T_f with respect to a set of LOFDs F, denoted as $CHASE(T_f$,F), is defined by $CHASE(T_f$,F) $= \{CHASE(r_f$,F) $\mid r_f \in T_f\}$. $CHASE(T_f$,F) *satisfies* $X \rightsquigarrow Y$, denoted by $CHASE(T_f$,F)$\models X \rightsquigarrow Y$, if, for all $r_f \in T_f$, $CHASE(r_f$,F) $\models X \rightsquigarrow Y$. Furthermore, $CHASE(T_f$,F) satisfies F, denoted by $CHASE(T_f$,F) $\models$ F, if, for all $X \rightsquigarrow Y \in$ F, $CHASE(T_f$,F) $\models X \rightsquigarrow Y$. A *valuation mapping* of T_f is a valuation mapping of some r_f in T_f.

The following theorem shows that the chase rules can be viewed as a sound and complete inference procedure for LOFDs.

THEOREM 5. *Let F be a set of LOFDs over R and f be a LOFD $X \rightsquigarrow Y$. Then $CHASE(T_f$,$F) \models f$ if and only if $F \models f$.*

PROOF.
IF: Assume $CHASE(T_f$,F) $\not\models f$. By Definition 25, there exists $r_f \in T_f$ such that $CHASE(r_f$,F) $\not\models f$ but $CHASE(r_f$,F) $\models$ F. Note that $CHASE(r_f$,F) is a

template which can be viewed as a relation instance. Therefore, we have a valuation mapping ρ to generate a relation $\rho(CHASE(r_f,\mathrm{F}))$ and by Proposition 4, $\rho(CHASE(r_f,\mathrm{F})) \models \mathrm{F}$ but $\rho(CHASE(r_f,\mathrm{F})) \not\models f$. This leads to a contradiction.

ONLY IF: We let w_1, w_2 be any two tuples in a relation r such that $w_1 \sqsubseteq^l_X w_2$. We claim $w_1 \sqsubseteq^l_Y w_2$. Let $s_f \in T_f$ be the template relation such that $\rho(t_1) = w_1$ and $\rho(t_2) = w_2$. We can always find such a template s_f because T_f exhausts all possibilities of two tuples which satisfy the condition $w_1 \sqsubseteq^l_X w_2$. Thus, we have $\rho(s_f) = \{w_1, w_2\}$ and $\rho(s_f) \models \mathrm{F}$. By Proposition 4, we have $s_f \models \mathrm{F}$. It follows by Proposition 3 that $s_f = CHASE(s_f,\mathrm{F})$. Since we have assumed that $CHASE(T_f,\mathrm{F}) \models f$, we have $CHASE(s_f,\mathrm{F}) \models f$. Thus, $\rho(CHASE(T_f,\mathrm{F}))= \rho(s_f) = \{w_1, w_2\}$, which implies that $w_1 \sqsubseteq^l_Y w_2$ as required. □

The following corollary is an immediate result of Theorem 5.

Corollary 3. *Let F be a set of LOFDs over R. The chase procedure is a decidable, sound and complete inference algorithm for LOFDs.*

The above corollary shows that the chase rules, together with tableaux, can be used to provide a systematic way to solve the implication problem for LOFDs.

4.5 OFDs Arising from Mixed Orderings

Pointwise-orderings and lexicographical orderings are two basic extensions of domain orderings, which have been used in defining POFDs and LOFDs in previous sections. We now further investigate other variant forms of OFDs arising from two possible combinations of pointwise-orderings and lexicographical orderings in a given OFD: (1) pointwise-orderings on the left hand side and lexicographical orderings on the right hand side, which give rise to PLOFDs, and (2) lexicographical orderings on the left hand side and pointwise-orderings on the right hand side, which give rise to LPOFDs. Using similar notations in Definitions 17 and 19, we give the formal semantics of PLOFDs and LPOFDs as follows:

Definition 26. (*Ordered Functional Dependencies Arising from Combination of Pointwise-Orderings and Lexicographical Orderings*) A *PLOFD*, denoted by $X \hookrightarrow\!\!\!\!\sim Y$, is satisfied in a relation r over R if, for all $t_1, t_2 \in r$, $t_1[X] \sqsubseteq^p_X t_2[X]$ implies that $t_1[Y] \sqsubseteq^l_Y t_2[Y]$. An *LPOFD*, denoted by $X \sim\!\!\!\!\hookrightarrow Y$, is satisfied in a relation r over R if, for all $t_1, t_2 \in r$, $t_1[X] \sqsubseteq^l_X t_2[X]$ implies that $t_1[Y] \sqsubseteq^p_Y t_2[Y]$. We call LPOFDs and PLOFDs collectively *mixed* OFDs.

The satisfaction of mixed OFDs is related to that of POFDs and LOFDs in a simple and interesting way. The results are presented in Lemma 13. Note that the converse of parts (2) and (3) in this Lemma does not hold in a relation r.

Lemma 13. *Let r be a relation. The following statements are true.*

(1) $r \models X \hookrightarrow\!\!\!\!\sim Y$, if and only if, $r \models X \rightsquigarrow A$ for all $A \in Y$.
(2) If $r \models X \hookrightarrow\!\!\!\!\sim Y$, then $r \models X \hookrightarrow Y$ and $r \models X \rightsquigarrow Y$.
(3) If $r \models X \rightsquigarrow Y$ or $r \models X \hookrightarrow Y$, then $r \models X \hookrightarrow\!\!\!\!\sim Y$.

Proof. Part (1) can be established by noting that the union and decomposition rules are sound for LPOFDs. Parts (2) and (3) follow from Definition 26 and the fact that, $\sqsubseteq^p_X$ implies that $\sqsubseteq^l_X$ and $\sqsubseteq^p_Y$ implies that $\sqsubseteq^l_Y$. □

It follows from part (1) of Lemma 13 that we are able to reduce the implication problem for LPOFDs into the implication problem for a corresponding set of LOFDs, each of which has only one attribute on the right hand side. Thus we are able to apply the proof procedure established in Section 4.4 to the set of LOFDs. The implication problem for PLOFDs can also be solved using the chase approach, similar to the case of LOFDs. We first give the corresponding chase rules adapted to the context of PLOFDs, which use similar notations as Definition 21.

Definition 27. (*Chase Rules for PLOFDs*) Let t_1 and t_2 be two tuples in r such that $t_1[X] \sqsubseteq^p_X t_2[X]$ but $t_1[Y] \not\sqsubseteq^l_Y t_2[Y]$, and the attributes A and B are defined as in Definition 21, then the *chase rules* for the PLOFD $X \hookrightarrow\!\!\rightsquigarrow Y$, are defined by the following two rules:

Equate rule: if $t_1[X] = t_2[X]$ but $t_1[B] \neq t_2[B]$, or if $t_1[X] \not\parallel t_2[X]$ but $t_1[B] \parallel t_2[B]$, then *equate*$(t_1[B], t_2[B])$;

Swap rule: if $t_1[X] \sqsubset^p t_2[X]$ but $t_2[B] \sqsubset t_1[B]$, then *swap*$(t_1[B], t_2[B])$, or if $t_2[X] \sqsubset^p t_1[X]$ but $t_1[B] \sqsubset t_2[B]$, then *swap*$(t_1[A], t_2[A])$.

Next, we need only two templates to cater for the general cases that violate a given PLOFD, $X \hookrightarrow\!\!\rightsquigarrow Y$. Using same notations as in Definition 23, we give the two templates of T_0 and T_1 as follows:

$T_0 =$

	X	$R - X$
t_1	$l_1 \cdots l_n$	$u_{n+1} \cdots u_m$
t_2	$l_1 \cdots l_n$	$v_{n+1} \cdots v_m$

$T_1 =$

	X	$R - X$
t_1	$l_1 \cdots l_n$	$u_{n+1} \cdots u_m$
t_2	$h_1 \cdots h_n$	$v_{n+1} \cdots v_m$

Fig. 18. Template relations for a PLOFD

Clearly, there are 4^{m-n} templates in T_0 and T_1, where $\mid R \mid = m$ and $\mid X \mid = n$, and the order of the upper bound of the number of templates in this case is also $O(4^m)$ using a similar argument to that in Section 4.4.

The following proposition summarises the relationship between the satisfaction of FDs and that of various forms of OFDs in a relation r. Note that the converse of this proposition does not hold in a relation r. This interesting relationship paves the way for studying normalisation of ordered databases when taking mixed OFDs into consideration.

Proposition 5. *Let r be a relation. If $r \models X\sigma Y$, then $r \models X \rightarrow Y$, where $\sigma \in \{\hookrightarrow, \rightsquigarrow, \rightsquigarrow\!\!\hookrightarrow, \hookrightarrow\!\!\rightsquigarrow\}$.*

From Proposition 5, it follows that the set of relations which satisfy a set of (any category of) OFDs F is a subset of the relations which satisfy the corresponding set of FDs F^*, where F^* is defined as $\{X \rightarrow Y \mid X\sigma Y \in \mathrm{F}\}$. We now let $SAT(f)$ be the set of relations that satisfy a data dependency f, and $f_1 = X \rightarrow Y$, $f_2 = X \hookrightarrow Y$, $f_3 = X \rightsquigarrow Y$, $f_4 = X \rightsquigarrow\!\!\hookrightarrow Y$, and $f_5 = X \hookrightarrow\!\!\rightsquigarrow Y$. From parts (2) and (3) of Lemma 13, it follows that $SAT(f_4) \subseteq (SAT(f_2) \cap SAT(f_3))$ and that $(SAT(f_2) \cup SAT(f_3)) \subseteq SAT(f_5)$. A comparison of the satisfaction of different categories of OFDs introduced so far in ordered relations can be represented by the diagram given in Figure 19 (the scale here is irrelevant). We remark that if X and

Y are unary, then in general we have $SAT(f_2) = SAT(f_3) = SAT(f_4) = SAT(f_5)$, but still we do not *always* have $SAT(f_1) = SAT(f_i)$ for $i \in \{2, 3, 4, 5\}$.

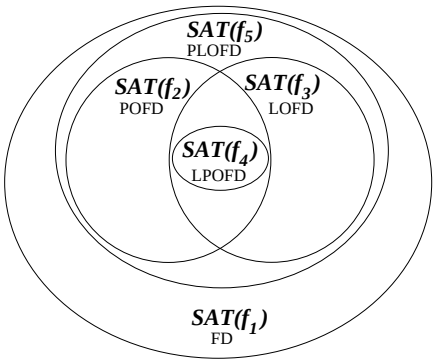

Fig. 19. Satisfaction of FDs and different categories of OFDs in ordered relations

5. CONCLUDING REMARKS

In this paper we have presented the ordered relational model and studied its impacts on the following three fundamental components of the conventional relational data model: data structures, query languages and data dependencies.

With respect to its data structures, the relational data model is extended to incorporate partial orderings into data domains. Hence, it provides the flexibility to manipulate tuples in an ordered database according to the semantics of underlying domains. With respect to its query languages, we have extended the relational algebra to the PORA by allowing the use of the ordering predicate, $\sqsubseteq$, in the language, whose expressive power has been formally stated in Theorem 1. Based on the features of the PORA, we have extended SQL to OSQL, which combines the capabilities of SQL with the power of semantic orderings as illustrated by the running example in Section 3.5. With respect to its data dependencies, we have formally defined OFDs, and have studied their semantics according to two categories of orderings: lexicographical orderings and pointwise-orderings. In the case of pointwise-orderings, we have presented a sound and complete axiom system for OFDs in Theorem 3. In the case of lexicographical orderings, we have presented a set of novel chase rules to OFDs in Definition 21, which are used to tackle the implication problem of OFDs as shown in Theorem 5. We have also discussed two other variant forms of OFDs arising from mixed orderings, and presented in Figure 19 the interesting relationship between the satisfaction of conventional FDs and different categories of OFDs in ordered relations.

The ordered relational model is a minimal extension of the relational data model. However, we have shown throughout the paper that partial orderings in data domains have an important part to play in modelling data. Our work is best evaluated in the context of the three successful factors of the relational model that we mentioned at the beginning.

(1) From the point of view of usability, the ordered relational model is as natural and simple as the conventional relational model. Ordered domains are easily understood by non-specialist users due to the fact that partial orderings are essential properties with respect to the structure of many types of data organisation in the real world. The ordered database model we have defined is easily compatible with the syntax and semantics of the conventional relational database model.

(2) From the point of view of applicability, the ordered relational model has been demonstrated to have the capabilities of capturing semantics in a wide spectrum of advanced applications such as tree-structured, temporal or incomplete information. It is also the only data model known to us that combines all the above application capabilities under a single unified model.

(3) From the point of view of formalism, the ordered relational model is elegant enough to support theoretical research in the areas of functional dependencies, the expressiveness of the PORA and the generic properties of queries over ordered databases. We can also build upon the rich mathematical research into the notion of order to investigate many important issues such as query completeness and axiomatisation of data dependencies.

There is still a wide range of research issues that can be carried out on the implementational aspects of the ordered relational model. For instance, one important issue is to integrate the facilitates of semantic orderings into the kernel of DBMSs at the physical level of a DBMS. We can consider a data structure called an *Ordered B-tree* (c.f. [Lynn 1982]), which may serve as a basis to implement ordered relations. Roughly speaking, an *Ordered B-tree* stores data, for example tuple identifiers, in its leaf pages, and a multi-level index is provided in each subtree to access data. In order to find a tuple identifier, the system is designed so that a scan can be performed from the root of the tree until a leaf page is encountered. Another important issue that we have not discussed is updating ordered domains. This can be investigated in terms of the algorithms and formal semantics of updating ordered domains, ordered databases and data dependencies. In particular, it is important to consider how to enforce data dependencies to ensure that updates do not cause inconsistencies of data with respect to a set of OFDs. We also believe the scope of the application of an internal hierarchy has not been fully developed. For example, in a parallel object-relational database environment, if a data stream is ordered according to semantic orderings then it helps to improve dynamic reconfiguration of query execution plans, as suggested by the empirical results in [Ng 1999]. It thus seems promising to study the possible benefits that can be obtained for query optimisation when a data stream is partitioned into an internal hierarchy in such an environment.

ACKNOWLEDGMENTS

The author would like to thank Mark Levene, Trevor Fenner, Nigel Martin, Ken Moody, Yoshifumi Masunaga, and anonymous referees for their constructive comments at different stages of this paper.

APPENDIX A: A Grammar of OSQL

Conventions:

—Key words are indicated by uppercase italicised characters.
—Non-terminal symbols are enclosed with "⟨⟩".
—Alternatives are separated by "|". If only one of the symbols is to be chosen out of several alternatives, then we enclose them with "{ }". In order not to cause confusion, we use "{{" and "}}" to represent the textual braces "{" and "}" used in OSQL expressions.
—Optional clauses are enclosed with "[ ]".
—"()" are just terminal symbols.
—Default keywords are underlined.
—A positive number begins with #.
—... at the end if a subclause indicates that it may be repeated.

A1. Data Definition Language

(1) *CREATE DOMAIN ORDER* ⟨ domain-order-name ⟩ ⟨ data-type ⟩ *AS*
⟨ ordering-specification ⟩
⟨ ordering-specification ⟩ ::= (⟨ data-pair ⟩ [, ⟨ data-pair ⟩ . . .])
⟨ data-pair ⟩ ::= [data-item | {{ data-item,. . .}}] < [data-item | {{ data-item,. . .}}]

(2) *CREATE TUPLE ORDER* ⟨ tuple-order-name ⟩ *ON* ⟨ data-type-list ⟩ *AS* ⟨ order-name-list ⟩
⟨ order-name-list ⟩ ::= (⟨ order-name ⟩ [, ⟨ order-name ⟩ . . .])
⟨ order-name ⟩ ::= { domain-order-name | *SYO* } [{ *ASC* | *DESC* }]
⟨ data-type-list ⟩ ::= (⟨ data-type ⟩ [, ⟨ data-type ⟩ . . .])

(3) *CREATE TABLE* ⟨ table-name ⟩ ⟨ ⟨ column-specification ⟩ [, ⟨ column-specification ⟩]. . . ⟩
⟨ column specification ⟩ ::= (attribute-name ⟨ data-type ⟩)
⟨ data-type ⟩ ::= { *CHAR*(#*n*) | *NUM*(#*n*) | *DATE* }

A2. Data Manipulation Language

(1) *SELECT* ⟨ attribute-list ⟩
FROM ⟨ ordered-relation-list ⟩
[*WHERE* { ⟨ comparison-expression ⟩ | *TUPLE* ⟨ tuple-level-set ⟩ }]
[*ORDER BY* ⟨ attribute-list ⟩]
⟨ attribute-list ⟩ ::= ⟨ attribute ⟩ [, ⟨ attribute ⟩]. . .
⟨ attribute ⟩ ::= { attribute-name | attribute-name *WITHIN* ⟨ order-name ⟩ | ∗ }
⟨ tuple-level-set ⟩ ::= ({ #n [, #n]) | *LAST* | #n1 . . #n2 })
⟨ ordered-relation-list ⟩ ::= ⟨ relation ⟩ [, ⟨ relation ⟩]. . .
⟨ relation ⟩ ::= { relation-name | relation-name *WITHIN* ⟨ tuple-order-name ⟩ [{ *ASC* | *DESC* }] }
⟨ comparison-expression ⟩ ::= ⟨ simple-comparison ⟩ [{ *AND* | *OR* } ⟨ simple-comparison ⟩]. . .
⟨ simple-comparison ⟩ ::= ⟨ { attribute-list | value-list } ⟩ ⟨ comparator ⟩
⟨ { attribute-list | value-list } ⟩ [*WITHIN* { ⟨ domain-order-name ⟩ | ⟨ tuple-order-name ⟩ }]
⟨ comparator ⟩ ::= { <|>|>=|<=|<> }
⟨ value-list ⟩ ::= (value [, value]. . .)

(2) *DELETE FROM* ⟨ table-name ⟩
[*WHERE* { ⟨ comparison-expression ⟩ | *TUPLE* ⟨ tuple-level-set ⟩ }]

REFERENCES

Abiteboul, S. and Ginsburg, S. 1986. Tuple Sequences and Lexicographical Indexes. *Journal of the Association for Computing Machinery*, 33, 3, 409-422.

Abiteboul, S., Hull, R., and Vianu, V. 1995. *Foundations of Databases*. Addison-Wesley.

ANSI/X3/SPARC 1975. Study Group on Database Management Systems, Interim Report. *FDT Bulletin of ACM SIGFIDET*, 7.

Armstrong, W.W. 1974. Dependency Structures of Data Base Relationships. In *Proceedings of the IFIP Congress*, Stockholm, 580-583.

Atzeni, P. and De Antonellis, V. 1993. *Relational Database Theory*. Benjamin/Cummings Publishing Company.

Bancilhon, F. 1978. On the Completeness of Query Languages for Relational Databases. In *LNCS 64: Mathematical Foundations of Computer Science*, Springer-Verlag, 112-124.

Beeri, C. and Bernstein, P.A. 1979. Computational Problems Related to the Design of Normal Form Relational Schemas. *ACM Transactions on Database Systems*, 4, 1, 30-59.

Biskup, J. 1990. An Extension of SQL for Querying Graph Relations. *Computing Language*, 15, 2, 65-82.

Blaha, M. and Premerlani, W. 1998. *Object-Oriented Modeling and Design for Database Applications*. Prentice Hall Publishing Company.

Buneman, P., Jung, A., and Ohori, A. 1991. Using Powerdomains to Generalise Relational Databases. *Theoretical Computer Science*, 9, 1, 23-55.

Chandra, A.K. and Harel, D. 1980. Computable Queries for Relational Databases. *Journal of Computer System Science*, 21, 2, 156-178.

Codd, E.F. 1970. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13, 6, 377-387.

Codd, E.F. 1979. Extending the Database Relational Model to Capture More Meaning. *ACM Transactions on Database Systems*, 4, 4, 397-434.

Codd, E.F. 1986. Missing Information (Applicable and Inapplicable) in Relational Databases. *ACM SIGMOD Record*, 15, 4, 53-78.

Date, C.J. 1990. *Relational Database Writings 1985-1989*. Addison-Wesley.

Date, C.J. 1997. *A Guide to the SQL Standard.*, 4th ed., Addison-Wesley.

Garey, M.R. and Johnson, D.S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York.

Ginsburg, S. and Hull, R. 1983. Order Dependency in the Relational Model. *Theoretical Computer Science*, 26, 1-2, 149-195.

Ginsburg, S. and Tanaka, K. 1986. Computation Tuple Sequences and Object Histories. *ACM Transactions on Database Systems*, 11, 2, 186-212.

Ginsburg, S. and Hull, R. 1986. Sort Sets in the Relational Model. *Journal of the Association for Computing Machinery*, 33, 3, 465-488.

Grahne, G. 1984. Dependency Satisfaction in Databases with Incomplete Information. In *Proceedings of the 10th VLDB International Conference*, 37-45.

Gratzer, G. 1978. *General Lattice Theory*. NewYork: Academic Press.

Guting, R.H., Zicari, R., and Choy, D.M. 1989. An Algebra for Structured Office Documents. *ACM Transactions on Office Information Systems*, 7, 4, 123-157.

Halmos, P. 1974. *Naive Set Theory,* Springer-Verlag, New York.

Honeyman, P. 1982. Testing Satisfaction of Functional Dependencies. *Journal of the ACM*, 29, 3, 668-677.

Jung, A., Libkin, L., and Puhlmann, H. 1991. Decomposition of Domains. In *LNCS 598: Proceedings of the Conference on Mathematical Foundations of Programming Semantics*, Springer-Verlag, 235-258.

Johnson, D.S. and Klug, A. 1984. Testing Containment of Conjunctive Queries under Functional and Inclusion Dependencies. *Journal of Computer and System Sciences*, 28, 1, 167-189.

Levene, M. and Loizou, G. 1996. Maintaining Consistency of Imprecise Relations. *The Computer Journal*, 39, 2, 114-123.

Levene, M. and Loizou, G. 1997. Null Inclusion Dependencies in Relational Databases. *Information and Computation*, 136, 2, 67-108.

Libkin, L. 1996. *Aspects of Partial Information in Databases*. Ph.D. Thesis, University of Pennsylvania, United States.

Lorentzos, N.A. 1992. DBMS Support for Time and Totally Ordered Compound Data Types. *Information Systems*, 17, 5, 347-358.

Lynn, N. 1982. *Implementation of Ordered Relations in a Data Base System*. Master Thesis, University of California, United States.

Maier, D., Mendelzon, A.O., and Sagiv, Y. 1979. Testing Implication of Data Dependencies. *ACM Transactions on Database Systems*, 4, 4, 455-469.

Maier, D. and Vance, B. 1993. A Call to Order. In *Proceedings of the Twelfth ACM Symposium on Principles of Databases Systems*, 1-16.

Mannila, H. and Raiha, K-J. 1988. Generating Armstrong Databases for Sets of Functional and Inclusion Dependencies. *Research Report* A-1988-7, University of Tampere, Finland.

Ng, W. K. 1999. *Dynamic Optimization of Query Execution Plans*. Ph.D. Thesis, University of California, Los Angeles, United States.

Ng, W. and Levene, M. 1997a. An Extension of OSQL to Support Ordered Domains in Relational Databases. In *IEEE Proceedings of the International Database Engineering and Applications Symposium*, 358-367.

Ng, W. and Levene, M. 1997b. The Development of Ordered SQL Packages for Modelling Advanced Applications. In *LNCS 1308: Proceedings of 8th International Conference of Database and Expert Systems Application*, Springer-Verlag, 529-538.

Ng, W. 1999a. Lexicographical Ordered Functional Dependencies and Their Application to Temporal Relations. In *IEEE Proceedings of the International Database Engineering and Applications Symposium*, 279-287.

Ng, W. 1999b. Ordered Functional Dependencies in Relational Databases. *Information Systems*, 24, 7, 535-554.

Paredaens, J. 1978. On the Expressive Power of the Relational Algebra. *Information Processing Letters*, 7, 2, 107-111.

Raymond, D. 1996. *Partial Order Databases*. Ph.D. Thesis, University of Waterloo, Canada.

Rumbaugh, J. 1988. Relational Database Design using an Object-Oriented Methodology. *Communications of the ACM*, 31, 4, 417-427.

Read, R. 1995. *Towards Multiresolution Data Retrieval via the Sandbag*. Ph.D. Thesis, University of Texas at Austin, United States.

Seshadri, P., Livny, M., and Ramakrishnan, R. 1996. The Design and Implementation of a Sequence Database System. *Proceedings of the 22nd VLDB Conference*, 99-110.

Tansel, A. et al. (editors) 1993. *Temporal Databases: Theory, Design and Implementation*. The Benjamin/Cummings Publishing Company.

Ullman, J.D. 1988. *Principles of Database and Knowledge-Base Systems, Vol. I*, Rockville, MD., Computer Science Press.

Wijsen, J. 1998. Reasoning about Qualitative Trends in Databases. *Information Systems*, 23, 7, 469-493.

Zaniolo, C. 1984. Database Relations with Null Values. *Journal of Computer and System Science*, 28, 1, 142-166.