A Formal Basis for Extending SQL to Object-Oriented Databases

Jan Van den Bussche*
University of Antwerp (UIA)

Abstract

A formal basis for extending SQL in a natural way to complex object databases is laid. $\mathcal{N}$SQL, a language equivalent to the standard algebra for nested relational databases, is obtained. It is also shown that, when allowing subqueries in the from-clause, the facility of subqueries in where-clauses becomes redundant with regard to expressive power.

1 Introduction

Much research work in database theory [6, 11, 13, 18] is spent on finding possible successors of the relational database model, as introduced by Codd in the early seventies [7, 8, 9]. Indeed, for more unconventional applications like CAD and office automation [4, 14], it became inacceptable that complex structures had to be decomposed in several different relations in order to be manageable by relational database systems [16]. In the quest for designing systems that overcome the drawbacks of the relational model, much attention has been paid to *object-oriented database systems* [2, 3].

The database language SQL is now widely accepted as the standard query language for relational database systems. Therefore it is meaningful to ask

*Author's address: Dept. Math. & Comp. Science, University of Antwerp (UIA), Universiteitsplein 1, B-2610 Antwerpen, Belgium. E-mail: `vdbuss@ccu.uia.ac.be` The author's work is supported by a grant from the I.W.O.N.L.

how this language can be extended in order to deal with object-oriented database systems. It is the aim of this paper to partially answer this question.

Two of the most appealing object-oriented concepts in the database context are *object identity* and *complex modelling*. In this paper, we concentrate on the latter one. General complex object databases were presented in [1, 5]. A particular and well-known model supporting complex objects is the *nested relational database model* [17, 19]. It is an extension of the relational database model in which a table element can be an atomic value, as in ordinary relations, but also a nested relation in turn. Complex structures can be represented more naturally using nested relations.

The paper is organized as follows. Section 2 introduces the necessary terminology regarding the nested relational database model and the associated formal query system, the *nested algebra*. Section 3 formally introduces $\mathcal{N}$SQL, an extension of SQL in the context of nested relations. In Section 4 it is shown that $\mathcal{N}$SQL has the full expressibility of the nested algebra. Section 5 gives conclusions and provides some issues for future work. This paper reports on a part of the work done by the author in [21].

It should be noted that Roth, Korth and Batory [15] introduced SQL/NF, which is also a query language for nested relations, based on SQL. However, the author feels that SQL/NF is a complete redesign of SQL (motivated by [10]), whereby the important aspect of extending from flat to nested relations is somewhat neglected. Therefore, the author cannot see SQL/NF as a solution for the problem this paper partially tries to solve.

2 Nested relations

A certain familiarity with the relational database model [7, 16, 20] is assumed. In this section, a formal model for working with nested relations is presented. It is essentially that of Gyssens and Van Gucht, as can be found in e.g. [12, 16].

Basically we have an infinitely enumerable set U of *atomic attributes* and an infinitely enumerable set V of *atomic values*. The set $\mathcal{U}$ of *attributes* can be defined inductively as follows: every atomic attribute is an attribute, and every finite set of attributes in which no atomic attribute appears[1] more than once is an attribute. As an example, suppose A, B, C are atomic attributes. Then $\{A, B, \{B, C\}\}$ is no legal attribute because B appears twice in it; $\{\{A\}\}$ is legal, as is $\{C, \{B\}\}$. Elements of $\mathcal{U} - U$ are called *composed attributes*. A *relation scheme* is simply a composed attribute.

For a scheme Ω, an *instance* ω over Ω is a finite set of *tuples* over Ω. A tuple t over a scheme Ω is a function on Ω such that for every atomic attribute $A \in \Omega$, $t(A) \in V$, and for every composed attribute $X \in \Omega$, $t(X)$ is an instance over X. Finally, a *nested relation* is an ordered pair (Ω, ω) such that Ω is a scheme and ω is an instance over Ω.

A scheme Ω is *flat* if $\Omega \subset U$. Note that relations over flat schemes are exactly the relations of the conventional relational model, so the nested relational model is a proper extension of this model. Of course, we need also a formal query system for nested relations, like the *relational algebra* is for flat relations. To this end, the algebra is extended to the *nested algebra* as follows. The *union, difference, cartesian product, equality selection* and *projection* are either canonical extensions or natural generalizations of their flat counterparts. Remain two restructuring operations that do not have a flat equivalent: *nesting* and *unnesting*. Because tuple components may be relations themselves, we need mechanisms to access the contents of these relations. Informally speaking, nesting a relation over a set of attributes X consists of grouping all tuples that are equal outside X together into one nested tuple, with a new composed attribute X containing all the X-values of the grouped tuples. Unnesting a relation over the composed attribute X is similar to undoing a nest operation over X. So unnesting can be used to reach the tuples of a subrelation; nesting provides a way back. Formal definitions of nest and unnest follow. Let ω be an instance over Ω. Then the nesting $\nu_X(\Omega, \omega)$ equals (Ω', ω'), where $\Omega' := \Omega - X \cup \{X\}$ and

$$\omega' := \big\{t \mid \exists t' \in \omega : t|_{\Omega - X} = t'|_{\Omega - X} \& t(X) = \{t''|_X \mid t'' \in \omega \& t''|_{\Omega - X} = t'|_{\Omega - X}\}\big\}$$

Let $Z \in \Omega - U$. Then the unnesting $\mu_Z(\Omega, \omega)$ equals (Ω', ω'), where $\Omega' := \Omega - \{Z\} \cup Z$ and

$$\omega' := \{t \mid \exists t' \in \omega : t|_{\Omega - \{Z\}} = t'|_{\Omega - \{Z\}} \;\&\; t|_Z \in t'(Z)\}$$

[1] The term *appears in* refers to any level: A appears in X if $A \in X$ or if A appears in Y for some $Y \in X$.

Concerning unnesting, it is frequently the case that we want to access a subrelation that is nested more than one level deep. Then, a sequence of unnest operations is needed. For this, we use the following notation: for a relation (Ω, ω), let $X_1 \in \cdots \in X_n \in \Omega$ with X_1 a composed attribute. Then $\overline{\mu}_{X_1}(\Omega, \omega)$ equals

$$\mu_{X_1} \cdots \mu_{X_n}(\Omega, \omega)$$

Finally we mention *renamings*. Applying a renaming to a relation changes one or more atomic attributes appearing in its scheme. Renaming is mostly used as a remedy in cases when the cartesian product is to be taken of two relations that do not have completely "independent" schemes, i.e., where one or more atomic attributes appear in both schemes. In such cases the resulting scheme would be ill-defined since these atomic attributes would appear more than once in it (recall the definition of composed attributes).

For an excellent overview of nested relational structures and their properties the reader is referred to chapter 7 of [16].

3 $\mathcal{N}$SQL

In the section above, the relational algebra was extended to the nested algebra. In this section the same will be done for the relational query language SQL. Although the language is widely used, formal definitions of the semantics of SQL are rare. In [16], a representative subset of SQL is formally proven to be equivalent with the relational algebra. This subset will be extended here to the query language $\mathcal{N}$SQL. The semantics of $\mathcal{N}$SQL will be defined formally using the nested algebra.

The syntax of $\mathcal{N}$SQL is shown in Figure 1. As usual, vertical bars denote logical or; square brackets denote an optional construct; curly brackets indicate that the construct may appear zero or more times; round brackets serve for grouping. Lexical entities are written in a different typestyle. When quoted, like in `"{"` or `")"`, the brackets are lexical entities, not meta characters.

Some remarks about the representation of attributes in $\mathcal{N}$SQL are in place here. We assume that every atomic attribute has its own attribute identifier. Composed attributes are sets of attributes; they can be syntactically

query ← *elementary-query* [(`difference`|`union`) *query*]
elementary-query ← *simple-query* [`group by` [*group-list*]]
simple-query ← `select` (`*`|*select-list*) `from` *relation-list* [`where` *condi-
tion*]
select-list ← *attribute-spec* {`,`*attribute-spec*}
attribute-spec ← [*tuple-id* `.`] *attribute-repr*
attribute-repr ← (*attribute-id*|*set-enum*)
set-enum ← `"{"` *attribute-repr* {`,`*attribute-repr*} `"}"`
relation-list ← *relation-spec* {`,`*relation-spec*}
relation-spec ← (*relation-id*|`"("`*query*`")"`) [*tuple-id*]
condition ← *elementary-cond*
condition ← `not` *condition*
condition ← *condition* (`or`|`and`) *condition*
condition ← `"("`*condition*`")"`
elementary-cond ← *comparison*
elementary-cond ← `exists` `"("`*simple-query*`")"`
comparison ← *attribute-spec* (`=`|`subset of`) *attribute-spec*
comparison ← *attribute-list* `element of` *attribute-spec*
group-list ← *attribute-spec* {`,`*attribute-spec*}

Figure 1: Syntax of $\mathcal{N}$SQL.

described as such (see *set-enum* in Figure 1). However, composed attributes may also have an attribute identifier[2] (see *attribute-repr* in Figure 1).

One difference between $\mathcal{N}$SQL and SQL is fundamental: the possibility for the relations in the `from`-clause to be queries themselves (see *relation-list* and *relation-spec* in the syntax description). This was proposed before in [10, 15]. It will turn out to be a powerful feature, with interesting properties.

The *labeling* technique of SQL is adopted by $\mathcal{N}$SQL. Labeling means providing a tuple identifier with a relation in a `from`-clause (see *relation-spec* in Figure 1). We could interpret the labeling of a relation as renaming the relation by prefixing every attribute appearing in the scheme with "t.". In this way, the attributes of labeled relations are specified in $\mathcal{N}$SQL; see *attribute-spec* in Figure 1.

We now can give a formal definition of the meaning of an $\mathcal{N}$SQL query, by inductively associating it with an equivalent expression in the nested algebra.

- As induction hypothesis, we assume that there is a nested algebra expression associated with each relation specifier of a relation list. In case this relation specifier is just an identifier, the expression is just the named relation; this is the basis of the induction. In each expression, the relation could be renamed due to labeling (see above). Then we can associate a nested algebra expression with a relation list (see *relation-list* in Figure 1) as follows: take the cartesian product of all the relations represented in the relation list.

- Consider

  ```
  select *
  from relation-list
  ```

  This query is equivalent with the nested algebra expression associated with *relation-list*.

- Let $(\Theta, \theta) \in \{(=,=), (\texttt{subset of}, \subseteq)\}$. Consider

[2] Such an identifier might be given in the data definition language, but this goes beyond the scope of this paper.

```
select *
from relation-list
where attribute-spec_1 Θ attribute-spec_2
```

If *attribute-spec$_i$* represents the attribute X_i, and if E is the nested algebra expression associated *relation-list*, this query is equivalent with $\sigma_{X_1 \theta X_2}(E)$.

- The query

  ```
  select *
  from relation-list
  where attribute-list element of attribute-spec
  ```

  is equivalent to $\sigma_{X \in Z}(E)$, where E is as above, X is the set of attributes represented by *attribute-list* and Z is the attribute represented by *attribute-spec*.

  The attentive reader will observe that, although we only defined selections with respect to equality, we use in this and the previous item selections with respect to set membership and inclusion. In an appendix we show that such selections can be expressed in the nested algebra.

- The result of the query

  ```
  select *
  from relation-list_1
  where exists
    (select  select-list
    from relation-list_2
    where condition)
  ```

  is equal to the result of the query

  ```
  select  select-list_1
  from relation-list_1,relation-list_2
  where condition
  ```

  Here, *select-list*$_1$ is a list representing the scheme of the nested algebra expression associated with *relation-list*$_1$.

- Let $(\Delta, \delta) \in \{(\texttt{and}, \cap), (\texttt{or}, \cup)\}$. The query

  ```
  select *
  from relation-list
  where condition_1 Δ condition_2
  ```

  is equivalent with $E_1 \; \delta \; E_2$, where E_i is the nested algebra expression associated with the query

  ```
  select *
  from relation-list
  where condition_i
  ```

- The query

  ```
  select *
  from relation-list
  where not condition
  ```

  is equivalent with $E - E'$ where E is the nested algebra expression associated with *relation-list*, and E' is the nested algebra expression associated with the query

  ```
  select *
  from relation-list
  where condition
  ```

- Let Q denote a *simple query* (see syntax of $\mathcal{N}$SQL, Figure 1). Consider the query

  ```
  Q
  group by group-list
  ```

  Let G be the set of attributes represented by *group-list*; if *group-list* is empty, put $G = \emptyset$. Then this query is equivalent with $\nu_{\Omega - G}(E)$ where E is the nested algebra expression associated with Q, and Ω is the resulting scheme of E.

- Let Q denote an *elementary query* (see Figure 1), with `select`-clause of the form

```
select select-list
```

Let Q' be the elementary query obtained from Q by replacing *select-list* with "`*`", and let E' be the nested algebra expression associated to Q', with resulting scheme Ω. Furthermore let X be the set of attributes represented by *select-list*. The set Y is defined as consisting of those composed attributes Z appearing in Ω such that Z contains an attribute of X. If $Y = \{Z_1, \ldots, Z_m\}$, the query Q is equivalent with

$$\pi_X \overline{\mu}_{Z_1} \cdots \overline{\mu}_{Z_m}(E')$$

- Finally, let $(\Phi, \phi) \in \{(\texttt{difference}, -), (\texttt{union}, \cup)\}$, and let Q_1 and Q_2 be two $\mathcal{N}$SQL queries. Then the query

  Q_1
  Φ
  Q_2

  is equivalent with $E_1 \; \phi \; E_2$ where E_i is the nested algebra expression associated with Q_i.

So, with every $\mathcal{N}$SQL-query a nested algebra expression can be associated. Rather straightforwardly, the meaning of an $\mathcal{N}$SQL-query is said to be well-defined iff the associated nested algebra expression is well-defined.

Comparing the formal query systems (the algebras) of the relational and the nested relational model, it is quite clear that in extending a relational language to a nested language, it is important to provide for nest and unnest mechanisms. In the case of $\mathcal{N}$SQL, nesting is achieved by means of an extension of the `group by` facility in SQL. Actually, the SQL `group by` facility already does a kind of nesting; the problem however is that nesting does not fit in the formalism of flat relations. Furthermore, $\mathcal{N}$SQL provides for some kind of "automatic unnesting": if a lower-level attribute is needed, simply listing it in the `select`-clause will cause repeated unnesting down to the desired level. So $\mathcal{N}$SQL really can be seen as a natural adaption of the SQL query system to the context of nested relations.

4 The expressiveness of $\mathcal{N}$SQL

We now turn to the expressiveness of $\mathcal{N}$SQL. We establish:

THEOREM *$\mathcal{N}$SQL and the nested algebra are equivalent with respect to the queries they can express.*

Since $\mathcal{N}$SQL was defined in terms of the nested algebra, every $\mathcal{N}$SQL-query is expressible in the nested algebra. So we only have to prove that also conversely, $\mathcal{N}$SQL can express the full range of nested algebra queries. The proof proceeds inductively on the size of nested algebra expressions.

As basis for the induction, an arbitrary fixed relation r is represented in $\mathcal{N}$SQL by

```
select *
from r
```

Suppose now that the nested algebra expressions E, E_1, E_2 are expressed by the $\mathcal{N}$SQL-queries Q, Q_1, Q_2. Ω is the resulting scheme of E.

- Let $(\Phi, \phi) \in \{(\texttt{union}, \cup), (\texttt{difference}, -)\}$. Then $E_1 \, \phi \, E_2$ is expressed by

  ```
  Q1
  Φ
  Q2
  ```

- $E_1 \times E_2$ is expressed by

  ```
  select *
  from (Q1),(Q2)
  ```

- Let $X = \{A_1, \ldots, A_m\} \subseteq \Omega$. $\pi_X(E)$ is expressed by

  ```
  select A1, ..., Am
  from (Q)
  ```

- With X as above, assume $\Omega - X = \{B_1, \ldots, B_k\}$. $\nu_X(E)$ is expressed by

```
select *
from (Q)
group by B_1, ..., B_k
```

- Let $Z = \{A_1, \ldots, A_m\} \in \Omega - U$, and assume $\Omega - \{Z\} = \{B_1, \ldots, B_k\}$. $\mu_Z(E)$ is expressed by

```
select A_1, ..., A_m, B_1, ..., B_k
from (Q)
```

- Finally let $X_1, X_2 \in \Omega$. $\sigma_{X_1=X_2}(E)$ is expressed by

```
select *
from (Q)
where X_1=X_2
```

Hence, every nested algebra expression has an equivalent counterpart in $\mathcal{N}$SQL. An important observation is that in the above reduction of the nested algebra to $\mathcal{N}$SQL, the `exists` facility, allowing emptyness test of subqueries in the `where`-clause, is not used. However, the ability to have subqueries in the `from`-clause is a key technique in the proof. This means that the construct of `exists`-subqueries does not add to expressive power in the presence of the construct of subqueries in the `from`-clause. Stated in another way, allowing subqueries in `from`-clauses makes the construct of subqueries in `where`-clauses redundant with respect to expressive power. This is an interesting property: in SQL, this construct *is* necessary in order to obtain full expressiveness, for it is shown in [16] that certain queries in the relational algebra cannot be expressed in SQL without using a subquery in the `where`-clause. On the other hand, it should be pointed out that `union`- and `difference`-like queries can be expressed using `where`-queries. So, these constructs in turn can be called redundant in the presence of `where`-subqueries. Therefore, it is preferrable to retain *all* the discussed facilities in the language $\mathcal{N}$SQL, as is done.

5 Conclusion

The relational query language SQL was extended to deal with nested relations. Emphasis was put on formal definitions, in analogy with the extension of the relational algebra to the nested algebra. Care was taken that the obtained extension, $\mathcal{N}$SQL, respects the original philosophy behind SQL as faithfully as possible.

$\mathcal{N}$SQL is of course not meant to be an operational database language. Only a representative subset of SQL, containing the most fundamental aspects, is extended. The language as it stands now is just a theoretical realization of the standard nested algebra, and does not allow an effective manipulation of nested relations. In order to allow for this, operations of *extended nested algebras*, such as nested application of projection [17] and more general, flexible selection conditions have to be incorporated. Other important aspects from a more practical point of view, such as aggregate functions and arithmetic, should not cause too much problems to be integrated in $\mathcal{N}$SQL. A more fundamental issue for further research, as indicated in the introduction, is to capture more object-oriented concepts, especially *object identity*. This problem is considerably more difficult, because up to now there is no appropriate generally accepted and well-defined formalism available yet, as it is for nested relations. The author is presently working on these subjects.

Acknowledgement

Jan Paredaens and Marc Gyssens are kindly acknowledged for their helpful comments and stimulating support.

References

[1] S. Abiteboul, C. Beeri: "On the power of languages for the manipulation of complex objects", *INRIA technical report 846*, May 1988.

[2] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, S. Zdonik: "The object-oriented database system manifesto", *Proceedings 1st DOODS*, Kyoto, December 1989.

[3] F. Bancilhon: "Object-oriented database systems", *Proceedings 7th PODS*, pp. 152–162, Austin, March 1988.

[4] J. Banerjee, H. Chou, J. Garza, W. Kim, D. Woelk, N. Ballou, H. Kim: "Data model issues for object-oriented applications", *ACM TOIS 5,1*, pp. 3–26, January 1987.

[5] C. Beeri: “Data models and languages for databases”, *Proceedings 2nd ICDT*, pp. 19–41, Bruges, August/September 1988.

[6] C. Beeri et al.: “Sets and negation in a logic database language (LDL1)”, *Proceedings 6th PODS*, pp. 21–37, San Diego, March 1987.

[7] E. Codd: “A relational model for large shared databanks”, *Communications of the ACM 13,6*, pp. 377–387, June 1970.

[8] E. Codd: “Further normalization of the data base relational model”, *Data base systems*, R. Rustin, ed., Prentice-Hall, 1972.

[9] E. Codd: “Relational completeness of data base sublanguages”, *Data base systems*, R. Rustin, ed., Prentice-Hall, 1972.

[10] C. Date: “A critique of the SQL database language”, *ACM SIGMOD Record 14,3*, pp. 8–54, 1984.

[11] M. Gyssens, J. Paredaens, D. Van Gucht: “A grammar-based approach towards unifying hierarchical data models”, *Proceedings ACM SIGMOD 89*, pp. 263–272, June 1989.

[12] M. Gyssens, D. Van Gucht: “The powerset operator as a result of adding programming constructs to the nested algebra”, *Proceedings ACM SIGMOD 88*, pp. 225–232, June 1988.

[13] R. Hull, R. King: “Semantic database modelling: Survey, applications, and research issues”, *ACM Computing surveys 19,3*, pp. 201–260, 1987.

[14] D. Maier, D. Price: “Data model requirements for engineering applications”, *Proceedings 1st IEEE international workshop on expert database systems*, pp. 759–765, 1984.

[15] M. Roth, H. Korth, D. Batory: “SQL/NF: A query language for ¬1NF relational databases”, *Information systems 12,1*, pp. 99–114, 1987.

[16] J. Paredaens, P. De Bra, M. Gyssens, D. Van Gucht: “The structure of the relational database model”, *EATCS Monographs on theoretical computer science*, W. Brauer, G. Rozenberg, A. Salomaa, eds., Springer-Verlag, 1989.

[17] H.-J. Scheck, M. Scholl: “The relational model with relation-valued attributes”, *Information systems 11,2*, pp. 137–147, 1986.

[18] D. Shipman: “The functional data model and the data language DAPLEX”, *ACM TODS 6,10*, pp. 140–173, 1981.

[19] S. Thomas, P. Fischer: “Nested relational structures”, *The theory of databases*, P. Kanellakis, ed., pp. 269–307, JAI Press, 1986.

[20] J. Ullman: “Principles of database and knowledge-base systems, Volume I”, Computer Science Press, 1988.

[21] J. Van den Bussche: “Query Systems in the Nested Relational Database Model”, *Graduate dissertation UIA*, 1989.

Appendix

We show that two additional kinds of selection operations, *inclusion selection* and *membership selection*, are expressible using the standard operations in the nested algebra. See also [12].

Let (Ω, ω) be a relation. Renaming operations are formally defined as follows: let ψ be a permutation of U. ψ can be extended to composed attributes, tuples and instances in a canonical way (make ψ the identity on V). The renaming $\rho^{\psi}(\Omega, \omega)$ equals $(\psi(\Omega), \psi(\omega))$.

Let $X, X' \in \Omega - U$. Suppose there exists a permutation φ on U such that $X' = \varphi(X)$ (φ is extended to composed attributes, tuples and instances in the canonical way). Then the inclusion selection $\sigma_{X \subseteq X'}(\Omega, \omega)$ equals (Ω, ω'), where

$$\omega' := \{t \in \omega \mid \varphi(t(X)) \subseteq t(X')\}$$

If ψ is a permutation on U such that Ω and Ω^{ψ} have no atomic attributes in common, the reader is invited to check the following equality:

$$\begin{aligned}\sigma_{X \subseteq X'}(x) = \\ &\pi_{\Omega}\sigma_{X=X^{\psi}}\nu_{X^{\psi}}\pi_{\Omega \cup X^{\psi}}\sigma_{X^{\psi}=X'_{\psi}}\mu_{X'_{\psi}}\mu_{X^{\psi}}\sigma_{\Omega=\Omega^{\psi}}(x \times \rho^{\psi}(x)) \\ &\qquad \cup \pi_{\Omega}\sigma_{X=X^{\psi}}(x \times (\{X^{\psi}\}, \{\emptyset\}))\end{aligned}$$

Thus, inclusion selection is expressible in the nested algebra.

Now let $X \subseteq \Omega$, and suppose $X^\varphi \in \Omega$ for some permutation φ on U. The membership selection $\sigma_{X \in X^\varphi}(\Omega, \omega)$ equals (Ω, ω'), where

$$\omega' := \{t \in \omega \mid \varphi(t|_X) \in t(X^\varphi)\}$$

If ψ is as above, a nested algebra expression for $\sigma_{X \in X^\varphi}(x)$ is

$$\pi_\Omega \sigma_{X^\psi = X^{\psi\varphi}} \mu_{X^{\psi\varphi}} \sigma_{\Omega = \Omega^\psi}(x \times \rho^\psi(x))$$