Copyright ©2006 Timothy Howard Merrett
Permission to make digital or hard copies of part or all of
this work for personal or classroom use is granted without fee
provided that copies are not made or distributed for profit or
commercial advantage and that copies bear this notice and
full citation in a prominent place. Copyright for components
of this work owned by others than T. H. Merrett must be
honoured. Abstracting with credit is permitted. To copy
otherwise, to republish, to post on servers, or to redistribute
to lists, requires prior specific permission and/or fee. Re-
quest permission to republish from: T. H. Merrett, School
of Computer Science, McGill University, fax 514 398 3883.

The author gratefully acknowledges support from the tax-
payers of Québec and of Canada who have paid his salary
and research grants while this work was developed at McGill
University, and from his students (who built the implemen-
tations and investigated the data structures and algorithms)
and their funding agencies.

T. H. Merrett
Semistructure from Relations
T. H. Merrett
McGill University

Part I
- A *programming language talk disguised as a database talk.*
- *Why relations?*
- *Why semistructure?*

Part II
- *Relations and path expressions.*

Part III
- *Irregular and unknown structure.*

Part IV
- *Markup and data on the web.*

Integration. Integration. Integration.

Dedicated to the late Alberto Oscar Mendelzon

T. H. Merrett
Part I  A PL talk disguised as a DB talk

• Not top-down (e.g., functional, logic, constraint paradigms).

• But bottom-up: programming for secondary storage.

• But not, for the moment, about
  – performance or optimization (NB terabytes or more),
  – compiler flexibility or debugging.
Why relations?

- Unit for bulk data:
  the essence of SS programming
  (hence necessary for DB).

- High-level operations (relational algebra).

- Abstraction over looping:
  like LISP, APL, SETL, .. but for SS.
Why semistructure?

• Bottom-up approach is empirical, not theoretical.

• So language must be tested in many applications.

• Semistructured data is a DB hot topic, but sufficiently worked out to provide a good test.

What is semistructure?

• Characterized by irregular or unknown structure.

• Notable language feature is path expressions.
Part II  Relations and path expressions

• Paths of attributes.

• Paths of conditions.

• Paths for updates.

T. H. Merrett ©06/2
Part II Relations and path expressions

Paths of attributes

• Operations on relations.

• Operations on attributes.

• Nested relations and level changes.

• Paths of attributes.
Part II Relations and path expressions

Paths of attributes: relations

Family tree example 1.

\[
Child(\text{Name} \ \ \text{DoB} \ \ \text{Pa} \ \ \text{Ma} \ )
\]

- Mary 1934 Ted Alice
- James 1935 Ted Alice
- Joe 1933 Max Sal

Unary operators.

\[
ChildND \leftarrow [\text{Name}, \text{DoB}] \ \text{where} \ \text{DoB} > 1933 \ \text{in} \ Child;
\]

\[
ChildND(\text{Name} \ \ \text{DoB} \ )
\]

- Mary 1934
- James 1935

\[
\text{Name in Child} \equiv [\text{Name}] \ \text{in} \ Child
\]

- (\text{Name} )
  - Mary
  - James
  - Joe

T. H. Merrett
Part II Relations and path expressions

Paths of attributes: relations, cont.

Family tree example 1.

\[
\text{Child}(\text{Name} \quad \text{DoB} \quad \text{Pa} \quad \text{Ma} \quad ) \\
\quad \text{Mary} \quad 1934 \quad \text{Ted} \quad \text{Alice} \\
\quad \text{James} \quad 1935 \quad \text{Ted} \quad \text{Alice} \\
\quad \text{Joe} \quad 1933 \quad \text{Max} \quad \text{Sal}
\]

\[
\text{Spouse}(\text{Ma} \quad \text{Pa} \quad \text{Wed} \quad ) \\
\quad \text{Alice} \quad \text{Ted} \quad 1933
\]

Binary operators. Note: infixed syntax.

\(([\text{Ma}, \text{Pa}] \text{ in Child}) \text{ ujoin } [\text{Ma}, \text{Pa}] \text{ in Spouse}\)

\[
(Ma \quad Pa \quad ) \\
\quad \text{Alice} \quad \text{Ted} \\
\quad \text{Sal} \quad \text{Max}
\]

djoin, ijoin, ..
Part II Relations and path expressions

Paths of attributes: attributes

Domain algebra: \texttt{red, equiv}

\texttt{let Oldest be red min of DoB; equiv min of DoB by Pa equiv min of DoB by Ma, Pa}

Family tree example 1.

\begin{tabular}{llllllll}
\hline
\textit{Child} & (\textit{Name} & \textit{DoB} & \textit{Pa} & \textit{Ma} ) & \textit{Oldest} & [\textit{min Pa}] & [\textit{min Ma,Pa}] \\
\hline
Mary     & 1934 & Ted & Alice & 1932 & 1932 & 1934 \\
James    & 1935 & Ted & Alice & 1932 & 1932 & 1934 \\
Joe      & 1933 & Max & Sal  & 1932 & 1933 & 1933 \\
Pete     & 1932 & Ted & Sal  & 1932 & 1932 & 1932 \\
\hline
\end{tabular}

T. H. Merrett
Part II Relations and path expressions

**Paths of attributes: nesting**

Domain algebra subsumes relational algebra.

\[
\text{let } \text{ChildN be } [\text{Name}] \text{ in Children;}
\]

\[
fam\text{ChildN } \leftarrow [\text{ChildN}] \text{ in Family;}
\]

\[
\Rightarrow fam\text{ChildN}(\text{ChildN}(\text{Name}))
\]

Level-raising through anonymous singleton

\[
\text{FamChildN } \leftarrow [\text{red ujoin of ChildN}] \text{ in Family;}
\]

Family tree example 2.

<table>
<thead>
<tr>
<th>Family</th>
<th>Pa</th>
<th>Wed</th>
<th>Name</th>
<th>DoB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Ted</td>
<td>1933</td>
<td>Mary</td>
<td>1934</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>James</td>
<td>1935</td>
</tr>
<tr>
<td>Sal</td>
<td>Ted</td>
<td>1930</td>
<td>Pete</td>
<td>1932</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Pete</td>
</tr>
</tbody>
</table>

Path expression

\[
\text{Family/ChildN } \equiv [\text{red ujoin of ChildN}] \text{ in Family}
\]

T. H. Merrett
Part II Relations and path expressions

**Paths of attributes**

### Family tree example 3

<table>
<thead>
<tr>
<th>Person</th>
<th>Family</th>
<th>Wed</th>
<th>Children</th>
<th>DoB</th>
<th>Family</th>
<th>Wed</th>
<th>Children</th>
<th>DoB</th>
<th>Family</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ted</td>
<td>Alice</td>
<td>1933</td>
<td>Mary</td>
<td>1934</td>
<td>Max</td>
<td>1956</td>
<td>Sue</td>
<td>1957</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Tom</td>
<td>1958</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>James</td>
<td>1935</td>
<td>Ann</td>
<td>1959</td>
<td>Joe</td>
<td>1960</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sal</td>
<td>1930</td>
<td>Pete</td>
<td>1932</td>
<td></td>
<td></td>
<td>Sal</td>
<td></td>
<td>Pete</td>
</tr>
</tbody>
</table>

\[
[\text{red ujoin of}\]

\[
[\text{red ujoin of}\]

\[
[\text{Name} \ \text{in}\]

\[
[\text{Children} \ \text{in}\]

\[
[\text{Family} \ \text{in}\]

\[
\text{Person}
\]

T. H. Merrett

©06/2
Part II Relations and path expressions

**Paths of attributes (cont.)**

(Family tree example 3.)

Option

\[
\text{Person}(/\text{Family} /\text{Children})?/\text{Name} \equiv \text{Name in Person} \text{ ujoin}
\]

\[
[\text{red ujoin of}]
\]

\[
[\text{red ujoin of}]
\]

\[
[\text{Name} \text{ in}]
\]

\[
\text{Children} \text{ in}
\]

\[
\text{Family} \text{ in}
\]

\[
\text{Person}
\]

Kleene Star (recursive domain algebra)

\[
\text{Person}(/\text{Family} /\text{Children})*/\text{Name} \equiv
\]

\[
\text{let Nom be Name ujoin}
\]

\[
[\text{red ujoin of}]
\]

\[
[\text{red ujoin of}]
\]

\[
\text{Nom} \text{ in}
\]

\[
\text{Children} \text{ in}
\]

\[
\text{Family};
\]

\[
[\text{red ujoin of} \text{ Nom} \text{ in} \text{ Person}
\]

T. H. Merrett

©06/2
Paths of conditions

Family tree example 1.

\[
\text{Child}(Name \ DoB \ Pa \ Ma)
\]

Mary 1934 Ted Alice
James 1935 Ted Alice
Joe 1933 Max Sal

Nullary relation is Boolean

\[
[\ ] \text{in Child} \equiv \text{“something in Child”} \equiv \text{“there is a child”}
\]

\[
[\ ] \text{where Name = "Joe" in Child}
\]

true

Path expression

(Family tree example 3).

Name where Family/Children/Name = "Mary" in Person \equiv

Name where

([ ] where

([ ] where Name = "Mary" in Children) in Family) in Person

T. H. Merrett
Part II Relations and path expressions

Paths of conditions, cont.

Recursive path expression

\[
\text{Name where } (\text{Family} / \text{Children} /)^* \text{Name} = "Mary"
\]

\[
\text{in Person } \equiv \text{func mary is}
\]

\{
\begin{align*}
\text{Name} &= "Mary" \text{ or} \\
\text{[ ] where} & \\
\text{[ ] where mary in Children} & \\
\text{in Family})
\end{align*}
\}

\[
\text{Name where mary in Person}
\]

NB \textbf{and, xor}, etc. have no syntactic sugar.
Part II Relations and path expressions

Paths for updates

Family tree example 1.

\[ \text{Child} \begin{pmatrix} \text{Name} & \text{DoB} & \text{Pa} & \text{Ma} \end{pmatrix} \]

<table>
<thead>
<tr>
<th>Name</th>
<th>DoB</th>
<th>Pa</th>
<th>Ma</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>1934</td>
<td>Ted</td>
<td>Alice</td>
</tr>
<tr>
<td>James</td>
<td>1935</td>
<td>Ted</td>
<td>Alice</td>
</tr>
<tr>
<td>Joe</td>
<td>1933</td>
<td>Max</td>
<td>Sal</td>
</tr>
</tbody>
</table>

update \text{Child} change

\[ \text{DoB} \leftarrow \text{if Name = "Mary" then "1933" else DoB;} \]

\[ \text{Child} \begin{pmatrix} \text{Name} & \text{DoB} & \text{Pa} & \text{Ma} \end{pmatrix} \]

<table>
<thead>
<tr>
<th>Name</th>
<th>DoB</th>
<th>Pa</th>
<th>Ma</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>1933</td>
<td>Ted</td>
<td>Alice</td>
</tr>
<tr>
<td>James</td>
<td>1935</td>
<td>Ted</td>
<td>Alice</td>
</tr>
<tr>
<td>Joe</td>
<td>1933</td>
<td>Max</td>
<td>Sal</td>
</tr>
</tbody>
</table>
Part II Relations and path expressions

Paths for updates, cont.

Path expression
(Family tree example 3).

\begin{verbatim}
update Person/Family/Children change
  DoB <- if Name = "Mary" then "1933"
       else DoB; ≡
update Person change
update Family change
update Children change
  DoB <- if Name = "Mary" then "1933"
       else DoB;
\end{verbatim}
Recursive path expression

```plaintext
update Person(/Family/Children)* change
   DoB <− if Name = "Mary" then "1933"
   else DoB; ≡
proc mary33 is
{  DoB <− if Name = "Mary" then "1933"
    else DoB;
    if [ ] in Family then update Family change
    if [ ] in Children then
        update Children change mary33;
};
update Person change mary33;
```

T. H. Merrett
Part III Irregular and unknown structure

- Schema query and update. 
  **Transpose** metadata operator, originally devised for association data mining.

- Missing and multiple values.

- Wildcards.

- Schema discovery.

T. H. Merrett  ©06/2
Part III Irregular and unknown structure

Schema query and update.

Union type
Family tree example 4.

\[
\text{domain } \text{DoB str} | \text{intg}; \\
\quad \text{Child}(\text{Name } \text{DoB } \text{Pa } \text{Ma }) \\
\quad \text{Mary } \text{intg:1934 Ted Alice} \\
\quad \text{James } \text{strg:1935 Ted Alice}
\]

Transpose operator
\[
\text{domain } \text{att attr; } \\
\text{domain } \text{typ type; } \\
\text{domain } \text{val any; } \\
\text{let } \text{xpose be transpose}(\text{att, typ, val}); \\
\text{transposeChild } \leftarrow \\
\quad [\text{Name, DoB, Pa, Ma, xpose}] \text{ in Child; }
\]

\[
\text{transposeChild} \\
\quad (\text{Name } \text{DoB } \text{Pa } \text{Ma } \text{xpose} ) \\
\quad (\text{att } \text{typ } \text{val} ) \\
\quad \text{Mary } \text{intg:1934 Ted Alice Name strg strg:Mary} \\
\quad \quad \text{DoB intg intg:1934} \\
\quad \quad \text{Pa strg strg:Ted} \\
\quad \quad \text{Ma strg strg:Alice} \\
\quad \text{James } \text{strg:1935 Ted Alice Name strg strg:James} \\
\quad \quad \text{DoB strg strg:1935} \\
\quad \quad \text{Pa strg strg:Ted} \\
\quad \quad \text{Ma strg strg:Alice}
\]
Part III Irregular and unknown structure

Schema query and update, cont.

Query on structure
Find all integer dates of birth

\[ \text{intgDoB} \leftarrow \text{where } \text{xpose/att} = \text{quote DoB and } \text{xpose/typ} = \text{intg in Child}; \]

\[ \text{intgDoB} \]
\[ (\text{Name } \text{DoB } \text{ Pa } \text{ Ma } ) \]
\[ \text{Mary intg:}1934 \text{ Ted Alice} \]

Update on structure

\textbf{domain DoB strg|intg;}

\textbf{Child(}\text{Name } \text{DoB } \text{ Pa } \text{ Ma } \text{)}

\[ \text{Mary intg:}1934 \text{ Ted Alice} \]
\[ \text{James strg:}1935 \text{ Ted Alice} \]

\textbf{update Child change DoB} \leftarrow \text{(strg)DoB }

using \text{where } \text{xpose/att} = \text{quote DoB and } \text{xpose/typ} = \text{intg in Child;}

\textbf{Child(}\text{Name } \text{DoB } \text{ Pa } \text{ Ma } \text{)}

\[ \text{Mary strg:}1934 \text{ Ted Alice} \]
\[ \text{James strg:}1935 \text{ Ted Alice} \]
Part III Irregular and unknown structure

Missing and multiple values

I By union type

domain child strg;
domain DoB intg;
domain Name strg;
domain chiln(Name, DoB);
domain Children child|chiln;
domain Conj strg;
domain Wed strg;

Family(Conj Wed Children )
Alice 1933 child:Bernice

relation Chiln(DoB,Name) <-
{(1934,"Mary"),(1935,"James")};

update Family/Children add Chiln ≡
update Family change
update Children add Chiln;

Family(Conj Wed Children )
Alice 1933 child: Bernice
chiln:
(Name DoB)
Mary 1934
James 1935

T. H. Merrett
Part III Irregular and unknown structure

**Missing and multiple values, cont.**

II By polymorphic relation

domain Conj strg;
domain Wed strg;
domain Child strg;
let Name be Child;
let Children be relation(Name);

Family(Conj Wed Child ) Name Children (Name)
    Alice  1933  Bernice  Bernice  Bernice

update Family change
replace Child with Children;
update Family/Children add Chiln

Family(Conj Wed Children (Name)
    Alice  1933  Bernice
    (DoB Name)
    1934 Mary
    1935 James

T. H. Merrett

©06/2
Part III Irregular and unknown structure

Wildcards

Family tree example 5.

\[
\text{FamEmp} = (\text{Name}, \text{Family}, \text{Employer})
\]

\[
(\text{Conj, Wed}) (\text{Boss, Conj, Subord})
\]

Ted Alice 1933 Pete Alan Carole

\[\text{famEmp} ./ . / \text{Conj} \equiv\]
\[\text{[red ujoin of [red ujoin of Conj in }\]
\[\text{.] in FamEmp}\]

...should give Alice, Alan:

T. H. Merrett
Part III Irregular and unknown structure

Wildcards, cont.

**Transpose** analyses leaves only:
\( \text{transposeAll}(att,typ) \) for non-leaf
as well as for leaf attributes.

\[
\text{let nonleaves be transposeAll}(att) \text{ djoin transpose}(att); \\
\]

\[
\begin{array}{ccc}
\text{FamEmp} & \text{Family} & \text{Employer} \\
\text{Name} & (..) & (..) \\
\end{array}
\]

\[
\text{let } FE \text{ be } \left[ \text{red ujoin of eval } att \right] \text{ in nonleaves}; \\
\text{famEmp} ./ ./ \text{Conj} \equiv \text{famEmp}/FE/\text{Conj} \equiv \\
\left[ \text{red ujoin of } \\
\left[ \text{red ujoin of Conj} \right] \text{ in } \\
FE \right] \text{ in FamEmp}
\]

(Conj)
Alice
Alan
Part III Irregular and unknown structure

Recursion and wildcards

\[ \text{Person} / / \text{Name} \equiv \text{Person}(./)^* / / \text{Name} \equiv \]
\begin{verbatim}
let Nom be Name ujoin
    [red ujoin of Nom] in .;
    [red ujoin of Nom] in Person;
\end{verbatim}

Family tree example 3

<table>
<thead>
<tr>
<th>Name</th>
<th>Conj</th>
<th>Wed</th>
<th>Children</th>
<th>DoB</th>
<th>Family</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ted</td>
<td></td>
<td></td>
<td>Mary</td>
<td>1934</td>
<td>Max</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Sue</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>James</td>
<td>1935</td>
<td>Ann</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Tom</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Pete</td>
<td>1932</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Joe</td>
</tr>
</tbody>
</table>

\(\text{Name} : \{(\text{Ted}), (\text{Mary}), (\text{James}), (\text{Pete}), (\text{Sue}), (\text{Tom}), (\text{Joe})\}\)
Part III Irregular and unknown structure

Schema discovery

```
Person
  (Name Family )
    (Conj Children )
      (Name Family )
        (Conj Children )
          (Name )

let attrib be self;
let schema be transpose(attrib) union
  [attrib, schema] in .;
Schema ← [attrib, schema] in Person;
```

```
Schema
  (attrib schema)
    (attrib schema)
      (attrib schema)
        (attrib schema)
          (attrib)

Person
  Name
  Family Conj
  Children Name
    Family Conj
      Children Name
```

T. H. Merrett  ©06/2 27
Part IV

Markup and data on the web.

Semistructure/text

• Specialized operator, **mu2nest**: marked-up $\rightarrow$ nest, including order information.

• Text querying: metadata relational operator, **grep**.

Other applications

• Data Streams? Skyline?

Highlights

• Binary operators must be infixed.

• Nullary relations are Boolean.

• Need domain algebra as well as relational algebra.

• Domain algebra subsumes relational algebra for nesting.

• Nesting => recursive nesting (no 2nd class).

• Metadata is important for advanced work:
  – transpose, transposeAll, quote, eval, self
Conclusion

• Thinking relations through carefully →

• Subtle adjustments to query syntax →

• General purpose SS programming language →

• Everything XML and XQuery can do and more.

Integration. Integration. Integration.