The author gratefully acknowledges support from the taxpayers 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 implementations and investigated the data structures and algorithms) and their funding agencies.

Relational Database Programming in Aldat

T. H. Merrett

McGill University

1. Chronology

2. Start with Databases — abstracting tuples, loops, grouping, & ordering

3. On to Programming Language — computations and predictive modelling; procedures and event programming

Chronology I Secondary Storage

... fixed length

1978 Static Multipaging

$S = N \ldots 2N$ $\quad$ $< T >= 1 \ldots 2$

1982 Dynamic Multipaging

$S = N \ldots 2N$ $\quad$ $< T >= 1 \ldots 2$

1983 Z-Order

$S = N$

1984 Tidy functions

$S = N \ldots 2N$ $\quad$ $< T >= 1 \ldots 2$

... variable length

1994 Tries for text

$S \approx 3N$ $\quad$ $< T >= \mid result \mid$

... fixed length

1994 Tries for maps

$S \approx 0.1N$ $\quad$ $< T >= \mid result \mid$

1996 Tries for orthogonal range queries

$S \approx 0.1N$ $\quad$ $< T >= \mid result \mid$

Chronology II
Programming Languages

	P.L.	**D.B.**
1975	iterator	relation
1984	recursion	fixed point
1992	scope	multidatabases
1992	process syncr.	selector
1992	reflection	metadata
1993	inheritance	join
1994	aggregation	join
1994	function	relation
1995	procedure	event
1995	A.D.T.	nesting
199?	type	constraint

Start with Databases

(Relix – SQL) =

no tuples, only relations

Difference I

iteration $\longrightarrow$ high level ops

Difference II

grouping $\longrightarrow$ high level ops

Difference III

ordering $\longrightarrow$ high level ops

Difference I — no iterators

a) Relational Algebra

StuCour(*Stu*	*Cour*	*Asst*	*Exam*)
Joe	Algol	45	42
Sue	Algol	47	38
Joe	Pascal	43	42
Sue	Pascal	45	42

- T-Selector is a loop:
  JoeCour <− [*Cour*,*Asst*,*Exam*] **where** *Stu*= `"Joe"` **in** *StuCour*

JoeCour(*Cour*	*Asst*	*Exam*)
Algol	45	42
Pascal	43	42

Difference I — no iterators

b) Domain Algebra

JoeCour(*Cour*	*Asst*	*Exam*)
Algol	45	42
Pascal	43	42

- Horizontal operations are a loop:
  let *Final* **be** *Asst* + *Exam*

- Vertical operations are a loop:
  let *Avg* **be** (**red** + **of** *Final*)/ (**red** + **of** 1)

$$JoeMark <\!- [Avg] \textbf{ in } JoeCour$$

JoeMark(*Avg*)
86

Difference II — grouping

a) Domain Algebra

StuCour(*Stu*	*Cour*	*Asst*	*Exam*)
Joe	Algol	45	42
Sue	Algol	47	38
Joe	Pascal	43	42
Sue	Pascal	45	42

- Vertical operations support grouping:
  let *Total* **be equiv** + **of** *Final* **by** *Cour*

TotMark <−[*Cour*,*Total*] **in** *StuCour*

TotMark(*Cour*	*Total*)
Algol	172
Pascal	172

Difference II — grouping

b) Relational Algebra

- Generalizing *division*, plus *recursion*, gives a one-line *inference engine*. (*Horn* is rulebase; *Facts* are known facts.)

NewFacts **is** *Facts* **ujoin** [*Concl*] **in** (*New-Facts*[*Concl* $\supseteq$ *Ante*]*Horn*

- Expands to a 50-line inference engine in a 200-line expert system shell, Relixpert (1987; 1 man-month; subsumes commercial shells AUGMENT, VP-Expert).

Difference III — ordering
a) Domain Algebra

- Here is a planimeter for geographical areas, using Stokes' Theorem

let X' **be par succ of** X **order** *Seq* **by** *Group*
let Y' **be par succ of** Y **order** *Seq* **by** *Group*
let *Area* **be equiv** $+$ of $(X \times Y' - Y \times X')/2$ **by** *Group*

- Similar calculations can find centroid, slope of an area in 3d, etc.

On to Programming Language

Computations

- Unify base concepts of DB and PL
- In mathematics, functions $\subseteq$ relations
- Let's generalize PL "function"

comp *Sum*(A, B, C) **is**

$C <\!\!- A + B$

alt $A <\!\!- C - B$

alt $B <\!\!- C - A$

- And let's invoke it by relational algebra

$T <\!\!-$ $[C]$ **where** $A = 2$ & $B = 2$ **in** *Sum*

$T(C)$	$R(A$	$B)$	$S(A$	B	$C)$
4	5	2	5	2	7
	6	3	6	3	9

$S <\!\!-$ *Sum* **ijoin** R

Computations and Predictive Modelling

- In addition to *data*, in relations like R,

- we now have *rules*, in computations like *Sum*.

- So as well as knowing the past,

- we can predict the future.

- All using the relational algebra.

For instance, we could have tree population data,
Pop
and a population model,
PopComp
and combine them in a database which can look up both past and future populations.

Computations and Interoperability

- Free-standing computations serve as procedures.

- (They have different invocation syntax.)

- *Events* are system invocations of procedures written by the programmer.

- Interoperability of diverse GIS (DB) across a network could use events as the common interface.

- Event programming is easily built into Aldat.

On to Programming Language II
Complex Objects

- A basic precept of PL is "no 2nd-class types".

- So values in relations should not be restricted.

- So relations can contain relations: "complex objects".

- How do we support this in the language?

The domain algebra subsumes the relational algebra:
any attribute type can be relational,
so relational operations are available for attributes.