Comments on
“Third Manifesto Concerns”

The slides are intended to provide feedback about concerns arising from the implementation of *The Third Manifesto* via the RAQUEL DBMS.

Scalar Types
The Third Manifesto (= *TTM*) has 4 categories of scalar data types that the DBMS must handle (although the ‘Plug-In’ category is not altogether clear).

It has been found simpler to have only 2 categories : scalar types that have possreps, and scalar types that don’t have possreps. The origin of a scalar type, be it user-defined or system-defined, is ignored.

It is considered better for the user because they will only need to consider 2 kinds of representation of scalar values :
1. those that use a Selector (and may have THE_ operators),
2. those that don’t use a Selector but rely on the innate representation universally used for a value of that type (e.g. the decimal notation for a number such as 372).

This is all the RAQUEL user needs to know.

It also makes the DBMS implementation simpler, because the DBMS can use the same method for every Selector representation, and only provide a means of handling every innate representation, which is logically necessary anyway[*].

Scalar Types : Logical Values Only
In keeping with *TTM*, scalar types should not involve physical representations of values within the logical model.

This contrasts with what typically happens in programming languages, say for numbers. A 32-bit integer number is typically considered to be of a different type to that of a 16-bit or 64-bit integer number. However a *TTM* DBMS must provide a logical set of permissible integer values as a type. If some attribute values of that type are stored internally as 32-bit integers and others as16-bit or 64-bit integers, and they are (say) added together, the DBMS must be able to add a 16-bit integer to a 64-bit integer and not fail with a type error.

Rational Numbers
It is unclear how they relate to ‘Real’ or ‘Floating Point’ numbers.

What representation is assumed ? The value ‘one third’ can be precisely stored in a Trinary representation but not in a Decimal representation.

[*] The only scalar types with innate representation that the RAQUEL notation currently supports are numbers, text, and the truth value ***true*** and ***false***.

Relational Truth Values
It would be useful to have a clear mapping between the scalar truth values, ***true*** and ***false***, and the relational truth values, DEE and DUM, respectively.

Consider a relvar *Emp* about employees that contains an attribute called 'Child' whose values are nested relation values (= relvalues) such that each tuple in a nested relvalue represents one child of an employee. It would be useful to express a relvalue representing all employees with children by

Emp **Restrict[** Child **Project[ ]]**

Project[ ] returns the relational truth value DEE for every nested relvalue containing one or more tuples, and DUM for every nested relvalue with no tuples. **Restrict** uses DEE and DUM, instead of ***true*** and ***false***, to determine which tuples of *Emp* appear in its result.

For a query concerning all employees located in Newcastle who have children, writing the following would be useful :-

Emp **Restrict[** Child **Project[ ] And (** Loc = 'Newcastle') **]**

The more relvars contain both nested relvalues and scalar values, the more significant the topic becomes.

View Updating to Support Assignments
The intent is to illustrate why it would be useful to have a comprehensive View Updating formalism as part of *TTM*.

In RAQUEL, insertions, deletions and amendments are made using assignments to a relvar. If an amendment (say) is to be made and it is to be assigned to only a subset of a relvar's tuples, then a Restriction defining that subset must appear to the left of the assignment. This is called an "Unnamed View" in RAQUEL because it corresponds to an unnamed view definition. It is called a Pseudovariable in *TTM*. Such Unnamed Views/Pseudovariables can be handled without view updating.

However RAQUEL's notation permits ***any*** meaningful Unnamed View/Pseudovariable to appear to the left of an assignment. For example assignments can be made to a Join of two or more relvars. To exploit the power of this notation to the full requires a comprehensive View Updating formalism in order to implement it.