# Presentation at SSDBM 2001

These slides are from the presentation given by Peter Gray at the Thirteenth International Conference on Scientific and Statistical Database Management, George Mason University, Fairfax, Virginia, USA.

- *Kemp, G.J.L., Gray, P.M.D. and Sjöstedt, A.R. (2001) Rewrite Rules for Quantified Subqueries in a Federated Database. In Kerschberg, L. and Kafatos, M. (eds.), Proceedings Thirteenth International Conference on Scientific and Statistical Database Management, IEEE Computer Society Press, pp 134-143.*

# Rewrite Rule Application

*Orig Query*

```
for each u in undergrad such that
    some c1 in takes(u) has code(c1) = 'C_331'
print(forename(u), surname(u));
```

*Rewritten Query*

```
for the u in takes_inv(the c2 in course such that
    code(c2) = 'C_331')
print(forename(u), surname(u));
```

The query now uses

- an *index* on `code` for direct access
- a *system-maintained inverse* `takes_inv` of the `takes` function;
- thus *avoiding enumeration*.

# Rewrite Rule Syntax

*with common*

```
    S in string
```

*rewrite*

```
   u in undergrad such that
       some c1 in takes(u) has code(c1) = S
```

*into*

```
   takes_inv(the c2 in course such that
                code(c2) = S);
```

*The ZF-expression (comprehension) in ICode is:*

```
u <- undergrad;
Exists{c1 | c1 <- takes(u); code(c1) = s}

u <- takes_inv({c2 | c2 <- course;
                     code(c2) = s})
```

- one or more `common` variables stand for well formed expressions denoting atomic values

(strings, integers, reals, booleans(predicate values)) or entity values (object identifiers).

- `rewrite` precedes a Daplex expression, usually denoting a set of objects, that can be described by a *ZF-expresssion*.

- rules are stored internally in P/FDM as Prolog term structures, unified by shared variables.

- rules are Horn-Clause rules based on unification and substitution, working top-down.

- assumes stratified rules

## Rewrite for Distributed Execution

Used to split the work up sensibly between two sites, and not send many *penny packet* queries. Assume `course` info on site 1 and `undergrad` info on site 2.

*Original Slow Version*

```
for each u in undergrad such that
    some c1 in course such that
        pred(c1) has level(c1) = year(u)
print(forename(u), surname(u));
```

*Rewritten Distributed Version:*

```
[Q1] print(level(c1 in course such that pred(c1)))

for each u1 in undergrad such that
    year(u1) in {RESULT OF Q1 e.g.{1,4,..}}
print(forename(u), surname(u));
```

Here Q1 need execute once only. This can be extended to more than 2 sites.

# Rewrite Rule Examples

Implemented by Unification in Prolog.

```
with common  u in undergrad
rewrite   ( some c1 in course such that pred(c1))
             has level(c1) = year( u)
into     year( u)   in
            level(c1 in course such that pred(c1))
```

Note that the rewritten expression denotes a predicate value and that the common expression denotes an `undergrad` entity.

```
with common  i in integer
rewrite      r in residue such that pos(r) =  i
into         absolutepos(chain, i)
```

```
with common  r in residue,
             st in integer,
             fin in integer
rewrite  some i in  st to  fin  has pos( r) =i
into   (pos( r) >=  st)   and (pos( r) =<  fin)
```

## Combination of rewrites

We can use the *generic rules* that flatten nested quantifiers, in combination with other *domain specific rules*, such as those that use indexes. Rewrite rules are used to replace iteration over residues by direct access.

```
with common i in integer
rewrite     r in residue such that pos(r) = i
into        absolutepos(chain,i);
```

We can then transform the following nested query:

```
for each c in structural_cdr such that
    name(c) = "L1"
    and some r in residue has
            name(r) = "CYS"
            and c in structural_cdr_domain_inv(
                      ig_domain_chain_inv(
                       residue_chain(r)))
            and pos(r) = start(c)
print(protein_code(domain_structure(
       structural_cdr_domain(c))));
```

as if it was written:

```
for each c in structural_cdr such that
    name(c) = "L1"
    and some r in absolutepos(chain,start(c)) has
            name(r) = "CYS"
            and c in structural_cdr_domain_inv(
                      ig_domain_chain_inv(
                       residue_chain(r)))
print(protein_code(domain_structure(
       structural_cdr_domain(c))));
```

## Generic Rewrites syntax extension

Allows Variables to denote *Predicate*, *Function or Class names* (recognised by a capital initial letter or an underscore)

*rewrite*

```
   U1 in ClassU such that
        some C in Rel(U1) has Prop(C) > Val
           and ( some U2 in Rel2(C) has
                    Prop2(U2) < Val2 )
```

*into*

```
   U1 in ClassU such that
        some C in Rel(U1) has Prop(C) > Val
           and Prop2( Rel2(C)) < Val2;
```

## Set-Theoretic Rules (Jarke and Koch)

*rewrite*

```
   E1 in Class1 such that
        all E2 in Class2 such that Pred(E2)
        have Fun1(E1) > Fun2(E2)
```

*into*

```
   E1 in Class1 such that
       Fun1(E1) > maximum(Fun2(E2 in Class2
                          such that Pred(E2)));
```

## Use of Where Clause in Rewrites

We can state *applicability* of rewrite rules by `where` clauses which test metadata about names in the ontology.

*where*

*function* **KeyFn** `is` *the key function of class* `Class and`

*function* **RelInv** `is` *the inverse of function* **Rel**

*rewrite*

```
Instance in Class such that
    some X in Rel(Instance)
            has KeyFn(X) = KeyValue
```

*into*

```
RelInv( the X in RelClass such that
            KeyFn(X) = KeyValue )
```

Example Instance: **Rel**=takes, **RelInv**=takes_inv, **KeyFn**=code, **Class**=undergrad.

N.B.: we do not use a "`with common`" clause to introduce `KeyValue` since we do not want to put a restriction on its type.

## Rewrite Rules for AMOS II

A P/FDM mediator can generate queries for an AMOS mediator (Risch-Uppsala). This uses AMOSQL -like OSQL or SQL-3. AMOS II usually recalls the compiler to re-plan nested queries. We can rewrite the query to avoid this.

ZF expression version:

```
{name(u1) | u1 <- undergrad;
   Exists {c | c <- takes(u1); level(c) > 3;
      Exists {u2 | u2 <- enrolled(c);
                   age(u2) < 19} } }
```

## Equivalent nested AMOSQL:

```
SELECT name(u1)
FROM   undergrad u1
WHERE  some(
   SELECT c
   FROM   course c
   WHERE  c = takes(u1)
   AND    level(c) > 3
   AND    some(
      SELECT u2
      FROM   undergrad u2
      WHERE  u2 = enrolled(c)
      AND    age(u2) < 19
   )
);
```

## *Flattened AMOSQL:*

```
SELECT name(u1)
FROM   undergrad u1
WHERE  some(
   SELECT c
   FROM   course c, undergrad u2
   WHERE  c = takes(u1)
   AND    level(c) > 3
   AND    u2 = enrolled(c)
   AND    age(u2) < 19
);
```

## Rewrite Rule (ICode version);

```
Exists{x | x <- generator; P(x);
           Exists{y | y <- h(x); Q(x,y)}}
=

Exists{x | x <- generator; P(x);
           y <- h(x); Q(x,y)}
```

- We depend on the AMOSQL optimiser to use the join predicates and selections concealed within $P$ and $Q$ to avoid a simplistic iteration over the Cartesian product of x(course) and y(undergrad).

- Otherwise the rule could actually worsen performance for a very selective $P$!

- Shows the importance of *knowledge about the remote query evaluation*.

## Pragmatic need for Rewrites in Wrappers

We introduce a final phase of rewriting that is target specific, as part of the mediator.

Accommodates known restrictions on remote DBMS optimisers, which may be given expressions they are not used to!

A common side-effect of query transformation and mapping between heterogeneous systems - very important in practise ($O(N) -> O(1)$).

## Comparison with Kleisli and TAMBIS

Both use comprehensions via the Collection Programming Language **CPL**(Buneman, Davidson et. al.)

- Kleisli works directly on Monad Composition form of queries (like ICode but more generic).

- Kleisli has impressive data integration applications in Bioinformatics through a variety of customised wrappers.

- Plans in TAMBIS follow the built-in Classification Hierarchy

- our plans follow Relationships in users SQL3-like queries.

## Common Functional Approach

- Our rules have a functional Syntax but a Semantics based on Unification

- We all depend on *referential transparency* for consistency of substitution.

## Readability of Rules: Can Users Maintain Them ??

- Kleisli functions are currently built into the optimiser but are extensible by the implementors.

- TAMBIS depends on user to extend the Classification Hierarchy

- Kleisli rules/functions are relatively few and very generic.

- Our Rules are more domain-specific - we want user to maintain them.

## Readability - Genericity Tradeoff

**Problem** There is a trade-off between readability and degree of abstract parametrisation.

- Rules that refer to very domain-specific situations involving specific named attributes are much easier to read and maintain.

- One way to extend this could be to include carefully formatted specific *instances of generic rules as comments*, for ease of understanding.

- We need to guard against users making rules *more general than intended* by not putting enough checks in the where-clause of a rule.

# Conclusions

- We advocate the use of an *object data model* with a high level declarative language including quantifiers that does not tie us either to relational or object storage.

- We have introduced a simple but powerful *rewrite language that includes FOL quantifiers* and allows a very general form of parametrisation, whichsuits unification in Prolog.

- rewrite rules provide a great deal of flexibility. They can implement:

  1. the set-theoretic rules given by Jarke and Koch;

  2. rewrites based on data semantics (J.J. King);

3. opportunities to replace iteration by indexed search;

4. unnesting and other transformations to assist remote optimisers;

5. rewrites that change the relative workload between two processors in a distributed query

6. many combinations, without our having to foresee them and code them individually

- This is **essential** in an Internet environment, where **data sources with different data management systems continue to be added!**