MATHEMATICAL MODELLING OF RELATIONAL DATABASE IN RISCAL

Joachim Borya

Johannes Kepler Universität

March 8th, 2022

The goal

In order to see the actual use of the following considerations, we take an actual SQL database as a model.

Figure 1: DDL script

```
BEGIN TRANSACTION;
CREATE TABLE IF NOT EXISTS ‘s‘ (
‘Field1‘ INTEGER,
‘Field2‘ INTEGER
);
INSERT INTO ‘s‘ VALUES (0,0);
INSERT INTO ‘s‘ VALUES (0,1);
INSERT INTO ‘s‘ VALUES (1,0);
CREATE TABLE IF NOT EXISTS ‘r‘ (
‘Field1‘ INTEGER,
‘Field2‘ INTEGER,
‘Field3‘ INTEGER
);
INSERT INTO ‘r‘ VALUES (1,1,0);
INSERT INTO ‘r‘ VALUES (0,1,0);
INSERT INTO ‘r‘ VALUES (0,0,0);
INSERT INTO ‘r‘ VALUES (1,1,1);
COMMIT;
```

The goal

Later on, we check if our algebraic approach leads to the same result as the query below.

Figure 2: Query

```
SELECT distinct *
FROM
(SELECT r.Field1 as 'a', r.Field3 as 'b'
FROM r WHERE r.Field2 = 1) as 't'
INNER JOIN s
ON s.Field1 = t.a;
```

Table 1: Output

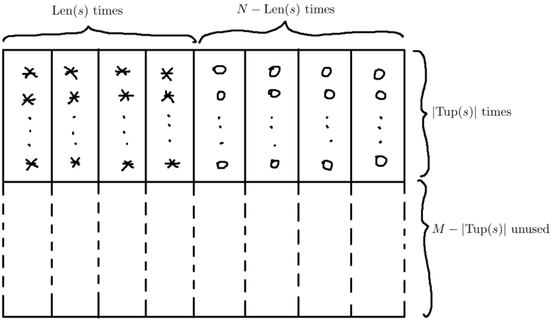
1	0	1	0
0	0	0	0
0	0	0	1
1	1	1	0

Algebra

- Theoretical foundation for the implementation in RISCAL
- The algebra we construct consists of ...
  - a domain `Relation`
  - and operations with signatures of the form $* \to$ `Relation`.
- For each operation we also define suitable preconditions.

Domain

- The domain will be parametrized by constants $M, N \in \mathbb{N}$ where M is the **maximum cardinality of relations** and N the **maximum length of tuples**.
- Let `Row` be the set of all functions $\{0, \dots, N\} \to \{0, 1\}$.
- The domain `Relation` consists of all $\langle n, r \rangle \in \{0, \dots, N\} \times \mathcal{P}(\texttt{Row})$ that satisfy
  - $|r| \leq M$
  - and $\forall t \in r, i \in \{n, \dots, N-1\} : t[i] = 0$. Note that $\{n, \dots, N-1\} = \emptyset$ for $n > N-1$.
- Notation: $\text{Len}(s) := n$ and $\text{Tup}(s) = r$ for $s \in \texttt{Relation}$
- Note: As a means of abstraction the "cells" of a "table" contain only bit values.


Len(s) times
$N-$ Len(s) times
|Tup(s)| times
$M-$ |Tup(s)| unused

Operations

- The actual operations we will construct are `cartesian`, `select`, `project`, `join`, `union`, `intersect` and `minus`.
- We will also have a `concat` function, which is not an actual operation. It will help to introduce `cartesian`.

concat

- Description: The function concatenates two rows.
- Signature: $\texttt{Row} \times \texttt{Row} \times \{0, \ldots, N\} \times \{0, \ldots, N\} \rightarrow \texttt{Row}$

Definition

$$\texttt{concat}(t_1, t_2, n_1, n_2) := n \mapsto \begin{cases} t_1(n), & \text{if } n < n_1 \\ t_2(n - n_1), & \text{if } n_1 \leq n < n_1 + n_2 \\ 0, & \text{else} \end{cases}$$

- Precondition: The parameters n_1, n_2 denote the actual length of a row. Therefore we need to ensure that $n_1 + n_2 \leq N$.

RISCAL

cartesian

- Description: The function constructs the cartesian product of two relations.
- Signature: $\texttt{Relation} \times \texttt{Relation} \rightarrow \texttt{Relation}$

Definition

$\texttt{cartesian}(r_1, r_2) = r :\Leftrightarrow$
$\mathsf{Tup}(r) = \{\mathrm{concat}(t_1, t_2) : t_1 \in \mathsf{Tup}(r_1), t_2 \in \mathsf{Tup}(r_2)\}$ and
$\mathsf{Len}(r) = \mathsf{Len}(r_1) + \mathsf{Len}(r_2)$.

- Precondition: The cartesian product is a relation where the rows have the length $\mathsf{Len}(r_1) + \mathsf{Len}(r_2)$, therefore we need to ensure that $\mathsf{Len}(r_1) + \mathsf{Len}(r_2) \leq N$. The maximum cardinality of this relation is $|\mathsf{Tup}(r_1)| \cdot |\mathsf{Tup}(r_2)|$, therefore we need to ensure that $|\mathsf{Tup}(r_1)| \cdot |\mathsf{Tup}(r_2)| \leq M$.

RISCAL

select

- Description: The function filters out rows whose columns have a certain value.
- Signature:
  $\texttt{Relation} \times \{0, \dots, N-1\} \times \{0,1\} \to \texttt{Relation}$

Definition

$\texttt{select}(r, a, e) := \langle \text{Len}(r), \{t \in r : t(a) = e\}\rangle$

- Precondition: We need to ensure that the column indicator a is not greater or equal the length of the rows of r, i.e. we need the precondition $a < \text{Len}(r)$.

RISCAL

project

- Description: The function can be used to create a new relation consisting of a rearrangement of certain columns of the previous relation.
- Signature:
  $\texttt{Relation} \times \{0, \ldots, N\}^{\{0,\ldots,N-1\}} \rightarrow \texttt{Relation}$

Definition

$\texttt{project}(r, c) = s :\Leftrightarrow \mathrm{Len}(s) = |\{i \in \{0, \ldots, N-1\} : c(i) \neq N\}|$
and
$\forall t_r \in \mathrm{Tup}(r) \exists t_s \in \mathrm{Tup}(s) \forall i \in \{0, \ldots, \mathrm{Len}(s)-1\} : t_s(i) = t_r(c(i))$

- Precondition: The parameter c should denote a choice of valid column indices in a certain order. A convenient precondition is given by

$$\exists i \in \{0, \ldots, N-1\} \forall j \in \{0, \ldots, N-1\} :$$

$$(j > i \Rightarrow c(i) = N) \wedge (j \leq i \Rightarrow c(i) < \mathrm{Len}(r))$$

RISCAL

join

- Description: The function filters out all rows in the cartesian product that have matching values in two certain columns.
- Signature: $\texttt{Relation}^2 \times \{0, \ldots, N-1\}^2 \to \texttt{Relation}$

Definition

$\texttt{join}(r_1, r_2, n_1, n_2) = s :\Leftrightarrow \mathsf{Len}(s) = \mathsf{Len}(r_1) + \mathsf{Len}(r_2)$ and $Tup(s) = \{\texttt{concat}(t_1, t_2, \mathsf{Len}(r_1), \mathsf{Len}(r_2)) : t_1 \in \mathsf{Tup}(r_1), t_2 \in \mathsf{Tup}(r_2), t_1(n_1) = t_2(n_2)\}$

- Precondition: Firstly n_1, n_2 need to denote valid columns, therefore we need a precondition $n_1 < \mathsf{Len}(r_1), n_2 < \mathsf{Len}(r_2)$. Secondly, just as in the cartesian product we need the preconditions $\mathsf{Len}(r_1) + \mathsf{Len}(r_2) \leq N$ and $|\mathsf{Tup}(r_1)| \cdot |\mathsf{Tup}(r_2)| \leq M$.

RISCAL

Set operations

- Description: The functions perform the regular set operations on relations.
- Signature: $\texttt{Relation} \times \texttt{Relation} \rightarrow \texttt{Relation}$

Definition

$\texttt{union}(r_1, r_2) := \langle \mathsf{Len}(r_1), \mathsf{Tup}(r_1) \cup \mathsf{Tup}(r_2) \rangle$
$\texttt{intersect}(r_1, r_2) := \langle \mathsf{Len}(r_1), \mathsf{Tup}(r_1) \cap \mathsf{Tup}(r_2) \rangle$
$\texttt{minus}(r_1, r_2) := \langle \mathsf{Len}(r_1), \mathsf{Tup}(r_1) \setminus \mathsf{Tup}(r_2) \rangle$

- For each of the three operations the relations r_1, r_2 need to be *union-compatible*, i.e. $\mathsf{Len}(r_1) = \mathsf{Len}(r_2)$. In case of `union` we additionally have to ensure that $|\mathsf{Tup}(r_1)| + |\mathsf{Tup}(r_2)| \leq M$.

RISCAL

Verification idea

1. Encoding of ...
   - the database
   - and the query

   ... in a single RISCAL procedure.
2. We prove as a theorem, that our model produces the same output as the query.

Encoding of the query

Figure 3: RISCAL procedure query()

```
proc query():Relation {
var dum:Map[Attribute,Element] := Map[Attribute,Element](0);

var r1:Relation := ⟨len: 3, tup: choose s:Set[Row] with |s|=0⟩;
var r2:Relation := ⟨len: 2, tup: choose s:Set[Row] with |s|=0⟩;

r1.tup := r1.tup ∪ {dum};
r2.tup := r2.tup ∪ {dum};
dum[1] := 1;
r1.tup := r1.tup ∪ {dum};
r2.tup := r2.tup ∪ {dum};
dum[0] := 1;
r1.tup := r1.tup ∪ {dum};
dum[1] := 0;
r2.tup := r2.tup ∪ {dum};
dum[1] := 1;
dum[2] := 1;
r1.tup := r1.tup ∪ {dum};
print r1;
print r2;

var columns:Array[N,Length] := Array[N,Length](N);
columns[0] := 0;
columns[1] := 2;
print columns;

return join(project2(select(r1,1,1),columns),r2,0,0);
}
```

Results

Figure 4: RISCAL procedure `result()`

```
proc result():Relation {
var dum:Map[Attribute,Element] := Map[Attribute,Element](0);
var r:Relation := ⟨len: 4, tup: choose s:Set[Row] with |s|=0⟩;

r.tup := r.tup ∪ {dum};

dum[3] := 1;
r.tup := r.tup ∪ {dum};

dum[3] := 0;
dum[0] := 1;
dum[2] := 1;
r.tup := r.tup ∪ {dum};

dum[1] := 1;
r.tup := r.tup ∪ {dum};

return r;
}

theorem correct_result() ⇔ query() = result();
```

In RISCAL it can be verified that the theorem above is true.

concat Go back

Figure 5: RISCAL implementation of concat

```
fun concat1(t1:Row, t2:Row, n1:Length, n2:Length):Row
requires n1 + n2 ≤ N;
= choose t:Row with ∀ i:Attribute. (
if i < n1 then t[i] = t1[i]
else if i ≥ n1 ∧ i < n1+n2 then t[i] = t2[i-n1]
else t[i] = 0
);

proc concat2(t1:Row, t2:Row, n1:Length, n2:Length):Row
requires n1 + n2 ≤ N; {
var t:Row = Array[N,Element](0);
for var i:Length:=0; i<n1; i:=i+1 do {
t[i] := t1[i];
}
for var i:Length:=n1; i<n1+n2; i:=i+1 do {
t[i] := t2[i-n1];
}
return t;
}

theorem concat_equiv(t1:Row, t2:Row, n1:Length, n2:Length)
requires n1 + n2 ≤ N; ⇔
concat1(t1,t2,n1,n2) = concat2(t1,t2,n1,n2);
```

cartesian Go back

Figure 6: RISCAL implementation of cartesian

```
fun cartesian(r1:Relation, r2:Relation):Relation
requires r1.len+r2.len ≤ N ∧ |r1.tup|*|r2.tup| ≤ M;
= ⟨len: r1.len+r2.len, tup: concat1(t1,t2,r1.len,r2.len) | t1∈r1.tup, t2∈r2.tup⟩;
```

select Go back

Figure 7: RISCAL implementation of `select`

```
fun select(r:Relation, a:Attribute, e:Element):Relation
requires a < r.len;
= ⟨len: r.len, tup: t | t∈r.tup with t[a] = e⟩;
```

project Go back

Figure 8: RISCAL implementation of project

```
fun project1(r:Relation, columns:Array[N,Length]):Relation
requires (∃ i:Attribute. ∀ j:Attribute.
(j>i ⇒ columns[j] = N) ∧ (j≤i ⇒ columns[j] < r.len));
= choose s:Relation with s.len = |i | i:Attribute with columns[i] ≠ N| ∧
(∀ tr:Row. tr∈r.tup ⇒
∃ ts:Row. ts∈s.tup ∧ ∀ i:Attribute. i < s.len ⇒ ts[i]=tr[columns[i]]);

proc project2(r:Relation, columns:Array[N,Length]):Relation
requires (∃ i:Attribute. ∀ j:Attribute.
(j>i ⇒ columns[j] = N) ∧ (j≤i ⇒ columns[j] < r.len)); {

var l:Length := |i | i:Attribute with columns[i] ≠ N|;
var q:Relation := ⟨len: l, tup: choose s:Set[Row] with |s|=0⟩;
var s:Set[Row] := r.tup;

choose t ∈ s do {
s := s \ {t};

var tn:Row := Array[N,Element](0);

var j:Length := 0;
for var i:Length := 0; i<N; i:=i+1 do {
if columns[i] ≠ N then {
tn[j] := t[columns[i]];
j := j+1;
}
}
q.tup := q.tup ∪ {tn};
}

return q;
}
```

join Go back

Figure 9: RISCAL implementation of `join`

```
fun join(r1:Relation, r2:Relation, n1:Attribute, n2:Attribute):Relation
requires n1<r1.len ∧ n2<r2.len ∧ r1.len+r2.len ≤ N ∧ |r1.tup|*|r2.tup| ≤ M;
= ⟨len: r1.len+r2.len,
tup: concat1(t1,t2,r1.len,r2.len) | t1∈r1.tup, t2∈r2.tup with t1[n1] = t2[n2]⟩;
```

Set operations

Go back

Figure 10: RISCAL implementation of the set operation

```
pred union_compatible(r1:Relation, r2:Relation) ⇔ r1.len=r2.len;

fun rUnion(r1:Relation, r2:Relation):Relation
requires union_compatible(r1,r2) ∧ |r1.tup| + |r2.tup| ≤ M;
= ⟨len: r1.len, tup: r1.tup ∪ r2.tup⟩;

fun rIntersect(r1:Relation, r2:Relation):Relation
requires union_compatible(r1,r2);
= ⟨len: r1.len, tup: r1.tup ∩ r2.tup⟩;

fun rMinus(r1:Relation, r2:Relation):Relation
requires union_compatible(r1,r2);
= ⟨len: r1.len, tup: r1.tup \ r2.tup⟩;
```