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**

```sql
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**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
The goal

Algebra

- Theoretical foundation for the implementation in RISCAL
- The algebra we construct consists of ...
  - a domain Relation
  - and operations with signatures of the form \( \ast \rightarrow \text{Relation} \).
- 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 $\text{Row}$ be the set of all functions $\{0, \ldots, N\} \rightarrow \{0, 1\}$.

- The domain $\text{Relation}$ consists of all $\langle n, r \rangle \in \{0, \ldots, N\} \times \mathcal{P}(\text{Row})$ that satisfy
  - $|r| \leq M$
  - and $\forall t \in r, i \in \{n, \ldots, N - 1\}: t[i] = 0$. Note that $\{n, \ldots, N - 1\} = \emptyset$ for $n > N - 1$.

- **Notation**: $\text{Len}(s) := n$ and $\text{Tup}(s) = r$ for $s \in \text{Relation}$

- **Note**: As a means of abstraction the "cells" of a "table" contain only bit values.
Mathematical Modelling of Relational Databases in RISCAL

Joachim Borya

The goal

Algebra

Domain

Operations

concat

cartesian

select

project

join

Set operations

RISCAL

Verification idea

Encoding of the query

Results

Appendix
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**: $\text{Row} \times \text{Row} \times \{0, \ldots, N\} \times \{0, \ldots, N\} \rightarrow \text{Row}$

**Definition**

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

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

- **Description:** The function constructs the cartesian product of two relations.

- **Signature:** Relation $\times$ Relation $\rightarrow$ Relation

**Definition**

$$\text{cartesian}(r_1, r_2) = r :\iff$$

$$\text{Tup}(r) = \{\text{concat}(t_1, t_2) : t_1 \in \text{Tup}(r_1), t_2 \in \text{Tup}(r_2)\} \text{ and}$$

$$\text{Len}(r) = \text{Len}(r_1) + \text{Len}(r_2).$$

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

RISCAL
select

- **Description:** The function filters out rows whose columns have a certain value.

- **Signature:**
  \[ \text{Relation} \times \{0, \ldots, N - 1\} \times \{0, 1\} \rightarrow \text{Relation} \]

**Definition**

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

- **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 \(a < \text{Len}(r)\).
project

- **Description:** The function can be used to create a new relation consisting of a rearrangement of certain columns of the previous relation.

- **Signature:**
  \[ \text{Relation} \times \{0,\ldots,N\}^{0,\ldots,N-1} \rightarrow \text{Relation} \]

**Definition**

\[
\text{project}(r, c) = s : \iff \text{Len}(s) = |\{i \in \{0,\ldots,N-1\} : c(i) \neq N\}| \quad \text{and} \\
\forall t_r \in \text{Tup}(r) \exists t_s \in \text{Tup}(s) \forall i \in \{0,\ldots,\text{Len}(s)-1\} : t_s(i) = t_r(c(i))
\]

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

\[
\exists i \in \{0,\ldots,N-1\} \forall j \in \{0,\ldots,N-1\} : (j > i \Rightarrow c(i) = N) \land (j \leq i \Rightarrow c(i) < \text{Len}(r))
\]
join

- **Description**: The function filters out all rows in the cartesian product that have matching values in two certain columns.

- **Signature**: $\text{Relation}^2 \times \{0, \ldots, N-1\}^2 \rightarrow \text{Relation}$

**Definition**

\[
\text{join}(r_1, r_2, n_1, n_2) = s : \iff \text{Len}(s) = \text{Len}(r_1) + \text{Len}(r_2) \text{ and } \text{Tup}(s) = \\
\{\text{concat}(t_1, t_2, \text{Len}(r_1), \text{Len}(r_2)) : t_1 \in \text{Tup}(r_1), t_2 \in \text{Tup}(r_2), t_1(n_1) = t_2(n_2)\}
\]

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

- **Description**: The functions perform the regular set operations on relations.

- **Signature**: Relation $\times$ Relation $\rightarrow$ Relation

### Definition

<table>
<thead>
<tr>
<th>Operation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>union</td>
<td>$\text{union}(r_1, r_2) := \langle \text{Len}(r_1), \text{Tup}(r_1) \cup \text{Tup}(r_2) \rangle$</td>
</tr>
<tr>
<td>intersect</td>
<td>$\text{intersect}(r_1, r_2) := \langle \text{Len}(r_1), \text{Tup}(r_1) \cap \text{Tup}(r_2) \rangle$</td>
</tr>
<tr>
<td>minus</td>
<td>$\text{minus}(r_1, r_2) := \langle \text{Len}(r_1), \text{Tup}(r_1) \setminus \text{Tup}(r_2) \rangle$</td>
</tr>
</tbody>
</table>

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

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()

```riscal
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.
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);
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⟩;
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⟩;
Figure 8: RISCAL implementation of project

fun project1(r:Relation, columns:Array[N,Length]):Relation
requires (\( \exists \ i: \text{Attribute.} \ \forall \ j: \text{Attribute.} \ (j > i \ \Rightarrow \ \text{columns}[j] = N) \land (j \leq i \ \Rightarrow \ \text{columns}[j] < r.len)));
= choose s:Relation with s.len = \mid i \mid i: \text{Attribute with columns}[i] \neq N \land \\
(\forall \ tr: \text{Row.} \ tr \in r.tup \Rightarrow \\
\exists \ ts: \text{Row.} \ ts \in s.tup \land \forall \ i: \text{Attribute.} \ i < s.len \Rightarrow ts[i] = tr[\text{columns}[i]]));

proc project2(r:Relation, columns:Array[N,Length]):Relation
requires (\( \exists \ i: \text{Attribute.} \ \forall \ j: \text{Attribute.} \ (j > i \ \Rightarrow \ \text{columns}[j] = N) \land (j \leq i \ \Rightarrow \ \text{columns}[j] < r.len)));
{
  var l:Length := \mid i \mid i: \text{Attribute with columns}[i] \neq N; 
  var q:Relation := \langle len: l, tup: choose s:Set[\text{Row}] with \mid s\mid = 0 \rangle; 
  var s:Set[\text{Row}] := r.tup; 

  choose t \in s do 
  {
    s := s \setminus \{t\};

    var tn:Row := Array[N,\text{Element}](0);

    var j:Length := 0;
    for var i:Length := 0; i < N; i := i+1 do 
      if columns[i] \neq N then 
        t[n[j]] := t[columns[i]];
        j := j+1;
    }

    q.tup := q.tup \cup \{tn\};
  }

  return q;
}
**Figure 9: RISCAL implementation of `join`**

```riscal
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]⟩;
```

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⟩;