A Coq mechanised formal semantics for real life SQL queries

Formally reconciling SQL and bag relational algebra

Véronique Benzaken, Évelyne Contejean

LRI - CNRS - Université Paris Sud

Coq Workshop@Floc18 Oxford 8th of July 2018
Motivations

Data are **pervasive and valuable** ...

... Little attention to **guarantee systems are reliable and safe**.

How to obtain strong guarantees?
Motivations

Data are pervasive and valuable ...

... Little attention to guarantee systems are reliable and safe.

How to obtain strong guarantees?

By using formal methods
Motivations

Data are pervasive and valuable ...

... Little attention to guarantee systems are reliable and safe.

How to obtain strong guarantees?

By using formal methods

Strong guarantees: proof assistants, Coq or Isabelle
Motivations

Data are pervasive and valuable ...

... Little attention to guarantee systems are reliable and safe.

How to obtain strong guarantees?

By using formal methods

Strong guarantees: proof assistants, Coq or Isabelle

Datacert project 2012-???

Coq formalisation of data-centric languages and systems
Relational database systems

Most widespread systems

Underlying theory is well known [Codd70]

One standard SQL the relational database programming language

Mature implementations

Oracle, DB2 IBM, SQLServer, Postgresql, MySql, SQLite ...

Mid term goal: provide a Coq verified SQL’s compiler
Sources (methodology) and goals

Foundations (studying) relational model and algebra

ANSI/ISO Standard (reading) SQL’s description

1500 pages natural language ...

Mainstream systems, Postgresql and Oracle™ (testing)
Sources (methodology) and goals

Foundations (studying)  relational model and algebra

ANSI/ISO Standard (reading)  1500 pages natural language ...

Mainstream systems, Postgresql and Oracle™ (testing)
Reconciling all of them
with strongest possible correctness guarantees (Coq)
Relational model and algebra

• **Information modeling:**
  
  through relations and tuples

Structure: relation name and sort (finite set of attributes)

\[ r(a, b) \quad \text{relation name } r \quad \text{sort: } \{a, b\} \]

• **Information extraction:**
  
  through query languages (relational algebra)
Relational model and algebra

Two perspectives:

unnamed vs named

\[ r = \{(1, 2); (3, 2); (1, 1)\} \]

\[ r = \{t1; t2; t3\} \]

\[ t1(a) = 1, \ t1(b) = 2 \]
\[ t2(a) = 3, \ t2(b) = 2 \]
\[ t3(a) = 1, \ t3(b) = 1 \]
Relational model and algebra

Two perspectives:

unnamed vs named

\[ r = \{(1, 2); (3, 2); (1, 1)\} \quad \text{vs} \quad r = \{t1; t2; t3\} \]

\[ t1.a = 1, t1.b = 2 \]
\[ t2.a = 3, t2.b = 2 \]
\[ t3.a = 1, t3.b = 1 \]
Relational model and algebra

**Unnamed (SPC)**

\[
q := r \mid \sigma_f(q) \mid \pi_W(q) \mid q \times q \\
| q \cup q \mid q \cap q \mid q \setminus q
\]

**Named (SPJR)**

\[
q := r \mid \sigma_f(q) \mid \pi_W(q) \mid \rho_g(q) \mid q \bowtie q \\
| q \cup q \mid q \cap q \mid q \setminus q
\]
Relational model and algebra

- \( \sigma_f(q) = \{ t \in q \mid f(t) \} \)
- \( \pi_W(q) = \{ t|_W \mid t \in q \} \)
- \( q_1 \circlearrowright q_2 = \{ t \mid \exists t_1 \in q_1, \exists t_2 \in q_2, t|_{\text{sort}(q_1)} = t_1 \land t|_{\text{sort}(q_2)} = t_2 \} \)
- \( \rho_g(q) = \{ t' \mid \exists t \in q, \forall a \in \text{sort}(q), t'.g(a) = t.a \} \)
Relational model and algebra

- $\sigma_f(q) = \{ t \in q \mid f(t) \}$
- $\pi_W(q) = \{ t|_W \mid t \in q \}$
- $q_1 \bowtie q_2 = \{ t \mid \exists t_1 \in q_1, \exists t_2 \in q_2, t|_{\text{sort}(q_1)} = t_1 \land t|_{\text{sort}(q_2)} = t_2 \}$
- $\rho_g(q) = \{ t' \mid \exists t \in q, \forall a \in \text{sort}(q), t'.g(a) = t.a \}$
Simple algebraic queries

Assuming tbl1(a, b, c) and tbl2(d, e)

\[ \pi\{a, c\}(\sigma_b > 3(tbl1)) \]

\[ \rho\{a \rightarrow a_1; c \rightarrow c_1\}(\pi\{a, c\}(\sigma_b > 3(tbl1))) \]

\[ \sigma_{b=d \land c=e}(tbl1 \bowtie tbl2) \]
SQL: a simple declarative language

SQL “inter-galactic” dialect for manipulating (relational) data

Declarative DSL describe what opposed as how

```
select expression
from query
where condition
group by expression
having condition
```

With attribute’s names as first-class citizens

⇒ name-based perspective
SQL’s compilation

**Syntactic analysis**

SQL $\rightarrow$ AST

**Semantic analysis**

AST $\rightarrow$ AST\textsubscript{sem}

- **Textbooks**
  - leaves = relations
  - nodes = relational algebra operators

- **Real life**
  - depends on DB vendors

**Optimisation / Query planning**

AST\textsubscript{sem} $\rightarrow$ AST\textsubscript{phys}

- **Logical**
  - rewritings / algebraic equivalences

- **Physical**
  - auxiliary data structures (B trees, Hash tables etc)
  - physical algebra – different implementations of operators
  - data dependent statistics
**SQL’s compilation**

### Syntactic analysis

- SQL → AST

### Semantic analysis

- Textbooks
  - leaves = relations
  - nodes = relational algebra operators

- Real life depends on DB vendors

### Optimisation / Query planning

- See Chantal Keller’s talk at ITP!
- Logical rewritings / algebraic equivalences
- Physical
  - auxiliary data structures (B trees, Hash tables etc)
  - physical algebra – different implementations of operators
  - data dependent statistics

- AST$_{sem}$ → AST$_{phys}$
SQL’s compilation

This talk

Semantic analysis

Textbooks
leaves = relations
nodes = relational algebra operators

Real life
depends on DB vendors

Providing a formal semantics to SQL

Formally relating SQL’s semantics with a relevant algebra
SQL’s compilation

This talk

Semantic analysis

Textbooks

leaves = relations
nodes = relational algebra operators

Real life

depends on DB vendors

Providing a formal semantics to SQL

Formally relating SQL’s semantics with a relevant algebra

30 years research efforts

[Ceri&al85, Negri&al91, Guagliardo&al17]

[Malecha&al10, Auerbach&al17, Chu&al17]
SQL: a Simple Declarative Language

Assuming tbl1(a,b,c) and tbl2(d,e)

```sql
select a, c from tbl1 where b>3;

\[ \pi_{\{a, c\}}(\sigma_{b>3}(tbl1)) \]

select a as a1, c as c1 from tbl1 where b>3;

\[ \rho_{\{a \to a1; c \to c1\}}(\pi_{\{a, c\}}(\sigma_{b>3}(tbl1))) \]

select * from tbl1, tbl2 where b=d and c=e;

\[ \sigma_{b=d \land c=e}(tbl1 \bowtie tbl2) \]

select b, sum(a) from tbl1 where a=7 group by b;

\[ \gamma_{b,sum(a)}(\sigma_{a=7}(tbl1)) \]

select b, 2*(a+c), sum(a) from tbl1 where a+b = 7 group by b, a+c having avg(b+c) > 6;

No corresponding expression in textbooks’ algebras
SQL: a Simple Declarative Language

Declarative DSL = ... simple

intended not to be Turing complete

But

Not so simple ...
SQL: not so simple

Based on relational algebra for the \texttt{select-from-where} part

Mixes two algebras: the \texttt{name} based SPJR and the \texttt{unnamed} SPC
SQL: not so simple

Based on relational algebra for the select-from-where part
Mixes two algebras: the name based SPJR and the unnamed SPC
Quoting page 51 of the ISO document attributes are specified by:

“The terms column, field, and attribute refer to structural components of tables, row types, and structured types, [...] in analogous fashion. As the structure of a table consists of one or more columns, so does the structure of a row type consist of one or more fields [...] Every structural element, whether a column, a field, or an attribute, is primarily a name paired with a declared type. The elements of a structure are ordered. Elements in different positions in the same structure can have the same declared type but not the same name. [...] in some circumstances [...] the compatibility [...] is determined solely by considering the declared types of each pair of elements at the same ordinal position.”
SQL: not so simple

Based on relational algebra for the `select-from-where` part

Mixes two algebras: the name based SPJR and the unnamed SPC

Enjoys a bag semantics

Manages complex expressions and aggregates

with NULL values which represent incomplete information handled by 3-valued logic with unknown

and nested, correlated queries
SQL: not so simple

Based on relational algebra for the **select-from-where** part

Mixes two algebras: the **name** based SPJR and the **unnamed** SPC

Enjoys a **bag** semantics

Manages **complex expressions** and **aggregates**

with **NULL** values which represent **incomplete** information handled by 3-valued logic with **unknown**

and **nested, correlated** queries $\implies$ **strange behaviours**
SQL: NULL's (1)

\[ r = \{(a=1), (a=NULL)\} \]
\[ s = \{(a=NULL)\} \]
\[ t = \{(a=1), (a=NULL), (a=NULL)\} \]

Q₁: select r.a+2 as b from r;
SQL: NULL's (I)

\[ r = \{ (a=1), (a=NULL) \} \]
\[ s = \{ (a=NULL) \} \]
\[ t = \{ (a=1), (a=NULL), (a=NULL) \} \]

Q₁: select r.a+2 as b from r;

\[ \{ (b = 1+2); (b = NULL+2) \} \]
SQL: NULL’s (1)

\[ r = \{ (a=1), (a=\text{NULL}) \} \]
\[ s = \{ (a=\text{NULL}) \} \]
\[ t = \{ (a=1), (a=\text{NULL}), (a=\text{NULL}) \} \]

Q₁: select r.a+2 as b from r;

\[ \{ (b = 1+2); (b = \text{NULL}+2) \} \]
\[ \{ (b = 3); (b = \text{NULL}) \} \]

NULL is an absorbing element
**SQL: NULL’s (II)**

\[
\begin{align*}
r &= \{(a=1), (a=NULL)\} \\
s &= \{(a=NULL)\} \\
t &= \{(a=1), (a=NULL), (a=NULL)\}
\end{align*}
\]

Q₂: \textit{select} r.a \textit{from} r \textit{where} r.a \textit{not in} (select s.a \textit{from} s);

Q₃: \textit{select} r.a \textit{from} r \textit{where} \\
\textit{not exists} (select * \textit{from} s \textit{where} s.a = r.a);

Q₄: \textit{select} r.a \textit{from} r \textit{except} select s.a \textit{from} s;
SQL: NULL's (II)

\[ r = \{ (a=1), (a=NULL) \} \]
\[ s = \{ (a=NULL) \} \]
\[ t = \{ (a=1), (a=NULL), (a=NULL) \} \]

Q2: select r.a from r where r.a not in (select s.a from s);

\[ \{ t.a \mid t \in r \land \neg (t.a \in \{ (a=NULL) \}) \} \]
\[ \{ t.a \mid t \in r \land (t.a \neq NULL) \} \]

Q3: select r.a from r where not exists (select * from s where s.a = r.a);

Q4: select r.a from r except select s.a from s;
SQL: NULL's (II)

\[ r = \{ (a=1), (a=NULL) \} \]
\[ s = \{ (a=NULL) \} \]
\[ t = \{ (a=1), (a=NULL), (a=NULL) \} \]

Q2: select r.a from r where r.a not in (select s.a from s);

\{ t.a \mid t \in r \land \neg(t.a \in \{ (a=NULL) \}) \}
\{ t.a \mid t \in r \land (t.a \neq NULL) \}

1 and NULL are not different from NULL

Q3: select r.a from r where not exists (select * from s where s.a = r.a);

Q4: select r.a from r except select s.a from s;
SQL: NULL's (II)

\[ r = \{(a=1), (a=NULL)\} \]
\[ s = \{(a=NULL)\} \]
\[ t = \{(a=1), (a=NULL), (a=NULL)\} \]

Q2: \( \text{select } r.a \text{ from } r \text{ where } r.a \text{ not in (select } s.a \text{ from } s); \)

\[ \{t.a \mid t \in r \land \neg(t.a \in \{(a=NULL)\})\} \]
\[ \{t.a \mid t \in r \land (t.a \neq NULL)\} \]

\[
\text{1 and NULL are not different from NULL}
\]

Q3: \( \text{select } r.a \text{ from } r \text{ where not exists (select * from } s \text{ where } s.a = r.a); \)

\[ \{t.a \mid t \in r \land \{u \mid u \in s \land u.a = t.a\} = \emptyset\} \]

Q4: \( \text{select } r.a \text{ from } r \text{ except select } s.a \text{ from } s; \)
**SQL: NULL's (II)**

\[ r = \{(a=1), (a=NULL)\} \]
\[ s = \{(a=NULL)\} \]
\[ t = \{(a=1), (a=NULL), (a=NULL)\} \]

Q2: select \( r.a \) from \( r \) where \( r.a \) not in (select \( s.a \) from \( s \));

\[ \{t.a \mid t \in r \land \neg (t.a \in \{(a=NULL)\})\} \]
\[ \{t.a \mid t \in r \land (t.a \neq NULL)\} \]

1 and NULL are not different from NULL

Q3: select \( r.a \) from \( r \) where
not exists (select * from \( s \) where \( s.a = r.a \));

\[ \{t.a \mid t \in r \land \{u \mid u \in s \land u.a = t.a\} = \emptyset\} \]

NULL neither equals to anything else

Q4: select \( r.a \) from \( r \) except select \( s.a \) from \( s \);
SQL: NULL’s (II)

\[ r = \{ (a=1), (a=\text{NULL}) \} \]
\[ s = \{ (a=\text{NULL}) \} \]
\[ t = \{ (a=1), (a=\text{NULL}), (a=\text{NULL}) \} \]

Q2: select r.a from r where r.a not in (select s.a from s);

\[ \{ t.a \mid t \in r \land \neg (t.a \in \{ (a=\text{NULL}) \}) \} \]
\[ \{ t.a \mid t \in r \land (t.a \neq \text{NULL}) \} \]

1 and NULL are not different from NULL

Q3: select r.a from r where not exists (select * from s where s.a = r.a);

\[ \{ t.a \mid t \in r \land \{ u \mid u \in s \land u.a = t.a \} = \emptyset \} \]

NULL neither equals to anything else

Q4: select r.a from r except select s.a from s;

\{ (a = 1) \} syntactic equality is used
SQL: NULL’s (III)

\[ r = \{ |(a=1), (a=NULL)\} \]
\[ s = \{ |(a=NULL)\} \]
\[ t = \{ |(a=1), (a=NULL), (a=NULL)\} \]

Q5: select t.a, count( * ) as c from t group by t.a;
SQL: NULL’s (III)

\[
\begin{align*}
    r &= \{(a=1), (a=\text{NULL})\}\, \\
    s &= \{(a=\text{NULL})\}\, \\
    t &= \{(a=1), (a=\text{NULL}), (a=\text{NULL})\}\,
\end{align*}
\]

Q₅: \textit{select t.a, count( * ) as c from t group by t.a;}

\[
\{(a = \text{NULL}, c = 2);(a = 1, c = 1)\}\}
\]

\textbf{NULL equals NULL for grouping}
SQL: aggregates, nesting, correlated queries

Designing instances and queries for testing against Postgresql and Oracle™

<table>
<thead>
<tr>
<th></th>
<th>(a_1)</th>
<th>(b_1)</th>
<th>(a_1)</th>
<th>(b_1)</th>
<th>(a_1)</th>
<th>(b_1)</th>
<th>(a_1)</th>
<th>(b_1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>\text{empty}</td>
<td>\text{empty}</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td></td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>\text{empty}</td>
<td>\text{empty}</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>\text{empty}</td>
<td>\text{empty}</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>\text{empty}</td>
<td>\text{empty}</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>\text{empty}</td>
<td>\text{empty}</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>6</td>
<td>2</td>
<td>6</td>
<td>\text{empty}</td>
<td>\text{empty}</td>
<td>\text{empty}</td>
<td>\text{empty}</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>7</td>
<td>2</td>
<td>7</td>
<td>4</td>
<td>6</td>
<td>\text{empty}</td>
<td>\text{empty}</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>\text{empty}</td>
<td>\text{empty}</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>9</td>
<td>2</td>
<td>9</td>
<td>4</td>
<td>8</td>
<td>\text{empty}</td>
<td>\text{empty}</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>10</td>
<td>2</td>
<td>10</td>
<td>4</td>
<td>9</td>
<td>\text{empty}</td>
<td>\text{empty}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(a_2)</th>
<th>(b_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>

19 / 49
SQL: aggregates, nesting, correlated queries

\[
t_1 = \{(a_1 = 1, b_1 = i) \mid 1 \leq i \leq 10\} \cup \\
\{(a_1 = 2, b_1 = i) \mid 1 \leq i \leq 10\} \cup \\
\{(a_1 = 3, b_1 = i) \mid 1 \leq i \leq 5\} \cup \\
\{(a_1 = 4, b_1 = i) \mid 6 \leq i \leq 10\} \\
\]

\[
t_2 = \{(a_2 = 7, b_2 = 7), (a_2 = 7, b_2 = 7)\} \\
\]

Q6: \texttt{select a1, max(b1) from t1 group by a1;
SQL: aggregates, nesting, correlated queries

\[ t_1 = \{(a_1 = 1, b_1 = i) \mid 1 \leq i \leq 10\} \cup \{(a_1 = 2, b_1 = i) \mid 1 \leq i \leq 10\} \cup \{(a_1 = 3, b_1 = i) \mid 1 \leq i \leq 5\} \cup \{(a_1 = 4, b_1 = i) \mid 6 \leq i \leq 10\} \]

\[ t_2 = \{(a_2 = 7, b_2 = 7), (a_2 = 7, b_2 = 7)\} \]

Q6: select a1, \textit{max}(b1) from \( t_1 \) group by a1;

\[ \{(a_1 = 1, \textit{max} = 10); (a_1 = 2, \textit{max} = 10); (a_1 = 3, \textit{max} = 5); (a_1 = 4, \textit{max} = 10)\} \]
SQL: aggregates, nesting, correlated queries

\[ t_1 = \{ (a_1 = 1, b_1 = i) \mid 1 \leq i \leq 10 \} \cup \{ (a_1 = 2, b_1 = i) \mid 1 \leq i \leq 10 \} \cup \{ (a_1 = 3, b_1 = i) \mid 1 \leq i \leq 5 \} \cup \{ (a_1 = 4, b_1 = i) \mid 6 \leq i \leq 10 \} \cup \]

\[ t_2 = \{ (a_2 = 7, b_2 = 7), (a_2 = 7, b_2 = 7) \} \]

Q7: select a1 from t1 group by a1 having exists (select a2 from t2 group by a2 having sum(1+0*a2) = 2);

sum(1+0*a2): computes the number of tuples in a group
SQL: aggregates, nesting, correlated queries

\[ t_1 = \{(a_1 = 1, b_1 = i) \mid 1 \leq i \leq 10\} \cup \{(a_1 = 2, b_1 = i) \mid 1 \leq i \leq 10\} \cup \{(a_1 = 3, b_1 = i) \mid 1 \leq i \leq 5\} \cup \{(a_1 = 4, b_1 = i) \mid 6 \leq i \leq 10\} \cup \]

\[ t_2 = \{(a_2 = 7, b_2 = 7), (a_2 = 7, b_2 = 7)\}\]

Q7: select a1 from t1 group by a1 having exists (select a2 from t2 group by a2 having sum(1+0*a2) = 2);

sum(1+0*a2): computes the number of tuples in a group which group?
SQL: aggregates, nesting, correlated queries

\[ t_1 = \{(a_1 = 1, b_1 = i) \mid 1 \leq i \leq 10\} \cup \{(a_1 = 2, b_1 = i) \mid 1 \leq i \leq 10\} \cup \{(a_1 = 3, b_1 = i) \mid 1 \leq i \leq 5\} \cup \{(a_1 = 4, b_1 = i) \mid 6 \leq i \leq 10\} \]

\[ t_2 = \{(a_2 = 7, b_2 = 7), (a_2 = 7, b_2 = 7)\} \]

Q7: select a_1 from t_1 group by a_1 having
exists (select a_2 from t_2 group by a_2 
    having sum(1+0*a_2) = 2);

sum(1+0*a_2): computes the number of tuples in a group

\[ \{(a_1 = 1); (a_1 = 2); (a_1 = 3); (a_1 = 4) \} \]
**SQL: aggregates, nesting, correlated queries**

\[ t_1 = \{(a_1 = 1, b_1 = i) \mid 1 \leq i \leq 10\} \cup \{(a_1 = 2, b_1 = i) \mid 1 \leq i \leq 10\} \cup \{(a_1 = 3, b_1 = i) \mid 1 \leq i \leq 5\} \cup \{(a_1 = 4, b_1 = i) \mid 6 \leq i \leq 10\} \]

\[ t_2 = \{(a_2 = 7, b_2 = 7), (a_2 = 7, b_2 = 7)\} \]

**Q7:** select a1 from t1 group by a1 having

\[ \exists \text{(select a2 from t2 group by a2 having sum(1+0*a2) = 2)}; \]

\text{sum}(1+0*a2): computes the number of tuples in a group which group?

\[ \{(a_1 = 1); (a_1 = 2); (a_1 = 3); (a_1 = 4) \} \]

\text{sum}(1+0*a2) is evaluated to 2 for each a2-group
SQL: aggregates, nesting, correlated queries

\[ t_1 = \{ (a_1 = 1, b_1 = i) \mid 1 \leq i \leq 10 \} \cup \{ (a_1 = 2, b_1 = i) \mid 1 \leq i \leq 10 \} \cup \{ (a_1 = 3, b_1 = i) \mid 1 \leq i \leq 5 \} \cup \{ (a_1 = 4, b_1 = i) \mid 6 \leq i \leq 10 \} \cup \]

\[ t_2 = \{ (a_2 = 7, b_2 = 7), (a_2 = 7, b_2 = 7) \} \]

Q8: \textit{select} \ a_1 \ \textit{from} \ t_1 \ \textit{group by} \ a_1 \ \textit{having} \ \
\quad \textit{exists (select} \ a_2 \ \textit{from} \ t_2 \ \textit{group by} \ a_2 \ \
\quad \quad \textit{having} \ \textit{sum(1+0*a2) = 10);}
SQL: aggregates, **nesting**, correlated queries

\[ t_1 = \begin{cases} (a_1 = 1, b_1 = i) & | 1 \leq i \leq 10 \\ (a_1 = 2, b_1 = i) & | 1 \leq i \leq 10 \\ (a_1 = 3, b_1 = i) & | 1 \leq i \leq 5 \\ (a_1 = 4, b_1 = i) & | 6 \leq i \leq 10 \end{cases} \cup \begin{cases} (a_1 = 7, b_2 = 7), (a_2 = 7, b_2 = 7) \end{cases} \]

Q8: \textbf{select} a_1 \textbf{from} t_1 \textbf{group by} a_1 \textbf{having}

\textbf{exists} (\textbf{select} a_2 \textbf{from} t_2 \textbf{group by} a_2

\textbf{having} \textbf{sum}(1+0*a_2) = 10);

\{\}
SQL: aggregates, nesting, correlated queries

\[ t_1 = \{(a_1 = 1, b_1 = i) \mid 1 \leq i \leq 10\} \cup \{(a_1 = 2, b_1 = i) \mid 1 \leq i \leq 10\} \cup \{(a_1 = 3, b_1 = i) \mid 1 \leq i \leq 5\} \cup \{(a_1 = 4, b_1 = i) \mid 6 \leq i \leq 10\} \]

\[ t_2 = \{(a_2 = 7, b_2 = 7),(a_2 = 7, b_2 = 7)\} \]

Q8: select a1 from t1 group by a1 having

exists (select a2 from t2 group by a2

having sum(1+0*a2) = 10);

\{\} \}

\text{sum}(1+0*a2) \text{ is evaluated to } 2 \neq 10 \text{ for each } a_2\text{-group}
SQL: aggregates, nesting, correlated queries

\[
t_1 = \{ (a_1 = 1, b_1 = i) \mid 1 \leq i \leq 10 \} \cup \{ (a_1 = 2, b_1 = i) \mid 1 \leq i \leq 10 \} \cup \{ (a_1 = 3, b_1 = i) \mid 1 \leq i \leq 5 \} \cup \{ (a_1 = 4, b_1 = i) \mid 6 \leq i \leq 10 \} \cup \]

\[
t_2 = \{ (a_2 = 7, b_2 = 7), (a_2 = 7, b_2 = 7) \}
\]

Q9(k): \text{select a1 from t1 group by a1 having}
\[
\text{exists (select a2 from t2 group by a2}
\text{havign sum(1) = k)};
\]
SQL: aggregates, nesting, correlated queries

\[ t_1 = \{(a_1 = 1, b_1 = i) | 1 \leq i \leq 10\} \cup \{(a_1 = 2, b_1 = i) | 1 \leq i \leq 10\} \cup \{(a_1 = 3, b_1 = i) | 1 \leq i \leq 5\} \cup \{(a_1 = 4, b_1 = i) | 6 \leq i \leq 10\} \cup \]

\[ t_2 = \{(a_2 = 7, b_2 = 7), (a_2 = 7, b_2 = 7)\} \]

Q9(k): select a1 from t1 group by a1 having 
exists (select a2 from t2 group by a2 
having sum(1) = k);

\[ k = 2 \leadsto \{| (a_1 = 1); (a_1 = 2); (a_1 = 3); (a_1 = 4) |\} \]

\[ k \neq 2 \leadsto \{| \} \]
SQL: aggregates, nesting, correlated queries

\[ t_1 = \{(a_1 = 1, b_1 = i) \mid 1 \leq i \leq 10\} \cup \{(a_1 = 2, b_1 = i) \mid 1 \leq i \leq 10\} \cup \{(a_1 = 3, b_1 = i) \mid 1 \leq i \leq 5\} \cup \{(a_1 = 4, b_1 = i) \mid 6 \leq i \leq 10\} \cup \]

\[ t_2 = \{(a_2 = 7, b_2 = 7), (a_2 = 7, b_2 = 7)\} \]

Q9(k): select a1 from t1 group by a1 having

exists (select a2 from t2 group by a2

having sum(1) = k);

\[ k = 2 \leadsto \{(a_1 = 1); (a_1 = 2); (a_1 = 3); (a_1 = 4) \} \]

\[ k \neq 2 \leadsto \{\} \]

sum(1) is evaluated to 2 for each a2-group
SQL: aggregates, nesting, correlated queries

\[ t_1 = \{ (a_1 = 1, b_1 = i) \mid 1 \leq i \leq 10 \} \cup \{ (a_1 = 2, b_1 = i) \mid 1 \leq i \leq 10 \} \cup \{ (a_1 = 3, b_1 = i) \mid 1 \leq i \leq 5 \} \cup \{ (a_1 = 4, b_1 = i) \mid 6 \leq i \leq 10 \} \cup t_2 = \{ (a_2 = 7, b_2 = 7), (a_2 = 7, b_2 = 7) \} \]

Q9(k): select a1 from t1 group by a1 having

\[ \exists (\text{select a2 from t2 group by a2 having sum(1) = k}); \]

\[ k = 2 \Rightarrow \{ (a_1 = 1); (a_1 = 2); (a_1 = 3); (a_1 = 4) \} \]

\[ k \neq 2 \Rightarrow \{ \} \]

sum(1) is evaluated to 2 for each a2-group

Tentative conclusion: \( 1 + 0 \times a_2 = 1 \)
SQL: aggregates, nesting, correlated queries

\[ t_1 = \big\{ (a_1 = 1, b_1 = i) \mid 1 \leq i \leq 10 \big\} \cup \]
\[ \big\{ (a_1 = 2, b_1 = i) \mid 1 \leq i \leq 10 \big\} \cup \]
\[ \big\{ (a_1 = 3, b_1 = i) \mid 1 \leq i \leq 5 \big\} \cup \]
\[ \big\{ (a_1 = 4, b_1 = i) \mid 6 \leq i \leq 10 \big\} \cup \]

\[ t_2 = \big\{ (a_2 = 7, b_2 = 7), (a_2 = 7, b_2 = 7) \big\} \]

Q10: select a1 from t1 group by a1 having
exists (select a2 from t2 group by a2
having sum(1+0*a1) = 10);
SQL: aggregates, nesting, correlated queries

\[ t_1 = \{(a_1 = 1, b_1 = i) \mid 1 \leq i \leq 10\} \cup \{(a_1 = 2, b_1 = i) \mid 1 \leq i \leq 10\} \cup \{(a_1 = 3, b_1 = i) \mid 1 \leq i \leq 5\} \cup \{(a_1 = 4, b_1 = i) \mid 6 \leq i \leq 10\} \cup \]

\[ t_2 = \{(a_2 = 7, b_2 = 7), (a_2 = 7, b_2 = 7)\} \]

Q10: select a1 from t1 group by a1 having
exists (select a2 from t2 group by a2
having sum(1+0*a1) = 10);

\[ \{ (a1 = 1); (a1 = 2) \} \]
SQL: aggregates, nesting, correlated queries

\[ t_1 = \{(a_1 = 1, b_1 = i) \mid 1 \leq i \leq 10\} \cup \{(a_1 = 2, b_1 = i) \mid 1 \leq i \leq 10\} \cup \{(a_1 = 3, b_1 = i) \mid 1 \leq i \leq 5\} \cup \{(a_1 = 4, b_1 = i) \mid 6 \leq i \leq 10\} \]

\[ t_2 = \{(a_2 = 7, b_2 = 7), (a_2 = 7, b_2 = 7)\} \]

Q10: \texttt{select a1 from t1 group by a1 having exists (select a2 from t2 group by a2 having sum(1+0*a1) = 10);} 
\[ \{(a1 = 1); (a1 = 2) \} \]

\texttt{sum(1+0*a1)} is evaluated for each \texttt{a1}-group
**SQL: aggregates, nesting, correlated queries**

\[ t_1 = \{(a_1 = 1, b_1 = i) \mid 1 \leq i \leq 10\} \cup \{(a_1 = 2, b_1 = i) \mid 1 \leq i \leq 10\} \cup \{(a_1 = 3, b_1 = i) \mid 1 \leq i \leq 5\} \cup \{(a_1 = 4, b_1 = i) \mid 6 \leq i \leq 10\} \cup t_2 = \{(a_2 = 7, b_2 = 7), (a_2 = 7, b_2 = 7)\} \]

Q10: \( \text{select } a_1 \text{ from } t_1 \text{ group by } a_1 \text{ having} \)
\( \exists (\text{select } a_2 \text{ from } t_2 \text{ group by } a_2 \)
\( \text{having } \text{sum}(1+0*a_1) = 10); \)
\( \{(a_1 = 1); (a_1 = 2) \} \)

\( \text{sum}(1+0*a_1) \text{ is evaluated for each } a_1 \text{-group} \)

Conclusion: \( 1 \not< 1+0*a_1 \text{ in some contexts (since } Q9(10) \neq Q10) \)
SQL: aggregates, nesting, correlated queries

\[
t_1 = \{(a_1 = 1, b_1 = i) \mid 1 \leq i \leq 10\} \cup \{(a_1 = 2, b_1 = i) \mid 1 \leq i \leq 10\} \cup \{(a_1 = 3, b_1 = i) \mid 1 \leq i \leq 5\} \cup \{(a_1 = 4, b_1 = i) \mid 6 \leq i \leq 10\}
\]

\[
t_2 = \{(a_2 = 7, b_2 = 7), (a_2 = 7, b_2 = 7)\}
\]

Q11(k): select a1 from t1 group by a1 having exists (select a2 from t2 group by a2 having sum(1+0*a1)+sum(1+0*a2) = k);
SQL: aggregates, nesting, correlated queries

\[ t_1 = \{ (a_1 = 1, b_1 = i) \mid 1 \leq i \leq 10 \}\cup \{ (a_1 = 2, b_1 = i) \mid 1 \leq i \leq 10 \}\cup \{ (a_1 = 3, b_1 = i) \mid 1 \leq i \leq 5 \}\cup \{ (a_1 = 4, b_1 = i) \mid 6 \leq i \leq 10 \}\cup \]

\[ t_2 = \{ (a_2 = 7, b_2 = 7), (a_2 = 7, b_2 = 7) \}\]

Q11(k): select a_1 from t_1 group by a_1 having
exists (select a_2 from t_2 group by a_2
having sum(1+0*a_1)+sum(1+0*a_2) = k);

\[ k = 7 \leadsto \{ (a_1 = 3); (a_1 = 4) \} \quad k = 12 \leadsto \{ (a_1 = 1); (a_1 = 2) \} \]

\[ k \neq 7, k \neq 12 \leadsto \{\} \]
SQL: aggregates, nesting, correlated queries

\[
t_1 = \{ (a_1 = 1, b_1 = i) \mid 1 \leq i \leq 10 \} \cup \\
\{ (a_1 = 2, b_1 = i) \mid 1 \leq i \leq 10 \} \cup \\
\{ (a_1 = 3, b_1 = i) \mid 1 \leq i \leq 5 \} \cup \\
\{ (a_1 = 4, b_1 = i) \mid 6 \leq i \leq 10 \} \cup \\
\]

\[
t_2 = \{ (a_2 = 7, b_2 = 7), (a_2 = 7, b_2 = 7) \} \\
\]

Q11(k): \select a_1 \from t_1 \group by a_1 \having \exists (\select a_2 \from t_2 \group by a_2 \having \sum(1+0*a_1)+\sum(1+0*a_2) = k); \\

\[
k = 7 \leadsto \{ (a_1 = 3); (a_1 = 4) \} \quad k = 12 \leadsto \{ (a_1 = 1); (a_1 = 2) \} \quad k \neq 7, k \neq 12 \leadsto \{ \} \\
\]

Different sub-expressions of same expression are evaluated in different environments
SQL: aggregates, nesting, **correlated queries**

\[ t_1 = \left\{ (a_1 = 1, b_1 = i) \mid 1 \leq i \leq 10 \right\} \cup \]
\[ \left\{ (a_1 = 2, b_1 = i) \mid 1 \leq i \leq 10 \right\} \cup \]
\[ \left\{ (a_1 = 3, b_1 = i) \mid 1 \leq i \leq 5 \right\} \cup \]
\[ \left\{ (a_1 = 4, b_1 = i) \mid 6 \leq i \leq 10 \right\} \cup \]

\[ t_2 = \left\{ (a_2 = 7, b_2 = 7), (a_2 = 7, b_2 = 7) \right\} \]

**Q12(k):**

select a₁ from t₁ group by a₁ having

exists (select a₂ from t₂ group by a₂
having sum(1+0*a₁+0*a₂) = k);
SQL: aggregates, nesting, correlated queries

\[ t_1 = \{(a_1 = 1, b_1 = i) \mid 1 \leq i \leq 10\} \cup \{(a_1 = 2, b_1 = i) \mid 1 \leq i \leq 10\} \cup \{(a_1 = 3, b_1 = i) \mid 1 \leq i \leq 5\} \cup \{(a_1 = 4, b_1 = i) \mid 6 \leq i \leq 10\} \]

\[ t_2 = \{(a_2 = 7, b_2 = 7), (a_2 = 7, b_2 = 7)\} \]

Q12(k): select \( a_1 \) from \( t_1 \) group by \( a_1 \) having 
exists (select \( a_2 \) from \( t_2 \) group by \( a_2 \) having \( \text{sum}(1+0*a_1+0*a_2) = k \));

\[ k = 2 \leadsto \{(a_1 = 1); (a_1 = 2); (a_1 = 3); (a_1 = 4)\} \]

\[ k \neq 2 \leadsto \{\} \]
SQL: aggregates, nesting, correlated queries

\[ t_1 = \{(a_1 = 1, b_1 = i) \mid 1 \leq i \leq 10\} \cup \{(a_1 = 2, b_1 = i) \mid 1 \leq i \leq 10\} \cup \{(a_1 = 3, b_1 = i) \mid 1 \leq i \leq 5\} \cup \{(a_1 = 4, b_1 = i) \mid 6 \leq i \leq 10\} \]

\[ t_2 = \{(a_2 = 7, b_2 = 7), (a_2 = 7, b_2 = 7)\} \]

Q12(k): select a1 from t1 group by a1 having

exists (select a2 from t2 group by a2

having \(\text{sum}(1+0*a_1+0*a_2) = k\));

\[ k = 2 \leadsto \{(a_1 = 1); (a_1 = 2); (a_1 = 3); (a_1 = 4) \} \]

\[ k \neq 2 \leadsto \{} \]

\[ \text{sum}(1+0*a_1+0*a_2) \text{ is evaluated for each } a_2\text{-group} \]
SQL: aggregates, nesting, correlated queries

\[ t_1 = \{ (a_1 = 1, b_1 = i) \mid 1 \leq i \leq 10 \} \cup \{ (a_1 = 2, b_1 = i) \mid 1 \leq i \leq 10 \} \cup \{ (a_1 = 3, b_1 = i) \mid 1 \leq i \leq 5 \} \cup \{ (a_1 = 4, b_1 = i) \mid 6 \leq i \leq 10 \} \cup \]

\[ t_2 = \{ (a_2 = 7, b_2 = 7), (a_2 = 7, b_2 = 7) \} \]

Q13(k): \text{select a}_1 \text{ from t}_1 \text{ group by a}_1 \text{ having exists (select a}_2 \text{ from t}_2 \text{ group by a}_2 \text{ having sum(1+0*a}_1\text{+0*b}_2\text{) = k});
SQL: aggregates, nesting, correlated queries

\[ t_1 = \{ (a_1 = 1, b_1 = i) \mid 1 \leq i \leq 10 \} \cup \{ (a_1 = 2, b_1 = i) \mid 1 \leq i \leq 10 \} \cup \{ (a_1 = 3, b_1 = i) \mid 1 \leq i \leq 5 \} \cup \{ (a_1 = 4, b_1 = i) \mid 6 \leq i \leq 10 \} \cup t_2 \]

\[ t_2 = \{ (a_2 = 7, b_2 = 7), (a_2 = 7, b_2 = 7) \} \]

Q13(k): \textbf{select a1 from t1 group by a1 having}
\begin{align*}
\text{exists (select a2 from t2 group by a2} \\
\text{having sum}(1+0*a_1+0*b_2) = k)\
\end{align*}

\[ k = 2 \iff \{ (a_1 = 1); (a_1 = 2); (a_1 = 3); (a_1 = 4) \} \]

\[ k \neq 2 \iff \{ \} \]
SQL: aggregates, nesting, **correlated** queries

\[
t_1 = \{(a_1 = 1, b_1 = i) \mid 1 \leq i \leq 10\} \cup \\
\{(a_1 = 2, b_1 = i) \mid 1 \leq i \leq 10\} \cup \\
\{(a_1 = 3, b_1 = i) \mid 1 \leq i \leq 5\} \cup \\
\{(a_1 = 4, b_1 = i) \mid 6 \leq i \leq 10\}
\]

\[
t_2 = \{(a_2 = 7, b_2 = 7), (a_2 = 7, b_2 = 7)\}
\]

**Q13(k):** select a1 from t1 group by a1 having 
exists (select a2 from t2 group by a2 
having sum(1+0*a1+0*b2) = k);

\[
k = 2 \leadsto \{(a_1 = 1); (a_1 = 2); (a_1 = 3); (a_1 = 4) \}
\]

\[
k \neq 2 \leadsto \{}
\]

\[
\text{sum}(1+0*a1+0*b2) \text{ is evaluated for each } a_2\text{-group}
\]
SQL: aggregates, nesting, correlated queries

\[ t_1 = \{ (a1 = 1, b1 = i) \mid 1 \leq i \leq 10 \} \cup \{ (a1 = 2, b1 = i) \mid 1 \leq i \leq 10 \} \cup \{ (a1 = 3, b1 = i) \mid 1 \leq i \leq 5 \} \cup \{ (a1 = 4, b1 = i) \mid 6 \leq i \leq 10 \} \]

\[ t_2 = \{ (a2 = 7, b2 = 7), (a2 = 7, b2 = 7) \} \]

Q14: \texttt{select a1 from t1 group by a1 having exists (select a2 from t2 group by a2 having sum(1+0*b1+0*b2) = 10);}
SQL: aggregates, nesting, correlated queries

\[ t_1 = \{(a_1 = 1, b_1 = i) \mid 1 \leq i \leq 10\} \cup \{(a_1 = 2, b_1 = i) \mid 1 \leq i \leq 10\} \cup \{(a_1 = 3, b_1 = i) \mid 1 \leq i \leq 5\} \cup \{(a_1 = 4, b_1 = i) \mid 6 \leq i \leq 10\} \]

\[ t_2 = \{(a_2 = 7, b_2 = 7), (a_2 = 7, b_2 = 7)\} \]

Q14: select a1 from t1 group by a1 having
   exists (select a2 from t2 group by a2
     having sum(1+0*b1+0*b2) = 10);

ERROR: subquery uses ungrouped column "t1.b1" from outer query
LINE 1: ...sts (select a2 from t2 group by a2 having sum(1+0*b1+0*b2) =
SQL: aggregates, nesting, **correlated** queries

\[
t_1 \quad = \quad \{(a_1 = 1, b_1 = i) \mid 1 \leq i \leq 10\} \cup \\
\{(a_1 = 2, b_1 = i) \mid 1 \leq i \leq 10\} \cup \\
\{(a_1 = 3, b_1 = i) \mid 1 \leq i \leq 5\} \cup \\
\{(a_1 = 4, b_1 = i) \mid 6 \leq i \leq 10\} \cup \\
\]

\[
t_2 \quad = \quad \{(a_2 = 7, b_2 = 7), (a_2 = 7, b_2 = 7)\}
\]

Q15: \textbf{select} a_1 \textbf{from} t_1 \textbf{group by} a_1 \textbf{having} \\
\textbf{exists} (\textbf{select} a_2 \textbf{from} t_2 \textbf{group by} a_2 \\
\textbf{having} \textbf{sum}(1+0*b_1+0*a_2) = 12);
SQL: aggregates, nesting, **correlated** queries

\[ t_1 = \{(a_1 = 1, b_1 = i) \mid 1 \leq i \leq 10\} \cup \\
\{(a_1 = 2, b_1 = i) \mid 1 \leq i \leq 10\} \cup \\
\{(a_1 = 3, b_1 = i) \mid 1 \leq i \leq 5\} \cup \\
\{(a_1 = 4, b_1 = i) \mid 6 \leq i \leq 10\} \cup \]

\[ t_2 = \{(a_2 = 7, b_2 = 7), (a_2 = 7, b_2 = 7)\} \]

Q15: \texttt{select a1 from t1 group by a1 having}

\texttt{exists (select a2 from t2 group by a2}

\texttt{having sum(1+0*b1+0*a2) = 12);}

ERROR: subquery uses ungrouped column "t1.b1" from outer query

LINE 1: ...sts (select a2 from t2 group by a2 having sum(1+0*b1+0*a2) =
Environments

A stack of slices, nesting levels, innermost on top

(attributes, grouping expressions, group of tuples)
Environments

A stack of slices, nesting levels, innermost on top

(attributes, grouping expressions, group of tuples)

Evaluation

• simple expression $\rightsquigarrow$ get the (unique) binding of each attribute
Environments

A stack of slices, nesting levels, innermost on top (attributes, grouping expressions, group of tuples)

Evaluation

- Simple expression $\rightarrow$ get the (unique) binding of each attribute
- Complex expression $\text{function}(\bar{e})$
  $\rightarrow$ evaluate independently each $e_i$ of $\bar{e}$
Environments

A **stack of slices**, nesting levels, innermost on top

(attributes, grouping expressions, group of tuples)

**Evaluation**

- **simple** expression $\rightsquigarrow$ get the (unique) binding of each attribute
- **complex** expression $\text{function}(\overline{e})$
  $\rightsquigarrow$ evaluate **independently** each $e_i$ of $(\overline{e})$
- **complex** expression $\text{aggregate(cst)}$
  $\rightsquigarrow$ use **innermost slice** (cardinality)
Environments

A stack of slices, nesting levels, innermost on top

(attributes, grouping expressions, group of tuples)

Evaluation

• **simple** expression $\leadsto$ get the (unique) binding of each attribute

• **complex** expression $\mathrm{function}(\overline{e})$

  $\leadsto$ evaluate independently each $e_i$ of $\overline{e}$

• **complex** expression $\mathrm{aggregate}(\mathrm{cst})$

  $\leadsto$ use innermost slice (cardinality)

• **complex** expression $\mathrm{aggregate}(e)$ in $[S_n; \ldots; S_1]$

  $\leadsto$ find the smallest ”suitable” suffix $[S_{i+1}; S_i; \ldots; S_1]$

  s.t. $e$ is built upon $A(S_{i+1}) \cup G(S_i) \cup \ldots \cup G(S_1)$

  $\leadsto$ split tuples of $(i + 1)$th slice

  $\leadsto [(A(S_{i+1}), G(S_{i+1}), [t_{i+1}]); S_i; \ldots; S_1] \quad t_{i+1} \in T(S_{i+1})$
Environments

A stack of slices, nesting levels, innermost on top

(attributes, grouping expressions, group of tuples)

Evaluation

- **simple** expression $\mapsto$ get the (unique) binding of each attribute
- **complex** expression $e$ function $\mapsto$ evaluate independently each $e_i$ of $(e)$
- **complex** expression $\text{aggregate}(cst)$ $\mapsto$ use innermost slice (cardinality)
- **complex** expression $\text{aggregate}(e)$ in $[S_n; \ldots ; S_1]$ $\mapsto$ find the smallest "suitable" suffix $[S_{i+1}; S_i; \ldots ; S_1]$ s.t. $e$ is built upon $A(S_{i+1}) \cup G(S_i) \cup \ldots \cup G(S_1)$ $\mapsto$ split tuples of $(i + 1)$th slice $\mapsto [(A(S_{i+1}), G(S_{i+1}), [t_{i+1}]); S_i; \ldots ; S_1] \ t_{i+1} \in T(S_{i+1})$
**SQL\textsubscript{Coq} Queries**

\texttt{Inductive set\_op} \(:=\) Union \mid Intersect \mid Except.
\texttt{Inductive select} \(:=\) Select\_As : aggterm \(\rightarrow\) attribute \(\rightarrow\) select.
\texttt{Inductive select\_item} \(:=\) Select\_Star \mid Select\_List : list select \(\rightarrow\) select\_item.
\texttt{Inductive group\_by} \(:=\) Finest\_P \mid Group\_By : list funterm \(\rightarrow\) group\_by.

\texttt{Inductive att\_renaming} \(:=\)
- Att\_As : attribute \(\rightarrow\) attribute \(\rightarrow\) att\_renaming.
\texttt{Inductive att\_renaming\_item} \(:=\)
- Att\_Ren\_Star \mid Att\_Ren\_List : list att\_renaming \(\rightarrow\) att\_renaming\_item.

\texttt{Inductive sql\_query} \(:=\)
| Table : relname \(\rightarrow\) sql\_query
| Set : set\_op \(\rightarrow\) sql\_query \(\rightarrow\) sql\_query \(\rightarrow\) sql\_query
| Select :
  - (** select *) select\_item \(\rightarrow\)
  - (** from *) list from\_item \(\rightarrow\)
  - (** where *) formula sql\_query \(\rightarrow\)
  - (** group by *) group\_by \(\rightarrow\)
  - (** having *) formula sql\_query \(\rightarrow\) sql\_query

\textit{with from\_item} \(:=\) From\_Item : sql\_query \(\rightarrow\) att\_renaming\_item \(\rightarrow\) sql\_from\_item.

no optional where, group by, nor having
no where \(\sim\) TTrue
no group by but having \(\sim\) Group\_By nil
no group by nor having \(\sim\) Finest\_P \(\uplus\) TTrue
SQL\textsubscript{Coq} Formulas

\[
\text{Inductive}\ \text{conjunct} \::= \ \text{And} \mid \text{Or}.
\]
\[
\text{Inductive}\ \text{quantifier} \::= \ \text{All} \mid \text{Any}.
\]

\[
\text{Inductive}\ \text{formula}\ (\text{dom} : \text{Type})\ ::= \n\]
\[
\mid \text{Conj} : \text{conjunct} \rightarrow \text{formula}\ \text{dom} \rightarrow \text{formula}\ \text{dom} \rightarrow \text{formula}\ \text{dom}
\]
\[
\mid \text{Not} : \text{formula}\ \text{dom} \rightarrow \text{formula}\ \text{dom}
\]
\[
\mid \text{TTrue} : \text{formula}\ \text{dom}
\]
\[
\mid \text{Pred} : \text{predicate} \rightarrow \text{list}\ \text{aggterm} \rightarrow \text{formula}\ \text{dom}
\]
\[
\mid \text{Quant} : \text{list}\ \text{aggterm} \rightarrow \text{predicate} \rightarrow \text{quantifier} \rightarrow \text{dom} \rightarrow \text{formula}\ \text{dom}
\]
\[
\mid \text{In} : \text{list}\ \text{select} \rightarrow \text{dom} \rightarrow \text{formula}\ \text{dom}
\]
\[
\mid \text{Exists} : \text{dom} \rightarrow \text{formula}\ \text{dom}.
\]

almost FO + in + exists
\[
\forall \rightsquigarrow \text{all}
\]
\[
\exists \rightsquigarrow \text{any}
\]
\[
in\ (\text{membership}) \rightsquigarrow _{\in} _{\cdot}\ (\text{not a usual predicate over values})
\]
exists \rightsquigarrow \text{non-emptiness test}

parameterised by dom
\[
\text{intended to be a finite domain of interpretation}
\]
Coq mechanised semantics

Simple expressions

(* The type of evaluation environments *)
Definition env_type := list (list attribute * group_by * list tuple).

(* get the (unique) binding of each attribute *)
Fixpoint interp_dot env (a : attribute) :=
  match env with
  | nil ⇒ default_value a
  | (sa, gb, nil) :: env' ⇒ interp_dot env' a
  | (sa, gb, t :: l) :: env' ⇒ if a inS? labels t then (dot t a) else interp_dot env' a
  end.

Fixpoint interp_funterm env t :=
  match t with
  | F_Constant c ⇒ c
  | F_Dot a ⇒ interp_dot env a
  | F_Expr f l ⇒ interp_symb f (List.map (fun x ⇒ interp_funterm env x) l)
  end.
Coq mechanised semantics

Complex expressions, environments

Fixpoint is_built_upon G f :=
  match f with
  | F_Constant _ ⇒ true
  | F_Dot _ ⇒ f inS? g
  | F_Expr s l ⇒ (f ins? G) || forallb (is_built_upon G) l
  end.

Definition is_a_suitable_env la env f :=
  is_built_upon
  (map (fun a ⇒ F_Dot a) la ++
    flat_map (fun slc ⇒ match slc with (_, G, _) ⇒ G end) env) f.

Fixpoint find_eval_env env f :=
  match env with
  | nil ⇒ if is_built_upon nil f then Some nil else None
  | (la1, g1, l1) :: env' ⇒
    match find_eval_env env' f with
    | Some _ as e ⇒ e
    | None ⇒ if is_a_suitable_env la1 env' f then Some env else None
    end
  end.

simply models SqHeLL, beginning
Coq mechanised semantics

Complex expressions, environments

**Fixpoint** `interp_aggterm env (ag : aggterm) :=
match ag with
| A_Expr ft ⇒ (* simple expression without aggregate *)
   interp_funterm env ft
| A_fun f lag ⇒
   (** simple recursive call in order to evaluate independently the sub-expressions
   when the top symbol is a function *)
   interp_symb f (List.map (fun x ⇒ interp_aggterm env x) lag)
| A_agg ag ft ⇒
   let env' := if is_empty (att_of_funterm ft)
   then (** the expression under ag is a constant *)
       Some env
   else (** find the outermost suitable level *)
       find_eval_env env ft in
   let lenv :=
     match env' with
     | None | Some nil ⇒
       (** this case should not happen for well-formed queries *) nil
     | Some ((la1, g1, l1) :: env'') ⇒
       (** the outermost group is split into *)
       map (fun t1 ⇒ (la1, g1, t1 :: nil) :: env'') l1
   end in
   interp_aggregate ag (List.map (fun e ⇒ interp_funterm e ft) lenv)
end.

simply models **SqHeLL**, end
irrelevant cases (ill-formed queries) due to totality
Coq mechanised semantics
parametric Booleans and 3-valued logic

Module Bool. (* parametric Booleans *)
Record Rcd : Type := mk_R {
    b : Type;
    true : b;
    false : b;
    andb : b → b → b;
    orb : b → b → b;
    negb : b → b;
    [...]
    true_is_true : ∀ b, is_true b = Datatypes.true ↔ b = true }.
End Bool.

split with bool true false andb orb negb [...]

Inductive bool3 : Type := true3 | false3 | unknown3.
Definition andb3 b1 b2 := [...]
Definition orb3 b1 b2 := [...]
Definition negb3 b := [...]

split with bool3 true3 false3 andb3 orb3 negb3 [...]

interpretation of formulas parameterised by a Booleans,
⇝ 2-valued logic or 3-valued logic
NULLs ⇝ 3-valued logic
**Coq mechanised semantics**

### Formulas

**Hypothesis** $B : \text{Bool.Rcd.}(\text{parametric Booleans})$

**Hypothesis** $I : \text{env_type} \to \text{dom} \to \text{bagT}(\text{bags of tuples})$.

**Fixpoint** `eval_formula env f : Bool.b B :=`  

- `Conj a f1 f2` $\Rightarrow$ (interp$_\text{conj}$ $B$ a) (eval$_\text{formula}$ env f1) (eval$_\text{formula}$ env f2)
- `Not f` $\Rightarrow$ Bool$_\text{negb}$ $B$ (eval$_\text{formula}$ env f)
- `TTrue` $\Rightarrow$ Bool$_\text{true}$ $B$
- `Pred p l` $\Rightarrow$ interp$_\text{predicate}$ p (map (interp$_\text{aggterm}$ env) l)
  
  $\mid$ `Quant qtf p l sq` $\Rightarrow$ `let lt := map (interp$_\text{aggterm}$ env) l in`
  
  interp$_\text{quant}$ $B$ qtf (fun $x$ $\Rightarrow$ `let la := Fset.elements _ (labels T x) in`
  
  interp$_\text{predicate}$ p (lt ++ map (dot T x) la))
  
  (Febag.elements _ (I env sq))

- `In s sq` $\Rightarrow$ `let p := (projection env (Select_List s)) in`
  
  interp$_\text{quant}$ $B$ Any
  
  (fun $x$ $\Rightarrow$ `match Oeset.compare (OTuple T) p x with`
   
  $\mid$ `Eq` $\Rightarrow$ `if contains_null p then unknown else Bool.true B`
   
  $\mid$ `_` $\Rightarrow$ `if (contains_null p || contains_null x) then unknown else Bool.false B end`
  
  (Febag.elements _ (I env sq))

- `Exists sq` $\Rightarrow$ `if Febag.is_empty _ (I env sq) then Bool.false B else Bool.true B end`

**evaluation parameterised by Booleans**

**subtleties in In for handling equality for NULLs:**

unknown may be unknown3 or false
Coq mechanised semantics
Queries

Fixpoint eval_sql_query env (sq : sql_query) {struct sq} :=
match sq with
| Sql_Table tbl ⇒ instance tbl
| Sql_Set o sq1 sq2 ⇒ [...]  
| Sql_Select s lsq f1 gby f2 ⇒
  let elsq := (** evaluation of the from part **)
    (List.map (eval_sql_from_item env) lsq) in
  let cc := (** selection of the from part by the formula f1, with old names **)
  Febag.filter _
    (fun t ⇒ Bool.is_true B (* casting parametric Booleans to Bool2 *))
    (eval_sql_formula eval_sql_query (env_t env t) f1))
  (N_product_bag elsq) in
(** computation of the groups grouped according to gby **)
let lg1 := make_groups env cc gby in
(** discarding groups according the having clause f2 **)
let lg2 :=
  List.filter
    (fun g ⇒ Bool.is_true B (* casting parametric Booleans to Bool2 *))
    (eval_sql_formula eval_sql_query (env_g env gby g) f2))
  lg1 in
(** applying the outermost projection and renaming, the select part s **)
  Febag.mk_bag BTupleT
  (List.map (fun g ⇒ projection (env_g env gby g) s) lg2)
end
Empirical assessment

Executable semantics \(\leadsto\) checked against Postgresql and Oracle\(^\text{TM}\)

Previous queries and similar ones (up to 4 levels of nesting)

Random instance generator, 5 parameters: number of tables, number of attributes for each table, max size of a relation’s instance, max integer value in relations’ instances, proportion of NULL’s in instances,

Random query generator, 5 parameters: proportion of constants among expressions, max number of expressions in `select`, max number of queries in `from`, max number of grouping expressions in `group by`, max level of nesting
Relating $\text{SQL}_{\text{Coq}}$ with an algebra

Define $\text{SQL}_{\text{Alg}}$

Extended relational algebra

Enjoying a bag semantics and

Natively accounting for group by having

Hence recovering well known algebraic equivalences
**SQL$_{\text{Alg}}$, a Coq mechanised algebra**

Inductive alg_query : Type :=
  | Q_Empty_Tuple : alg_query
  | Q_Table : relname → alg_query
  | Q_Set : set_op → alg_query → alg_query → alg_query
  | Q_Join : alg_query → alg_query → alg_query
  | Q_Pi : list select → alg_query → alg_query
  | Q_Sigma : (formula alg_query) → alg_query → alg_query
  (* extending the usual $\gamma$ textbook operator *)
  | Q_Gamma :
    (* aggregated (output) expressions *) list select →
    (* grouping expressions *) list funterm →
    (* handling having condition *) (formula alg_query) →
    (* query *) alg_query → alg_query.

**usual relational algebra + a generalized $\gamma$ operator:** Q_Gamma

**formulas** are **shared** with SQL$_{\text{Coq}}$
Fixpoint eval_alg_query env q {struct q} : bagT :=
    match q with
    | Q_Empty_Tuple ⇒ Febag.singleton _ (empty_tuple T)
    | Q_Table r ⇒ instance r
    | Q_Set o q1 q2 ⇒ [...]
    | Q_Join q1 q2 ⇒ natural_join (eval_alg_query env q1) (eval_alg_query env q2)
    | Q_Pi s q ⇒
        Febag.map _ _
        (\ t ⇒ projection (env_t env t) (Select_List s)) (eval_alg_query env q)
    | Q_Sigma f q ⇒
        Febag.filter _ _
        (\ t ⇒Bool.is_true B (eval_formula _ eval_alg_query (env_t env t) f))
        (eval_alg_query env q)
    | Q_Gamma s lf f q ⇒
        Febag.mk_bag _
        (map (\ l ⇒ projection (env_g env (Group_By lf) l) (Select_List s))
        (filter (\ l ⇒Bool.is_true B
            (eval_formula _ eval_alg_query (env_g env (Group_By lf) l) f))
            (make_groups env (eval_alg_query env q) (Group_By lf)))
        end.

environments and formula’s evaluation are shared with SQLCoq
Equivalence

\[ \text{SQL}_{\text{Coq}} \equiv \text{SQL}_{\text{Alg}} \]
Fixpoint sql_query_to_alg basesort (sq : sql_query) :=
match sq with
| Sql_Table r ⇒ Q_Table r
| Sql_Set o sq1 sq2 ⇒ [...]
| Sql_Select s lsq f1 g f2 ⇒
  match s with
  | Select_Star ⇒ [...]
  | Select_List s ⇒
  let q1 := (** from clause is translated thanks to n-ary natural join *)
    N_Q_Join (map sql_item_to_alg lsq) in
  let q2 := (** filtering against where condition *)
    Q_Sigma (formula_to_alg f1) q1 in
  match g with
  | Finest_P ⇒
    (** no grouping, filtering against having condition, and then evaluation of select *)
    Q_Pi s (Q_Sigma (formula_to_alg f2) q2)
  | Group_By g ⇒
    (** grouping, using extended γ *)
    Q_Gamma s g (formula_to_alg f2) q2
end
end
end with [...]

⚠️ from ↝ ⊳
Back translation from SQL\textsubscript{Alg} to SQL\textsubscript{Coq}

Hypothesis fresh (la : list attribute) : attribute.
Hypothesis fresh_is_fresh : \( \forall \ s, \text{Oset.mem_bool (}\text{OAtt T}) \ (\text{fresh } s) \ s = \text{false}. \)

Fixpoint alg_query_to_sql (q : alg_query) : sql_query :=
match q with [...]
| Q_Join q1 q2 ⇒
  let rho1 := (** fresh names for attributes of q1 *) [...] in
  let rho2 := (** fresh names for attributes of q2 *) [...] in
  let rho1’ := (** inverse of rho1 *) [...] in
  let rho2’ := (** inverse of rho2, over for attributes which do not belong to q1 *) [...] in
  let f_join :=
    (* formula stating that new names for the same old shared attributes
       correspond to the same value : rho1(q1.a) = rho2(q2.a) *) [...] in
  Sql_Select (Select_List (rho1’ ++ rho2’))
    (From_Item (alg_sql_query_to_sql q1) (Att_Ren_List rho1) ::
     From_Item (alg_sql_query_to_sql q2) (Att_Ren_List rho2) :: nil)
    f_join Finest_P (Sql_True _)
| Q_Pi s q ⇒ [...] 
| Q_Sigma f q ⇒ [...] 
| Q_Gamma s g h q ⇒
  Sql_Select (Select_List s) (From_Item (alg_query_to_sql q) Att_Ren_Star :: nil)
    (Sql_True _) (Group_By g) (alg_formula_to_sql h)
end.
Definition **well_sorted_sql_table** :=
\[ \forall \text{tbl} \ t, \ t \in \text{BE} \ (\text{instance tbl}) \rightarrow \text{labels t} = \text{S} = \text{basesort tbl}. \]

Fixpoint **all_distinct** lsa :=
match lsa with
| nil \rightarrow true
| sa1 :: lsa \Rightarrow \text{Fset.is_empty} (A T) (sa1 \text{interS} (\text{Fset.Union} _ lsa)) && all_distinct lsa
end.

Fixpoint **well_formed_q** (sq : sql_query) :=
match sq with
| Sql_Table _ \Rightarrow true
| Sql_Set _ sq1 sq2 \Rightarrow well_formed_q sq1 && well_formed_q sq2
| Sql_Select s lsq f1 g f2 \Rightarrow
  (all_distinct (map (fun x \Rightarrow sql_from_item_sort) x) lsq)
  && (forallb (fun x \Rightarrow match x with From_Item sq _ \Rightarrow well_formed_q sq end) lsq)
  && (well_formed_f f1) && (well_formed_f f2)
end.

Lemma **sql_query_to_alg_is_sound** :
well_sorted_sql_table \rightarrow
\forall \text{env sq}, \text{well_formed_q sq} = \text{true} \rightarrow
\text{eval_alg_query env (sql_query_to_alg basesort sq)} = \text{BE} = \text{eval_sql_query env sq}.

Lemma **alg_query_to_sql_is_sound** :
well_sorted_sql_table \rightarrow
\forall \text{env q}, \text{eval_alg_query env q} = \text{BE} = \text{eval_sql_query env (alg_query_to_sql q)}.
Equivalence’s theorems

Definition well_sorted_sql_table :=
    ∀ tbl t, t inBE (instance tbl) → labels t =S= basesort tbl.

Fixpoint all_distinct lsa :=
    match lsa with
    | nil ⇒ true
    | sa1 :: lsa ⇒ Fset.is_empty (A T) (sa1 interS (Fset.Union _ lsa)) && all_distinct lsa
    end.

Fixpoint well_formed_q (sq : sql_query) :=
    match sq with
    | Sql_Table _ ⇒ true
    | Sql_Set _ sq1 sq2 ⇒ well_formed_q sq1 && well_formed_q sq2
    | Sql_Select s lsq f1 g f2 ⇒
        (all_distinct (map (fun x => sql_from_item_sort) x) lsq)
        && (forallb (fun x ⇒ match x with From_Item sq _ ⇒ well_formed_q sq end) lsq)
        && (well_formed_f f1) && (well_formed_f f2)
    end.

Lemma sql_query_to_alg_is_sound :
    well_sorted_sql_table → (* cartesian product = natural join thanks to to well-formedness *)
    ∀ env sq, well_formed_q sq = true ->
        eval_alg_query env (sql_query_to_alg basesort sq) =BE= eval_sql_query env sq.

Lemma alg_query_to_sql_is_sound :
    well_sorted_sql_table →
    ∀ env q, eval_alg_query env q =BE= eval_sql_query env (alg_query_to_sql q).
Equivalence’s theorems

Definition well_sorted_sql_table :=
\( \forall \text{tbl t}, \text{t inBE (instance tbl)} \rightarrow \text{labels t} =_{\text{S}} \text{basesort tbl} \).

Fixpoint all_distinct lsa :=
match lsa with
| nil \Rightarrow true
| sa1 :: lsa \Rightarrow \text{Fset.is_empty (A T) (sa1 interS (Fset.Union _ lsa))} \& \& \text{all_distinct lsa}
end.

Fixpoint well_formed_q (sq : sql_query) :=
match sq with
| Sql_Table _ \Rightarrow true
| Sql_Set _ sq1 sq2 \Rightarrow \text{well_formed_q sq1} \& \& \text{well_formed_q sq2}
| Sql_Select s lsq f1 g f2 \Rightarrow
  \text{(all_distinct (map (fun x => sql_from_item_sort) x) lsq)}
  \& \& \text{(forallb (fun x \Rightarrow match x with From_Item sq _ \Rightarrow well_formed_q sq end) lsq)}
  \& \& \text{(well_formed_f f1)} \& \& \text{(well_formed_f f2)}
end.

Lemma sql_query_to_alg_is_sound :
well_sorted_sql_table \rightarrow (* cartesian product = natural join thanks to to well-formedness *)
\( \forall \text{env sq, well_formed_q sq = true} \rightarrow \text{eval_alg_query env (sql_query_to_alg basesort sq) =}_{\text{BE}} \text{eval_sql_query env sq} \).

Lemma alg_query_to_sql_is_sound :
well_sorted_sql_table \rightarrow (* cartesian product = natural join thanks to to fresh names *)
\( \forall \text{env q, eval_alg_query env q} =_{\text{BE}} \text{eval_sql_query env (alg_query_to_sql q)} \).
Lessons : Coq side

Modelling a real-life language

⇝ pushing Coq to the very limits

⇝ discovering some practical restrictions with no theoretical reason
(* an abstracted version of formula’s sharing between SQL queries and algebraic queries *)

Section FirstVersion.
Hypothesis A : Type.
Inductive (* first version of formula *) b : Type := B : A → b
with (* first version of sql query *) mut : Type := M : b → mut.
End FirstVersion.

Inductive (* first tentative version of algebraic query *) x : Type := X : (b x) → x.
(* Error: Non strictly positive occurrence of "x" in "b x → x". *)

Section SecondVersion.
Hypothesis A : Type.
Inductive (* new version formula *) b’ : Type := B’ : A → b’.
Inductive (* new version of sql query *) mut’ : Type := M’ : b’ → mut’.
End SecondVersion.

Inductive (* new style algebraic query *) x1 : Type := X1 : (b’ x1) → x1.

(*
x1 is defined
x1_rect is defined
x1_ind is defined
x1_rec is defined
*)
Lessons : DB side

first version : set-semantics $\leadsto$ second version: bag-semantics

  technical, not a problem

NULL’s

  at expression level, absorbing elements
  at formula level, use 3-valued logic...
  not so difficult

real difficulty

  complex expressions and nested and correlated queries
  environments management

remains to be done:

outer, inner join (syntactic sugar) order by, windows, rank,
recursive queries
like handling regular expressions for strings
more data types: date
Data centric languages : a fantastic bestiary

NoSQL, Cassandra, MongoDB, Neo4j, etc weird

SQL purposely not Turing complete
⇝ overtime, new primitives and features:
  aggregates, nested / correlated queries, functions, NULL’s
  ⇝ uncontrolled interactions
⇝ departing from its elegant theoretical foundation
⇝ pay tribute to pioneers: Codd, Chamberlin, Boyce

use Coq to design new languages
  ⇝ completely formalised, clear and well-understood semantics