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

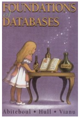
Mature implementations

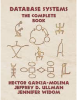
Oracle, DB_2 IBM, SQLServer, Postgresql, MySql, SQLite ...

Mid term goal: provide a Coq verified SQL's compiler

Sources (methodology) and goals

Foundations (studying) relational model and algebra





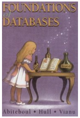
ANSI/ISO Standard (reading)

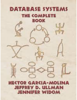
1500 pages natural language ... SQL's description

Mainstream systems, Postgresql and Oracle$^{\text{TM}}$ (testing)

Sources (methodology) and goals

Foundations (studying) relational model and algebra





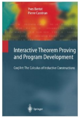
ANSI/ISO Standard (reading)

1500 pages natural language ... SQL's description

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) relation name r 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
$r = \{(1, 2); (3, 2); (1, 1)\}$		$r = \{t1; t2; t3\}$

$$\begin{aligned} t1.a &= 1, \; t1.b = 2 \\ t2.a &= 3, \; t2.b = 2 \\ t3.a &= 1, \; t3.b = 1 \end{aligned}$$

Relational model and algebra

unamed (SPC)

$$
\begin{array}{rcl}
q & := & r \mid \sigma_f(q) \mid \pi_W(q) \mid q \times q \\
  & \mid & q \cup q \mid q \cap q \mid q \setminus q
\end{array}
$$

named (SPJR)

$$
\begin{array}{rcl}
q & := & r \mid \sigma_f(q) \mid \pi_W(q) \mid \rho_g(q) \mid q \bowtie q \\
  & \mid & q \cup q \mid q \cap q \mid q \setminus q
\end{array}
$$

Relational model and algebra

- $\sigma_f(q) = \{t \in q \mid f(t)\}$
- $\pi_W(q) = \{t|_W \mid t \in q\}$
- $q_1 \bowtie q_2 = \{t \mid \exists t_1 \in q_1, \exists t_2 \in q_2, t|_{sort(q_1)} = t_1 \wedge t|_{sort(q_2)} = t_2\}$

- $\rho_g(q) = \{t' \mid \exists t \in q, \forall a \in sort(q), t'.g(a) = t.a\}$

Relational model and algebra

- $\sigma_f(q) = \{t \in q \mid f(t)\}$
- $\pi_W(q) = \{t|_W \mid t \in q\}$
- $q_1 \bowtie q_2 = \{t \mid \exists t_1 \in q_1, \exists t_2 \in q_2, t|_{sort(q_1)} = t_1 \wedge t|_{sort(q_2)} = t_2\}$

- $\rho_g(q) = \{t' \mid \exists t \in q, \forall a \in sort(q), t'.g(a) = t.a\}$

Simple algebraic queries

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

$\pi_{\{\mathtt{a},\mathtt{c}\}}(\sigma_{\mathtt{b}>3}(\mathtt{tbl1}))$

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

$\sigma_{\mathtt{b}=\mathtt{d}\wedge\mathtt{c}=\mathtt{e}}(\mathtt{tbl1} \bowtie \mathtt{tbl2})$

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

$\Rightarrow$ name-based perspective

SQL's compilation

Syntactic analysis $SQL \rightarrow AST$

Semantic analysis $AST \rightarrow AST_{sem}$

Textbooks
: leaves = relations
: nodes = relational algebra operators

Real life depends on DB vendors

Optimisation / Query planning $AST_{sem} \rightarrow AST_{phys}$

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 $\text{SQL} \rightarrow \text{AST}$

Semantic analysis $\text{AST} \rightarrow \text{AST}_{sem}$

Textbooks
- leaves = relations
- nodes = relational algebra operators

Real life: depends on DB vendors

Optimisation / Query planning $\text{AST}_{sem} \rightarrow \text{AST}_{phys}$

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

SQL's compilation

This talk

Semantic analysis $\text{AST} \rightarrow \text{AST}_{sem}$

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 $AST \rightarrow AST_{sem}$

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)`

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

$$\pi_{\{a,c\}}(\sigma_{b>3}(\texttt{tbl1}))$$

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

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

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

$$\sigma_{b=d \wedge c=e}(\texttt{tbl1} \bowtie \texttt{tbl2})$$

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

$$\gamma_{b, sum(a)}(\sigma_{a=7}(\texttt{tbl1}))$$

```
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

intented not to be Turing complete

But

Not so simple ...

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

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
$\Longrightarrow$ strange behaviours

SQL: NULL's (I)

$$r = \{\!|(\mathtt{a}{=}1), (\mathtt{a}{=}\mathtt{NULL})|\!\}$$
$$s = \{\!|(\mathtt{a}{=}\mathtt{NULL})|\!\}$$
$$t = \{\!|(\mathtt{a}{=}1), (\mathtt{a}{=}\mathtt{NULL}), (\mathtt{a}{=}\mathtt{NULL})|\!\}$$

Q_1: `select r.a+2 as b from r;`

SQL: NULL's (I)

$$r = \{\!|(\mathtt{a}{=}1), (\mathtt{a}{=}\mathtt{NULL})|\!\}$$
$$s = \{\!|(\mathtt{a}{=}\mathtt{NULL})|\!\}$$
$$t = \{\!|(\mathtt{a}{=}1), (\mathtt{a}{=}\mathtt{NULL}), (\mathtt{a}{=}\mathtt{NULL})|\!\}$$

Q_1:
```
select r.a+2 as b from r;
```

$$\{\!|\ (\mathtt{b}\ =\ 1{+}2); (\mathtt{b}\ =\ \mathtt{NULL}{+}2)\ |\!\}$$

SQL: NULL's (I)

$$r = \{\!|(\mathtt{a}{=}1), (\mathtt{a}{=}\mathtt{NULL})|\!\}$$
$$s = \{\!|(\mathtt{a}{=}\mathtt{NULL})|\!\}$$
$$t = \{\!|(\mathtt{a}{=}1), (\mathtt{a}{=}\mathtt{NULL}), (\mathtt{a}{=}\mathtt{NULL})|\!\}$$

Q_1: `select r.a+2 as b from r;`

$$\{\!|\ (\mathtt{b}\ =\ 1{+}2); (\mathtt{b}\ =\ \mathtt{NULL}{+}2)\ |\!\}$$
$$\{\!|\ (\mathtt{b}\ =\ 3); (\mathtt{b}\ =\ \mathtt{NULL})\ |\!\}$$

NULL is an absorbing element

SQL: NULL's (II)

$$r = \{\!|(\mathrm{a}{=}1), (\mathrm{a}{=}\mathtt{NULL})|\!\}$$
$$s = \{\!|(\mathrm{a}{=}\mathtt{NULL})|\!\}$$
$$t = \{\!|(\mathrm{a}{=}1), (\mathrm{a}{=}\mathtt{NULL}), (\mathrm{a}{=}\mathtt{NULL})|\!\}$$

Q_2:
```
select r.a from r where r.a not in (select s.a from s);
```

Q_3:
```
select r.a from r where
     not exists (select * from s where s.a = r.a);
```

Q_4:
```
select r.a from r except select s.a from s;
```

SQL: NULL's (II)

$$r = \{\!|(a{=}1), (a{=}\mathtt{NULL})|\!\}$$
$$s = \{\!|(a{=}\mathtt{NULL})|\!\}$$
$$t = \{\!|(a{=}1), (a{=}\mathtt{NULL}), (a{=}\mathtt{NULL})|\!\}$$

Q_2: `select r.a from r where r.a not in (select s.a from s);`

$$\{t.a \mid t \in r \wedge \neg(t.a \in \{\ (a{=}\mathtt{NULL})\ \})\}$$
$$\{t.a \mid t \in r \wedge (t.a \neq \mathtt{NULL})\}$$

Q_3:
```
select r.a from r where
       not exists (select * from s where s.a = r.a);
```

Q_4: `select r.a from r except select s.a from s;`

SQL: NULL's (II)

$$r = \{\!|(\mathrm{a}{=}1), (\mathrm{a}{=}\texttt{NULL})|\!\}$$
$$s = \{\!|(\mathrm{a}{=}\texttt{NULL})|\!\}$$
$$t = \{\!|(\mathrm{a}{=}1), (\mathrm{a}{=}\texttt{NULL}), (\mathrm{a}{=}\texttt{NULL})|\!\}$$

Q_2: `select r.a from r where r.a not in (select s.a from s);`

$$\{t.a \mid t \in r \wedge \neg(t.a \in \{\ (\mathrm{a}{=}\texttt{NULL})\ \})\}$$
$$\{t.a \mid t \in r \wedge (t.a \neq \texttt{NULL})\}$$

$\{\!|\quad|\!\}$ 1 and NULL are not different from NULL

Q_3: `select r.a from r where`
`not exists (select * from s where s.a = r.a);`

Q_4: `select r.a from r except select s.a from s;`

SQL: NULL's (II)

$$r = \{\!|(\mathrm{a}{=}1), (\mathrm{a}{=}\texttt{NULL})|\!\}$$
$$s = \{\!|(\mathrm{a}{=}\texttt{NULL})|\!\}$$
$$t = \{\!|(\mathrm{a}{=}1), (\mathrm{a}{=}\texttt{NULL}), (\mathrm{a}{=}\texttt{NULL})|\!\}$$

Q_2: `select r.a from r where r.a not in (select s.a from s);`

$$\{t.a \mid t \in r \wedge \neg(t.a \in \{\ (\mathrm{a}{=}\texttt{NULL})\ \})\}$$
$$\{t.a \mid t \in r \wedge (t.a \neq \texttt{NULL})\}$$

$\{\!|\quad|\!\}$ 1 and NULL are not different from NULL

Q_3: `select r.a from r where`
`not exists (select * from s where s.a = r.a);`

$$\{t.a \mid t \in r \wedge \{u \mid u \in s \wedge u.a = t.a\} = \emptyset\}$$

Q_4: `select r.a from r except select s.a from s;`

SQL: NULL's (II)

$$r = \{\!|(\mathrm{a}{=}1), (\mathrm{a}{=}\mathtt{NULL})|\!\}$$
$$s = \{\!|(\mathrm{a}{=}\mathtt{NULL})|\!\}$$
$$t = \{\!|(\mathrm{a}{=}1), (\mathrm{a}{=}\mathtt{NULL}), (\mathrm{a}{=}\mathtt{NULL})|\!\}$$

Q_2: `select r.a from r where r.a not in (select s.a from s);`

$$\{t.a \mid t \in r \wedge \neg(t.a \in \{\ (\mathrm{a}{=}\mathtt{NULL})\ \})\}$$
$$\{t.a \mid t \in r \wedge (t.a \neq \mathtt{NULL})\}$$

$\{\!|\quad|\!\}$ 1 and NULL are not different from NULL

Q_3: `select r.a from r where`
`not exists (select * from s where s.a = r.a);`

$$\{t.a \mid t \in r \wedge \{u \mid u \in s \wedge u.a = t.a\} = \emptyset\}$$

$\{\!|\ (\mathrm{a}\ =\ 1); (\mathrm{a}\ =\ \mathtt{NULL})\ |\!\}$ NULL neither equals to anything else

Q_4: `select r.a from r except select s.a from s;`

SQL: NULL's (II)

$$r = \{\!|(\text{a}{=}1), (\text{a}{=}\texttt{NULL})|\!\}$$
$$s = \{\!|(\text{a}{=}\texttt{NULL})|\!\}$$
$$t = \{\!|(\text{a}{=}1), (\text{a}{=}\texttt{NULL}), (\text{a}{=}\texttt{NULL})|\!\}$$

Q_2: `select r.a from r where r.a not in (select s.a from s);`

$$\{t.a \mid t \in r \wedge \neg(t.a \in \{\ (\text{a}{=}\texttt{NULL})\ \})\}$$
$$\{t.a \mid t \in r \wedge (t.a \neq \texttt{NULL})\}$$

$\{\!|\quad|\!\}$ 1 and NULL are not different from NULL

Q_3: `select r.a from r where`
`not exists (select * from s where s.a = r.a);`

$$\{t.a \mid t \in r \wedge \{u \mid u \in s \wedge u.a = t.a\} = \emptyset\}$$

$\{\!|\ (\text{a} = 1); (\text{a} = \texttt{NULL})\ |\!\}$ NULL neither equals to anything else

Q_4: `select r.a from r except select s.a from s;`

$\{\ (\text{a} = 1)\ \}$ syntactic equality is used

SQL: NULL's (III)

$$r = \{\!|(\mathrm{a}{=}1), (\mathrm{a}{=}\texttt{NULL})|\!\}$$
$$s = \{\!|(\mathrm{a}{=}\texttt{NULL})|\!\}$$
$$t = \{\!|(\mathrm{a}{=}1), (\mathrm{a}{=}\texttt{NULL}), (\mathrm{a}{=}\texttt{NULL})|\!\}$$

Q_5: `select t.a, count( * ) as c from t group by t.a;`

SQL: NULL's (III)

$$r = \{\!|(\mathrm{a}{=}1), (\mathrm{a}{=}\texttt{NULL})|\!\}$$
$$s = \{\!|(\mathrm{a}{=}\texttt{NULL})|\!\}$$
$$t = \{\!|(\mathrm{a}{=}1), (\mathrm{a}{=}\texttt{NULL}), (\mathrm{a}{=}\texttt{NULL})|\!\}$$

Q_5: `select t.a, count( * ) as c from t group by t.a;`

$\{\!|$ (a = NULL, c = 2); (a = 1, c = 1) $|\!\}$

NULL equals NULL for grouping

SQL: aggregates, nesting, correlated queries

Designing instances and queries for testing against Postgresql and Oracle™

t_1

a_1	b_1	a_1	b_1	a_1	b_1	a_1	b_1
1	1	2	1	3	1		
1	2	2	2	3	2		
1	3	2	3	3	3		
1	4	2	4	3	4		
1	5	2	5	3	5		
1	6	2	6			4	6
1	7	2	7			4	7
1	8	2	8			4	8
1	9	2	9			4	9
1	10	2	10			4	10

t_2

a_2	b_2
7	7
7	7

SQL: aggregates, nesting, correlated queries

$$
\begin{array}{rcl}
t_1 & = & \{\!|(a1 = 1, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
 & & \{\!|(a1 = 2, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
 & & \{\!|(a1 = 3, b1 = i) \mid 1 \leq i \leq 5|\!\} \cup \\
 & & \{\!|(a1 = 4, b1 = i) \mid 6 \leq i \leq 10|\!\} \cup \\
 & & \\
t_2 & = & \{\!|(a2 = 7, b2 = 7), (a2 = 7, b2 = 7)|\!\}
\end{array}
$$

Q6: `select a1, max(b1) from t1 group by a1;`

SQL: aggregates, nesting, correlated queries

$$
\begin{array}{rcl}
t_1 & = & \{\!|(a1 = 1, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
 & & \{\!|(a1 = 2, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
 & & \{\!|(a1 = 3, b1 = i) \mid 1 \leq i \leq 5|\!\} \cup \\
 & & \{\!|(a1 = 4, b1 = i) \mid 6 \leq i \leq 10|\!\} \cup \\
 & & \\
t_2 & = & \{\!|(a2 = 7, b2 = 7), (a2 = 7, b2 = 7)|\!\}
\end{array}
$$

Q6: `select a1, max(b1) from t1 group by a1;`

$$
\left\{\!\!\left| \begin{array}{l} (a1 = 1, max = 10); (a1 = 2, max = 10); \\ (a1 = 3, max = 5); (a1 = 4, max = 10) \end{array} \right|\!\!\right\}
$$

SQL: aggregates, nesting, correlated queries

$$
\begin{aligned}
t_1 &= \{\!|(a1 = 1, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
&\quad \{\!|(a1 = 2, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
&\quad \{\!|(a1 = 3, b1 = i) \mid 1 \leq i \leq 5|\!\} \cup \\
&\quad \{\!|(a1 = 4, b1 = i) \mid 6 \leq i \leq 10|\!\} \cup
\end{aligned}
$$

$$
t_2 = \{\!|(a2 = 7, b2 = 7), (a2 = 7, b2 = 7)|\!\}
$$

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

$$
\begin{array}{rcl}
t_1 & = & \{\!|(a1 = 1, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
 & & \{\!|(a1 = 2, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
 & & \{\!|(a1 = 3, b1 = i) \mid 1 \leq i \leq 5|\!\} \cup \\
 & & \{\!|(a1 = 4, b1 = i) \mid 6 \leq i \leq 10|\!\} \cup
\end{array}
$$

$$
t_2 = \{\!|(a2 = 7, b2 = 7), (a2 = 7, b2 = 7)|\!\}
$$

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

$$
\begin{aligned}
t_1 \quad = \quad & \{\!|(a1 = 1, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
& \{\!|(a1 = 2, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
& \{\!|(a1 = 3, b1 = i) \mid 1 \leq i \leq 5|\!\} \cup \\
& \{\!|(a1 = 4, b1 = i) \mid 6 \leq i \leq 10|\!\} \cup
\end{aligned}
$$

$$
t_2 \quad = \quad \{\!|(a2 = 7, b2 = 7), (a2 = 7, b2 = 7)|\!\}
$$

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?

$$
\{\!| \ (a1 = 1); (a1 = 2); (a1 = 3); (a1 = 4) \ |\!\}
$$

SQL: aggregates, nesting, correlated queries

$$
\begin{aligned}
t_1 &= \{\!|(a1 = 1, b1 = i) \mid 1 \le i \le 10|\!\} \cup \\
&\quad \{\!|(a1 = 2, b1 = i) \mid 1 \le i \le 10|\!\} \cup \\
&\quad \{\!|(a1 = 3, b1 = i) \mid 1 \le i \le 5|\!\} \cup \\
&\quad \{\!|(a1 = 4, b1 = i) \mid 6 \le i \le 10|\!\} \cup
\end{aligned}
$$

$$
t_2 = \{\!|(a2 = 7, b2 = 7), (a2 = 7, b2 = 7)|\!\}
$$

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?

$$
\{\!| \ (a1 = 1); (a1 = 2); (a1 = 3); (a1 = 4) \ |\!\}
$$

`sum(1+0*a2)` is evaluated to 2 for each a2-group

SQL: aggregates, nesting, correlated queries

$$
\begin{aligned}
t_1 \quad = \quad & \{\!|(a1 = 1, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
& \{\!|(a1 = 2, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
& \{\!|(a1 = 3, b1 = i) \mid 1 \leq i \leq 5|\!\} \cup \\
& \{\!|(a1 = 4, b1 = i) \mid 6 \leq i \leq 10|\!\} \cup
\end{aligned}
$$

$$
t_2 \quad = \quad \{\!|(a2 = 7, b2 = 7), (a2 = 7, b2 = 7)|\!\}
$$

Q8:
```
select a1 from t1 group by a1 having
          exists (select a2 from t2 group by a2
                         having sum(1+0*a2) = 10);
```

SQL: aggregates, nesting, correlated queries

$$
\begin{array}{rcl}
t_1 & = & \{\!|(a1 = 1, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
& & \{\!|(a1 = 2, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
& & \{\!|(a1 = 3, b1 = i) \mid 1 \leq i \leq 5|\!\} \cup \\
& & \{\!|(a1 = 4, b1 = i) \mid 6 \leq i \leq 10|\!\} \cup
\end{array}
$$

$$
t_2 = \{\!|(a2 = 7, b2 = 7), (a2 = 7, b2 = 7)|\!\}
$$

Q8:
```
select a1 from t1 group by a1 having
         exists (select a2 from t2 group by a2
                 having sum(1+0*a2) = 10);
```

$$
\{\!| \quad |\!\}
$$

SQL: aggregates, nesting, correlated queries

$$
\begin{array}{rcl}
t_1 & = & \{\!|(a1 = 1, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
& & \{\!|(a1 = 2, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
& & \{\!|(a1 = 3, b1 = i) \mid 1 \leq i \leq 5|\!\} \cup \\
& & \{\!|(a1 = 4, b1 = i) \mid 6 \leq i \leq 10|\!\} \cup
\end{array}
$$

$$
t_2 = \{\!|(a2 = 7, b2 = 7), (a2 = 7, b2 = 7)|\!\}
$$

Q8:
```
select a1 from t1 group by a1 having
         exists (select a2 from t2 group by a2
                   having sum(1+0*a2) = 10);
```

$$\{\!|\quad|\!\}$$

`sum(1+0*a2)` is evaluated to $2 \neq 10$ for each `a2`-group

SQL: aggregates, nesting, correlated queries

$$
\begin{aligned}
t_1 \quad = \quad & \{\!|(a1 = 1, b1 = i) \mid 1 \leq i \leq 10|\!\}\cup \\
& \{\!|(a1 = 2, b1 = i) \mid 1 \leq i \leq 10|\!\}\cup \\
& \{\!|(a1 = 3, b1 = i) \mid 1 \leq i \leq 5|\!\}\cup \\
& \{\!|(a1 = 4, b1 = i) \mid 6 \leq i \leq 10|\!\}\cup
\end{aligned}
$$

$$
t_2 \quad = \quad \{\!|(a2 = 7, b2 = 7), (a2 = 7, b2 = 7)|\!\}
$$

Q9(k):

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

SQL: aggregates, nesting, correlated queries

$$
\begin{array}{rcl}
t_1 & = & \{\!|(a1 = 1, b1 = i) \mid 1 \leq i \leq 10|\!\}\cup \\
 & & \{\!|(a1 = 2, b1 = i) \mid 1 \leq i \leq 10|\!\}\cup \\
 & & \{\!|(a1 = 3, b1 = i) \mid 1 \leq i \leq 5|\!\}\cup \\
 & & \{\!|(a1 = 4, b1 = i) \mid 6 \leq i \leq 10|\!\}\cup
\end{array}
$$

$$
t_2 = \{\!|(a2 = 7, b2 = 7), (a2 = 7, b2 = 7)|\!\}
$$

Q9(k):

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

$$
k = 2 \rightsquigarrow \{\!|\ (a1 = 1); (a1 = 2); (a1 = 3); (a1 = 4)\ |\!\}
$$

$$
k \neq 2 \rightsquigarrow \{\!|\quad|\!\}
$$

SQL: aggregates, nesting, correlated queries

$$
\begin{aligned}
t_1 \quad = \quad & \{\!|(a1 = 1, b1 = i) \mid 1 \leq i \leq 10|\!\}\cup \\
& \{\!|(a1 = 2, b1 = i) \mid 1 \leq i \leq 10|\!\}\cup \\
& \{\!|(a1 = 3, b1 = i) \mid 1 \leq i \leq 5|\!\}\cup \\
& \{\!|(a1 = 4, b1 = i) \mid 6 \leq i \leq 10|\!\}\cup
\end{aligned}
$$

$$
t_2 \quad = \quad \{\!|(a2 = 7, b2 = 7), (a2 = 7, b2 = 7)|\!\}
$$

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

$$
k = 2 \rightsquigarrow \{\!|\ (a1 = 1); (a1 = 2); (a1 = 3); (a1 = 4)\ |\!\}
$$

$$
k \neq 2 \rightsquigarrow \{\!|\quad|\!\}
$$

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

SQL: aggregates, nesting, correlated queries

$$
\begin{aligned}
t_1 &= \{\!|(a1 = 1, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
&\quad \{\!|(a1 = 2, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
&\quad \{\!|(a1 = 3, b1 = i) \mid 1 \leq i \leq 5|\!\} \cup \\
&\quad \{\!|(a1 = 4, b1 = i) \mid 6 \leq i \leq 10|\!\} \cup
\end{aligned}
$$

$$
t_2 = \{\!|(a2 = 7, b2 = 7), (a2 = 7, b2 = 7)|\!\}
$$

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

$$k = 2 \rightsquigarrow \{\!|\ (a1 = 1); (a1 = 2); (a1 = 3); (a1 = 4)\ |\!\}$$

$$k \neq 2 \rightsquigarrow \{\!|\ \ |\!\}$$

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

Tentative conclusion: `1+0*a2 = 1`

SQL: aggregates, nesting, correlated queries

$$
\begin{array}{rcl}
t_1 & = & \{\!|(a1 = 1, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
 & & \{\!|(a1 = 2, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
 & & \{\!|(a1 = 3, b1 = i) \mid 1 \leq i \leq 5|\!\} \cup \\
 & & \{\!|(a1 = 4, b1 = i) \mid 6 \leq i \leq 10|\!\} \cup
\end{array}
$$

$$
t_2 = \{\!|(a2 = 7, b2 = 7), (a2 = 7, b2 = 7)|\!\}
$$

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

$$
\begin{array}{rcl}
t_1 & = & \{\!|(a1 = 1, b1 = i) \mid 1 \leq i \leq 10|\!\}\cup \\
 & & \{\!|(a1 = 2, b1 = i) \mid 1 \leq i \leq 10|\!\}\cup \\
 & & \{\!|(a1 = 3, b1 = i) \mid 1 \leq i \leq 5|\!\}\cup \\
 & & \{\!|(a1 = 4, b1 = i) \mid 6 \leq i \leq 10|\!\}\cup
\end{array}
$$

$$
t_2 = \{\!|(a2 = 7, b2 = 7), (a2 = 7, b2 = 7)|\!\}
$$

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

$$
\begin{array}{rcl}
t_1 & = & \{\!|(a1 = 1, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
 & & \{\!|(a1 = 2, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
 & & \{\!|(a1 = 3, b1 = i) \mid 1 \leq i \leq 5|\!\} \cup \\
 & & \{\!|(a1 = 4, b1 = i) \mid 6 \leq i \leq 10|\!\} \cup
\end{array}
$$

$$
t_2 = \{\!|(a2 = 7, b2 = 7), (a2 = 7, b2 = 7)|\!\}
$$

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

$$
\{\!|\ (a1 = 1); (a1 = 2)\ |\!\}
$$

`sum(1+0*a1)` is evaluated for each `a1`-group

SQL: aggregates, nesting, correlated queries

$$
\begin{aligned}
t_1 \quad = \quad & \{\!|(a1 = 1, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
& \{\!|(a1 = 2, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
& \{\!|(a1 = 3, b1 = i) \mid 1 \leq i \leq 5|\!\} \cup \\
& \{\!|(a1 = 4, b1 = i) \mid 6 \leq i \leq 10|\!\} \cup
\end{aligned}
$$

$$
t_2 \quad = \quad \{\!|(a2 = 7, b2 = 7), (a2 = 7, b2 = 7)|\!\}
$$

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

$$
\{\!|\ (a1 = 1); (a1 = 2)\ |\!\}
$$

`sum(1+0*a1)` is evaluated for each `a1`-group

Conclusion: `1 <> 1+0*a1` in some contexts (since Q9(10) $\neq$ Q10)

SQL: aggregates, nesting, correlated queries

$$
\begin{aligned}
t_1 \quad = \quad & \{\!|(a1 = 1, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
& \{\!|(a1 = 2, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
& \{\!|(a1 = 3, b1 = i) \mid 1 \leq i \leq 5|\!\} \cup \\
& \{\!|(a1 = 4, b1 = i) \mid 6 \leq i \leq 10|\!\} \cup
\end{aligned}
$$

$$
t_2 \quad = \quad \{\!|(a2 = 7, b2 = 7), (a2 = 7, b2 = 7)|\!\}
$$

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

$$
\begin{aligned}
t_1 &= \{\!|(a1 = 1, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
&\quad \{\!|(a1 = 2, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
&\quad \{\!|(a1 = 3, b1 = i) \mid 1 \leq i \leq 5|\!\} \cup \\
&\quad \{\!|(a1 = 4, b1 = i) \mid 6 \leq i \leq 10|\!\} \cup
\end{aligned}
$$

$$
t_2 = \{\!|(a2 = 7, b2 = 7), (a2 = 7, b2 = 7)|\!\}
$$

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);
```

$$
k = 7 \rightsquigarrow \{\!|\ (a1 = 3); (a1 = 4)\ |\!\} \quad k = 12 \rightsquigarrow \{\!|\ (a1 = 1); (a1 = 2)\ |\!\}
$$

$$
k \neq 7, k \neq 12 \rightsquigarrow \{\!|\quad|\!\}
$$

SQL: aggregates, nesting, correlated queries

$$
\begin{aligned}
t_1 \quad = \quad & \{\!|(a1 = 1, b1 = i) \mid 1 \leq i \leq 10|\!\}\cup \\
& \{\!|(a1 = 2, b1 = i) \mid 1 \leq i \leq 10|\!\}\cup \\
& \{\!|(a1 = 3, b1 = i) \mid 1 \leq i \leq 5|\!\}\cup \\
& \{\!|(a1 = 4, b1 = i) \mid 6 \leq i \leq 10|\!\}\cup
\end{aligned}
$$

$$
t_2 \quad = \quad \{\!|(a2 = 7, b2 = 7), (a2 = 7, b2 = 7)|\!\}
$$

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);
```

$k = 7 \leadsto \{\!|\ (a1 = 3); (a1 = 4)\ |\!\}$ $\quad k = 12 \leadsto \{\!|\ (a1 = 1); (a1 = 2)\ |\!\}$

$$k \neq 7, k \neq 12 \leadsto \{\!|\quad|\!\}$$

Different sub-expressions of same expression are evaluated
in different environments

SQL: aggregates, nesting, correlated queries

$$
\begin{array}{rcl}
t_1 & = & \{\!|(a1 = 1, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
 & & \{\!|(a1 = 2, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
 & & \{\!|(a1 = 3, b1 = i) \mid 1 \leq i \leq 5|\!\} \cup \\
 & & \{\!|(a1 = 4, b1 = i) \mid 6 \leq i \leq 10|\!\} \cup
\end{array}
$$

$$
t_2 = \{\!|(a2 = 7, b2 = 7), (a2 = 7, b2 = 7)|\!\}
$$

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

$$
\begin{aligned}
t_1 \quad = \quad & \{\!|(a1 = 1, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
& \{\!|(a1 = 2, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
& \{\!|(a1 = 3, b1 = i) \mid 1 \leq i \leq 5|\!\} \cup \\
& \{\!|(a1 = 4, b1 = i) \mid 6 \leq i \leq 10|\!\} \cup
\end{aligned}
$$

$$
t_2 \quad = \quad \{\!|(a2 = 7, b2 = 7), (a2 = 7, b2 = 7)|\!\}
$$

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);
```

$$
k = 2 \rightsquigarrow \{\!|\ (a1 = 1); (a1 = 2); (a1 = 3); (a1 = 4)\ |\!\}
$$

$$
k \neq 2 \rightsquigarrow \{\!|\quad|\!\}
$$

SQL: aggregates, nesting, correlated queries

$$
\begin{array}{rcl}
t_1 & = & \{\!|(a1 = 1, b1 = i) \mid 1 \leq i \leq 10|\!\}\cup \\
 & & \{\!|(a1 = 2, b1 = i) \mid 1 \leq i \leq 10|\!\}\cup \\
 & & \{\!|(a1 = 3, b1 = i) \mid 1 \leq i \leq 5|\!\}\cup \\
 & & \{\!|(a1 = 4, b1 = i) \mid 6 \leq i \leq 10|\!\}\cup
\end{array}
$$

$$
t_2 = \{\!|(a2 = 7, b2 = 7), (a2 = 7, b2 = 7)|\!\}
$$

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);
```

$$
k = 2 \rightsquigarrow \{\!|\ (a1 = 1); (a1 = 2); (a1 = 3); (a1 = 4)\ |\!\}
$$

$$
k \neq 2 \rightsquigarrow \{\!|\quad|\!\}
$$

`sum(1+0*a1+0*a2)` is evaluated for each a2-group

SQL: aggregates, nesting, correlated queries

$$
\begin{aligned}
t_1 &= \{\!|(a1 = 1, b1 = i) \mid 1 \leq i \leq 10|\!\}\cup\\
&\quad \{\!|(a1 = 2, b1 = i) \mid 1 \leq i \leq 10|\!\}\cup\\
&\quad \{\!|(a1 = 3, b1 = i) \mid 1 \leq i \leq 5|\!\}\cup\\
&\quad \{\!|(a1 = 4, b1 = i) \mid 6 \leq i \leq 10|\!\}\cup
\end{aligned}
$$

$$
t_2 = \{\!|(a2 = 7, b2 = 7), (a2 = 7, b2 = 7)|\!\}
$$

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);
```

SQL: aggregates, nesting, correlated queries

$$
\begin{aligned}
t_1 \quad = \quad & \{\!|(a1 = 1, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
& \{\!|(a1 = 2, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
& \{\!|(a1 = 3, b1 = i) \mid 1 \leq i \leq 5|\!\} \cup \\
& \{\!|(a1 = 4, b1 = i) \mid 6 \leq i \leq 10|\!\} \cup
\end{aligned}
$$

$$
t_2 \quad = \quad \{\!|(a2 = 7, b2 = 7), (a2 = 7, b2 = 7)|\!\}
$$

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);
```

$$
k = 2 \rightsquigarrow \{\!| \ (a1 = 1); (a1 = 2); (a1 = 3); (a1 = 4) \ |\!\}
$$

$$
k \neq 2 \rightsquigarrow \{\!| \quad |\!\}
$$

SQL: aggregates, nesting, correlated queries

$$
\begin{array}{rcl}
t_1 & = & \{\!|(a1 = 1, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
 & & \{\!|(a1 = 2, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
 & & \{\!|(a1 = 3, b1 = i) \mid 1 \leq i \leq 5|\!\} \cup \\
 & & \{\!|(a1 = 4, b1 = i) \mid 6 \leq i \leq 10|\!\} \cup
\end{array}
$$

$$
t_2 = \{\!|(a2 = 7, b2 = 7), (a2 = 7, b2 = 7)|\!\}
$$

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);
```

$$
k = 2 \rightsquigarrow \{\!| \; (a1 = 1); (a1 = 2); (a1 = 3); (a1 = 4) \; |\!\}
$$

$$
k \neq 2 \rightsquigarrow \{\!| \quad |\!\}
$$

`sum(1+0*a1+0*b2)` is evaluated for each a2-group

SQL: aggregates, nesting, correlated queries

$$
\begin{array}{rcl}
t_1 & = & \{\!|(a1 = 1, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
 & & \{\!|(a1 = 2, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
 & & \{\!|(a1 = 3, b1 = i) \mid 1 \leq i \leq 5|\!\} \cup \\
 & & \{\!|(a1 = 4, b1 = i) \mid 6 \leq i \leq 10|\!\} \cup
\end{array}
$$

$$
t_2 = \{\!|(a2 = 7, b2 = 7), (a2 = 7, b2 = 7)|\!\}
$$

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);
```

SQL: aggregates, nesting, correlated queries

$$
\begin{aligned}
t_1 &= \{\!|(a1 = 1, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
&\quad \{\!|(a1 = 2, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
&\quad \{\!|(a1 = 3, b1 = i) \mid 1 \leq i \leq 5|\!\} \cup \\
&\quad \{\!|(a1 = 4, b1 = i) \mid 6 \leq i \leq 10|\!\} \cup
\end{aligned}
$$

$$
t_2 = \{\!|(a2 = 7, b2 = 7), (a2 = 7, b2 = 7)|\!\}
$$

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

$$
\begin{aligned}
t_1 &= \{\!|(a1 = 1, b1 = i) \mid 1 \leq i \leq 10|\!\}\cup \\
&\quad \{\!|(a1 = 2, b1 = i) \mid 1 \leq i \leq 10|\!\}\cup \\
&\quad \{\!|(a1 = 3, b1 = i) \mid 1 \leq i \leq 5|\!\}\cup \\
&\quad \{\!|(a1 = 4, b1 = i) \mid 6 \leq i \leq 10|\!\}\cup
\end{aligned}
$$

$$
t_2 = \{\!|(a2 = 7, b2 = 7), (a2 = 7, b2 = 7)|\!\}
$$

Q15:
```
select a1 from t1 group by a1 having
      exists (select a2 from t2 group by a2
                    having sum(1+0*b1+0*a2) = 12);
```

SQL: aggregates, nesting, correlated queries

$$
\begin{aligned}
t_1 &= \{\!|(a1 = 1, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
&\quad \{\!|(a1 = 2, b1 = i) \mid 1 \leq i \leq 10|\!\} \cup \\
&\quad \{\!|(a1 = 3, b1 = i) \mid 1 \leq i \leq 5|\!\} \cup \\
&\quad \{\!|(a1 = 4, b1 = i) \mid 6 \leq i \leq 10|\!\} \cup
\end{aligned}
$$

$$
t_2 = \{\!|(a2 = 7, b2 = 7), (a2 = 7, b2 = 7)|\!\}
$$

Q15:
```
select a1 from t1 group by a1 having
      exists (select a2 from t2 group by a2
                  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 $\rightsquigarrow$ get the (unique) binding of each attribute
- complex expression `function`$(\overline{e})$

  $\rightsquigarrow$ evaluate independently each e_i of $(\overline{e})$

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

Environments

A stack of slices, nesting levels, innermost on top

(attributes, grouping expressions, group of tuples)

Evaluation

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

SQHeLL

SQL$_{Coq}$ Queries

```
Inductive set_op := Union | Intersect | Except.
Inductive select := Select_As : aggterm → attribute → select.
Inductive select_item := Select_Star | Select_List : list select → select_item.
Inductive group_by := Finest_P | Group_By : list funterm → group_by.

Inductive att_renaming :=
  Att_As : attribute → attribute → att_renaming.
Inductive att_renaming_item :=
   Att_Ren_Star | Att_Ren_List : list att_renaming → att_renaming_item.

Inductive sql_query :=
  | Table : relname → sql_query
  | Set : set_op → sql_query → sql_query → sql_query
  | Select :
      (** select *) select_item  →
      (** from *) list from_item →
      (** where *) formula sql_query →
      (** group by *) group_by →
      (** having *) formula sql_query → sql_query

with from_item := From_Item : sql_query → att_renaming_item → sql_from_item.
```

no optional `where`, `group by`, nor `having`

no `where` ⇝ `TTrue`

no `group by` but `having` ⇝ `Group_By nil`

no `group by` nor `having` ⇝ `Finest_P`+ `TTrue`

SQL_{Coq} Formulas

```
Inductive conjunct := And | Or.
Inductive quantifier := All | Any.

Inductive formula (dom : Type) :=
  | Conj : conjunct → formula dom → formula dom → formula dom
  | Not : formula dom → formula dom
  | TTrue : formula dom
  | Pred : predicate → list aggterm → formula dom
  | Quant : list aggterm → predicate → quantifier → dom → formula dom
  | In : list select → dom → formula dom
  | Exists : dom → formula dom.
```

almost FO + `in` + `exists`

$\forall \rightsquigarrow$ `all`

$\exists \rightsquigarrow$ `any`

`in` (membership) $\rightsquigarrow$ $\_ \in \_$ (not a usual predicate over values)

`exists` $\rightsquigarrow$ non-emptiness test

parameterised by `dom`

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.

Definition Bool2 : Bool.Rcd.
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 := [...]

Definition Bool3 : Bool.Rcd.
 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 : Bool.Rcd. (* parametric Booleans *)
Hypothesis I : env_type → dom → bagT (* bags of tuples *).
Fixpoint eval_formula env f : Bool.b B := match f with
  | Conj a f1 f2 ⇒ (interp_conj B a) (eval_formula env f1) (eval_formula env f2)
  | Not f ⇒ Bool.negb B (eval_formula env f)
  | TTrue ⇒ Bool.true B
  | Pred p l ⇒ interp_predicate p (map (interp_aggterm env) l)

  | Quant qtf p l sq ⇒ let lt := map (interp_aggterm env) l in
     interp_quant B qtf (fun x ⇒ let la := Fset.elements _ (labels T x) in
                                  interp_predicate p (lt ++ map (dot T x) la))
                 (Febag.elements _ (I env sq))

  | In s sq ⇒ let p := (projection env (Select_List s)) in
     interp_quant B Any
       (fun x ⇒ match Oeset.compare (OTuple T) p x with
         | Eq ⇒ if contains_null p then unknown else Bool.true B
         | _ ⇒ if (contains_null p || contains_null x) then unknown else Bool.false B end)
       (Febag.elements _ (I env sq))

  | Exists sq ⇒ 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 `NULL`s:

`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 ⇝ checked against Postgresql and Oracle™

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 SQL_{Coq} with an algebra

Define SQL_{Alg}

Extended relational algebra

Enjoying a bag semantics and

Natively accounting for `group by having`

Hence recovering well known algebraic equivalences

SQL_{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 γ 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 γ operator: `Q_Gamma`

`formula`s are shared with SQL_{Coq}

SQL$_{Alg}$'s mechanised semantics

```
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 _ _
      (fun t ⇒ projection (env_t env t) (Select_List s)) (eval_alg_query env q)
  | Q_Sigma f q ⇒
    Febag.filter _
      (fun 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 (fun l ⇒ projection (env_g env (Group_By lf) l) (Select_List s))
      (filter (fun 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 SQL$_{Coq}$

Equivalence

$$SQL_{Coq} \equiv SQL_{Alg}$$

From SQL_{Coq} to SQL_{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 $\rightsquigarrow$ $\bowtie$

Back translation from SQL_{Alg} to SQL_{Coq}

```
Hypothesis fresh (la : list attribute) : attribute.
Hypothesis fresh_is_fresh : ∀s, Oset.mem_bool (OAtt T) (fresh s) s = 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.
```

$\bowtie \rightsquigarrow$ from $\rightsquigarrow$ fresh names needed

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 →
    ∀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 :=
  ∀ 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 :=
  ∀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 → (* cartesian product = natural join thanks to to fresh names *)
    ∀env q, eval_alg_query env q =BE= 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

Lessons : Coq side

```
(* 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 algbraic 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 ⇝ 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`

Epilogue

Data centric languages : a fantastic bestiary

NoSQL, Cassandra, MongoDB, Neo4j, etc weird

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

use Coq to design new languages
$\rightsquigarrow$ completely formalised, clear and well-understood semantics