Can We Trust SQL as a Data Analytics Tool?

SQL

- **The** query language for relational databases
- International Standard since 1987
- Implemented in all systems (free and commercial)
- **$30B/year** business
- Most common tool used by data scientists

Main Questions

- Do we understand SQL queries, even simple ones?
- And if we think we do, do query results make sense?

Asking these questions now?

A bit of history:
before 1969, various ad-hoc database modes (network, hierarchical)

writing queries: a very elaborate task

All changed in 1969: Codd's relational model;
now dominates the world.

Query writing made easy: **SQL**

Relational Model

Orders

ORDER_ID	TITLE	PRICE
Ord1	“Big Data”	30
Ord2	“SQL”	35
Ord3	“Logic”	50

Pay

CUST_ID	ORDER
c1	Ord1
c2	Ord2

Customer

CUST_ID	NAME
c1	John
c2	Mary

Relational Model

Orders

ORDER_ID	TITLE	PRICE
Ord1	"Big Data"	30
Ord2	"SQL"	35
Ord3	"Logic"	50

Pay

CUST_ID	ORDER
c1	Ord1
c2	Ord2

Customer

CUST_ID	NAME
c1	John
c2	Mary

Language: **Relational Algebra (RA)**

- projection **π** (find book titles)
- selection **σ** (find books that cost at least £40)
- Cartesian product **×**
- union **U**
- difference **-**

Queries

Find ids of customers who buy all books:

$$\pi_{\text{cust\_id}}\,(\text{Pay})\;-$$

$$\pi_{\text{cust\_id}}\left((\pi_{\text{cust\_id}}(\text{Pay}) \times \pi_{\text{title}}(\text{Order})\right) -$$

$$\pi_{\text{cust\_id,title}}\left(\sigma_{\text{order\_id=order}}\,(\text{Order} \times \text{Pay})\right)\Big)$$

Queries

Find ids of customers who buy all books:

$$\pi_{\text{cust\_id}} (\text{Pay}) - \pi_{\text{cust\_id}} \Big((\pi_{\text{cust\_id}}(\text{Pay}) \times \pi_{\text{title}}(\text{Order})\Big) - \pi_{\text{cust\_id,title}} \Big(\sigma_{\text{order\_id=order}} (\text{Order} \times \text{Pay})\Big)\Big)$$

That's not pretty. But here is a better idea (1971): express queries in **logic**

Queries

Find ids of customers who buy all books:

$$\pi_{\text{cust\_id}}(\text{Pay}) - \pi_{\text{cust\_id}}\Big(\big(\pi_{\text{cust\_id}}(\text{Pay}) \times \pi_{\text{title}}(\text{Order})\big) - \pi_{\text{cust\_id,title}}\big(\sigma_{\text{order\_id=order}}(\text{Order} \times \text{Pay})\big)\Big)$$

That's not pretty. But here is a better idea (1971):
express queries in **logic**

$$\{c \mid \forall(o,t,p) \in \text{Order}\ \ \exists\,(o',t,p') \in \text{Order}: (c,o') \in \text{Pay}\}$$

Queries

Find ids of customers who buy all books:

π_{cust_id} (Pay) -

π_{cust_id} ((π $_{cust\_id}$(Pay) $\times$ π_{title}(Order)) -

$\pi_{cust\_id,title}$ ($\sigma_{order\_id=order}$ (Order $\times$ Pay)))

That's not pretty. But here is a better idea (1971):
express queries in **logic**

{c | ∀(o,t,p) ∊ Order ∃ (o',t,p') ∊ Order: (c,o') ∊ Pay}

This is *first-order logic* (FO).
Codd 1971: ***RA = FO***.

History continued

Of course programmers don't write logical sentences, they need a programming syntax. Enters **SQL:**

```
SELECT P.cust_id FROM P
WHERE NOT EXISTS
     (SELECT * FROM Order O
      WHERE NOT EXISTS
          (SELECT * FROM Order O1
           WHERE O1.title=O.title AND O1.order_id=P.order))
```

History continued

Of course programmers don't write logical sentences, they need a programming syntax. Enters **SQL**:

```
SELECT P.cust_id FROM P
WHERE NOT EXISTS
     (SELECT * FROM Order O
      WHERE NOT EXISTS
           (SELECT * FROM Order O1
            WHERE O1.title=O.title AND O1.order_id=P.order))
```

$$\forall x F(x) = \neg \exists x \neg F(x)$$

History continued

Of course programmers don't write logical sentences, they need a programming syntax. Enters **SQL**:

```
SELECT P.cust_id FROM P
WHERE NOT EXISTS
     (SELECT * FROM Order O
      WHERE NOT EXISTS
          (SELECT * FROM Order O1
           WHERE O1.title=O.title AND O1.order_id=P.order))
```

$$\forall x F(x) = \neg \exists x \neg F(x)$$

- Take FO and turn into into programming syntax:
- Committee design!
- Then use RA to implement queries.

SQL development

- Standards: SQL-86, SQL-89, SQL-92, SQL:1999, SQL:2003, SQL:2008, SQL:2011, SQL:2016
  - The latest standard will make you $1000 poorer
- The core remains the same.
- And yet things are not as obvious as they should be.
- Now a few quiz-type slides….

TASK: Relations R(A), S(A)

Compute R - S.

TASK: Relations R(A), S(A)
Compute R - S.

Every student will write:

```
select R.A from R where R.A not in (select S.A from S)
```

TASK: Relations R(A), S(A)

Compute R - S.

Every student will write:

```
select R.A from R where R.A not in (select S.A from S)
```

And they are taught it is equivalent to :

```
select R.A from R
where not exists (select S.A from S where S.A=R.A)
```

TASK: Relations R(A), S(A)
Compute R - S.

Every student will write:

```
select R.A from R where R.A not in (select S.A from S)
```

And they are taught it is equivalent to :

```
select R.A from R
where not exists (select S.A from S where S.A=R.A)
```

and that they can do it directly in SQL:

```
select * from r except select * from s
```

TASK: Relations R(A), S(A)
Compute R - S.

R

A
1
null

S

A
null

Every student will write:

```
select R.A from R where R.A not in (select S.A from S)
```

And they are taught it is equivalent to :

```
select R.A from R
where not exists (select S.A from S where S.A=R.A)
```

and that they can do it directly in SQL:

```
select * from r except select * from s
```

TASK: Relations R(A), S(A)
Compute R - S.

R

A
1
null

S

A
null

Outputs:

Every student will write:

```
select R.A from R where R.A not in (select S.A from S)
```

And they are taught it is equivalent to :

```
select R.A from R
where not exists (select S.A from S where S.A=R.A)
```

and that they can do it directly in SQL:

```
select * from r except select * from s
```

TASK: Relations R(A), S(A)
Compute R - S.

R

A
1
null

S

A
null

Outputs:

A

Every student will write:

```
select R.A from R where R.A not in (select S.A from S)
```

And they are taught it is equivalent to :

```
select R.A from R
where not exists (select S.A from S where S.A=R.A)
```

and that they can do it directly in SQL:

```
select * from r except select * from s
```

TASK: Relations R(A), S(A)
Compute R - S.

R

A
1
null

S

A
null

Outputs:

A

Every student will write:

```
select R.A from R where R.A not in (select S.A from S)
```

A
1
null

And they are taught it is equivalent to :

```
select R.A from R
where not exists (select S.A from S where S.A=R.A)
```

and that they can do it directly in SQL:

```
select * from r except select * from s
```

TASK: Relations R(A), S(A)
Compute R - S.

R

A
1
null

S

A
null

Outputs:

Every student will write:

```
select R.A from R where R.A not in (select S.A from S)
```

A

And they are taught it is equivalent to :

```
select R.A from R
where not exists (select S.A from S where S.A=R.A)
```

A
1
null

and that they can do it directly in SQL:

```
select * from r  except  select * from s
```

A
1

SELECT * should be simple, no?

R

Q = SELECT R.A, R.A FROM R on

A
1

gives

A	A
1	1

SELECT * should be simple, no?

`Q = SELECT R.A, R.A FROM R` on

R

A
1

gives

A	A
1	1

Let’s use it as a subquery:

`Q’ = SELECT * FROM (Q) AS T`

SELECT * should be simple, no?

`Q = SELECT R.A, R.A FROM R` on

R

A
1

gives

A	A
1	1

Let's use it as a subquery:

`Q' = SELECT * FROM (Q) AS T`

<u>Output:</u>

- Postgres: as above
- Oracle, MS SQL Server: compile-time error

SELECT * should be simple, no?

R

Q = SELECT R.A, R.A FROM R on

A
1

gives

A	A
1	1

Let's use it as a subquery:

Q' = SELECT * FROM (Q) AS T

<u>Output:</u>

- Postgres: as above
- Oracle, MS SQL Server: compile-time error

SELECT R.A FROM R WHERE EXISTS (Q')

SELECT * should be simple, no?

Q = `SELECT R.A, R.A FROM R` on R

A
1

gives

A	A
1	1

Let's use it as a subquery:

Q' = `SELECT * FROM (Q) AS T`

Output:

- Postgres: as above
- Oracle, MS SQL Server: compile-time error

`SELECT R.A FROM R WHERE EXISTS (Q')`

Answer:

A
1

SELECT * should be simple, no?

`Q = SELECT R.A, R.A FROM R` on

R

A
1

gives

A	A
1	1

Let’s use it as a subquery:

`Q’ = SELECT * FROM (Q) AS T`

<u>Output:</u>

- Postgres: as above
- Oracle, MS SQL Server: compile-time error

`SELECT R.A FROM R WHERE EXISTS (Q’ )`

Answer:

A
1

Except in MySQL

Another example: Query equivalences

Q1(x) :- T(x,y)
Q2(x) :- T(x,y), T(u,v)

Another example: Query equivalences

Q1(x) :- T(x,y)
Q2(x) :- T(x,y), T(u,v)

In theory: equivalent; on

A	B
1	2
3	4

return

A
1
3

Another example: Query equivalences

Q1(x) :- T(x,y)
Q2(x) :- T(x,y), T(u,v)

In theory: equivalent; on

A	B
1	2
3	4

return

A
1
3

Now the same in SQL:

Another example: Query equivalences

Q1(x) :- T(x,y)
Q2(x) :- T(x,y), T(u,v)

In theory: equivalent; on

A	B
1	2
3	4

return

A
1
3

Now the same in SQL:

Q1 = `SELECT R.A FROM R` returns

A
1
3

Another example: Query equivalences

Q1(x) :- T(x,y)
Q2(x) :- T(x,y), T(u,v)

In theory: equivalent; on

A	B
1	2
3	4

return

A
1
3

Now the same in SQL:

Q1 = `SELECT R.A FROM R` returns

A
1
3

Q2 = `SELECT R1.A FROM R R1, R R2` returns

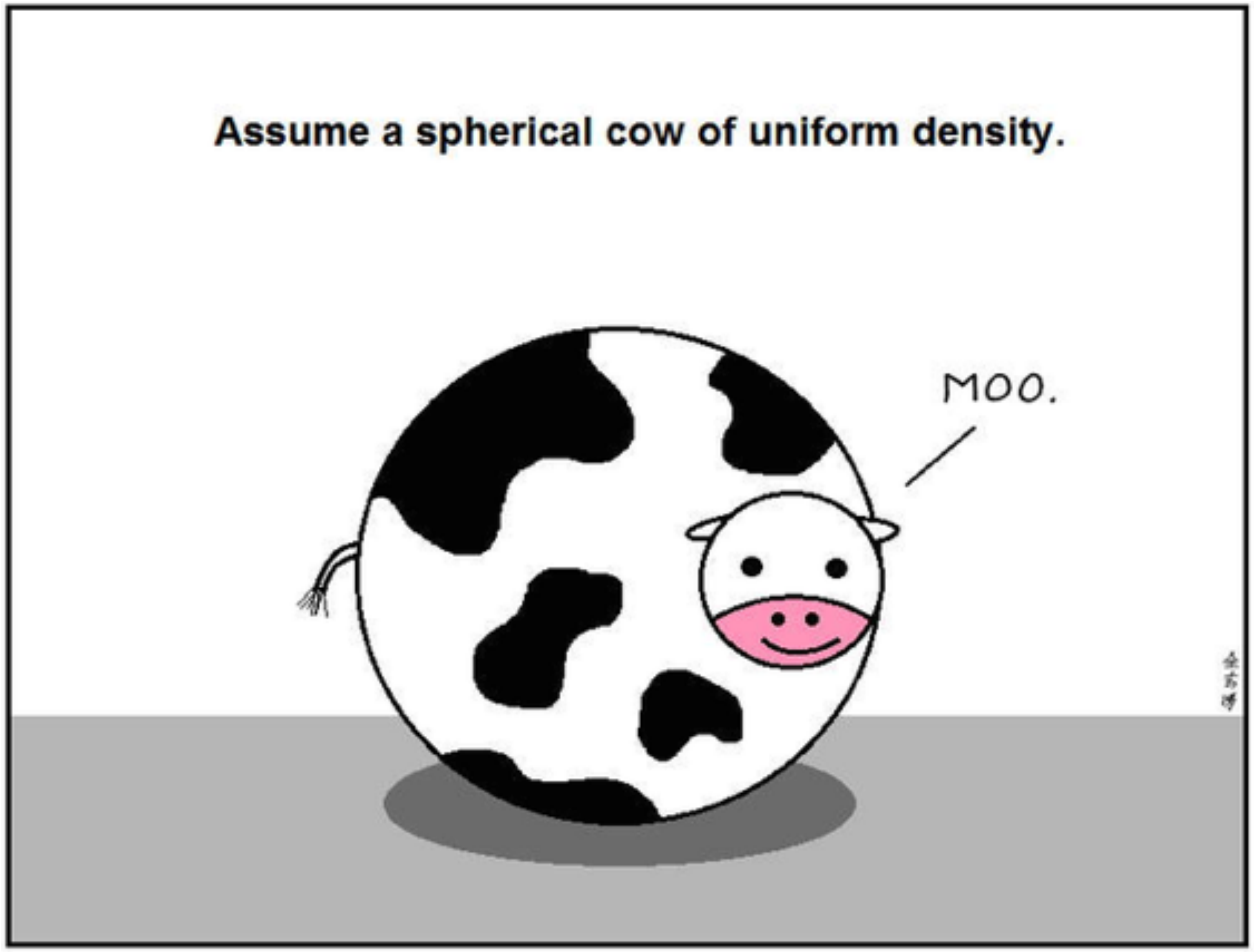
A
1
1
3
3

Why do we find these questions difficult?

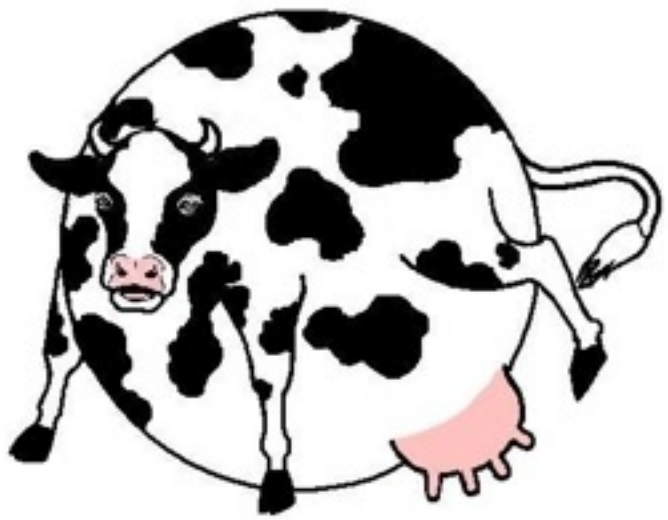
- Reason 1: there is no formal semantics of SQL.
  - The Standard is rather vague, not written formally, and different vendors interpret it differently.
- Reason 2: theory works with a simplified model, no nulls, no duplicates, no repeated attributes.
  - Under these assumptions several semantics exist (1985 - 2017) but they do not model the real language.

It is much harder to deal with the real thing than with theoretical abstractions

It is much harder to deal with the real thing than with theoretical abstractions



From spherical to real cows



- We do it for the basic fragment of SQL:
    - `SELECT-FROM-WHERE` without aggregation
    - but with pretty much everything else

[G.,L. *A Formal Semantics of SQL Queries, its Validation and Applications*. PVLDB 2017]

Syntax

$$\tau : \beta \;:=\; T_1 \;\texttt{AS}\; N_1,\; \ldots,\; T_k \;\texttt{AS}\; N_k \quad \text{for } \tau = (T_1, \ldots, T_k),\; \beta = (N_1, \ldots, N_k), \quad k > 0$$

$$\alpha : \beta' \;:=\; t_1 \;\texttt{AS}\; N'_1,\; \ldots,\; t_m \;\texttt{AS}\; N'_k \quad \text{for } \alpha = (t_1, \ldots, t_m),\; \beta' = (N'_1, \ldots, N'_m),\; m > 0$$

QUERIES:

$$\begin{aligned} Q \;:=\;& \texttt{SELECT}\; [\texttt{DISTINCT}]\; \alpha : \beta' \;\texttt{FROM}\; \tau : \beta \;\texttt{WHERE}\; \theta \\ \mid\;& \texttt{SELECT}\; [\texttt{DISTINCT}]\; * \;\texttt{FROM}\; \tau : \beta \;\texttt{WHERE}\; \theta \\ \mid\;& Q\; (\texttt{UNION} \mid \texttt{INTERSECT} \mid \texttt{EXCEPT})\; [\texttt{ALL}]\; Q \end{aligned}$$

CONDITIONS:

$$\begin{aligned} \theta \;:=\;& \texttt{TRUE} \mid \texttt{FALSE} \mid P(t_1, \ldots, t_k),\; P \in \mathcal{P} \\ \mid\;& t \;\texttt{IS}\; [\texttt{NOT}]\; \texttt{NULL} \\ \mid\;& \bar{t}\; [\texttt{NOT}]\; \texttt{IN}\; Q \mid \texttt{EXISTS}\; Q \\ \mid\;& \theta \;\texttt{AND}\; \theta \mid \theta \;\texttt{OR}\; \theta \mid \texttt{NOT}\; \theta \end{aligned}$$

Names: either simple (R, A) or composite (R.A)

Terms t: constants, nulls, or composite names

Predicates: anything you want on constants

Semantics: labels

$$\ell(R) = \text{tuple of names provided by the schema}$$

$$\ell(\tau) = \ell(T_1)\cdots\ell(T_k) \quad \text{for } \tau = (T_1,\ldots,T_k)$$

$$\ell\begin{pmatrix} \texttt{SELECT}\ [\texttt{DISTINCT}]\ \alpha : \beta' \\ \texttt{FROM}\ \tau : \beta\ \texttt{WHERE}\ \theta \end{pmatrix} = \beta'$$

$$\ell\big(\texttt{SELECT}\ [\texttt{DISTINCT}] * \texttt{FROM}\ \tau : \beta\ \texttt{WHERE}\ \theta\big) = \ell(\tau)$$

$$\ell\big(Q_1\ (\texttt{UNION} \mid \texttt{INTERSECT} \mid \texttt{EXCEPT})\ [\texttt{ALL}]\ Q_2\big) = \ell(Q_1)$$

Semantics

$$[\![Q]\!]_{D,\eta,x}$$

Q: query

D: database

η: environment (values for composite names)

x: Boolean switch to account for non-compositional nature of SELECT * (to show where we are in the query)

Semantics of terms

$$\llbracket t \rrbracket_\eta = \begin{cases} \eta(A) & \text{if } t = A \\ c & \text{if } t = c \in \mathsf{C} \\ \texttt{NULL} & \text{if } t = \texttt{NULL} \end{cases}$$

$$\llbracket (t_1, \ldots, t_n) \rrbracket_\eta = \big(\llbracket t_1 \rrbracket_\eta, \ldots, \llbracket t_n \rrbracket_\eta \big)$$

Semantics: queries

$$[\![R]\!]_{D,\eta,x} = R^D$$

$$[\![\tau:\beta]\!]_{D,\eta,x} = [\![T_1]\!]_{D,\eta,0} \times \cdots \times [\![T_k]\!]_{D,\eta,0} \quad \text{for } \tau = (T_1,\ldots,T_k)$$

$$\left[\!\!\left[\begin{array}{rl} \textbf{FROM} & \tau:\beta \\ \textbf{WHERE} & \theta \end{array}\right]\!\!\right]_{D,\eta,x} = \left\{ \underbrace{\bar r,\ldots,\bar r}_{k \text{ times}} \;\middle|\; \bar r \in_k [\![\tau:\beta]\!]_{D,\eta,0},\ \ [\![\theta]\!]_{D,\eta'} = \mathbf{t},\ \ \eta' = \eta \overset{\bar r}{\oplus} \ell(\tau:\beta) \right\}$$

$$\left[\!\!\left[\begin{array}{rl} \textbf{SELECT} & \alpha:\beta' \\ \textbf{FROM} & \tau:\beta \\ \textbf{WHERE} & \theta \end{array}\right]\!\!\right]_{D,\eta,x} = \left\{ \underbrace{[\![\alpha]\!]_{\eta'},\ldots,[\![\alpha]\!]_{\eta'}}_{k \text{ times}} \;\middle|\; \eta' = \eta \overset{\bar r}{\oplus} \ell(\tau:\beta),\ \ \bar r \in_k \left[\!\!\left[\begin{array}{rl} \textbf{FROM} & \tau:\beta \\ \textbf{WHERE} & \theta \end{array}\right]\!\!\right]_{D,\eta,x} \right\}$$

$$\left[\!\!\left[\begin{array}{rl} \textbf{SELECT} & * \\ \textbf{FROM} & \tau:\beta \\ \textbf{WHERE} & \theta \end{array}\right]\!\!\right]_{D,\eta,0} = \left[\!\!\left[\begin{array}{rl} \textbf{SELECT} & \ell(\tau:\beta):\ell(\tau) \\ \textbf{FROM} & \tau:\beta \\ \textbf{WHERE} & \theta \end{array}\right]\!\!\right]_{D,\eta,0}$$

$$\left[\!\!\left[\begin{array}{rl} \textbf{SELECT} & * \\ \textbf{FROM} & \tau:\beta \\ \textbf{WHERE} & \theta \end{array}\right]\!\!\right]_{D,\eta,1} = \left[\!\!\left[\begin{array}{rl} \textbf{SELECT} & c \ \textbf{AS} \ N \\ \textbf{FROM} & \tau:\beta \\ \textbf{WHERE} & \theta \end{array}\right]\!\!\right]_{D,\eta,1} \quad \text{for arbitrary } c \in \mathsf{C} \text{ and } N \in \mathsf{N}$$

$$\left[\!\!\left[\begin{array}{l} \textbf{SELECT DISTINCT } \alpha:\beta' \mid * \\ \textbf{FROM } \tau:\beta \textbf{ WHERE } \theta \end{array}\right]\!\!\right]_{D,\eta,x} = \varepsilon\left(\left[\!\!\left[\begin{array}{l} \textbf{SELECT } \alpha:\beta' \mid * \\ \textbf{FROM } \tau:\beta \textbf{ WHERE } \theta \end{array}\right]\!\!\right]_{D,\eta,x} \right)$$

Semantics: conditions

$$[\![P(t_1,\ldots,t_k)]\!]_{D,\eta} = \begin{cases} \mathbf{t} & \text{if } P\big([\![t_1]\!]_\eta,\ldots,[\![t_k]\!]_\eta\big) \text{ holds and } [\![t_i]\!]_\eta \neq \texttt{NULL} \text{ for all } i \in \{1,\ldots,k\} \\ \mathbf{f} & \text{if } P\big([\![t_1]\!]_\eta,\ldots,[\![t_k]\!]_\eta\big) \text{ does not hold and } [\![t_i]\!]_\eta \neq \texttt{NULL} \text{ for all } i \in \{1,\ldots,k\} \\ \mathbf{u} & \text{if } [\![t_i]\!]_\eta = \texttt{NULL} \text{ for some } i \in \{1,\ldots,k\} \end{cases}$$

$$[\![t \texttt{ IS NULL}]\!]_{D,\eta} = \begin{cases} \mathbf{t} & \text{if } [\![t]\!]_\eta = \texttt{NULL} \\ \mathbf{f} & \text{if } [\![t]\!]_\eta \neq \texttt{NULL} \end{cases}$$

$$[\![t \texttt{ IS NOT NULL}]\!]_{D,\eta} = \neg[\![t \texttt{ IS NULL}]\!]_{D,\eta}$$

$$[\![(t_1,\ldots t_n) = (t'_1,\ldots,t'_n)]\!]_{D,\eta} = \bigwedge_{i=1}^{n} [\![t_i = t'_i]\!]_{D,\eta} \qquad [\![(t_1,\ldots t_n) \neq (t'_1,\ldots,t'_n)]\!]_{D,\eta} = \bigvee_{i=1}^{n} [\![t_i \neq t'_i]\!]_{D,\eta}$$

$$[\![\bar{t} \texttt{ IN } Q]\!]_{D,\eta} = \begin{cases} \mathbf{t} & \text{if } \exists \bar{r} \in [\![Q]\!]_{D,\eta,0} \text{ s.t. } [\![\bar{t} = \bar{r}]\!]_{D,\eta} = \mathbf{t} \\ \mathbf{f} & \text{if } \forall \bar{r} \in [\![Q]\!]_{D,\eta,0} \text{ s.t. } [\![\bar{t} = \bar{r}]\!]_{D,\eta} = \mathbf{f} \\ \mathbf{u} & \text{if } \nexists \bar{r} \in [\![Q]\!]_{D,\eta,0} \text{ s.t. } [\![\bar{t} = \bar{r}]\!]_{D,\eta} = \mathbf{t} \text{ and } \exists \bar{r} \in [\![Q]\!]_{D,\eta,0} \text{ s.t. } [\![\bar{t} = \bar{r}]\!]_{D,\eta} \neq \mathbf{f} \end{cases}$$

$$[\![\bar{t} \texttt{ NOT IN } Q]\!]_{D,\eta} = \neg[\![\bar{t} \texttt{ IN } Q]\!]_{D,\eta}$$

$$[\![\texttt{EXISTS } Q]\!]_{D,\eta} = \begin{cases} \mathbf{t} & \text{if } [\![Q]\!]_{D,\eta,1} \neq \varnothing \\ \mathbf{f} & \text{if } [\![Q]\!]_{D,\eta,1} = \varnothing \end{cases}$$

$$[\![\texttt{TRUE}]\!]_{D,\eta} = \mathbf{t} \qquad [\![\theta_1 \texttt{ AND } \theta_2]\!]_{D,\eta} = [\![\theta_1]\!]_{D,\eta} \wedge [\![\theta_2]\!]_{D,\eta} \qquad [\![\texttt{NOT } \theta]\!]_{D,\eta} = \neg[\![\theta]\!]_{D,\eta}$$

$$[\![\texttt{FALSE}]\!]_{D,\eta} = \mathbf{f} \qquad [\![\theta_1 \texttt{ OR } \theta_2]\!]_{D,\eta} = [\![\theta_1]\!]_{D,\eta} \vee [\![\theta_2]\!]_{D,\eta}$$

Truth Tables:

$\wedge$	t	f	u
t	**t**	**f**	**u**
f	**f**	**f**	**f**
u	**u**	**f**	**u**

$\vee$	t	f	u
t	**t**	**t**	**t**
f	**t**	**f**	**u**
u	**t**	**u**	**u**

	$\neg$
t	**f**
f	**t**
u	**u**

Semantics: operations

$$
\begin{aligned}
[\![Q_1 \ \textbf{UNION ALL}\ Q_2]\!]_{D,\eta,x} &= [\![Q_1]\!]_{D,\eta,0} \cup [\![Q_2]\!]_{D,\eta,0} \\
[\![Q_1 \ \textbf{INTERSECT ALL}\ Q_2]\!]_{D,\eta,x} &= [\![Q_1]\!]_{D,\eta,0} \cap [\![Q_2]\!]_{D,\eta,0} \\
[\![Q_1 \ \textbf{EXCEPT ALL}\ Q_2]\!]_{D,\eta,x} &= [\![Q_1]\!]_{D,\eta,0} - [\![Q_2]\!]_{D,\eta,0} \\
[\![Q_1 \ \textbf{UNION}\ Q_2]\!]_{D,\eta,x} &= \varepsilon\big([\![Q_1 \ \textbf{UNION ALL}\ Q_2]\!]_{D,\eta,x}\big) \\
[\![Q_1 \ \textbf{INTERSECT}\ Q_2]\!]_{D,\eta,x} &= \varepsilon\big([\![Q_1 \ \textbf{INTERSECT ALL}\ Q_2]\!]_{D,\eta,x}\big) \\
[\![Q_1 \ \textbf{EXCEPT}\ Q_2]\!]_{D,\eta,x} &= \varepsilon\big([\![Q_1]\!]_{D,\eta,0}\big) - [\![Q_2]\!]_{D,\eta,0}
\end{aligned}
$$

Bag interpretation of operations; ϵ is duplicate elimination

Looks simple, no?

- It does not. Such basic things as variable binding changed several times till we got them right.

- The meaning of the new environment:

$$\left[\!\!\left[\begin{array}{ll} \mathbf{FROM} & \tau : \beta \\ \mathbf{WHERE} & \theta \end{array} \right]\!\!\right]_{D,\eta,x} = \left\{ \underbrace{\bar{r},\ldots,\bar{r}}_{k \text{ times}} \;\middle|\; \bar{r} \in_k [\![\tau : \beta]\!]_{D,\eta,0},\;\; [\![\theta]\!]_{D,\eta'} = \mathbf{t},\; \boxed{\eta' = \eta \overset{\bar{r}}{\oplus} \ell(\tau : \beta)} \right\}$$

  - in η, unbind every name that occurs among labels of the FROM clause

  - then bind non-repeated names among those to values taken from record r

How do we know we this is correct?

- Since the Standard is rather vague, there is only one way — **experiments.**
- But what kind of benchmark can we use?
- For performance studies there are standard benchmarks like TPC-H. But they won’t work for us: not enough queries.

Experimental Validation

- Benchmarks have rather few queries (22 in TPC-H). Validating on 22 queries is not a good evidence.
- But we can look at benchmarks, and then generate lots of queries that look the same.
- In TPC-H:
  - 8 tables,
  - maximum nesting depth = 3,
  - average number of tables per query = 3.2,
  - at most 8 conditions in WHERE (except two queries)

Validation: results

- Small adjustments of the Standard semantics (for Postgres and Oracle)
- Random query generator
- Naive implementation of the semantics
- Finally: experiments on 100,000 random queries

Validation: results

- Small adjustments of the Standard semantics (for Postgres and Oracle)
- Random query generator
- Naive implementation of the semantics
- Finally: experiments on 100,000 random queries
- **Yes, it is correct!**

What can we do with this?

- Equivalence of basic SQL and Relational Algebra: formally proved for the first time
    - Previous attempts (Ceri and Gottlob, Van den Bussche and Vansummeren restricted the language severely: no nulls, for example).
- 3-valued logic of SQL vs the usual Boolean logic: 3-valued logic does not add expressiveness.
    - Although it does not mean we should get rid of it now…

Does it matter which DBMS we use?

- We already saw it does. In fact in our experiments we adjusted things a bit for Postgres and Oracle.
- But how much of a difference does it make?
- We have a random query generator, so let's experiment:
  - generate lots of queries (over 150K)
  - send to standard DBMSs (Oracle, MySQL, MS SQL Server, PostgreSQL, IBM DB2)
  - and see what happens…

Discrepancies between RDBMSs

- About 2% of queries do not behave the same way on different DBMSs
    - and they come from the most basic fragment
- Lots of issues are minor and syntactic
    - different syntax for set operations (eg `EXCEPT` vs `MINUS`) or functions (eg `%` vs `MOD`, or `substring` vs `substr`)
- But some are serious - and surprise even people with good SQL knowledge. Four of the most surprising examples to follow…

Is empty string equal to itself?

```
SELECT *
FROM   R
WHERE  ''=''
```

Is empty string equal to itself?

```
SELECT *
FROM   R
WHERE ‘’=‘’
```

- Usually it is, but not in Oracle: the above query always returns the empty table.
- Because Oracle implements NULL as ‘’
- Madness? Yes. With a string operation that produces ‘’ you deal with 3-valued logic before you realize it!

Can you divide by zero?

```
SELECT  R.A/S.B
FROM    R, S

R={1},  S={0}
```

Can you divide by zero?

```
SELECT R.A/S.B
FROM   R, S

R={1},  S={0}
```

- Usually not except in MySQL 5.6
- It returns NULL
- OK, they realized it in MySQL 5.7 and now by default it's a warning. But one can go back to the 5.6 mode if one wishes…

Is equality transitive?

`x=y and y=z imply x=z, right?`

Is equality transitive?

`x=y and y=z imply x=z, right?`

- Usually yes, but not in MySQL
- x=’1a’, y=1, z=‘1b’
- Why is this a problem? SQL books teach programmers to overspecify join conditions: to `R.A=S.A AND S.A=T.A` add explicitly `R.A=T.A`
- But now it can turn a true condition into false!

Can you compare tuples in IN subqueries?

```
SELECT  *
FROM    R
WHERE  (R.A, R.B) IN SELECT (S.A, S.B FROM S)
```

Can you compare tuples in IN subqueries?

```
SELECT  *
FROM    R
WHERE  (R.A, R.B) IN SELECT (S.A, S.B FROM S)
```

- Usually yes, except in MS SQL Server.
- Why? No clue…

- Also SQL Server has UNION but no UNION ALL.
- Please explain this.

A simple tool

- We actually have a tool that lets you:
  - specify parameters of a query workload
  - generate lots of random queries, and
  - run against DBMSs you want to compare
- Have fun with results… at least you know what to expect.