Automated Reasoning of Database Queries

Shumo Chu, University of Washington
with Konstantin Weitz, Chenglong Wang, Daniel Li, Brendan Murphy,
Jared Roesch, Alvin Cheung, Dan Suciu

PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

- SQL: a language supported by all relational databases

- Restricted abstraction enabling powerful optimizations

- 30 years of research results in many optimizations based on semantic equivalent rewrites

https://insights.stackoverflow.com/survey/2017

Lack tools that can reason about SQL equivalences

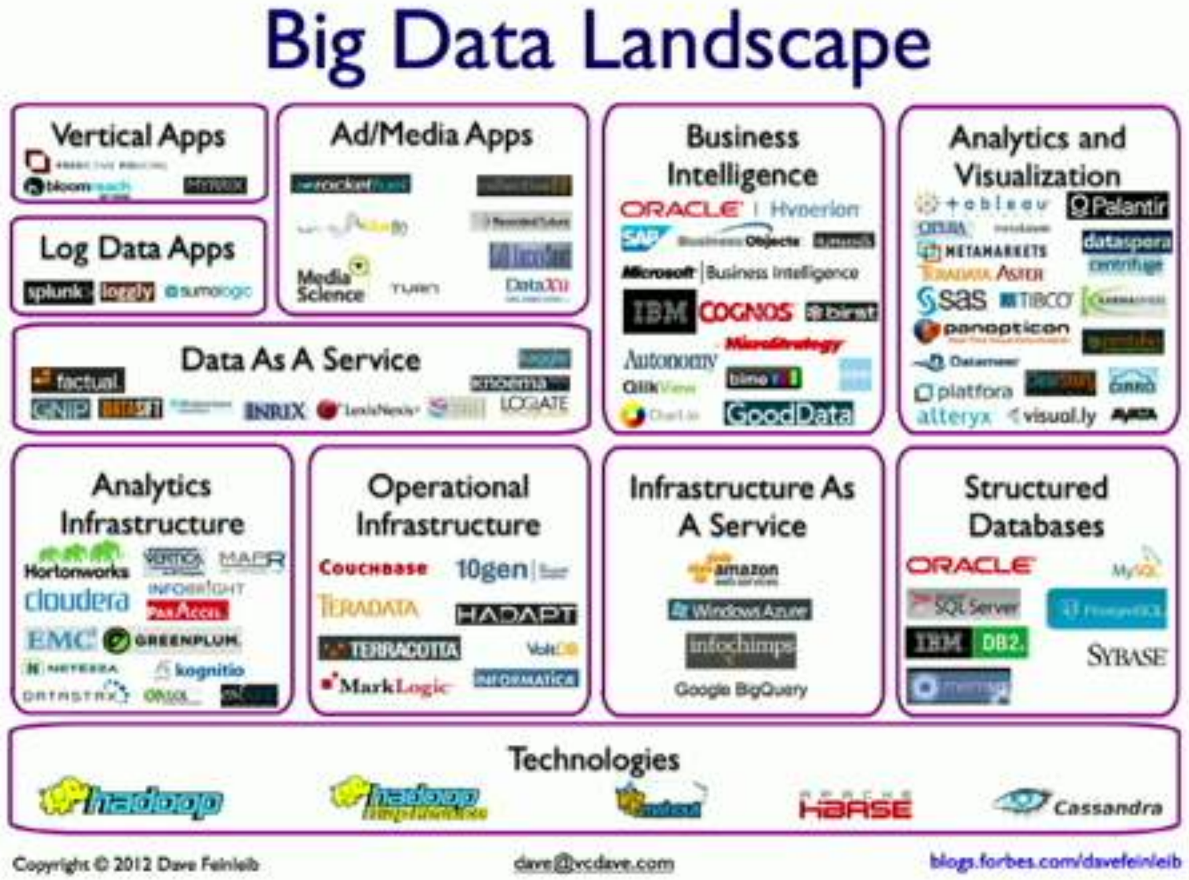
Automated Solver for SQL: Q1 = Q2?

Automated Solver for SQL:

∀ possible input, Q1 = Q2?

Automated Solver for SQL:
∀ possible input, Q1 = Q2?

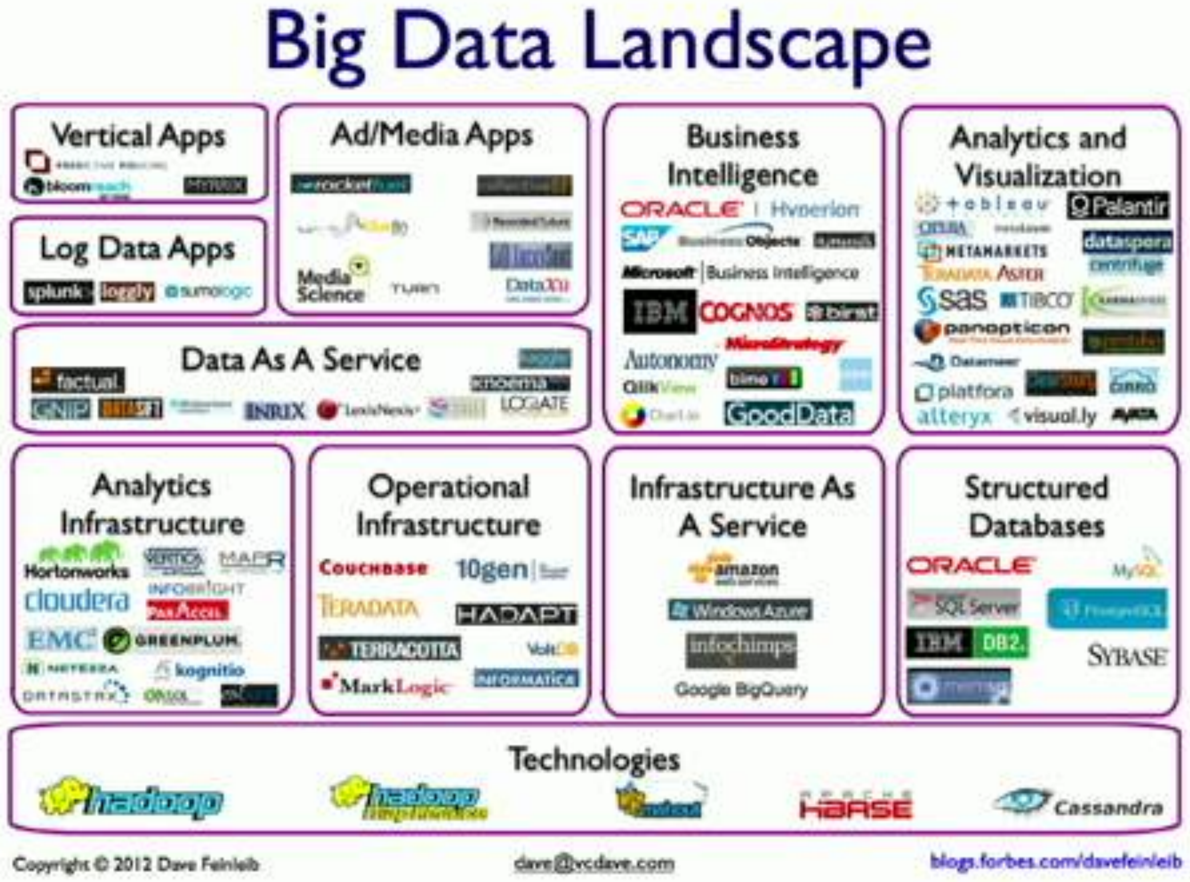
- Correctness of query rewrite



Automated Solver for SQL:

$\forall$ possible input, Q1 = Q2?

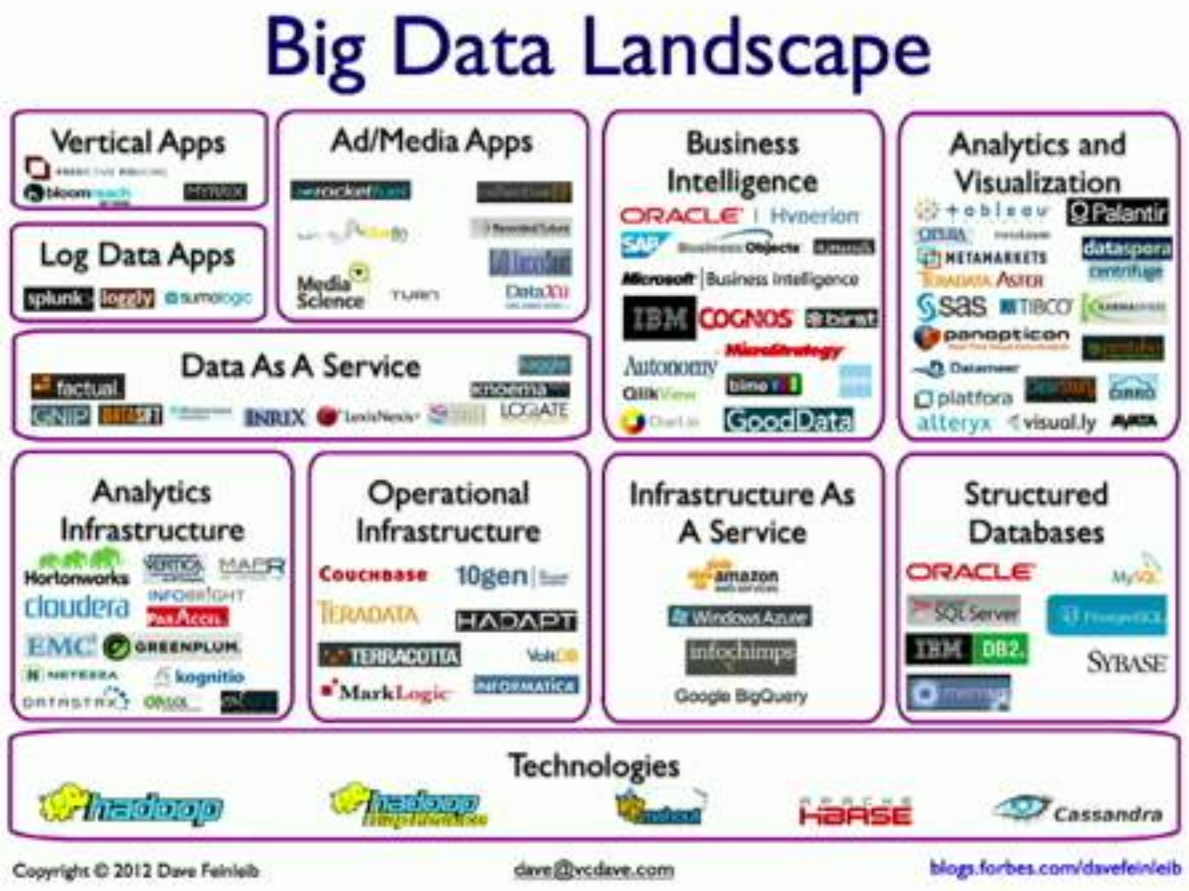
- Correctness of query rewrite
- Semantics layer of data systems



Automated Solver for SQL:

∀ possible input, Q1 = Q2?

- Correctness of query rewrite
- Semantics layer of data systems
- Auto grading on SQL assignments



Challenges

- Checking equivalences of two FO sentences over finite models is undecidable ☹
- Rich language features
  - Aggregation and grouping
  - Index and integrity constraints
  - Correlated subqueries

Boris A. Trakhtenbrot

Undecidability ≠ No Proof

→

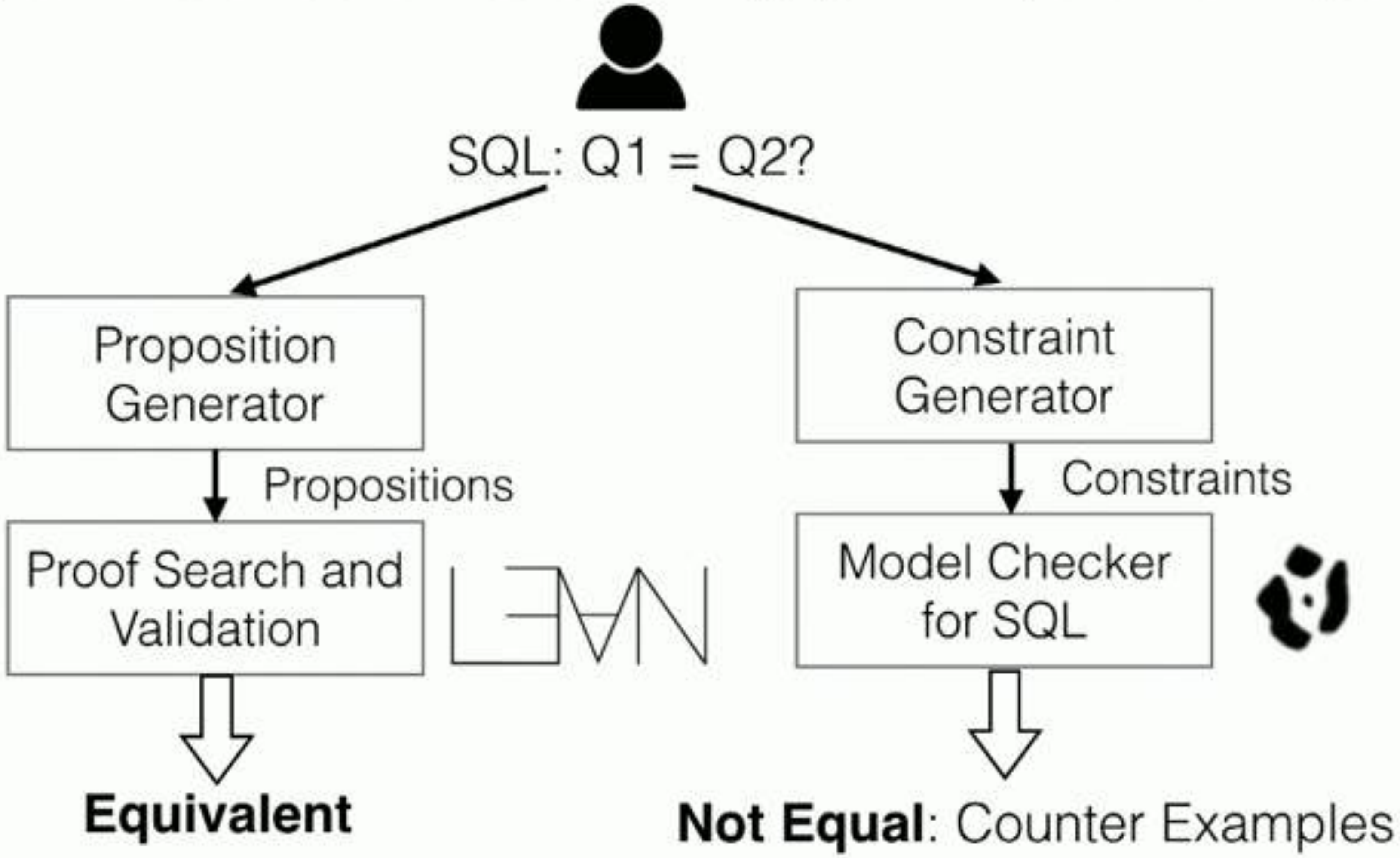
Interactive Theorem Prover that validates mechanized proofs

Undecidability ≠ No Proof → **Interactive Theorem Prover** that validates mechanized proofs

The model of an inequivalence is usually not large → **Constraint Solver** for Model Checking

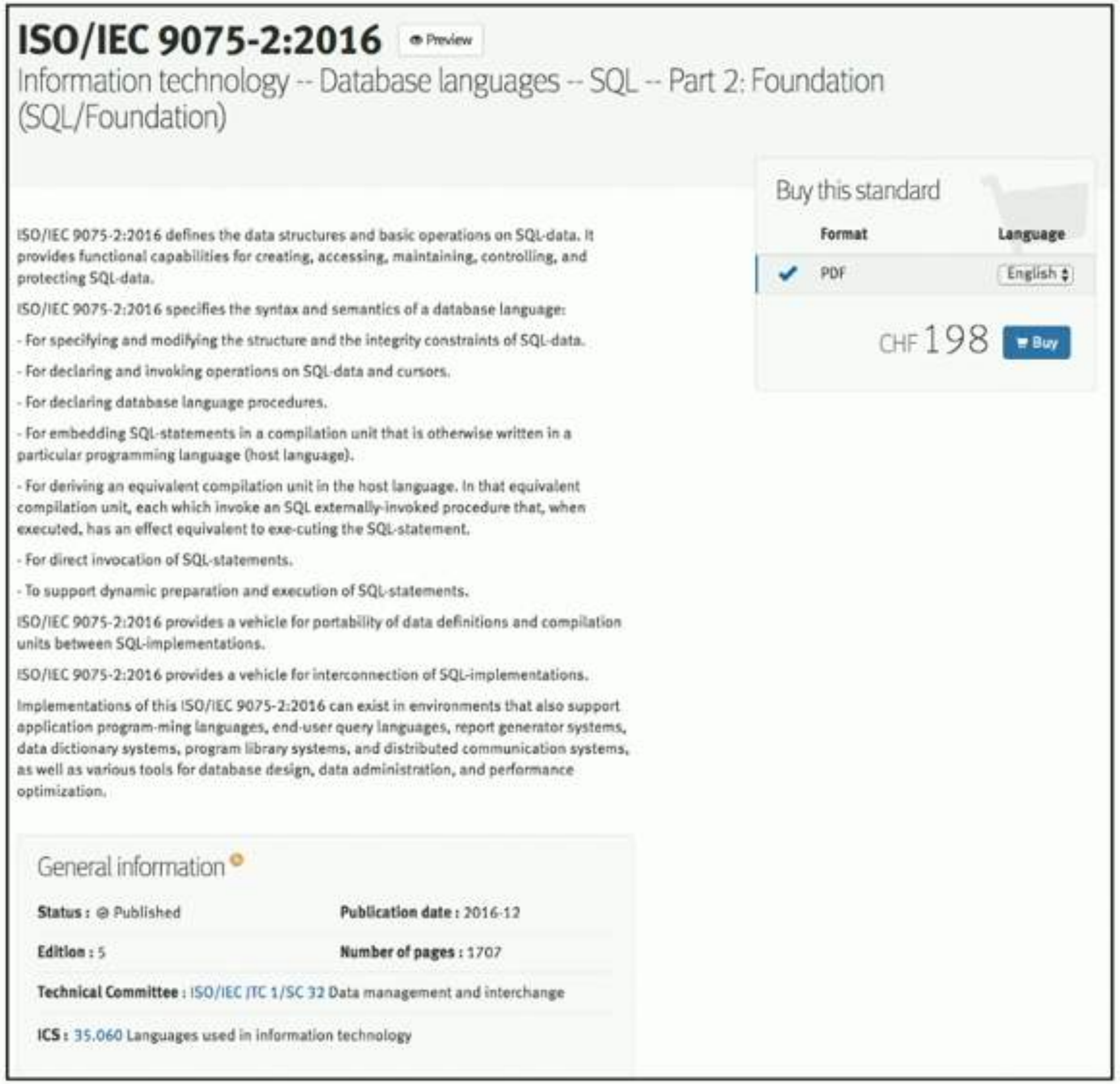
Construct a SQL Solver

- Cosette: An (almost) automated solver for SQL by *combining interactive theorem proving and constraints solving* (PLDI 17, SIGMOD17, CIDR17, VLDB18)



The Search for Formal SQL Semantics

What is SQL?



ISO/IEC 9075-2:2016

Preview

Information technology -- Database languages -- SQL -- Part 2: Foundation (SQL/Foundation)

ISO/IEC 9075-2:2016 defines the data structures and basic operations on SQL-data. It provides functional capabilities for creating, accessing, maintaining, controlling, and protecting SQL-data.

ISO/IEC 9075-2:2016 specifies the syntax and semantics of a database language:

- For specifying and modifying the structure and the integrity constraints of SQL-data.
- For declaring and invoking operations on SQL-data and cursors.
- For declaring database language procedures.
- For embedding SQL-statements in a compilation unit that is otherwise written in a particular programming language (host language).
- For deriving an equivalent compilation unit in the host language. In that equivalent compilation unit, each which invoke an SQL externally-invoked procedure that, when executed, has an effect equivalent to exe-cuting the SQL-statement.
- For direct invocation of SQL-statements.
- To support dynamic preparation and execution of SQL-statements.

ISO/IEC 9075-2:2016 provides a vehicle for portability of data definitions and compilation units between SQL-implementations.

ISO/IEC 9075-2:2016 provides a vehicle for interconnection of SQL-implementations.

Implementations of this ISO/IEC 9075-2:2016 can exist in environments that also support application program-ming languages, end-user query languages, report generator systems, data dictionary systems, program library systems, and distributed communication systems, as well as various tools for database design, data administration, and performance optimization.

Buy this standard

Format	Language
PDF	English

CHF 198 Buy

General information

Status : Published
Publication date : 2016-12
Edition : 5
Number of pages : 1707
Technical Committee : ISO/IEC JTC 1/SC 32 Data management and interchange
ICS : 35.060 Languages used in information technology

- More than **1,700** pages natural language description
- Insufficient for a rigorous SQL semantics

What is SQL?

- Data model: relations
    - A relation is an unordered collection of tuples
- Schema: set of attributes, each tuple in a relation must have same schema
- SQL query: $Q(R_1, R_2, \ldots, R_k) = R$

Relational Algebra

Relational Algebra (**×**, **σ**, **Π**, ∪, **-**)

- **×**: e.g. *Courses* **×** *Student* = **SELECT** * **FROM** *Course, Student*
- **σ**: e.g. $\sigma_{len\,>\,100}(Movie)$ = **SELECT** * **FROM** *Movie* **WHERE** *len > 100*
- **Π**:e.g. $\Pi_{first}(Name)$ = **SELECT** *first* **FROM** *Name*
- ∪: e.g. $R \cup S$ = (**SELECT** * **FROM** R) **UNION ALL** (**SELECT** * **FROM** S)

Relational Algebra

Relational Algebra ($\boxed{\times, \sigma, \Pi, \cup}$, **-**)

UCQ

- **×**: e.g. *Courses* **×** *Student* = `SELECT * FROM Course, Student`
- **σ**: e.g. $\sigma_{len > 100}(Movie)$ = `SELECT * FROM Movie WHERE len > 100`
- **Π**:e.g. $\Pi_{first}(Name)$ = `SELECT first FROM Name`
- ∪: e.g. $R \cup S$ = `(SELECT * FROM R) UNION ALL (SELECT * FROM S)`

Relational Algebra

- More operators needed
  - $\boldsymbol{\delta}$: e.g. $\boldsymbol{\delta}$**(R)** = `SELECT DISTINCT * FROM R`
  - $\boldsymbol{\gamma}$: e.g. $\boldsymbol{\gamma}_{\texttt{name,count(*)}}$`Person` = `SELECT name, count(*) FROM Person GROUP BY name`
- Constraints: keys, foreign keys …
- Lacking “meta-theory”

Relational Algebra

- More operators needed
  - $\boldsymbol{\delta}$: e.g. $\boldsymbol{\delta}$**(R)** = `SELECT DISTINCT * FROM R`
  - $\boldsymbol{\gamma}$: e.g. $\boldsymbol{\gamma}_{\texttt{name,count(*)}}$`Person = SELECT name, count(*) FROM Person GROUP BY name`
- Constraints: keys, foreign keys …
- Lacking "meta-theory"

Leaking Abstraction: What is the semantics of RA?

Relation as List

Existing Coq Formalization [Malecha et al., POPL 10]

- Two queries are equivalent if returning the **same** result for **all possible** input relations

- Need to reason about two lists are equivalent up to permutations

```
Q1 = SELECT *
     FROM (R UNION ALL S)
     WHERE b
```

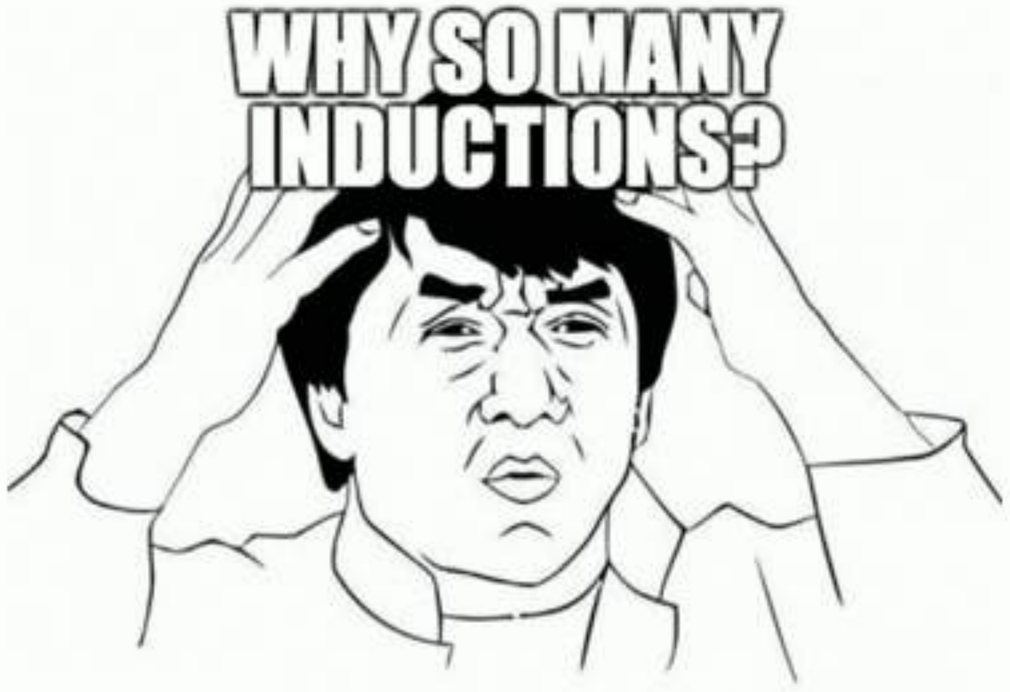
```
Q2 = (SELECT * FROM R WHERE b)
      UNION ALL
     (SELECT * FROM S WHERE b)
```

```
Q1 = SELECT *
     FROM (R UNION ALL S)
     WHERE b
```

```
Q2 = (SELECT * FROM R WHERE b)
      UNION ALL
     (SELECT * FROM S WHERE b)
```

Q1 = Q2 ?

```
Induction on R:
  Assume Q1 == Q2 when R has N tuples
  Then when R is of size N+1:
    ...

Induction on S:
  Assume Q1 == Q2 when S has N tuples
  Then when S is of size N+1:
    ...
```

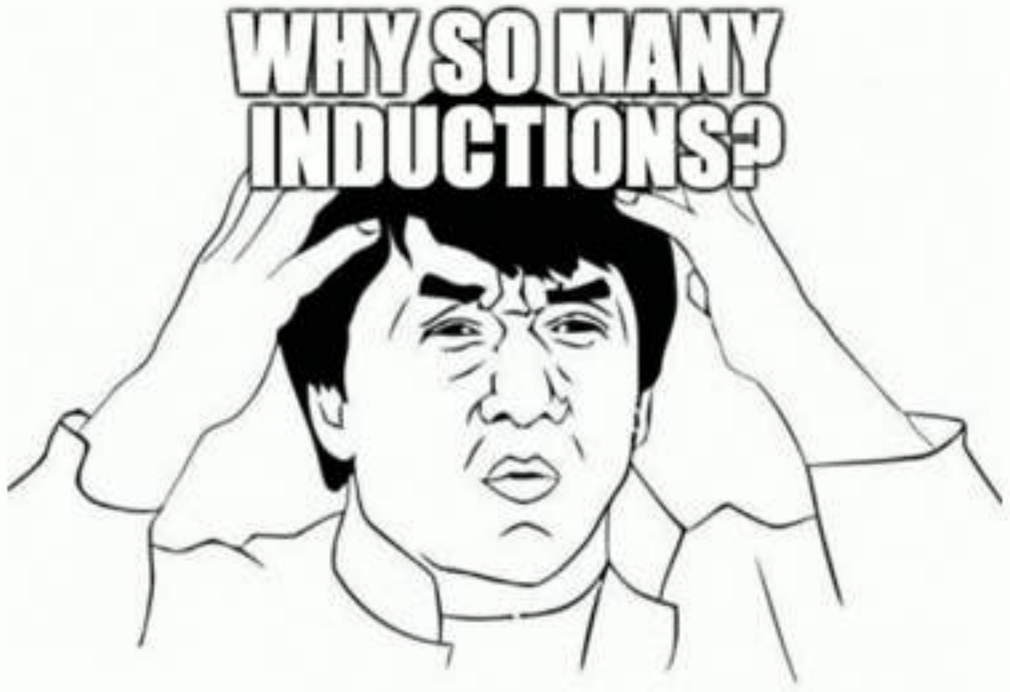


```
Q1 = SELECT *
     FROM (R UNION ALL S)
     WHERE b
```

```
Q2 = (SELECT * FROM R WHERE b)
     UNION ALL
     (SELECT * FROM S WHERE b)
```

Q1 = Q2 ?

```
Induction on R:
  Assume Q1 == Q2 when R has N tuples
  Then when R is of size N+1:
    ...

Induction on S:
  Assume Q1 == Q2 when S has N tuples
  Then when S is of size N+1:
    ...
```



400 Line of Coq for a simple rewrite, very limited rewrites has been proven

Univalent SQL Semantics

[Chu et al., PLDI 17]

Relation as Function

$\texttt{R:Relation} \rightarrow [\![\texttt{R}]\!]\texttt{:Tuple -> } \mathbb{N}$

Relation as Function

R:Relation → ⟦R⟧:Tuple -> $\mathbb{N}$

b: Predicate → ⟦b⟧:Tuple -> {0, 1}

SELECT * FROM R
WHERE b → Q(t) = ⟦R⟧ t × ⟦b⟧ t

R UNION ALL S → Q(t) = ⟦R⟧ t + ⟦S⟧ t

```
Q1 = SELECT *
     FROM (R UNION ALL S)
     WHERE b
```

```
Q2 = (SELECT * FROM R WHERE b)
      UNION ALL
     (SELECT * FROM S WHERE b)
```

```
Q1 = SELECT *
     FROM (R UNION ALL S)
     WHERE b
```

```
Q2 = (SELECT * FROM R WHERE b)
      UNION ALL
     (SELECT * FROM S WHERE b)
```

$$Q1(t) = (⟦R⟧(t) + ⟦S⟧(t)) \times ⟦b⟧(t)$$

$$Q2(t): ⟦R⟧(t) \times ⟦b⟧(t) + ⟦S⟧(t) \times ⟦b⟧(t)$$

```
Q1 = SELECT *
     FROM (R UNION ALL S)
     WHERE b
```

```
Q2 = (SELECT * FROM R WHERE b)
      UNION ALL
     (SELECT * FROM S WHERE b)
```

$$Q1(t) = (\llbracket R \rrbracket(t) + \llbracket S \rrbracket(t)) \times \llbracket b \rrbracket(t)$$

$$Q2(t): \llbracket R \rrbracket(t) \times \llbracket b \rrbracket(t) + \llbracket S \rrbracket(t) \times \llbracket b \rrbracket(t)$$

$$Q1 = Q2 \ ?$$

Algebraic Reasoning

QED

Problem 1: Projection

$$[\![\textbf{SELECT}\ \texttt{first}\ \textbf{FROM}\ \texttt{Name}]\!] = ?$$

Name

First	Last
Michael	Shulman
Michael	Jordan
Alex	Aiken
Alex	Krizhevsky

→

First
Michael
Michael
Alex
Alex

Problem 1: Projection

$[\![\texttt{SELECT first FROM Name}]\!] = ?$

Name

First	Last
Michael	Shulman
Michael	Jordan
Alex	Aiken
Alex	Krizhevsky

First
Michael
Michael
Alex
Alex

Add up multiplicities:
Q("Michael") = 2
Q("Alex")=2

Problem 1: Projection

$[\![$**SELECT** first **FROM** Name$]\!] = ?$

$Q(t') = \sum_{t:\ Tuple} [t.first = t'.first] \times [\![Name]\!](t)$

Problem 1: Projection

$[\![$**SELECT** first **FROM** Name$]\!] = ?$

0 or 1

$$Q(t') = \sum_{t:\ \mathrm{Tuple}} \boxed{[t.\mathrm{first} = t'.\mathrm{first}]} \times [\![\mathrm{Name}]\!](t)$$

Issues:

- Sum can potentially be infinite
- Need to reason about equality with algebraic expressions (now with sums …)

Problem 2: Duplicate Elimination

$[\![$ **SELECT DISTINCT** first **FROM** Name $]\!] = ?$

Proposal: A step functions over $\mathbb{N}$: e.g. $S(0) = 0$, $S(x) = 1$

Issues: Back to inductive proof again 🙃

Univalent Semantics

- HoTT Type instead of $\mathbb{N}$
- Dependent Pair Type ($\boldsymbol{\Sigma}$-Type) to represent summation
- Squash Type to represent duplicate elimination
- **Bonus: Unifies types and propositions**

P. Seshadri, J. Hellerstein, H. Pirahesh, T. Y. Leung,
R. Ramakrishnan, D. Srivastava, P. Stuckey, S. Sudarshan

Cost-Based Optimization for Magic: Algebra and Implementation.
SIGMOD 1996

Introduction of θ-semijoin:

$$R_1 \bowtie_\theta R_2 \equiv R_1 \bowtie_\theta (R_2 \ltimes_\theta R_1)$$

Pushing θ-semijoin through join:

$$(R_1 \bowtie_{\theta_1} R_2) \ltimes_{\theta_2} R_3 \equiv (R_1 \bowtie_{\theta_1} R_2') \ltimes_{\theta_2} R_3$$

$$R_2' = E_2 \ltimes_{\theta_1 \wedge \theta_2} (R_1 \bowtie R_3)$$

Pushing θ-semijoin through aggregation:

$${}_{\bar{g}}\mathcal{F}_{\bar{f}}(R_1) \ltimes_{c_1=c_2} R_2 \equiv_{\bar{g}} \mathcal{F}_{\bar{f}}(R_1 \ltimes_{c_1=c_2} R_2)$$

P. Seshadri, J. Hellerstein, H. Pirahesh, T. Y. Leung,
R. Ramakrishnan, D. Srivastava, P. Stuckey, S. Sudarshan

Cost-Based Optimization for Magic: Algebra and Implementation.
SIGMOD 1996

Introduction of θ-semijoin:

$$R_1 \bowtie_\theta R_2 \equiv R_1 \bowtie_\theta (R_2 \ltimes_\theta R_1)$$

Pushing θ-semijoin through join:

$$(R_1 \bowtie_{\theta_1} R_2) \ltimes_{\theta_2} R_3 \equiv (R_1 \bowtie_{\theta_1} R_2') \ltimes_{\theta_2} R_3$$

$$R_2' = E_2 \ltimes_{\theta_1 \wedge \theta_2} (R_1 \bowtie R_3)$$

Pushing θ-semijoin through aggregation:

$${}_{\bar{g}}\mathcal{F}_{\bar{f}}(R_1) \ltimes_{c_1=c_2} R_2 \equiv_{\bar{g}} \mathcal{F}_{\bar{f}}(R_1 \ltimes_{c_1=c_2} R_2)$$

Can we do better?

Axiomatic Foundations and Semi-Decision Procedures

[Chu et al., VLDB 18]

Idea 1: projection requires summation

$$[\![\texttt{SELECT first FROM Name}]\!] \equiv$$

$$[\![Q]\!](t') = \textstyle\sum_t [t.name = t'.name] \times [\![R]\!](t)$$

Solution: new* infinitary operator $\sum_t$

Idea 2: **DISTINCT** converts bags to sets

$$[\![\textbf{SELECT DISTINCT } \texttt{first} \textbf{ FROM } \texttt{Name}]\!] \equiv$$

$$[\![Q]\!](t') = \| \textstyle\sum_t [t.\text{first}=t'.\text{first}] \times [\![\text{Name}]\!](t) \|$$

Solution: add the squash* operator $\|\ldots\|$

Intuition: $\|0\| = 0$ and $\|x\| = 1$ otherwise

Idea 3: non-monotone operators e.g. **`EXCEPT`**, **`NOT EXIST`**

Solution: add a `not(…)` operator

Intuition: `not(0) = 1`, `not(x) = 0` otherwise

U-Semirings

Definition: An **Unbounded-semiring** is a structure **(U, 0, 1, +, ×, ||…||, not,** $(\sum_D)_{D \in \mathfrak{D}}$**)** satisfying a list of axioms

U-Semirings

Comm. Semiring

Definition: An **Unbounded-semiring** is a structure $(\boxed{U, 0, 1, +, \times,}\ \|\ldots\|,$ **not,** $(\sum_D)_{D \in \mathfrak{D}})$ satisfying a list of axioms

Example: Axiomization of Sum

$$\sum_t (f_1(t) + f_2(t)) = \sum_t f_1(t) + \sum_t f_2(t)$$

Distributivity

$$\sum_{t_1} \sum_{t_2} f(t_1, t_2) = \sum_{t_2} \sum_{t_1} f(t_1, t_2)$$

Commutativity

$$x \times \sum_t f(t) = \sum_t x \times f(t)$$

Associativity

$$\| \sum_t f(t) \| = \| \sum_t \| f(t) \| \|$$

Idempoten on Squash

Axiomization of Integrity Constraints

- The **key constraint** on R.k is the identity:

$$[t.k=t'.k]\times R(t)\times R(t') = [t=t']\times R(t)$$

Axiomization of Integrity Constraints

- The **key constraint** on R.k is the identity:

$$\boxed{[t.k=t'.k]}\times R(t)\times R(t') = [t=t']\times R(t)$$

Axiomization of Integrity Constraints

- The **key constraint** on R.k is the identity:

$$\boxed{[t.k=t'.k]}\times R(t)\times R(t') = \boxed{[t=t']}\times R(t)$$

- The **foreign key constraint** from S.fk to R.k is :

$$S(t') = S(t') \times \textstyle\sum_t R(t) \times [t.k=t'.fk]$$

Proving Equivalences with ICs

```
-- Q1
SELECT ip.np, itm.type, itm.itemno
FROM (SELECT DISTINCT itp.itemno AS itn,
                      itp.np AS np
      FROM price WHERE price.np > 1000) ip, itm
WHERE ip.itn = itm.itemno;

-- Q2
SELECT DISTINCT price.np, itm.type, itm.itemno
FROM price, itm
WHERE price.np > 1000 AND itp.itemno = itm.itemno;
```

`itemno` **is a key of** `itm`

Proving Equivalences with ICs

$$\begin{aligned} Q_1(t) \quad = \quad & \textstyle\sum_{t_1,t_2} [t_1.\mathsf{np} = t.\mathsf{np}] \times [t_2.\mathsf{type} = t.\mathsf{type}] \times \\ & [t_2.\mathtt{itemno} = t.\mathtt{itemno}] \times [t_1.\mathtt{itn} = t_2.\mathtt{itemno}] \times \\ & \| \textstyle\sum_{t'} [t'.\mathtt{itemno} = t_1.\mathtt{itn}] \times [t'.\mathsf{np} = t_1.np] \times \\ & [t'.\mathsf{np} > 1000] \times \mathsf{price}(t') \,\| \times \mathtt{itm}(t_2) \end{aligned}$$

$$\begin{aligned} Q_2(t) \quad = \quad & \| \textstyle\sum_{t_1,t_2} [t_2.\mathsf{type} = t.\mathsf{type}] \times [t_2.\mathtt{itemno} = t.\mathtt{itemno}] \times \\ & [t_1.\mathtt{itemno} = t.\mathtt{itemno}] \times [t_2.\mathtt{itemno} = t_1.\mathtt{itemno}] \times \\ & [t_1.\mathsf{np} = t.np] \times [t_1.\mathsf{np} > 1000] \times \mathsf{price}(t_1) \times \mathtt{itm}(t_2) \,\| \end{aligned}$$

Proving Equivalences with ICs

$$Q_1(t) = \sum_{t_1,t_2} [t_1.\mathsf{np} = t.\mathsf{np}] \times [t_2.\mathsf{type} = t.\mathsf{type}] \times [t_2.\mathtt{itemno} = t.\mathtt{itemno}] \times [t_1.\mathtt{itn} = t_2.\mathtt{itemno}] \times \| \sum_{t'} [t'.\mathtt{itemno} = t_1.\mathtt{itn}] \times [t'.\mathsf{np} = t_1.np] \times [t'.\mathsf{np} > 1000] \times \mathsf{price}(t') \| \times \mathtt{itm}(t_2)$$

Rewrite Using
U-Semiring Axioms

$$Q_1(t) = \| \sum_{t_2,t'} [t_2.\mathsf{type} = t.\mathsf{type}] \times [t_2.\mathtt{itemno} = t.\mathtt{itemno}] \times [t'.\mathtt{itemno} = t.\mathtt{itemno}] \times [t'.\mathsf{np} = t.np] \times [t'.\mathsf{np} > 1000] \times \mathsf{price}(t') \times \mathtt{itm}(t_2) \|$$

$$Q_2(t) = \| \sum_{t_1,t_2} [t_2.\mathsf{type} = t.\mathsf{type}] \times [t_2.\mathtt{itemno} = t.\mathtt{itemno}] \times [t_1.\mathtt{itemno} = t.\mathtt{itemno}] \times [t_2.\mathtt{itemno} = t_1.\mathtt{itemno}] \times [t_1.\mathsf{np} = t.np] \times [t_1.\mathsf{np} > 1000] \times \mathsf{price}(t_1) \times \mathtt{itm}(t_2) \|$$

Algorithm: Main Idea

Check $[\![Q_1]\!] = [\![Q_2]\!]$ by generalizing two known cases:

- UCQ under set semantics:
  - Check for homomorphisms Q1 ↔ Q2
- UCQ under bag semantics:
  - Check for isomorphisms Q1 → Q2
- "Chase" the axiomatization of constraints

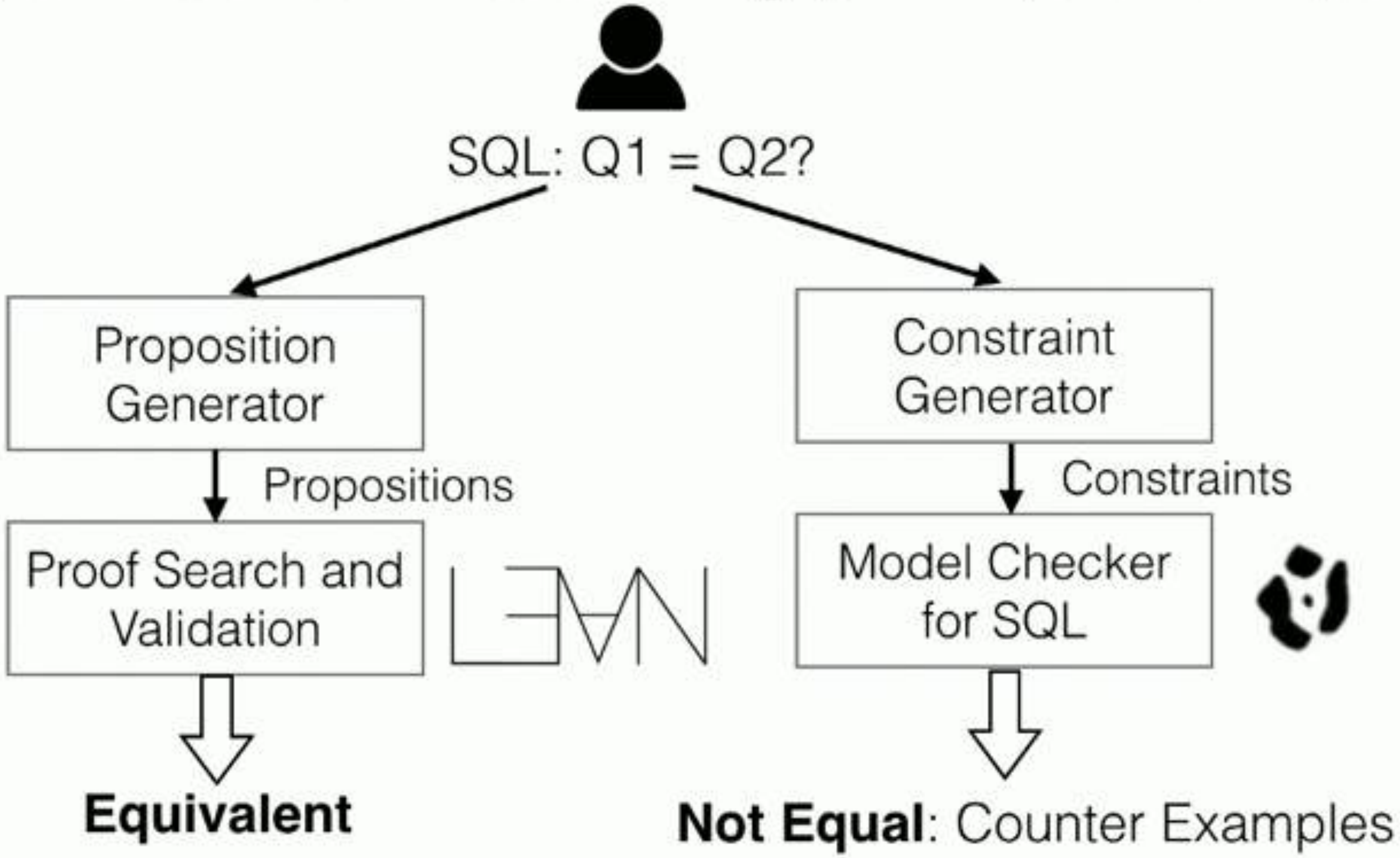
What about inequivalent queries?

Finding Counterexamples using Constraint Solver

[Chu et al., CIDR17]

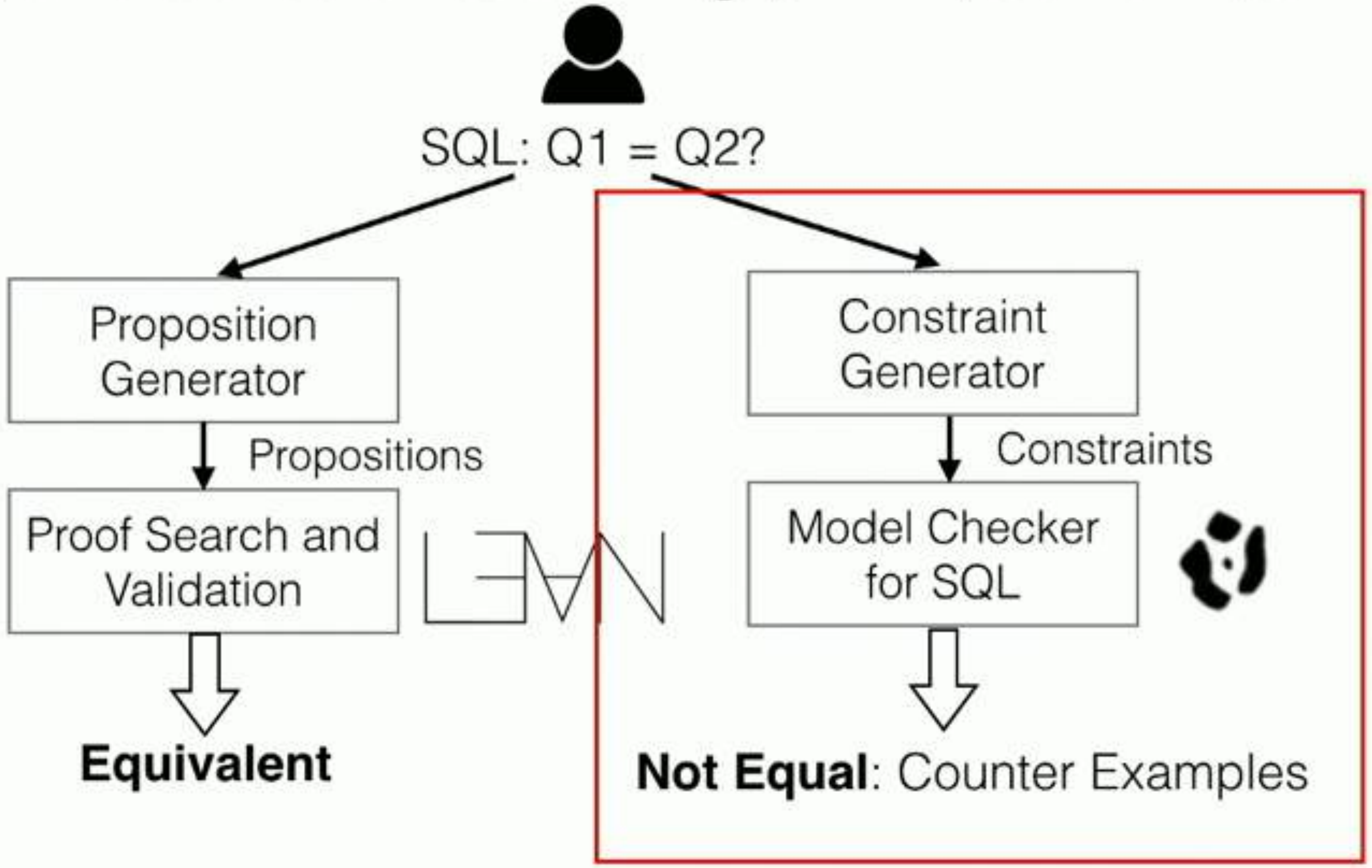
Cosette Architecture

- Cosette: An (almost) automated solver for SQL by *combining interactive theorem proving and constraints solving* (PLDI 17, SIGMOD17, CIDR17, VLDB18)

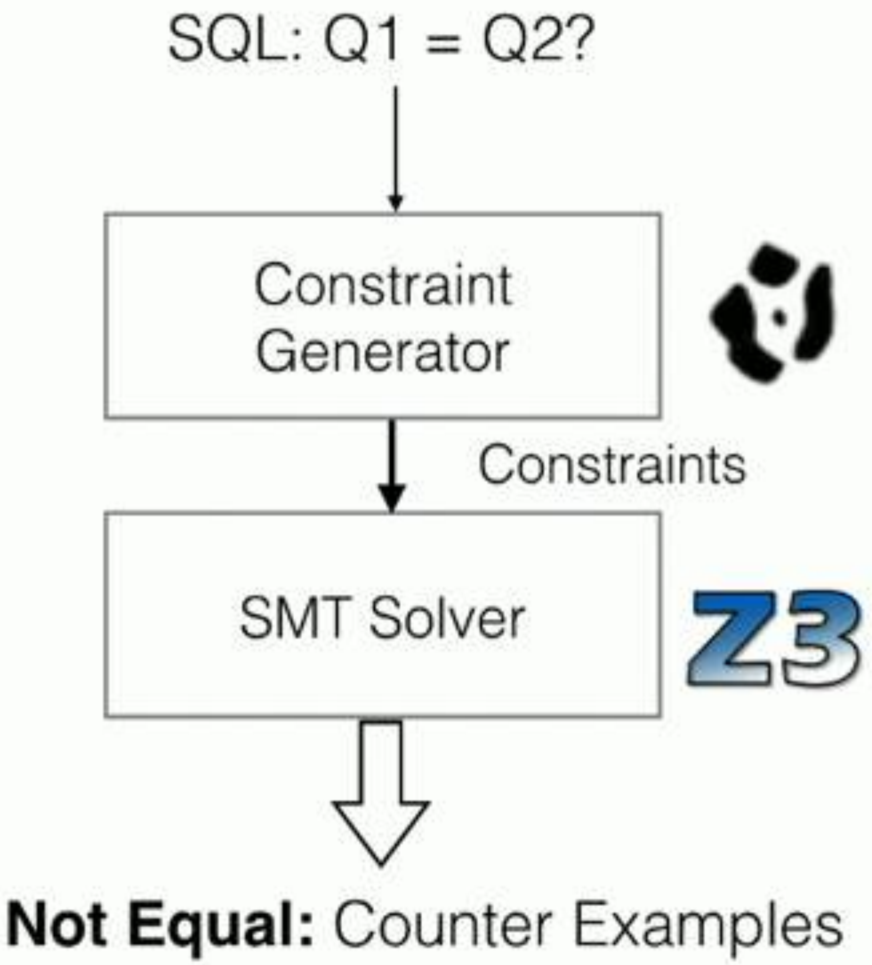


Cosette Architecture

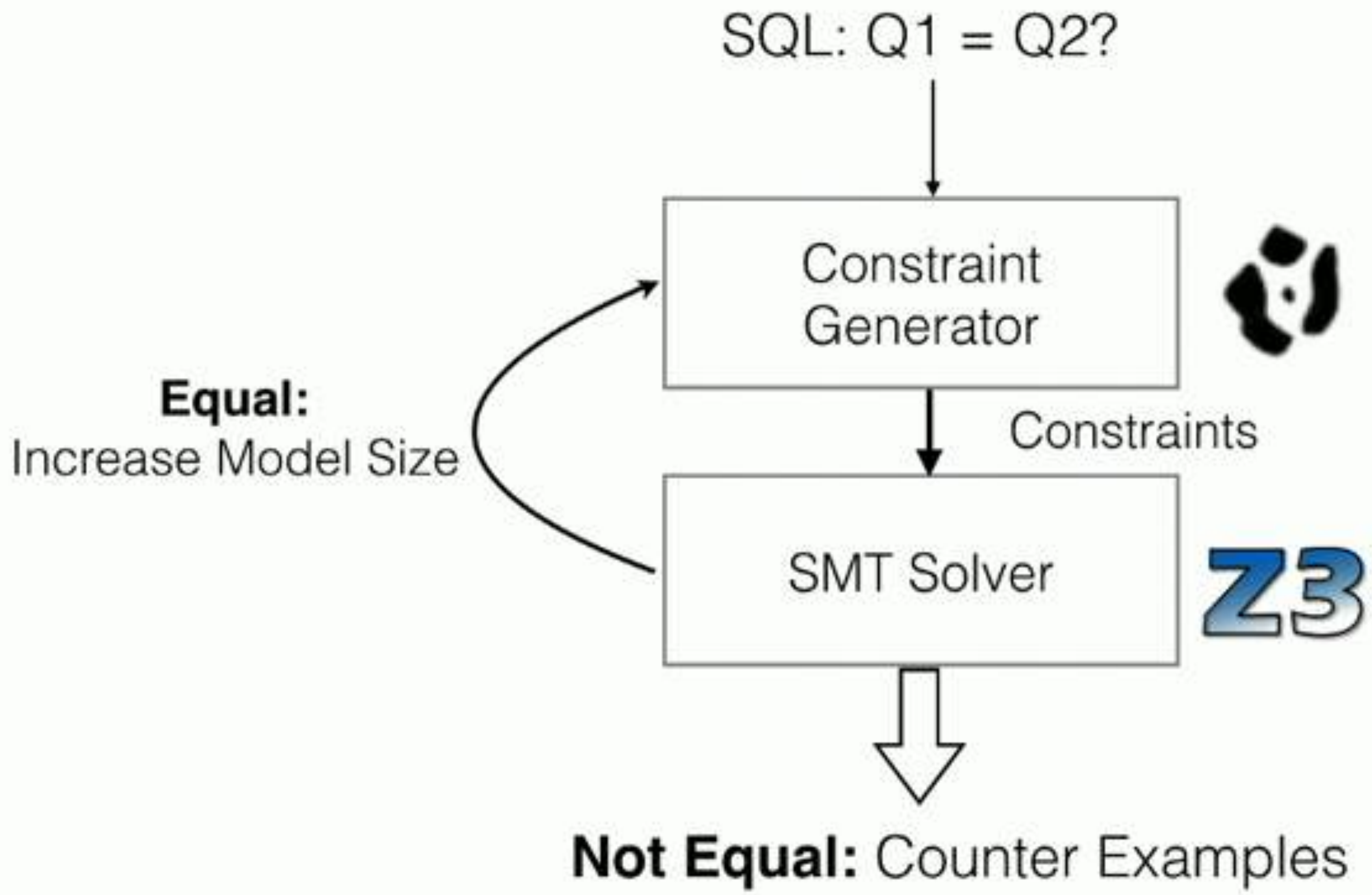
- Cosette: An (almost) automated solver for SQL by *combining interactive theorem proving and constraints solving* (PLDI 17, SIGMOD17, CIDR17, VLDB18)



Finding Counter Examples with SMT Solver



Finding Counter Examples with SMT Solver



Encoding SQL in Solvers

- A tuple as a list

```
Tuple := List <Integer>
```

Encoding SQL in Solvers

- A tuple as a list

```
Tuple := List <Integer>
```

- A relation as tuples tagged with multiplicity

```
Relation := List <Pair<Tuple, Integer>>
```

- A SQL query as constraints over symbolic values

Encoding SQL in Solvers

- A tuple as a list

sv
Tuple := List <~~Integer~~>

- A relation as tuples tagged with multiplicity

sv
Relation := List <Pair<Tuple, ~~Integer~~>>

- A SQL query as constraints over symbolic values

Performance Optimizations

- Maximize concrete evaluation/ Minimize Symbolic Execution
- Symmetry breaking

Show API Key

Cosette

Cosette is an automated solver for checking equivalences of SQL queries. Check out the Cosette Guide on how to use Cosette. Your star and feedback are appreciated!

Star 444 Issue

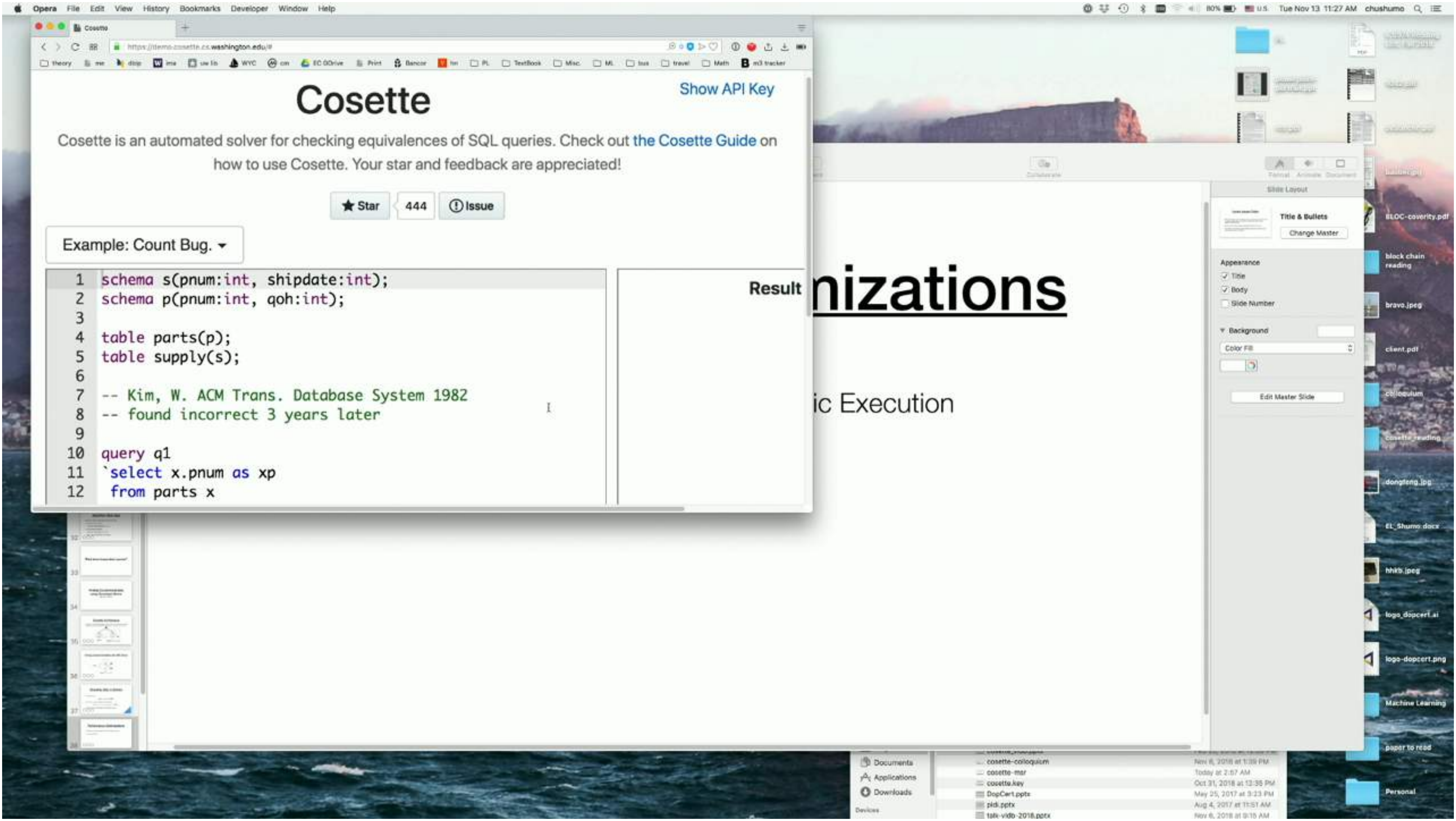
Example: Count Bug.

```
schema s(pnum:int, shipdate:int);
schema p(pnum:int, qoh:int);

table parts(p);
table supply(s);

-- Kim, W. ACM Trans. Database System 1982
-- found incorrect 3 years later

query q1
`select x.pnum as xp
 from parts x
 where x.qoh = (select count(y.shipdate) as cnt
                 from supply y
          where y.pnum = x.pnum AND y.shipdate < 10)`;

query q2
`select x.pnum as xp
 from parts x, (select y.pnum as suppnum, count(y.shipdate) as ct
               from supply y where y.shipdate < 10
               group by y.pnum) temp
 where x.qoh = temp.ct AND x.pnum = temp.suppnum`;

verify q1 q2;
```

Result

Submit

Back to Cosette Website


Cosette
Show API Key
Cosette is an automated solver for checking equivalences of SQL queries. Check out the Cosette Guide on how to use Cosette. Your star and feedback are appreciated!
Star
444
Issue
Example: Count Bug.
schema s(pnum:int, shipdate:int);
schema p(pnum:int, qoh:int);
table parts(p);
table supply(s);
-- Kim, W. ACM Trans. Database System 1982
-- found incorrect 3 years later
query q1
`select x.pnum as xp
from parts x
Result
nizations
ic Execution
Slide Layout
Title & Bullets
Change Master
Appearance
Title
Body
Slide Number
Background
Color Fill
Edit Master Slide
Documents
Applications
Downloads
cosette-colloquium
cosette-msr
cosette.key
DopCert.pptx
pldi.pptx
talk-vldb-2018.pptx

Cosette

Show API Key

Cosette is an automated solver for checking equivalences of SQL queries. Check out the Cosette Guide on how to use Cosette. Your star and feedback are appreciated!

Star 444 Issue

Example: Count Bug.

```
schema s(pnum:int, shipdate:int);
schema p(pnum:int, qoh:int);

table parts(p);
table supply(s);

-- Kim, W. ACM Trans. Database System 1982
-- found incorrect 3 years later

query q1
`select x.pnum as xp
 from parts x
 where x.qoh = (select count(y.shipdate) as cnt
                from supply y
          where y.pnum = x.pnum AND y.shipdate < 10)`;

query q2
`select x.pnum as xp
 from parts x, (select y.pnum as suppnum, count(y
     .shipdate) as ct
               from supply y where y.shipdate < 10
               group by y.pnum) temp
 where x.qoh = temp.ct AND x.pnum = temp.suppnum`;

verify q1 q2;
```

Result

Submit

Evaluating **Cosette**

- **Bug**: 3 real-world optimizer bugs
- **XData**: query and mutant pairs collected from XData, a test generation framework
- **Exams**: a set of questions from the undergraduate data management class
- **Rules**: 68 query rewrite rules from database literatures and real-world optimizers

Evaluating **Cosette**

- **Bug**: 3 real-world optimizer bugs
- **XData**: query and mutant pairs collected from XData, a test generation framework
- **Exams**: a set of questions from the undergraduate data management class
- **Rules**: 68 query rewrite rules from database literatures and real-world optimizers

Unequal SQLs

Equivalent SQLs

Evaluating **Cosette**

Dataset	Total #	Average time taken
Bugs	3	8.3s
XData	9	<1s
Exams	5	1.3s

Dataset	Equiv?	Total Number	Automatically Decided		Manually proved
			No.	Avg. Time	
Rules	Yes	68	62	< 20 s	6
Exams	Yes	4	3	< 1 s	1

Evaluating **Cosette**

Dataset	Total #	Average time taken
Bugs	3	8.3s
XData	9	<1s
Exams	5	1.3s

Rule No. 29:
400 LOC in
[Malecha et al., POPL 10]
to 15 LOC in Cosette

Dataset	Equiv?	Total Number	Automatically Decided		Manually proved
			No.	Avg. Time	
Rules	Yes	68	62	< 20 s	6
Exams	Yes	4	3	< 1 s	1

Cosette in Action

- Full stack: solver core, web service, online demo, automated grader
- Deployed for UW 344/544 automated grading since 2017
- SIGMOD 2017 Best Demo
- Top 1 Trending Racket Project in GitHub (July, 2017)

```
/* define schema s1,
   here s1 can contain any number of attributes,
   but it has to at least contain integer attributes
   x and y */
schema s1(x:int, a:int, ??);

schema s2(a:int, ??);         -- define schema s2

table a(s1);                  -- define table a using schema s1
table b(s2);                  -- define table b using schema s1

query q1                      -- define query q1 on tables a
    and b
`select distinct x.x as ax from a x, b y
 where x.a = y.a`;

query q2                      -- define query q2 likewise
`select distinct x.x as ax from a x, a y, b z
 where x.x = y.x and x.a = z.a`;

verify q1 q2;                 -- does q1 equal to q2?
```

Conclusions and Takeaways

Cosette: The first practical SQL solver

- A new axiomatic semantics for SQL
- Semi-decision procedure for UCQ under bag/set with ICs
- Integrated interactive theorem proving and constraints solving techniques

- Automated reasoning brought by ***Formal Methods + Domain Specific Semantics*** *leads to more reliable, more optimized future data systems*
- Website: cosette.cs.washington.edu