Bringing SQL to the Masses with Program Synthesis

Chenglong Wang, Alvin Cheung, Ras Bodik
University of Washington

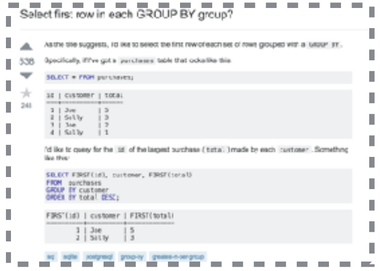
End-User

Select rows with maximum value for each group.

Find rows containing duplicate values.

Calculate the running total for a table.

stackoverflow



SQL

```
Select x.id, x.customer, x.total
From PURCHASES x
Join (Select p.customer,
             Max(total)
       From PURCHASES p
       Group By p.customer) y
On y.customer = x.customer
   And y.max_total = x.total
```

```
Select *
From  Users a
Where Exists
   (Select *
    From Users b
    Where (a.name = b.name
           Or  a.email = b.email)
          And a.ID <> b.id)
```

```
Select a.ord, a.total,
       Sum(b.total)
From t As a Join t As b
Where b.ord <= a.ord
Group By a.ord,a.total
Order By a.ord
```

Observations

A lot of **common tasks** require using **complex SQL constructs**.

greatest-n-per-group	Aggregation
running-total	Subquery
duplicates	Exists/In-clauses

Many tasks can be concisely expressed with **input-output exar**

Idea: summarize our observation on StackOverflow

Transition: these problems can be concisely expressed with examples, can we build some system that allows users to ask question using examples only?

Programming by Example System

Join two tables and return the rows containing the maximum **val** below 50 for each group.

T1

id	date	uid
1	12/25	1
2	11/21	3
4	12/24	2

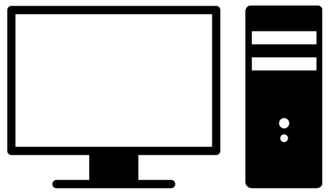
T2

oid	val
1	30
1	10
1	10
2	50
2	10

Out

oid	date	uid	oid	MaxVal
1	12/25	1	1	30
4	12/24	2	2	10

Synthesize



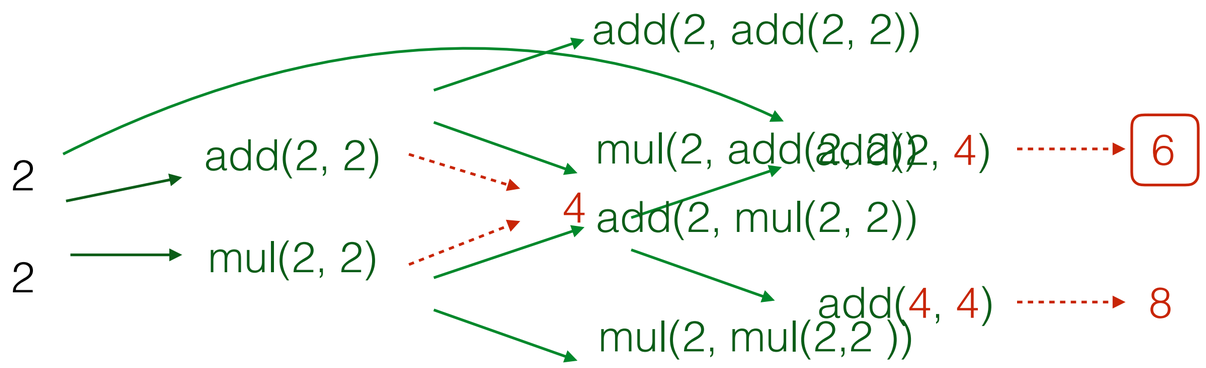
```
Select *
From   T1
Join   (Select id, Max(val)
        From  T2
        Where val < 50
        Group By oid) T3
On     T3.oid = T1.uid
```

Idea: introduce what we want to do: build a PBE system.

Transition: let's first see what is the traditional algorithm to build such system.

Synthesis Algorithm: Value-directed Search

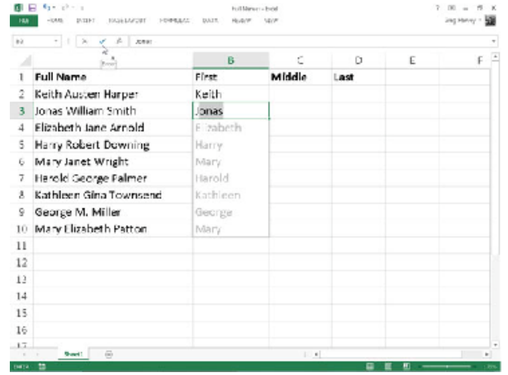
Input: 2, 2, **Output**: 6, **Operators**: add, mul



add(2, add(2, 2)) = 6

add(2, mul(2, 2)) = 6

FlashFill



SuperOptimizer

```
cmp   r1, #0
add   r2, r1, #7
movge r2, r1
```

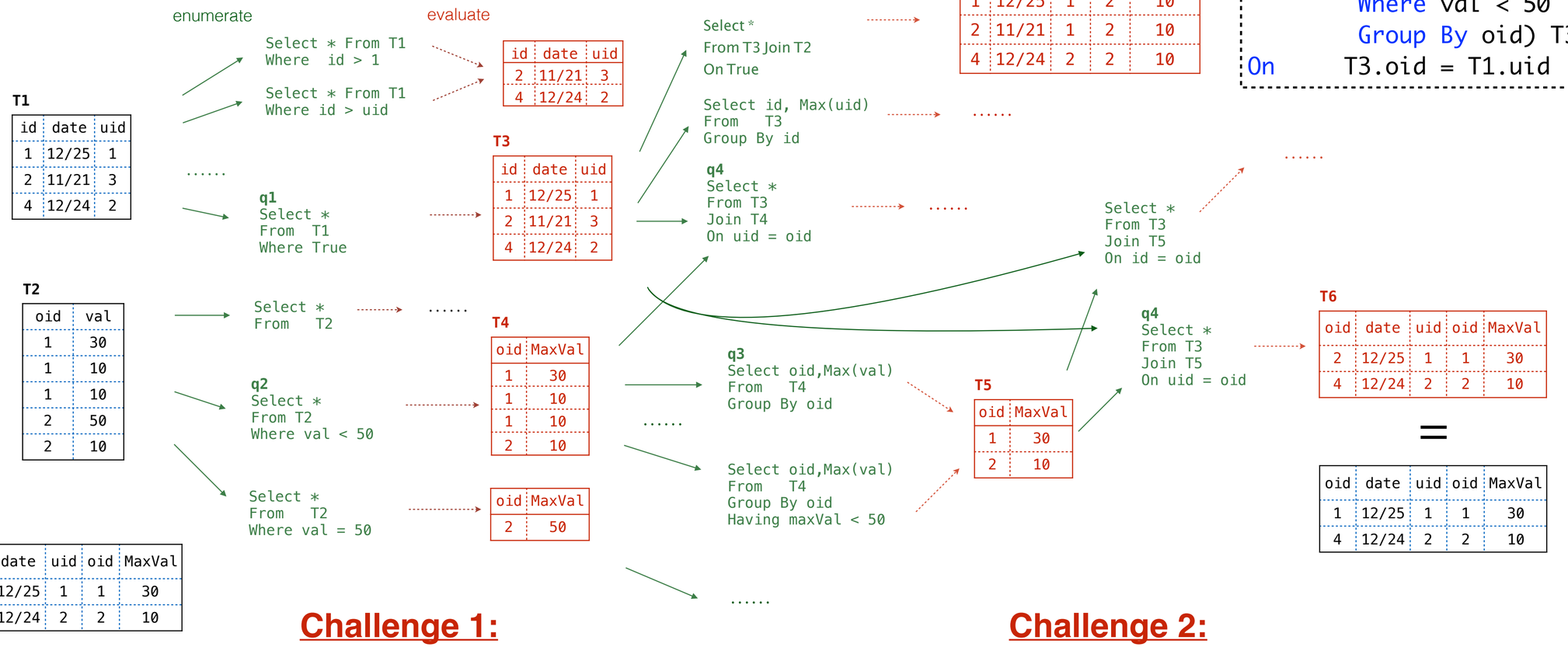
```
asr r3, r1, #2
add r2, r1, r3, lsr #29
```

Enumerative Searc QL

Input: T1, T2, **Output:** T_{out}, **Operators:** Select, Join, Aggr

id	date	uid	oid	MaxVal
1	12/25	1	1	30
2	11/21	1	1	30
4	12/24	2	1	30
1	12/25	1	1	10
2	11/21	1	1	10
4	12/24	2	1	10
1	12/25	1	2	50
2	11/21	1	2	50
4	12/24	2	2	50
1	12/25	1	2	10
2	11/21	1	2	10
4	12/24	2	2	10

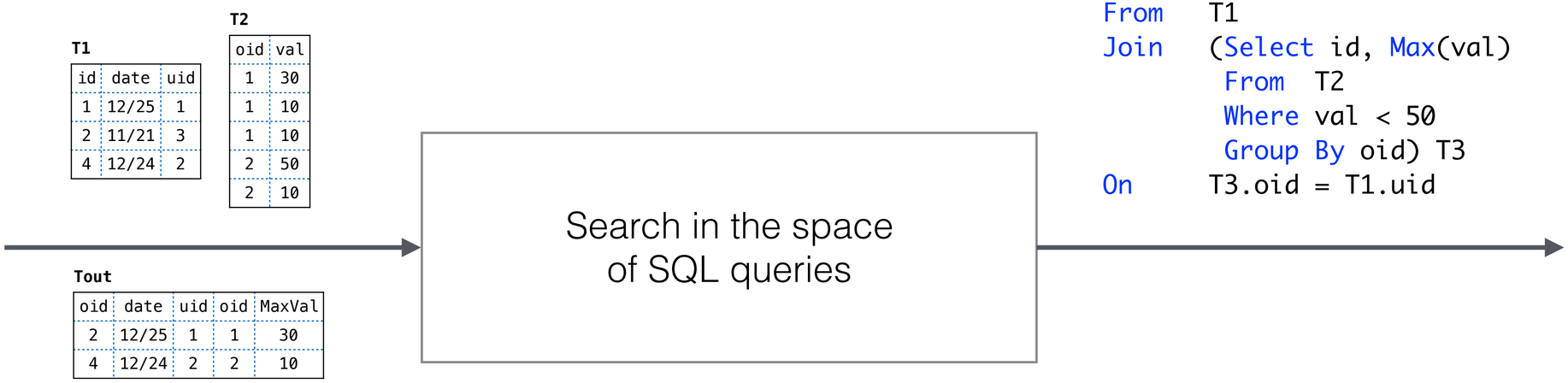
```
Select *
From   T1
Join   (Select id, Max(val) As MaxVal
        From  T2
        Where val < 50
        Group By oid) T3
On     T3.oid = T1.uid
```



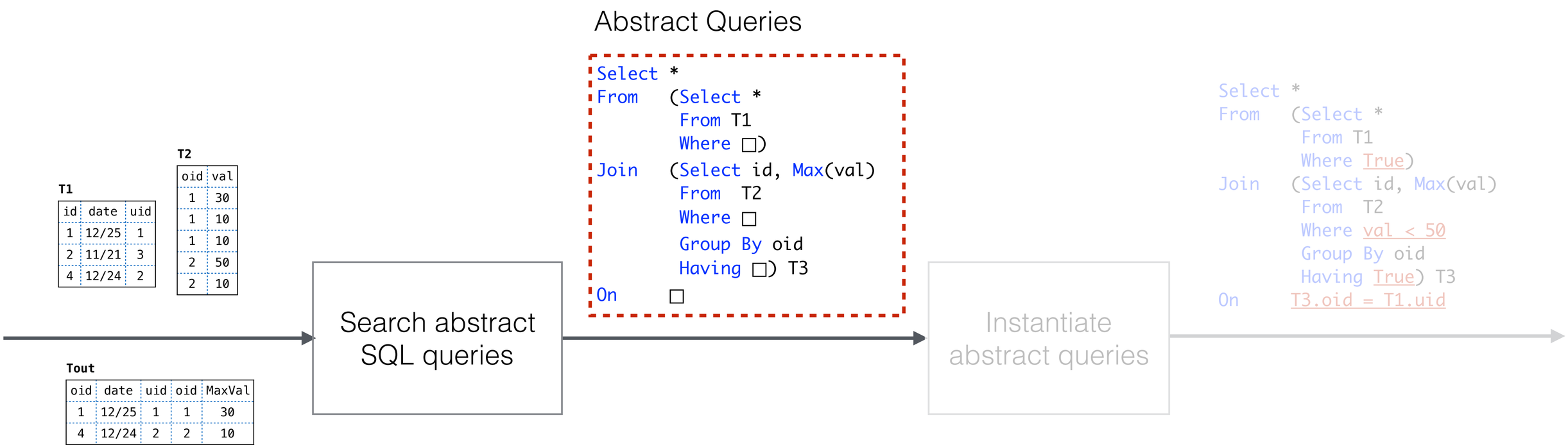
Challenge 1: Large number of queries

Challenge 2: Large tables

Insight: Decomposition



Insight: Decomposition



Pro: Smaller space of programs.

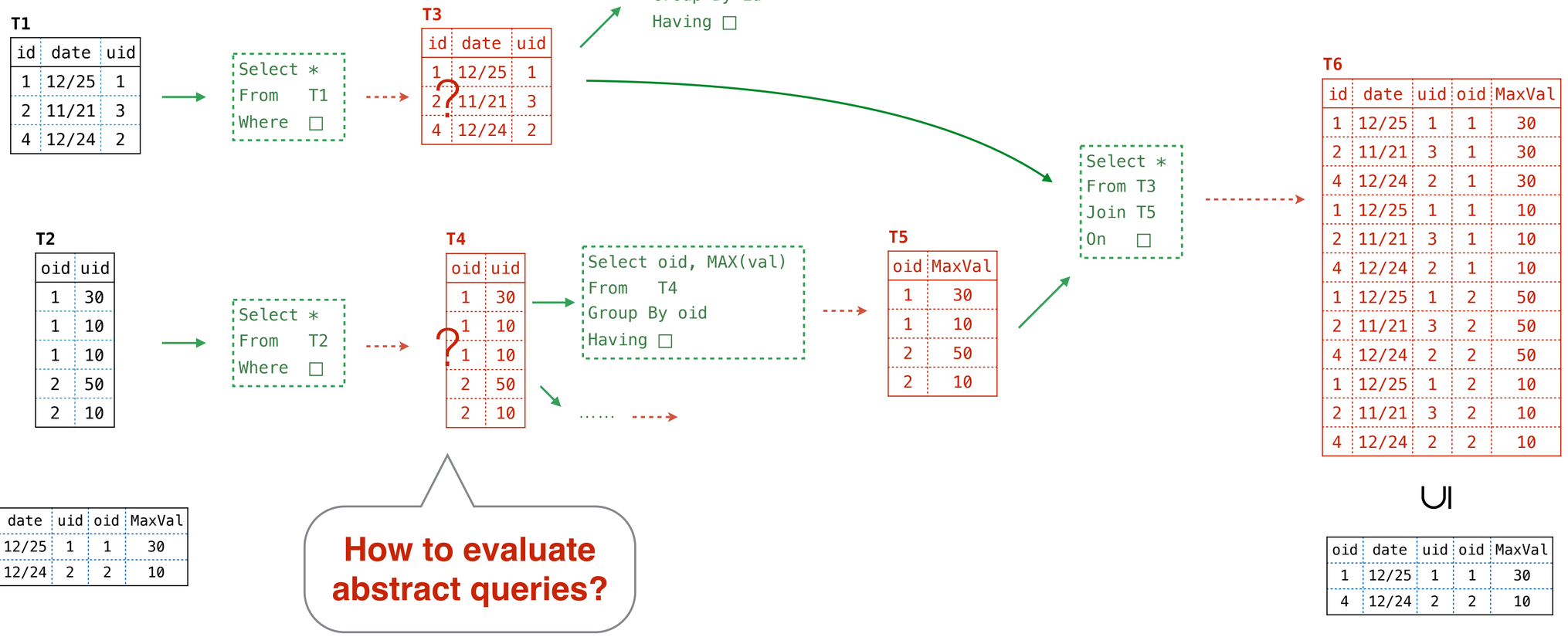
Challenge: which ones to search for?

Search with Abstract Queries

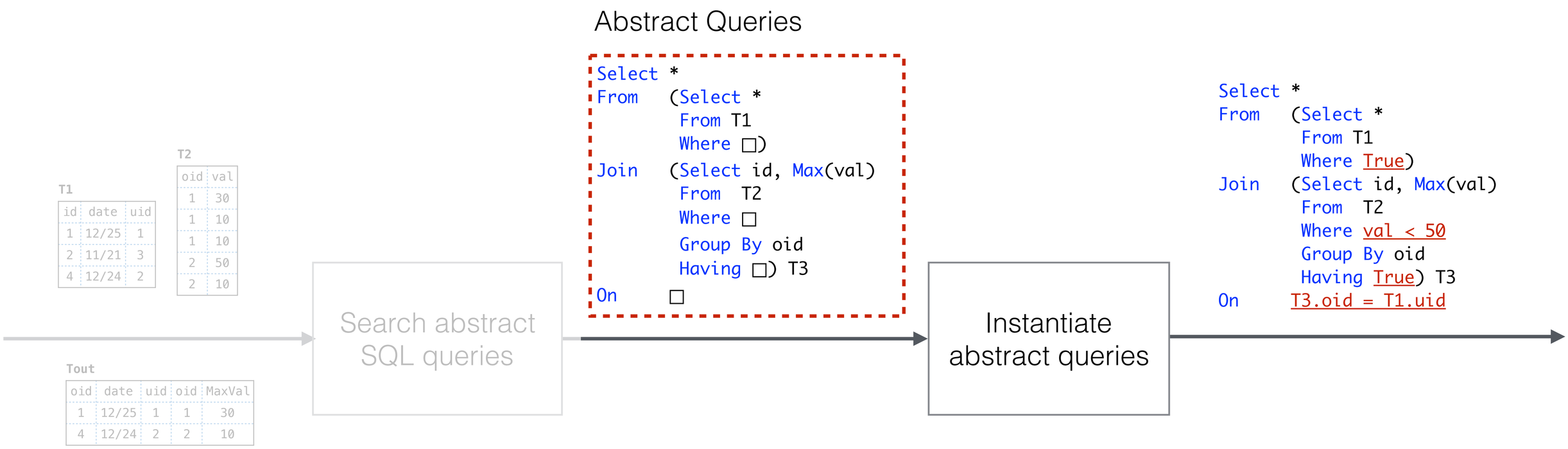
Goal:

```
Select *
From   (Select *
        From T1
        Where □)
Join   (Select id, Max(val)
        From  T2
        Where □
        Group By oid
        Having □) T3
On     □
```

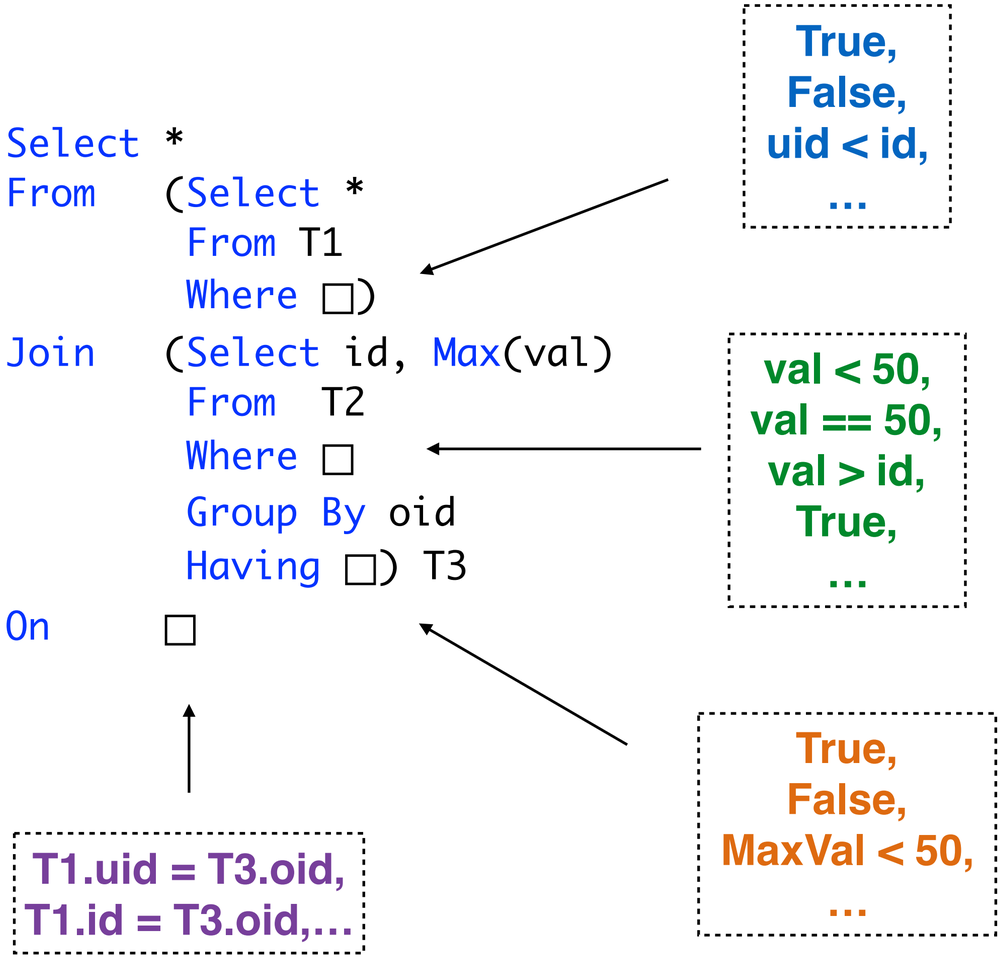
Input: T1, T2, **Output:** T_{out}, **Operators:** abstract query operators



Instantiate Abstract Queries



Instantiate Abstract Queries



True + **val < 50** + **False** + **T1.uid = T3.oid**

False + **val < 50** + **False** + **T1.uid = T3.oid**

True + **val == 50** + **False** + **T1.uid = T3.oid**

True + **val == 50** + **MaxVal < 50** + **T1.uid = T3.oid**

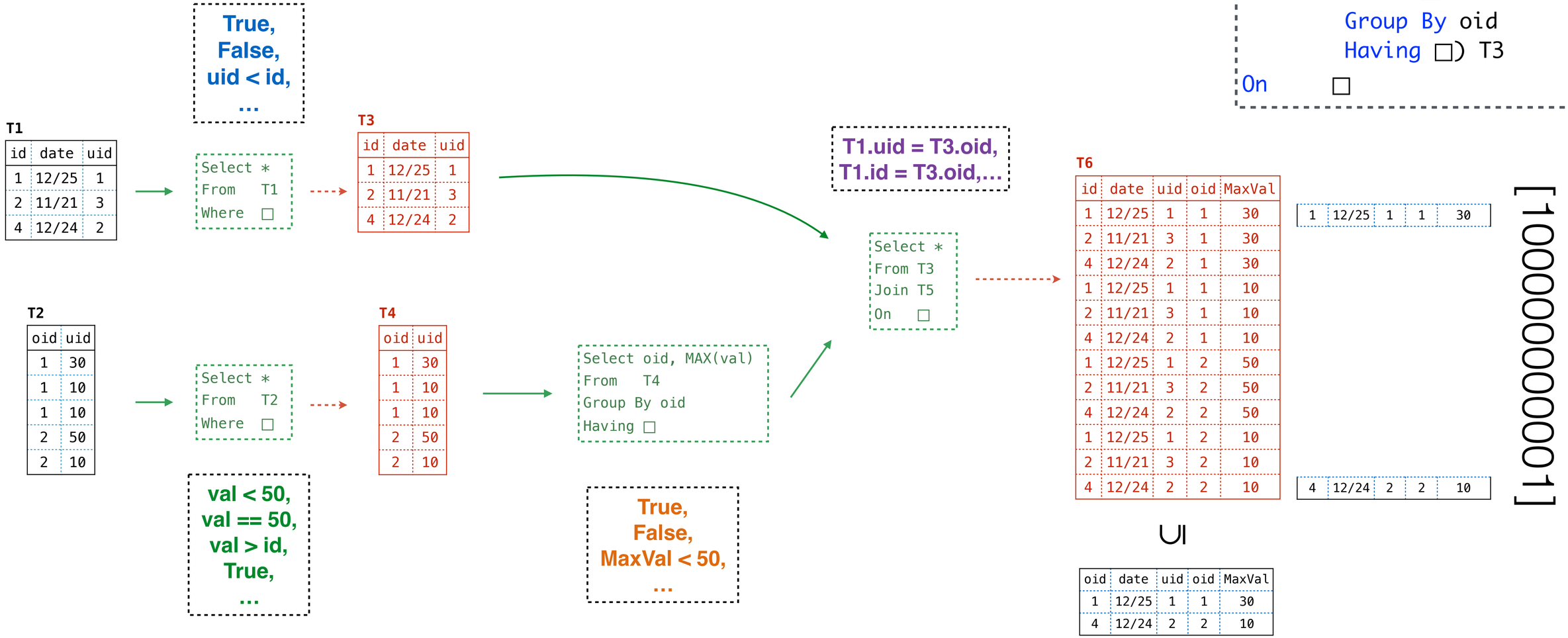
……

A intuitive solution that does no scale.

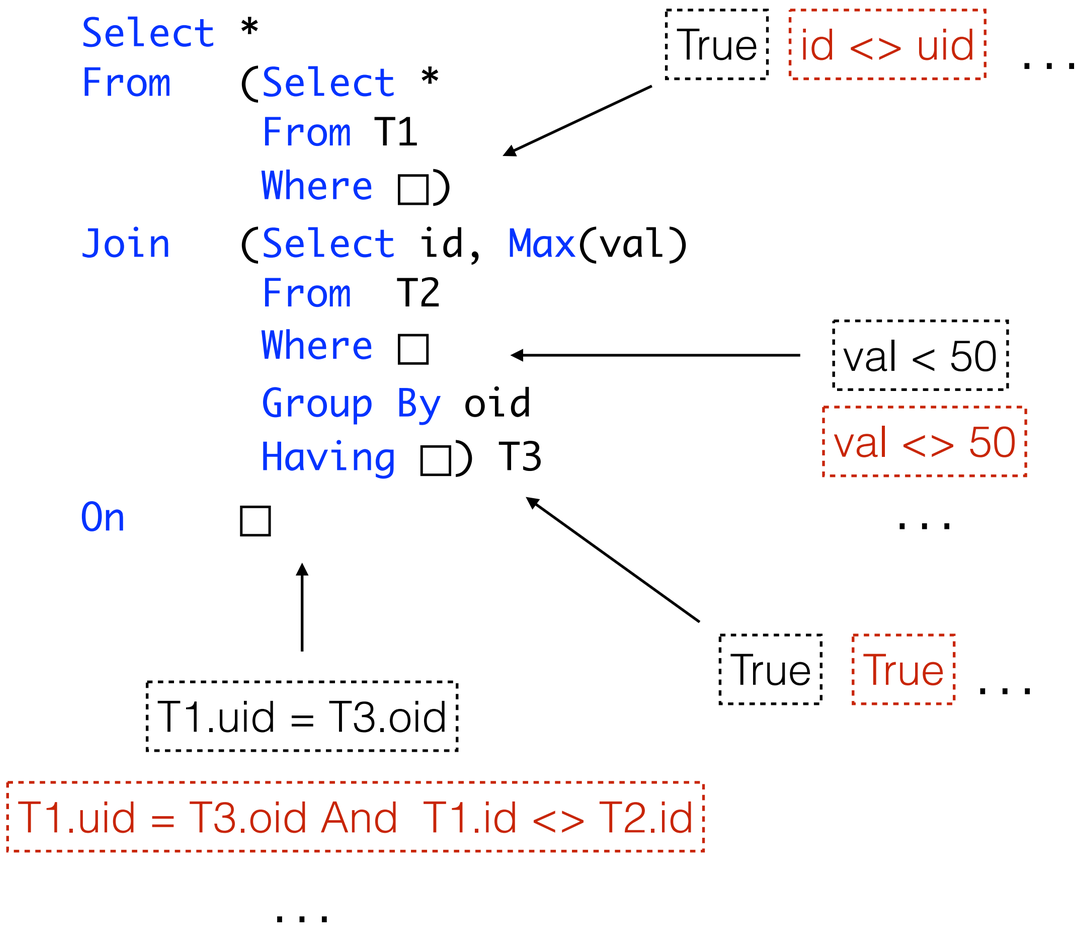
Transition: can we use properties of the abstract query to optimize this?

Instantiate Abstract Queries

```
Select *
From   (Select *
        From T1
        Where □)
Join   (Select id, Max(val)
        From  T2
        Where □
        Group By oid
        Having □) T3
On     □
```



Generating Solutions




```
Select *
From   T1
Join   (Select id, Max(val)
From  T2
        Where val < 50
        Group By oid) T3
On     T3.oid = T1.uid
```

.....

```
Select *
From   (Select * From T1 Where id <> uid)
Join   (Select id, Max(val)
        From  T2
        Where val <> 50
        Group By oid) T3
On     T1.uid = T3.oid And  T1.id <> T2.id
```

Ranking & Interaction

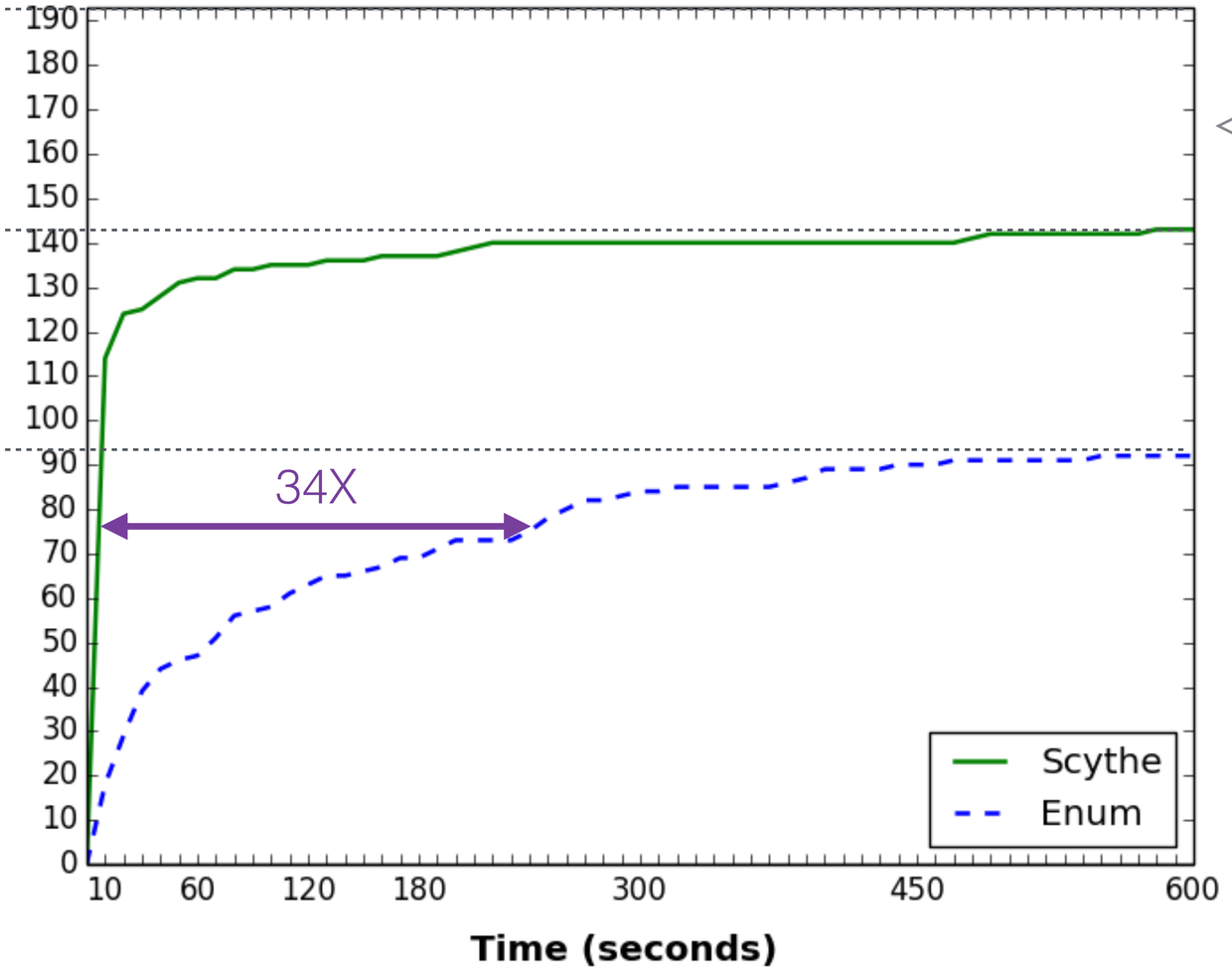
- Heuristically rank candidate queries.
  - Criteria: complexity, naturalness etc.
- When the result is not desirable:
  - Provide new input-output examples.

Evaluation

Benchmark: 193

Scythe: 143

Enum: 92



34: more features
15: run out of time
1: fail to disambiguate

Demo

Conclusion

- Goal: Helping end users to program SQL with input-output examples.
- Solution: An efficient two-phase synthesis algorithm.
- Evaluation: Able to solve 143/193 problems on StackOverflow.