Synthesizing Highly Expressive SQL Queries From Input-Output Examples

http://scythe.cs.washington.edu

Chenglong Wang, Alvin Cheung, Ras Bodík
University of Washington
Tasks

Select the id for user “Tom”

Select rows with maximum value for each user.

Calculate moving average over id.

SQL Query

Select id
From table
Where name = “Tom”

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 a.ord, a.val, Avg(b.val)
From t As a Join t As b
Where b.ord <= a.ord
Group By a.ord,a.val
Order By a.ord

Problem: Advanced SQL operators make SQL powerful but hard to master.
How to select the first N rows of each group?

I have two SQLite tables like this:

<table>
<thead>
<tr>
<th>AuthorId</th>
<th>AuthorName</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alice</td>
</tr>
<tr>
<td>2</td>
<td>Bob</td>
</tr>
<tr>
<td>3</td>
<td>Carol</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BookId</th>
<th>AuthorId</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>aaa1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>aaa2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>aaa3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>ddd1</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>ddd2</td>
</tr>
<tr>
<td>19</td>
<td>3</td>
<td>fff1</td>
</tr>
<tr>
<td>20</td>
<td>3</td>
<td>fff2</td>
</tr>
<tr>
<td>21</td>
<td>3</td>
<td>fff3</td>
</tr>
<tr>
<td>22</td>
<td>3</td>
<td>fff4</td>
</tr>
</tbody>
</table>

I want to make a SELECT query that will return the first N (e.g., two) rows for each AuthorId, ordering by Title ("Select the first two books of each author")..

Sample output:

<table>
<thead>
<tr>
<th>BookId</th>
<th>AuthorId</th>
<th>AuthorName</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Alice</td>
<td>aaa1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>Alice</td>
<td>aaa2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>Alice</td>
<td>aaa3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>Bob</td>
<td>ddd1</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>Bob</td>
<td>ddd2</td>
</tr>
<tr>
<td>19</td>
<td>3</td>
<td>Carol</td>
<td>fff1</td>
</tr>
<tr>
<td>20</td>
<td>3</td>
<td>Carol</td>
<td>fff2</td>
</tr>
<tr>
<td>21</td>
<td>3</td>
<td>Carol</td>
<td>fff3</td>
</tr>
<tr>
<td>22</td>
<td>3</td>
<td>Carol</td>
<td>fff4</td>
</tr>
</tbody>
</table>

How can I build this query?

(Yes I found a similar topic and I know how to return only one row (first or top). The problem is with the two).

Key: The synthesizer takes inputs that users can provide online.

Synthesize queries from …?

### Input Example

<table>
<thead>
<tr>
<th>BookId</th>
<th>AuthorId</th>
<th>AuthorName</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Alice</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>Bob</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>Carol</td>
</tr>
</tbody>
</table>

### Output Example

<table>
<thead>
<tr>
<th>BookId</th>
<th>AuthorId</th>
<th>AuthorName</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Alice</td>
<td>aaa1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>Alice</td>
<td>aaa2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>Bob</td>
<td>ddd1</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>Bob</td>
<td>ddd2</td>
</tr>
<tr>
<td>19</td>
<td>3</td>
<td>Carol</td>
<td>fff1</td>
</tr>
<tr>
<td>20</td>
<td>3</td>
<td>Carol</td>
<td>fff2</td>
</tr>
</tbody>
</table>

### Constants

\{2\}

### Aggregation Functions (Optional)

\{ Count, Max, Min, Sum, Avg \}
Talk Outline

• Motivation & Problem Definition

• Synthesis Algorithm

```sql
Select b.BookId, a.AuthorId, a.AuthorName, b.Title
From Author a
Join Book b
On a.AuthorId = b.AuthorId
Where (Select count(*)
    From book b2
    Where b2.bookId <= b.BookId
    And b2.AuthorId = b.AuthorId
) <= 2;
```

• Evaluation on Stack Overflow Posts
Running Example

**Task:** Collect the max vals below 50 for all oid groups in T2 and join them with T1.

<table>
<thead>
<tr>
<th>T1</th>
<th></th>
<th>T2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>date</td>
<td>uid</td>
<td>oid</td>
</tr>
<tr>
<td>1</td>
<td>12/25</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>11/21</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>12/24</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Out

<table>
<thead>
<tr>
<th>oid</th>
<th>date</th>
<th>uid</th>
<th>oid</th>
<th>MaxVal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12/25</td>
<td>1</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>12/24</td>
<td>2</td>
<td>2</td>
<td>10</td>
</tr>
</tbody>
</table>

Constants = { 50 }

AggrFunc = { Max, Min }
Basic Algorithm: Enumerative Search

Input: 2  Output: 6  Operators: add, mul

Key: Compressing the search space by memoizing values.
Input: T1, T2  Output: T_out  Operators: Select, Join, Aggr

Select *
From T1
Where id > 1

Select *
From T2
Where val < 50

Select *
From T1
Where id >= uid
Input: T1, T2  Output: T_{out}  Operators: Select, Join, Aggr

Select * From T1 Where id > 1
Select * From T1 Where True
Select * From T2 Where val < 50
Select * From T1 Where id \geq uid

Out
<table>
<thead>
<tr>
<th>oid</th>
<th>date</th>
<th>uid</th>
<th>oid</th>
<th>MaxVal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12/25</td>
<td>1</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>11/21</td>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>12/24</td>
<td>2</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>11/21</td>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>12/25</td>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>11/21</td>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>12/24</td>
<td>2</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>11/21</td>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
</tbody>
</table>

Problem: Value-based compression is inefficient & ineffective.
Insight: Decompose Search Process

Search SQL queries

T1
<table>
<thead>
<tr>
<th>id</th>
<th>date</th>
<th>uid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12/25</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>11/21</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>12/24</td>
<td>2</td>
</tr>
</tbody>
</table>

T2
<table>
<thead>
<tr>
<th>oid</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
</tbody>
</table>

Tout
<table>
<thead>
<tr>
<th>oid</th>
<th>date</th>
<th>uid</th>
<th>oid</th>
<th>MaxVal</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>12/25</td>
<td>1</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>12/24</td>
<td>2</td>
<td>2</td>
<td>10</td>
</tr>
</tbody>
</table>
Insight: Decompose Search Process With Abstract Queries

1. Prune query families
   If a skeleton cannot be instantiated to return output, prune all queries with the skeleton

2. Speed up predicate synthesis
Insight: Decompose Search Process With Abstract Queries

1. Prune query families
   If a skeleton cannot be instantiated to return output, prune all queries with the skeleton

How?
Evaluating Abstract Queries with Over-Approximation

**T1**

<table>
<thead>
<tr>
<th>id</th>
<th>date</th>
<th>uid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12/25</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>11/21</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>12/24</td>
<td>2</td>
</tr>
</tbody>
</table>

Select id, date
From T1
Where □

Summary

<table>
<thead>
<tr>
<th>id</th>
<th>date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12/25</td>
</tr>
<tr>
<td>2</td>
<td>11/21</td>
</tr>
<tr>
<td>4</td>
<td>12/24</td>
</tr>
</tbody>
</table>

**T2**

<table>
<thead>
<tr>
<th>oid</th>
<th>MaxVal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
</tbody>
</table>

T1 Join T2
On □

Summary2

<table>
<thead>
<tr>
<th>id</th>
<th>date</th>
<th>uid</th>
<th>oid</th>
<th>MaxVal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12/25</td>
<td>1</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>11/21</td>
<td>1</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>12/24</td>
<td>2</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>1</td>
<td>12/25</td>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>11/21</td>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>12/24</td>
<td>2</td>
<td>1</td>
<td>10</td>
</tr>
</tbody>
</table>

Key: Evaluating abstract queries into over-approximations of concrete query results.

Inductively defined over abstract SQL operators.
Pruning with Abstract Queries

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

On average, number of tables generated is 7× less v.s. concrete case.
Search with **Abstract Queries**

### Search abstract SQL queries

<table>
<thead>
<tr>
<th>T1</th>
<th>id</th>
<th>date</th>
<th>uid</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>12/25</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>11/21</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>12/24</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tout</th>
<th>oid</th>
<th>date</th>
<th>uid</th>
<th>oid</th>
<th>MaxVal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>12/25</td>
<td>1</td>
<td></td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>11/21</td>
<td>3</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>12/24</td>
<td>2</td>
<td>2</td>
<td>10</td>
</tr>
</tbody>
</table>

### Instantiate Abstract Queries

1. **Prune query families**
   In average over 90% of queries skeletons are pruned.

2. **Speed up predicate synthesis**
Challenge: Large number of predicate combinations to search.
Enumerative Predicate Synthesis
Enumerative Predicate Synthesis

Select *
From (Select *
    From T1
    Where □)
    As T3
Join (Select id, Max(val)
    From T2
    Where □
    Group By oid
    Having □)
    T5
On □

Select *
From T2
Where □

Select * 
From T1
Where □
Select * 
From T1
Where id=uid
Select * 
From T1
Where id<uid
Select * 
From T1
Where True

Select oid, 
       MAX(val)
From   T4
Group By oid
Having □

Select oid, 
       MAX(val)
From   T4
Group By oid
Having □

Select * 
From T1
Where True

Select * 
From T1
Where □

Select * 
From T2
Where □

Select * 
From T3
Join T5
On □

Select * 
From T3
Join T5
On id < oid
Select * 
From T3
Join T5
On uid = oid

Select * 
From T3
Join T5
On True

Out
oid date uid oid MaxVal
1 12/25 1 1 30
1 12/25 1 1 30
2 12/24 2 2 10
Enumerative Predicate Synthesis

Select *
From (Select *
   From T1
   Where □)
As T3
Join (Select id, MAX(val)
   From T2
   Where □
   Group By oid
   Having □)
T5
On □
"Evaluating abstract queries into over-approximations of concrete query results."
Encoding Tables using Bit-vectors

Select *
From (Select *
    From T1
    Where □)
As T3
Join (Select id, Max(val)
    From T2
    Where □
    Group By oid
    Having □)
T5
On □

T1
id date uid
1 12/25 1
2 11/21 3
4 12/24 2

Select oid, MAX(val)
From T4
Group By oid
Having mVal < 50

T2
oid val
1 30
1 10
1 10
2 10
2 50
1 1
1 0
1 1
1 0
1 1

Select *
From (Select *
    From T2
    Where □)
As T3
Join T5
On □

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

Computation overhead
Optimize computation: Grouping Predicates

<table>
<thead>
<tr>
<th>oid</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
</tbody>
</table>

Alternative inputs from its subquery

<table>
<thead>
<tr>
<th>oid</th>
<th>mVal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
</tbody>
</table>

All possible outputs of this query

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Problem: need to perform 2 × 3 operations to get only 3 results.

Discovery: Grouping predicates on the summary table.

Number of predicates reduced by 40,000×
Enumerative Predicate Synthesis
Grouping Predicates + Bit-vector Representation

Select *
From (Select *
From T1
Where □) As T3
Join (Select id, Max(val)
From T2
Where □
Group By oid
Having □) T5
On □

T2
oid val
1 30
1 10
2 50
2 10

Select *
From T2
Where □

T1
id date uid
1 12/25 1
2 11/21 3
4 12/24 2

Select oid, MAX(val)
From T4
Group By oid
Having mVal < 50

Select oid, MAX(val)
From T4
Join T5
On uid = oid
Having True

Select oid,
MAX(val)
From T4
Group By oid
Having mVal < 50

Select * 
From T1
Where □

Select oid, MAX(val)
From T4
Join T5
On id < oid
On True

Select *
From T2
Where □

Select *
From T2
Where val < 50

Select *
From T1
Where True

Select *
From T1
Where id=uid

Select *
From T1
Where id<uid

Select * 
Join T5 
On True

Select * 
Join T5 
On id < oid

Select *
From T3
Join T5
On True

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

……
1 12/25 1 1 10
2 11/21 1 1 10
4 12/24 2 1 10

1 0.
1 0.
1 0.
1 0.
1 0.
As a Programming-by-Example System

- **Synthesis process**
  - Iterating over the search depth for abstract queries
  - Instantiate abstract queries in the current depth and check results

- **Dealing with ambiguity**
  - Ranking programs by heuristic complexity, naturalness, constant coverage
  - Provide a new example / restrict aggregation functions.
Implementation — Scythe

http://scythe.cs.washington.edu

• Supported features:
  • Select, Join, Group By, Aggregation,
  • Subqueries, Outer Join, Exists, Union

• Unsupported
  • Arithmetics, Pivot, Window functions, Limit, Insert
Evaluation

• Benchmarks from Stack Overflow:
  • 57 used in development
  • 57 top-voted posts
  • 51 recent posts
• Benchmarks from prior work:
  [Zhang et al. ASE’13]
  • 23 textbook questions.
  • 5 forum posts.

In total 193 benchmarks.
Avg. Example Size: 34 cells

• Algorithms
  • Enumerative Search
    [Udupa et al. PLDI’13]
  • SqlSynthesizer
    (Decision tree algorithm)
    [Zhang et al. ASE’13]
  • Scythe

• Evaluation Condition
  • 4G memory, 600s timeout
Evaluation

Benchmarks:

- **Scythe**: 143
- **Enum**: 92
- **Benchmark**: 193

34x faster on avg.

59% can be answered within 10 seconds

Reasons for failures:

- **34**: missing features
- **15**: timeout
- **1**: failed to disambiguate

Comparing with SQLSynthesizer:

- **Scythe**: 18/28 in 120s
- **SQLSynthesizer**: 15/28 in 120s
Some Related Work

- **Enumerative search**
  - Value-based Memoization [Udupa et al. PLDI’13]

- **Search optimization with approximation**
  - Synthesizing regex from examples [Lee et al. GPCE’16]
  - Monotonicity [Hu et al. PLDI’17]

- **Synthesizing table manipulation programs**
  - Pruning search space using partial programs [Feng et al. PLDI’17]

<table>
<thead>
<tr>
<th></th>
<th>Pruning Approach</th>
<th>Pruning Overhead</th>
<th>Pruning Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scythe</td>
<td>Over-approximation</td>
<td>Higher</td>
<td>Higher</td>
</tr>
<tr>
<td>Feng et al.</td>
<td>Constraint encoded</td>
<td>Lower</td>
<td>Lower</td>
</tr>
</tbody>
</table>

Benefit from value-based search space compression.
Algorithm: Decompose Search Process
With Abstract Queries

Search abstract SQL queries

Instantiate Abstract Queries

T1
<table>
<thead>
<tr>
<th>id</th>
<th>date</th>
<th>uid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12/25</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>11/21</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>12/24</td>
<td>2</td>
</tr>
</tbody>
</table>

T2
<table>
<thead>
<tr>
<th>oid</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
</tr>
</tbody>
</table>

Tout
<table>
<thead>
<tr>
<th>oid</th>
<th>date</th>
<th>uid</th>
<th>oid</th>
<th>MaxVal</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>12/25</td>
<td>1</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>12/24</td>
<td>2</td>
<td>2</td>
<td>10</td>
</tr>
</tbody>
</table>

Try demo on http://scythe.cs.washington.edu!