Automatically Synthesizing SQL Queries from Input-Output Examples

Sai Zhang

University of Washington

Joint work with: Yuyin Sun

Computer Science & Engineering
UNIVERSITY of WASHINGTON

Goal: making it easier for non-expert users to write correct SQL queries

- Non-expert database end-users
  - Business analysts, scientists, marketing managers, etc.

can describe *what* the query task is

***do not* know *how* write a correct query**

This paper: bridge the gap!

An example

Table: student

name	stu_id
Alice	1
Bob	2
Charlie	3
Dan	4

Table: enrolled

stu_id	course_id	score
1	504	100
1	505	99
2	504	96
3	501	60
3	502	88
3	505	68

Output table

name	MAX(score)
Alice	100
Charlie	88

Find the name and the maximum course score of each student enrolled in more than 1 course.

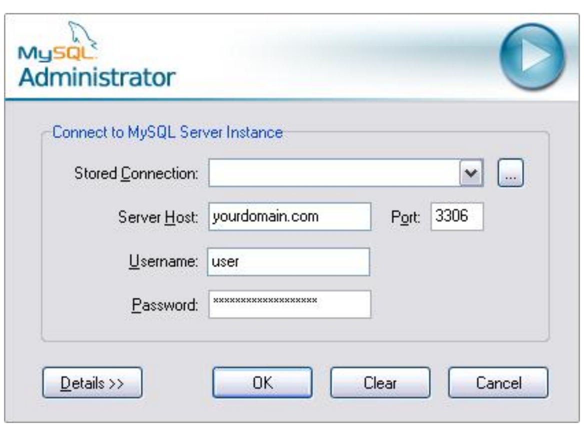
The correct SQL query:

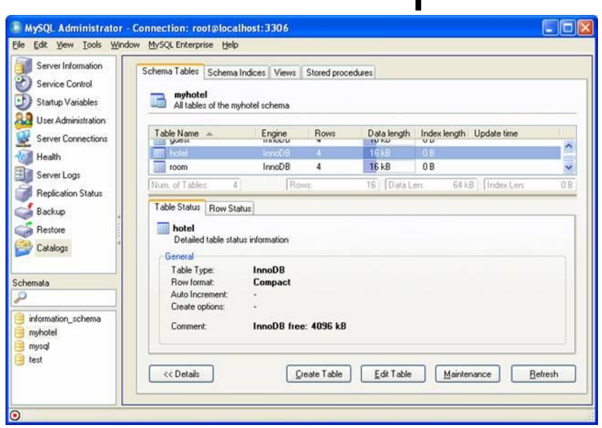
```
SELECT    name, MAX(score)
FROM      student, enrolled
WHERE     student.stu_id = enrolled.stu_id
GROUP BY student.stu_id
HAVING   COUNT(enrolled.course_id) > 1
```

Existing solutions for querying a database

- General programming languages
  - \+ powerful
  - − learning barriers

- GUI tools
  - \+ easy to use
  - − limited in customization and personalization
  - − hard to discover desired features in complex GUIs





Our solution: programming by example

Table: student

name	stu_id
Alice	1
Bob	2
Charlie	3
Dan	4

Table: enrolled

stu_id	course_id	score
1	504	100
1	505	99
2	504	96
3	501	60
3	502	88
3	505	68

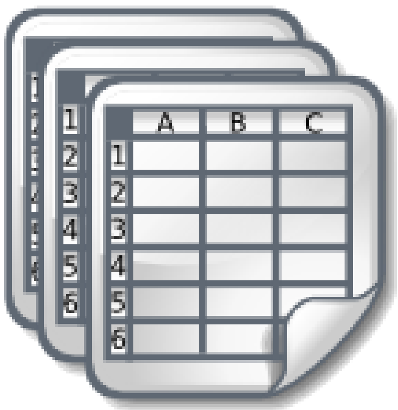
Output table

name	MAX(score)
Alice	100
Charlie	88

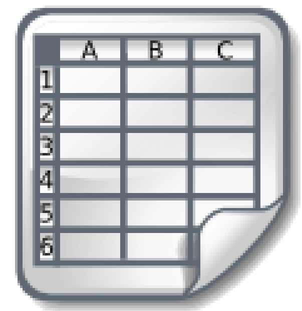
SQLSynthesizer

```
SELECT   name, MAX(score)
FROM     student, enrolled
WHERE    student.stu_id = enrolled.stu_id
GROUP BY student.stu_id
HAVING   COUNT(enrolled.course_id) > 1
```

How do end-users use SQLSynthesizer?

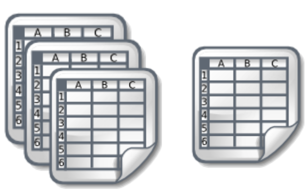


SQL?



Real, large database tables

Desired output result



SQLSynthesizer

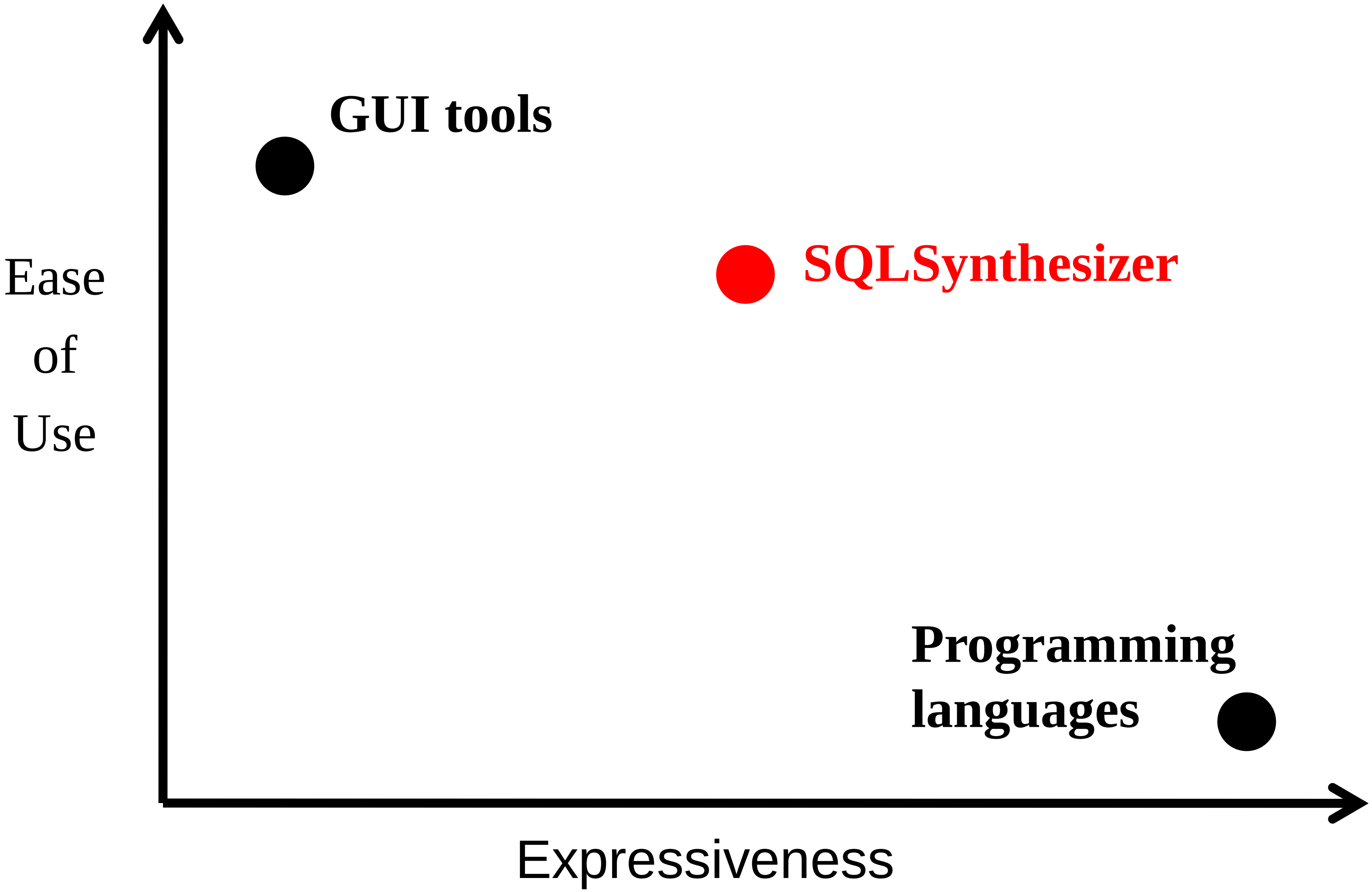
SQL

Small, *representative*
Input-output examples

SQLSynthesizer's advantages

- Fully automated
  - Only requires input-output examples
  - No need of annotations, hints, or specification of any form
- Support a wide range of SQL queries
  - Beyond the "select-from-where" queries [Tran'09]

Comparison of solutions

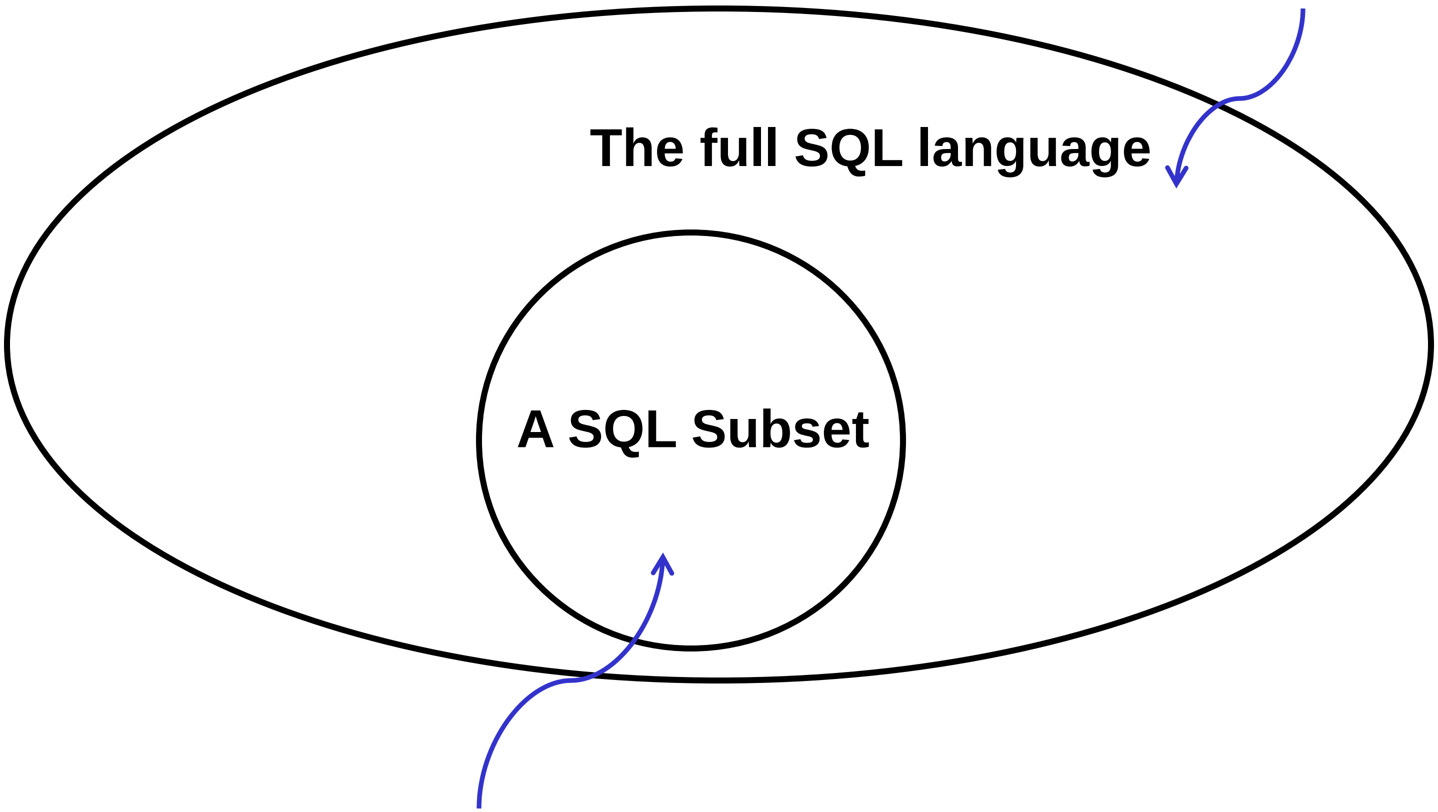


Outline

- Motivation
- A SQL Subset
- Synthesis Approach
- Evaluation
- Related Work
- Conclusion

Designing a SQL subset

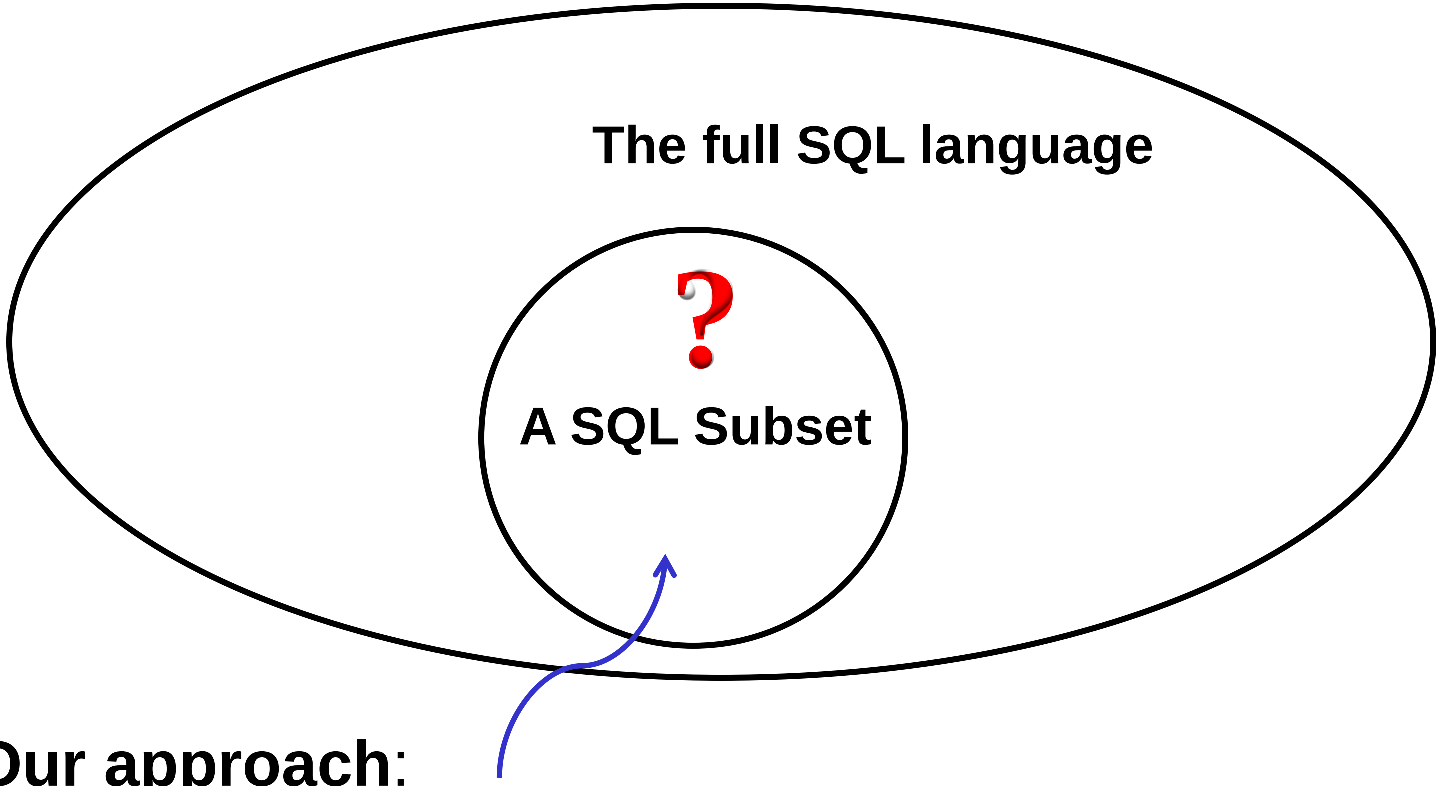
- 1000+ pages specification
- PSPace-Completeness [Sarma'10]
- Some features are rarely used



SQLSynthesizer's focus: a widely-used SQL subset

How to design a SQL subset?

- Previous approaches:
  - Decided by the paper authors [Kandel'11] [Tran'09]



- **Our approach**:
  - Ask experienced IT professionals for the most widely-used SQL features

Our approach in designing a SQL subset

1. **Online survey**: eliciting design requirement
   - Ask each participant to select 10 most widely-used SQL features
   - Got 12 responses

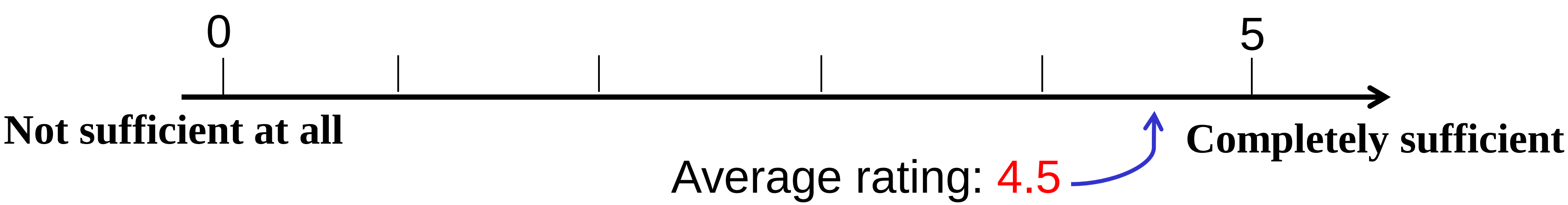
2. **Designing** the SQL subset

   Supported SQL features

   1) SELECT.. FROM…WHERE
   2) JOIN
   3) GROUP BY / HAVING
   4) Aggregators (e.g., MAX, COUNT, SUM, etc)
   5) ORDER BY

   Supported in the previous work [Tran'09]

3. **Follow-up interview**: obtaining feedback
   - Ask each participant to rate the sufficiency of the subset



Our approach in designing a SQL subset

1. **Online survey**: eliciting design requirement
   - Ask each participant to select 10 most widely-used SQL features
   - Got 12 respondents

1. **Designing** the SQL subset

   Supported SQL features

   - SELECT.. FROM…WHERE
   - JOIN
   - GROUP BY, HAVING
   - Aggregators (e.g., MAX, COUNT, SUM, etc)
   - ORDER BY

   Supported in the previous work [Tran'09]

2. **Follow-up interview**: obtaining feedback
   - Ask each participant to rate the sufficiency of the subset

0 5

Not sufficient at all **Completely sufficient**

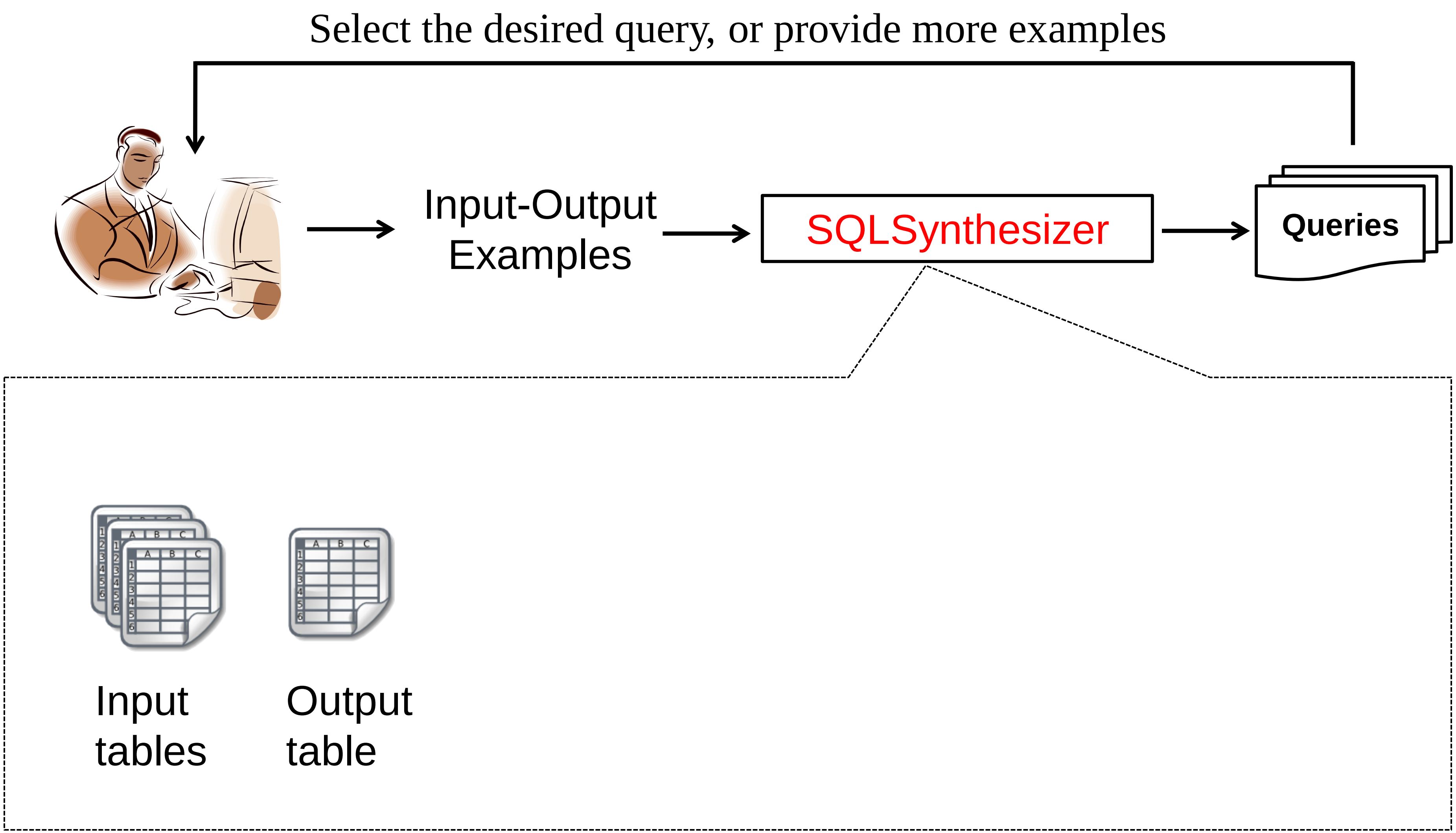
Average rating: 4.5

The SQL subset is enough to write most common queries.

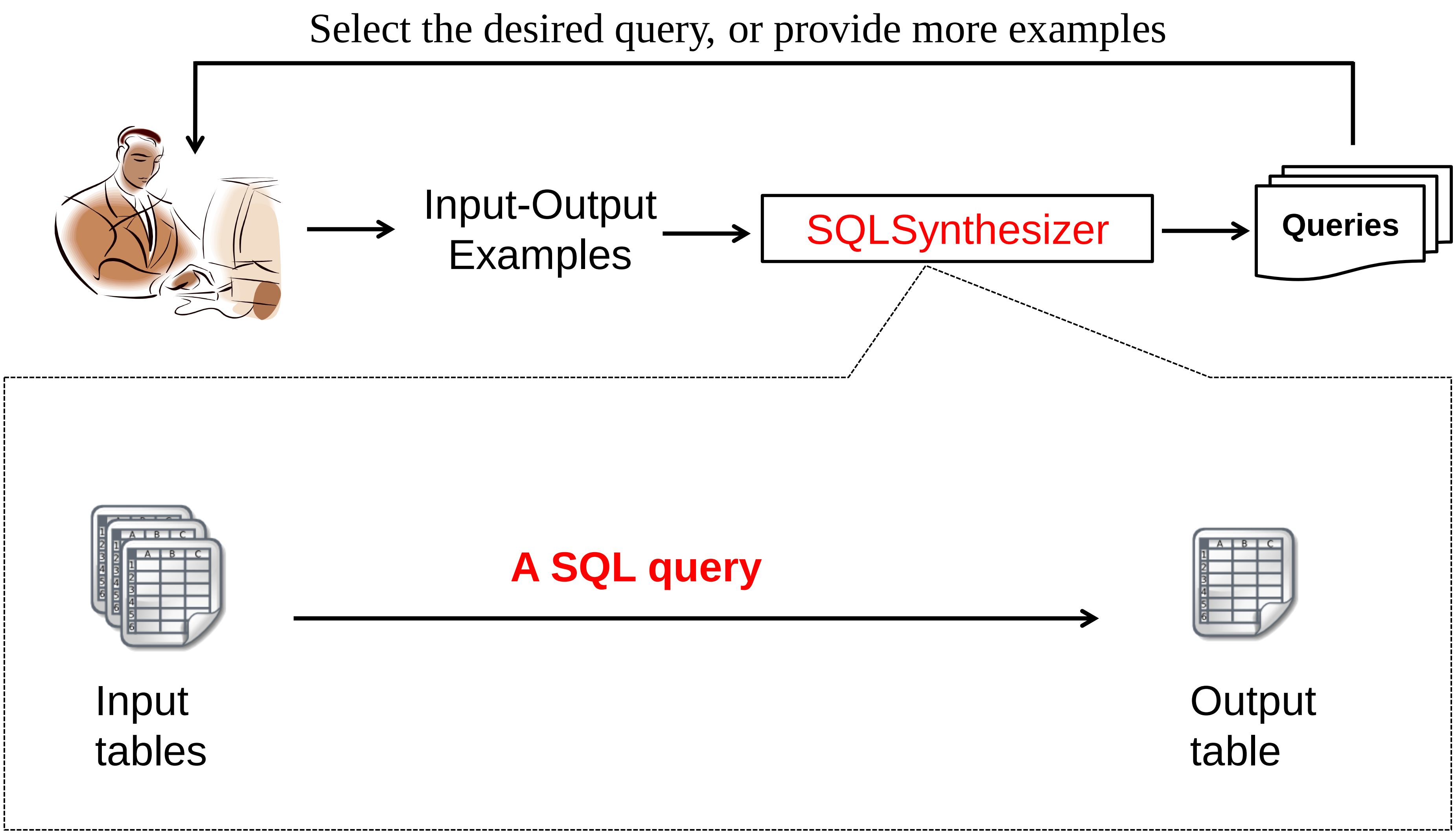
Outline

- Motivation
- Language Design
- Synthesis Approach
- Evaluation
- Related Work
- Conclusion

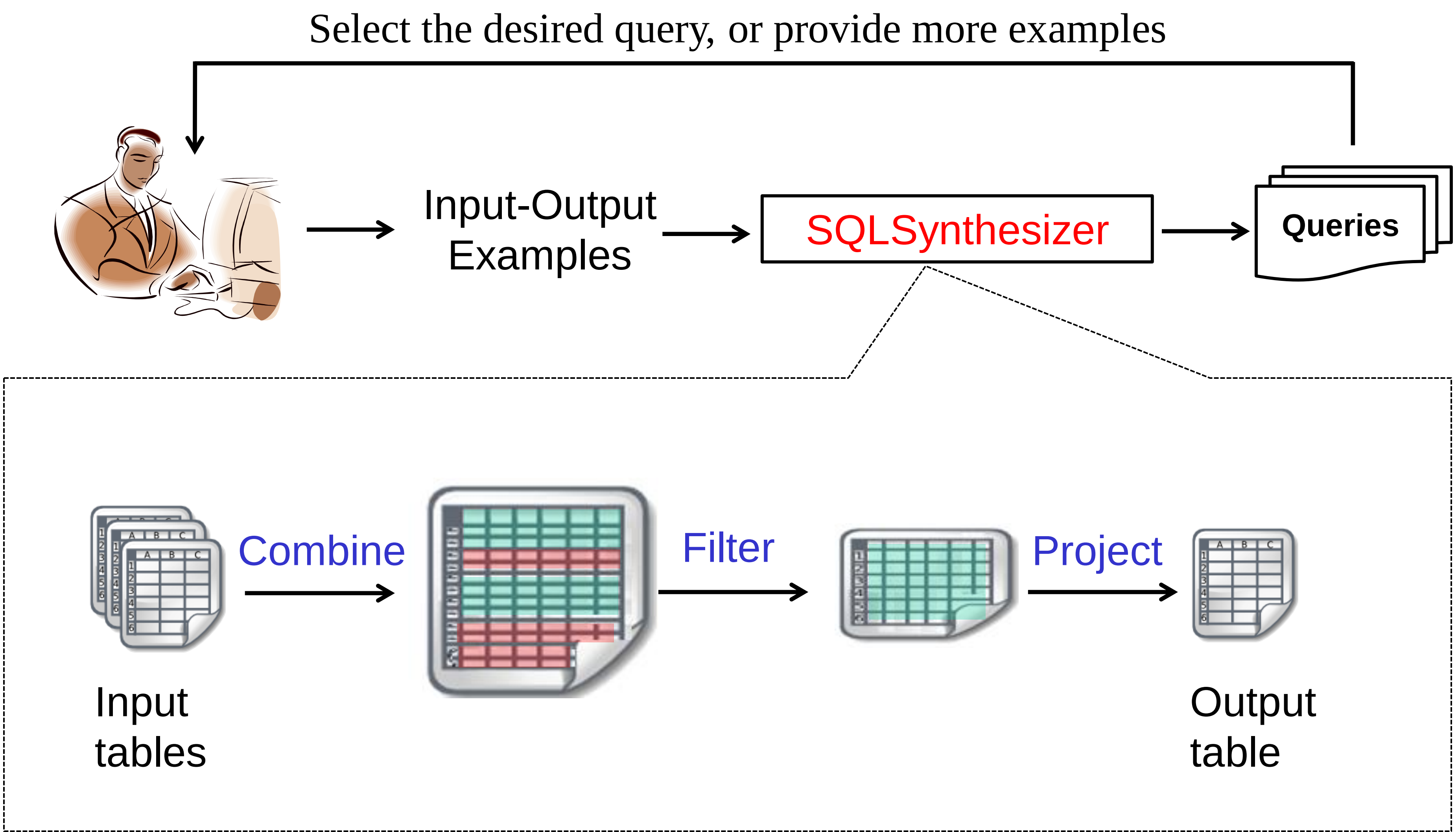
SQLSynthesizer Workflow



SQLSynthesizer Workflow



SQLSynthesizer Workflow



SQLSynthesizer Workflow

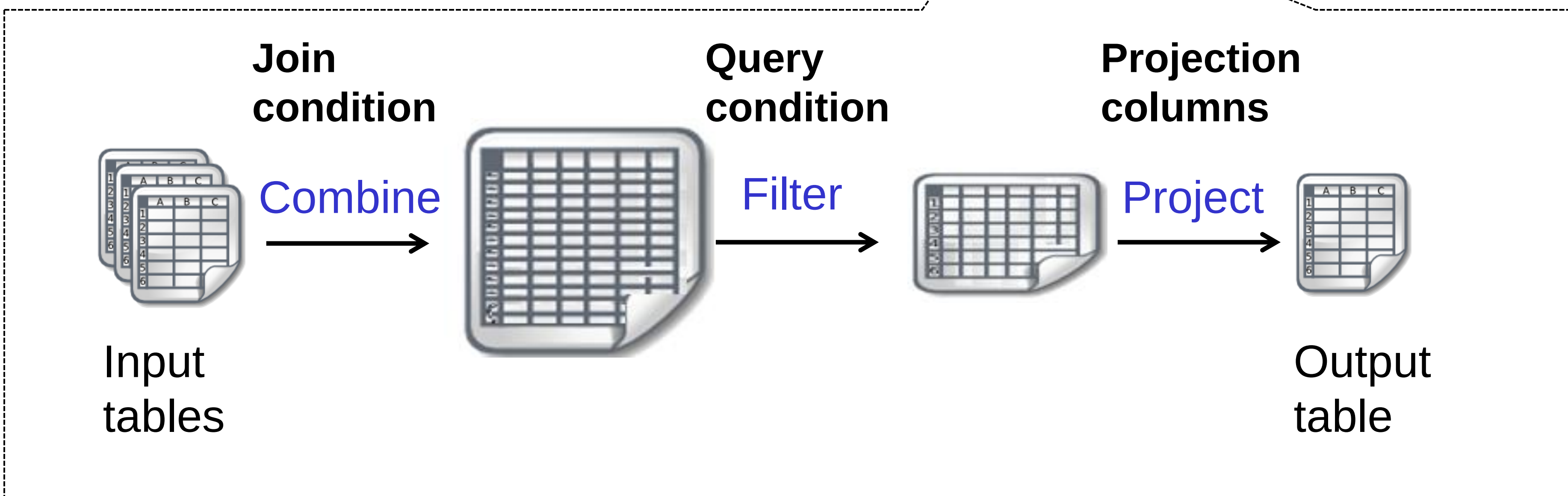
A complete SQL:

Projection columns

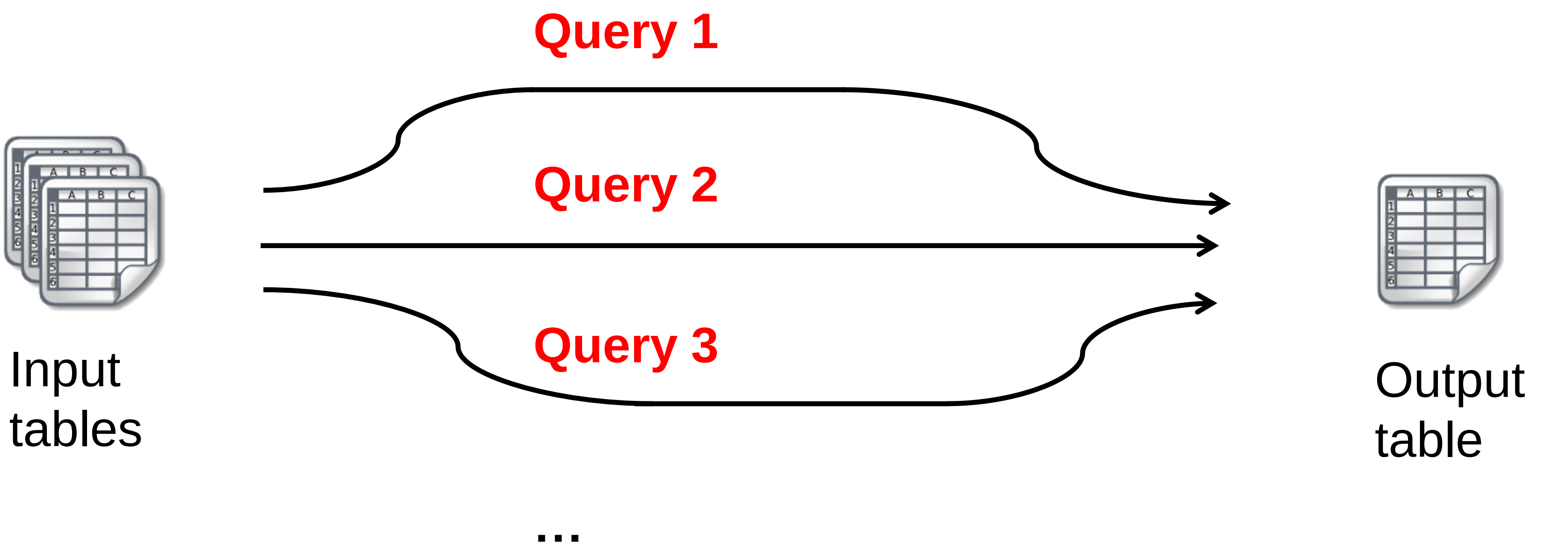
Join condition

Query condition

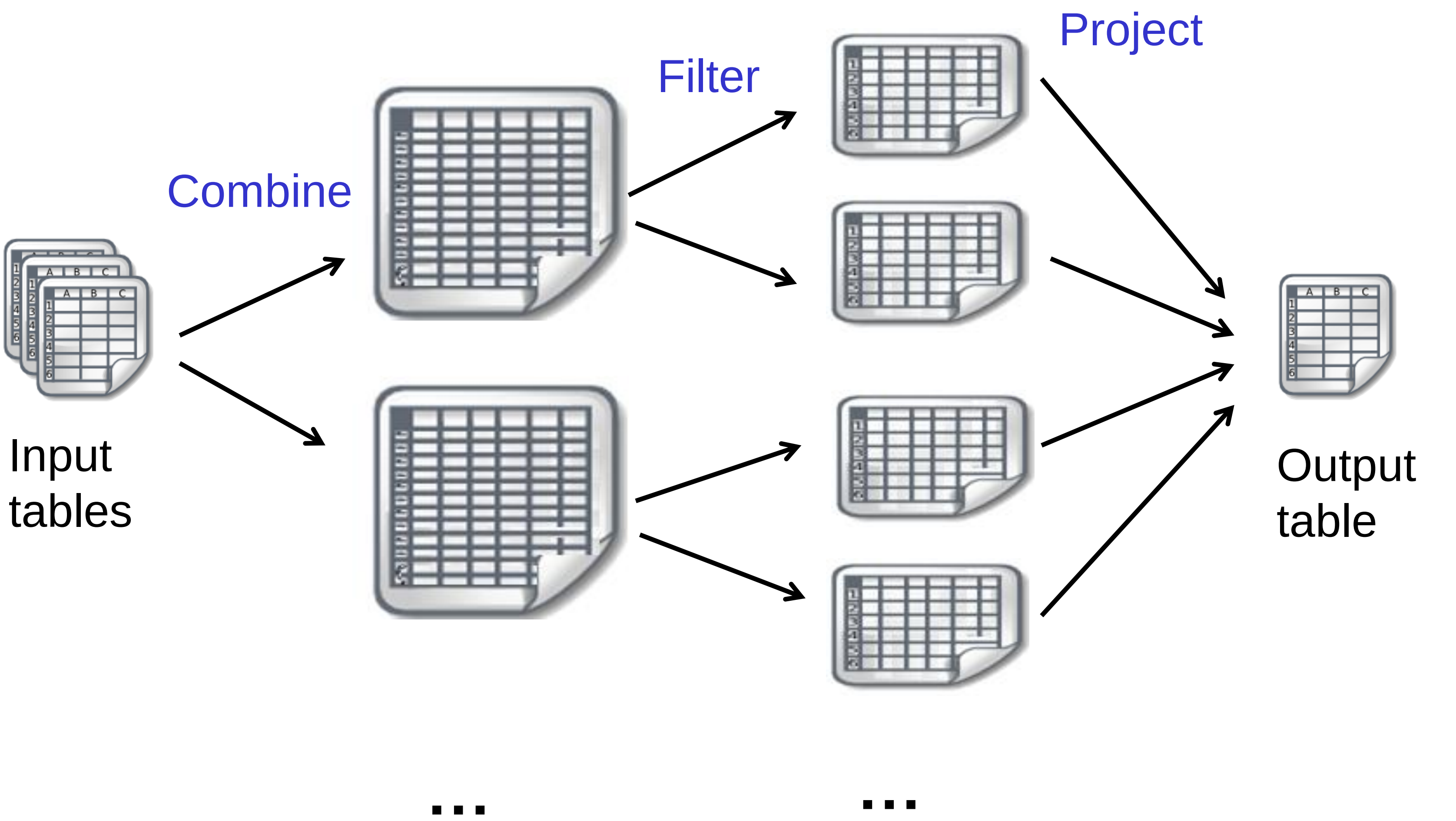
```
SELECT    name, MAX(score)
FROM      student, enrolled
WHERE     student.stu_id = enrolled.stu_id
GROUP BY  student.stu_id
HAVING    COUNT(enrolled.course_id) > 1
```



Multiple solutions

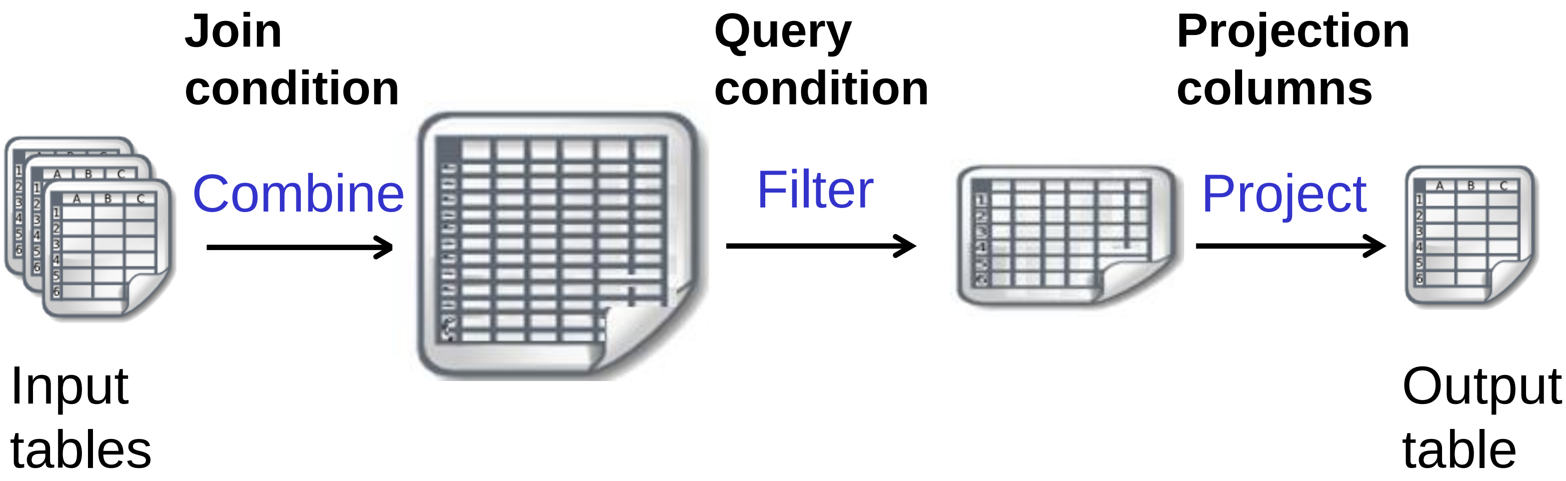


Multiple solutions



Computes all solutions, ranks them, and shows them to the user.

Key techniques



1. Combine:
   Exhaustive search over legal combinations
   (e.g., cannot join columns with different types)
2. Filter:
   A machine learning approach to infer query conditions

Key contribution

3. Project:
   Exhaustive search over legal columns
   (e.g., cannot apply `AVG` to a string column)

Learning query conditions

Cast as a rule learning problem:

Finding rules that can perfectly divide a search space into a positive part and a negative part

Rows contained in the output table

The rest of the rows

All rows in the joined table

Search space: the joined table

Table: student

name	stu_id
Alice	1
Bob	2
Charlie	3
Dan	4

Table: enrolled

stu_id	course_id	score
1	504	100
1	505	99
2	504	96
3	501	60
3	502	88
3	505	68

Join on the **`stu_id`** column →

(inferred in the Combine step)

The joined table

Name	stu_id	course_id	score
Alice	1	504	100
Alice	1	505	99
Bob	2	504	96
Charlie	3	501	60
Charlie	3	502	88
Charlie	3	505	68

Finding rules selecting rows contained in the output table

The joined table

name	stu_id	course_id	score
Alice	1	504	100
Alice	1	505	99
Bob	2	504	96
Charlie	3	501	60
Charlie	3	502	88
Charlie	4	505	68

Output table

name	MAX(score)
Alice	100
Charlie	88

Finding rules selecting rows containing the output table

The joined table

name	stu_id	course_id	score
Alice	1	504	100
Alice	1	505	99
Bob	2	504	96
Charlie	3	501	60
Charlie	3	502	88
Charlie	4	505	68

rules?

name	stu_id	course_id	score
Alice	1	504	100
Alice	1	505	99
Charlie	3	501	60
Charlie	3	502	88
Charlie	4	505	68

Finding rules selecting rows containing the output table

The joined table

name	stu_id	course_id	score
Alice	1	504	100
Alice	1	505	99
Bob	2	504	96
Charlie	3	501	60
Charlie	3	502	88
Charlie	4	505	68

rules?

No good rules!

name	stu_id	course_id	score
Alice	1	504	100
Alice	1	505	99
Charlie	3	501	60
Charlie	3	502	88
Charlie	4	505	68

Solution: computing additional features

- Key idea:
  - ***Expand*** the search space with additional features
    - ***Enumerate*** all possibilities that a table can be aggregated
    - ***Precompute*** aggregation values as features

Suppose grouping it by **stu_id**

name	stu_id	course_id	score
Alice	1	504	100
Alice	1	505	99
Bob	2	504	96
Charlie	3	501	60
Charlie	3	502	88
Charlie	3	505	68

The joined table

additional features

MAX(score)
100
100
96
88
88
88

Finding rules without additional features

The joined table

name	stu_id	course_id	score
Alice	1	504	100
Alice	1	505	99
Bob	2	504	96
Charlie	3	501	60
Charlie	3	502	88
Charlie	4	505	68

No good rules!

name	stu_id	course_id	score
Alice	1	504	100
Alice	1	505	99
Charlie	3	501	60
Charlie	3	502	88
Charlie	4	505	68

Finding rules with additional features

The joined table

after the table is grouped by **stu_id**

name	stu_id	course_id	score	COUNT(course_id)	MIN(score)	…
Alice	1	504	100	2	99	
Alice	1	505	99	2	99	
Bob	2	504	96	1	96	
Charlie	3	501	60	3	60	
Charlie	3	502	88	3	60	
Charlie	4	505	68	3	60	

COUNT(course_id) > 1
(after groupping by **stu_id**)

name	stu_id	course_id	score
Alice	1	504	100
Alice	1	505	99
Charlie	3	501	60
Charlie	3	502	88
Charlie	4	505	68

```
SELECT    name, MAX(score)
FROM      student, enrolled
WHERE     student.stu_id =
             enrolled.stu_id
GROUP BY  student.stu_id
HAVING    COUNT(enrolled.course_id) > 1
```

Ranking multiple SQL queries

- Occam's razor principle: rank simpler queries higher
  - A simpler query is less likely to overfit the examples

- Approximate a query's complexity by its text length

Input table: student

name	age	score
Alice	20	100
Bob	20	99
Charlie	30	99

Output table

name
Alice
Bob

Query 1: **`select name from student where age < 30`**

Query 2: **`select name from student where name = ‘Alice’ || name = ‘Bob’`**

Outline

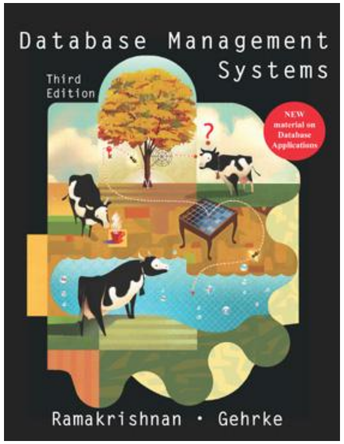
- Motivation
- Language Design
- Synthesis Approach
- Evaluation
- Related Work
- Conclusion

Research Questions

- Success ratio in synthesizing SQL queries?
- What is the tool time cost?
- How much human effort is needed in writing examples?
- Comparison to existing techniques.

Benchmarks

- 23 SQL query related exercises from a classic textbook
  - ***All*** exercises in chapters 5.1 and 5.2



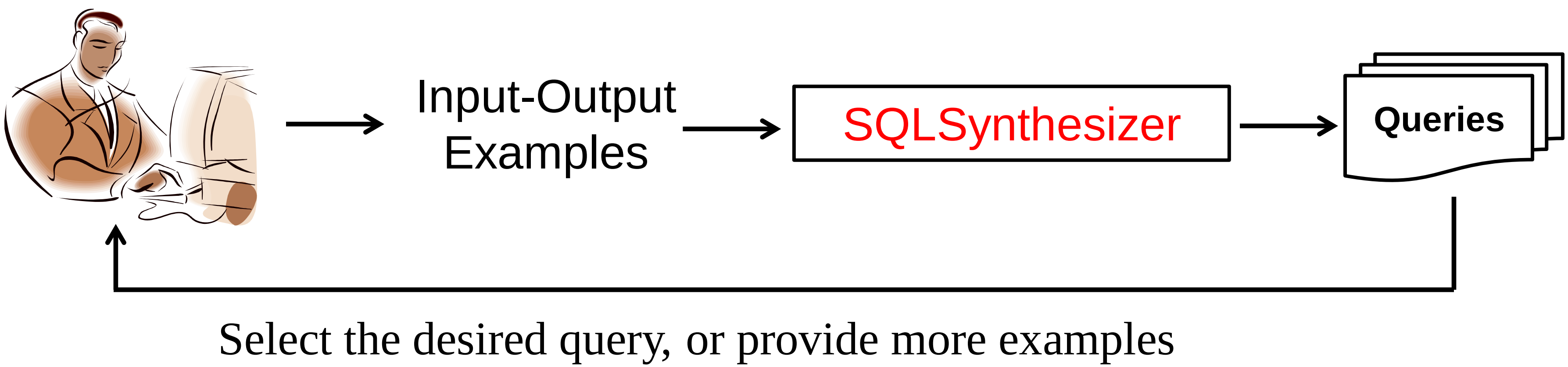
- 5 forum questions
  - Can be answered by using standard SQL (Most questions are vendor-specific)
  - 2 questions contain example tables

stackoverflow

DATABASE JOURNAL
The Knowledge Center for Database Professionals

Tutorialized™ Forums

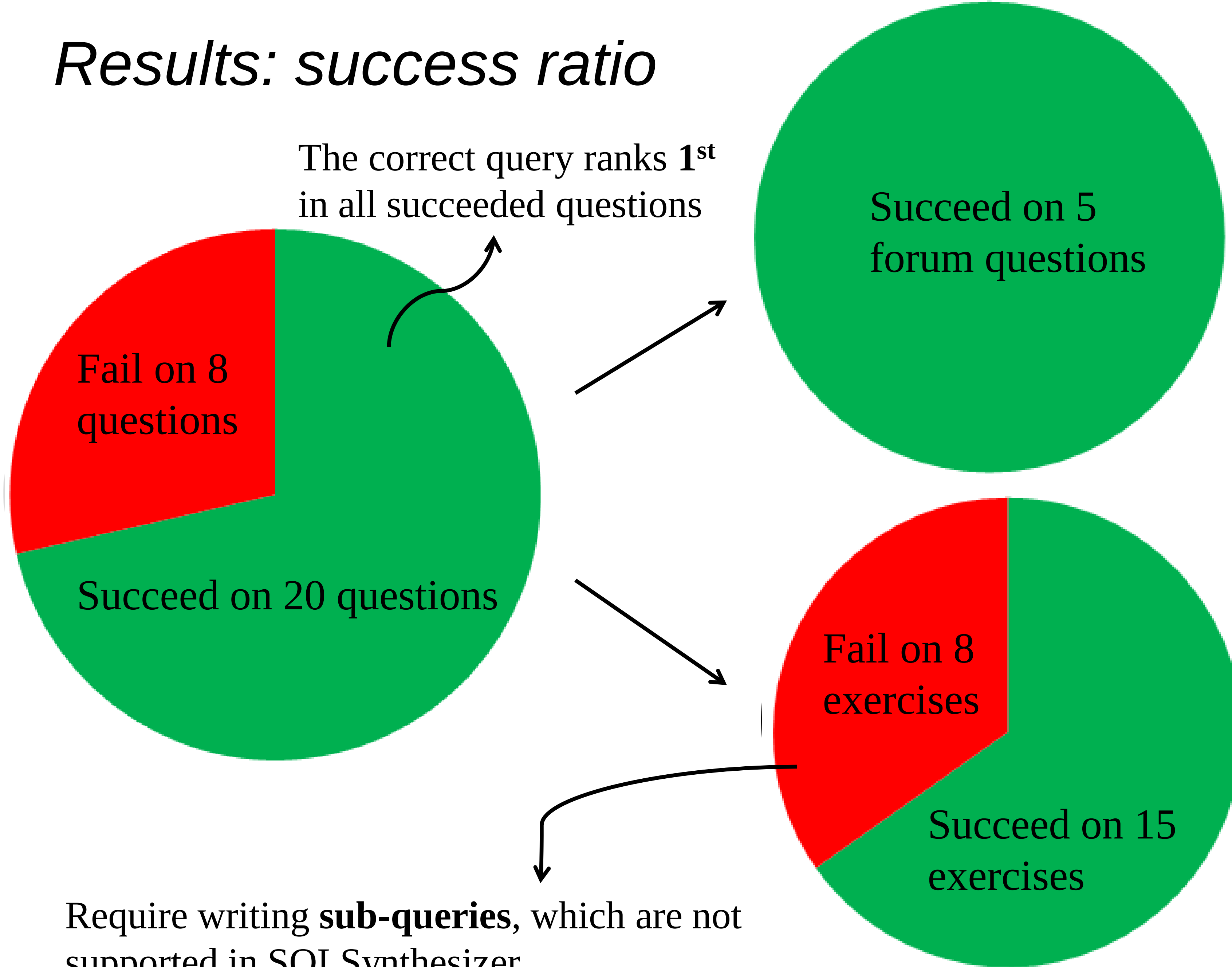
Evaluation Procedure



- Rank of the correct SQL query
- Tool time cost
- Manual cost
  - Example size, time cost, and the number of interaction rounds

(All experiments are done by the second author)

Results: success ratio



Result: tool time cost

- On average, 8 seconds per benchmark
  - Min: 1 second, max: 120 seconds
  - Roughly proportional to the #table and #column

Results: manual cost

- Example size
  - 22 rows, on average (min: 8 rows, max: 52 rows)
- Time cost in writing examples
  - 3.6 minutes per benchmark, on average
    (min: 1 minute, max: 7 minutes)
- Number of interaction rounds
  - 2.3 rounds per benchmark, on average
    (min: 1 round, max: 5 rounds)

Comparison with an existing approach

- Query-by-Output (QBO) [Tran'09]
  - Support simple "select-from-where" queries
  - Use data values as machine learning features

Table: student

name	age	score
Alice	20	100
Bob	20	99
Charlie	30	80

```
select name from student
   where age < 30
```

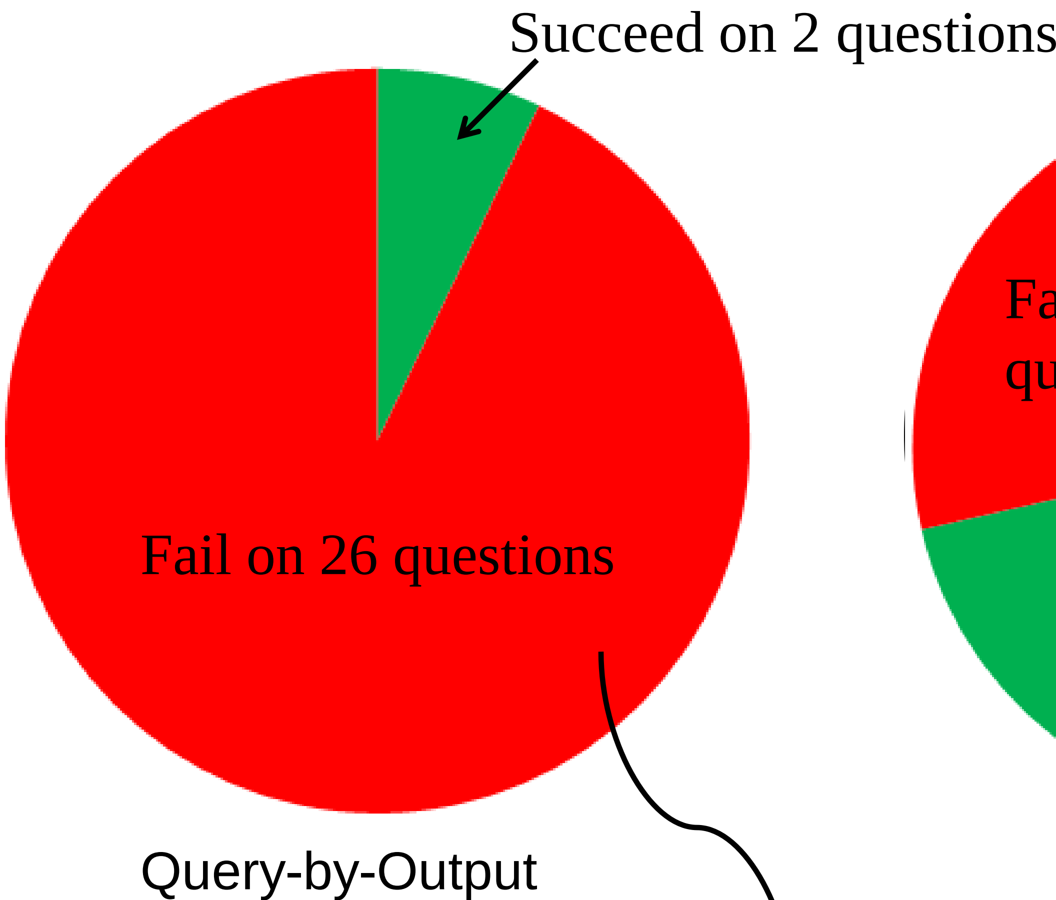
Output table

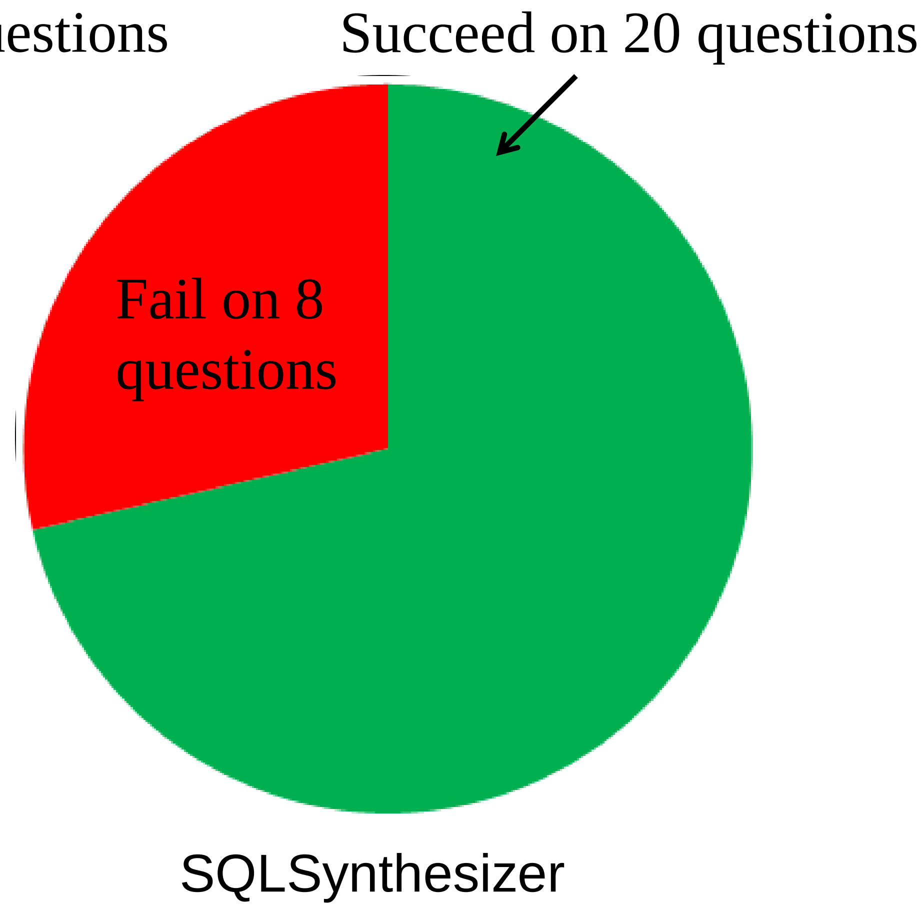
name
Alice
Bob

```
select age, max(score) from student
group by age
```

age	MAX(score)
20	100
30	80

Query-by-Output vs. SQLSynthesizer





- *Many realistic SQL queries use aggregation features.*
- *Users are unlikely to get stuck on simple "select-from-where" queries.*

Experimental Conclusions

- Good success ratio (71%)
- Low tool time cost
  - 8 seconds on average
- Reasonable manual cost
  - 3.6 minutes on average
  - 2.3 interaction rounds
- Outperform an existing technique
  - Success ratio: QBO (7%) vs. SQLSynthesizer (71%)

Outline

- Motivation
- Language Design
- Synthesis Approach
- Evaluation
- Related Work
- Conclusion

Related Work

- **Reverse engineering SQL queries**

  Query-by-Examples [Zloof'75]

  A new GUI with a domain-specific language to write queries

  Query-by-Output [Tran'09]

  Uses data values as features, and supports a small SQL subset.

  View definition Synthesis [Sarma'10]

  Theoretical analysis, and is limited to 1 input/output table.

- **Automated program synthesis**

  PADS [Fisher'08], Wrangler [Kandel'11], Excel Macro [Harris'11], SQLShare [Howe'11], SnippetSuggest [Khoussainova'11],

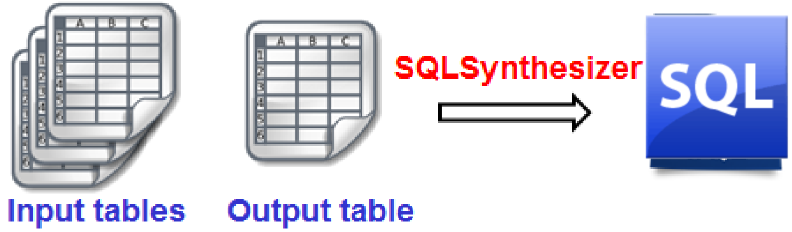
  SQL Inference from Java code [Cheung'13]

  - *Targets different problems, or requires different input.*
  - *Inapplicable to SQL synthesis*

Outline

- Motivation
- Language Design
- Synthesis Approach
- Evaluation
- Related Work
- Conclusion

Contributions



- A programming-by-example technique
  - Synthesize SQL queries from input-output examples
  - Core idea: using machine learning to infer query conditions

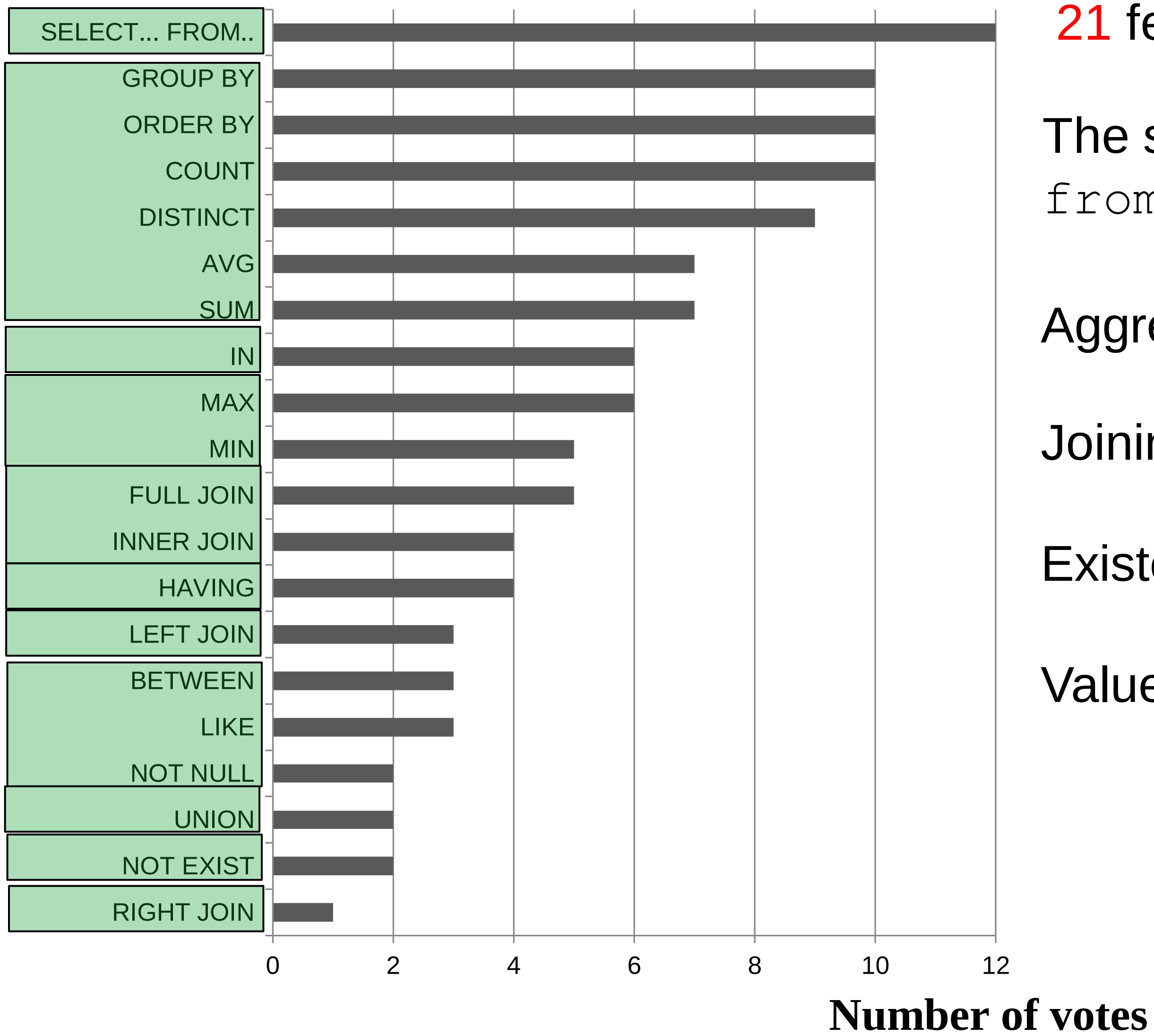
- Experiments that demonstrate its usefulness
  - Accurate and efficient
    - Inferred correct answers for 20 out of 28 SQL questions
    - 8 seconds for each question
  - Outperforms an existing technique

- The SQLSynthesizer implementation

  http://sqlsynthesizer.googlecode.com

[Backup Slides]

The most widely-used SQL features



21 features

The standard `select .. from.. where..` feature

Aggregation features

Joining features

Existential features

Value matching features

Design a SQL subset

Covered 15 features