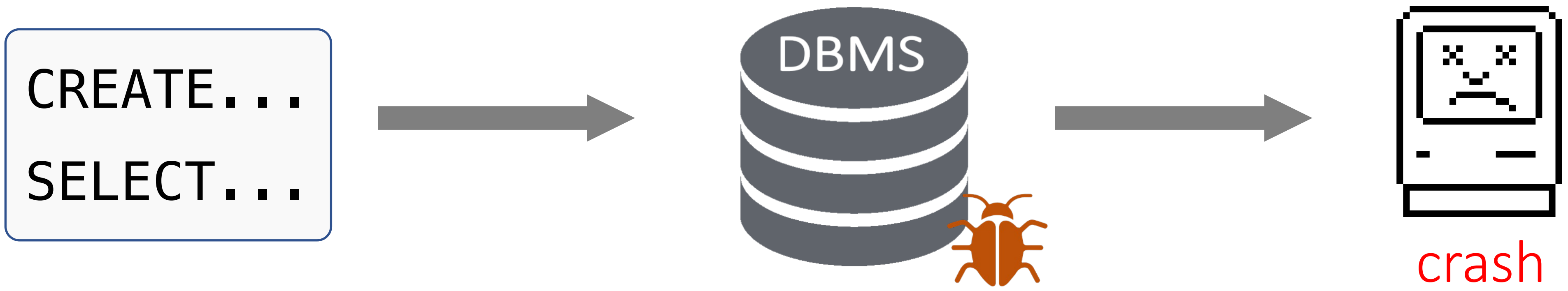
Detecting Logical Bugs of DBMS with Coverage-based Guidance

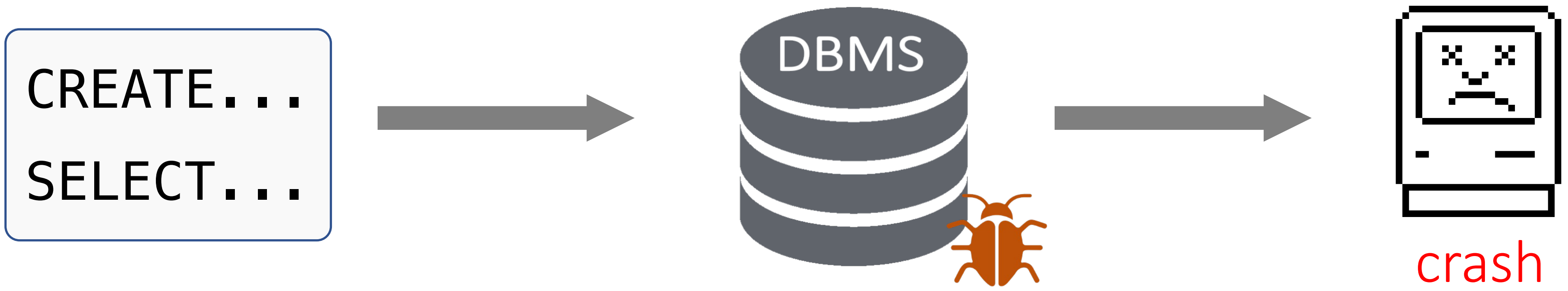
Yu Liang Song Liu Hong Hu

PennState QI-ANXIN

Memory Bugs in DBMS: Well Studied

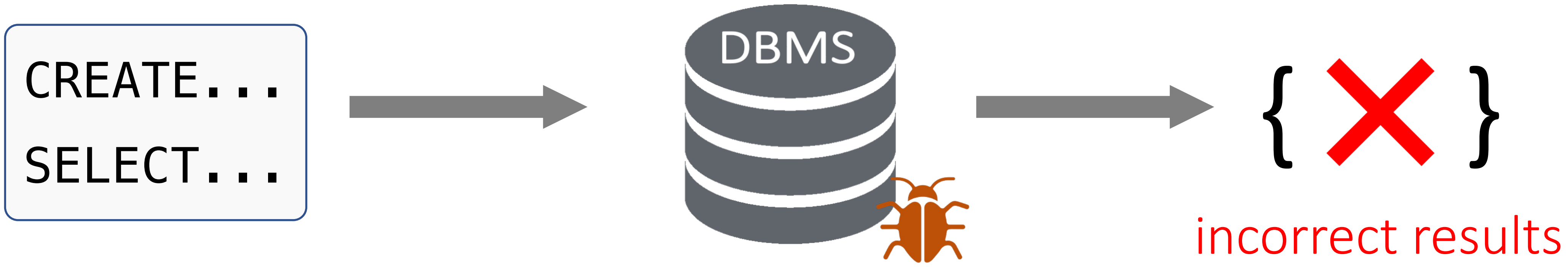

```
CREATE...
SELECT...
```

Memory Bugs in DBMS: Well Studied

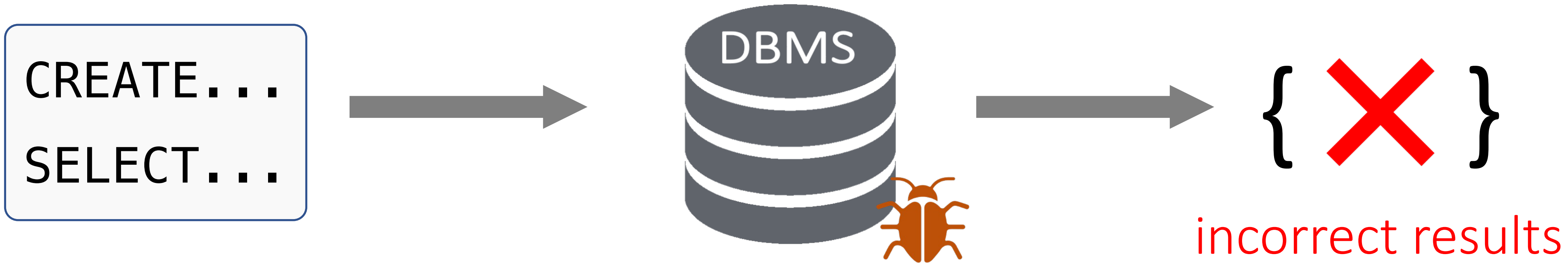


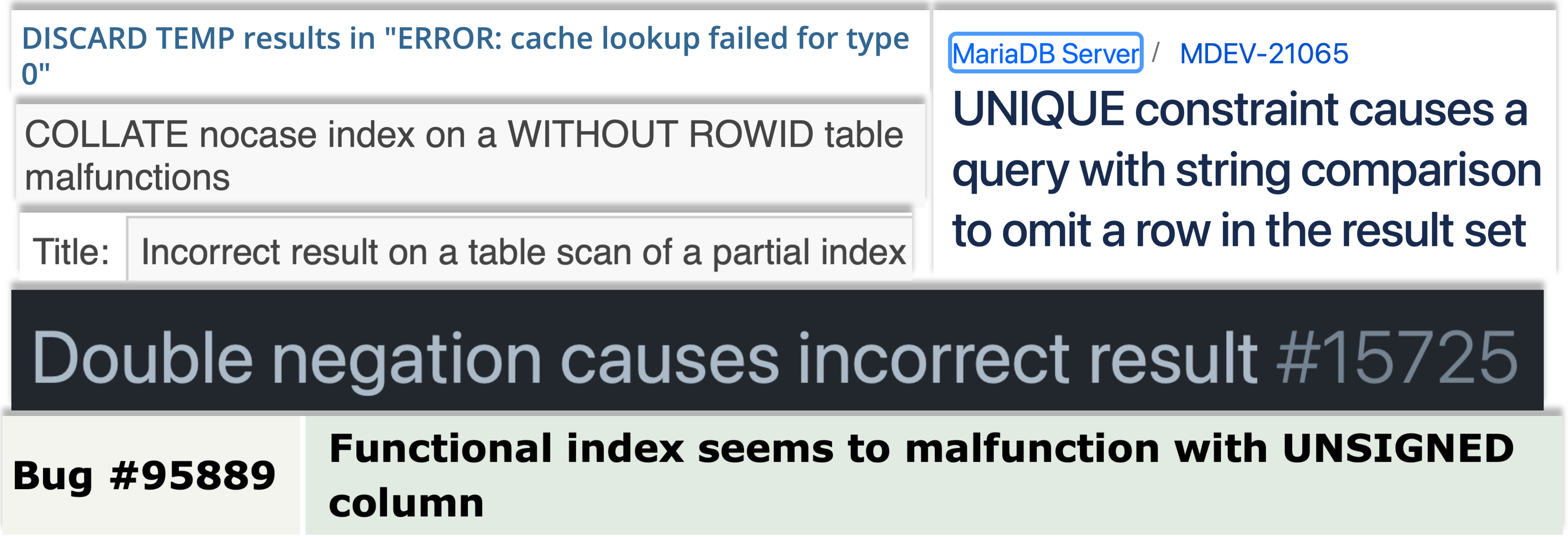
- Generation-based testing
  - *SQLsmith, QAGen [SIGMOD'07], QGEN [VLDB'04] ...*
- Mutation-based fuzzing
  - *Squirrel [CCS'20], PolyGlot [Oakland'21], RATEL [ICSE-SEIP'21] ...*

Logical Bugs in DBMS: Limited Exploration



Logical Bugs in DBMS: Limited Exploration





DISCARD TEMP results in "ERROR: cache lookup failed for type 0"

COLLATE nocase index on a WITHOUT ROWID table malfunctions

Title: Incorrect result on a table scan of a partial index

MariaDB Server / MDEV-21065

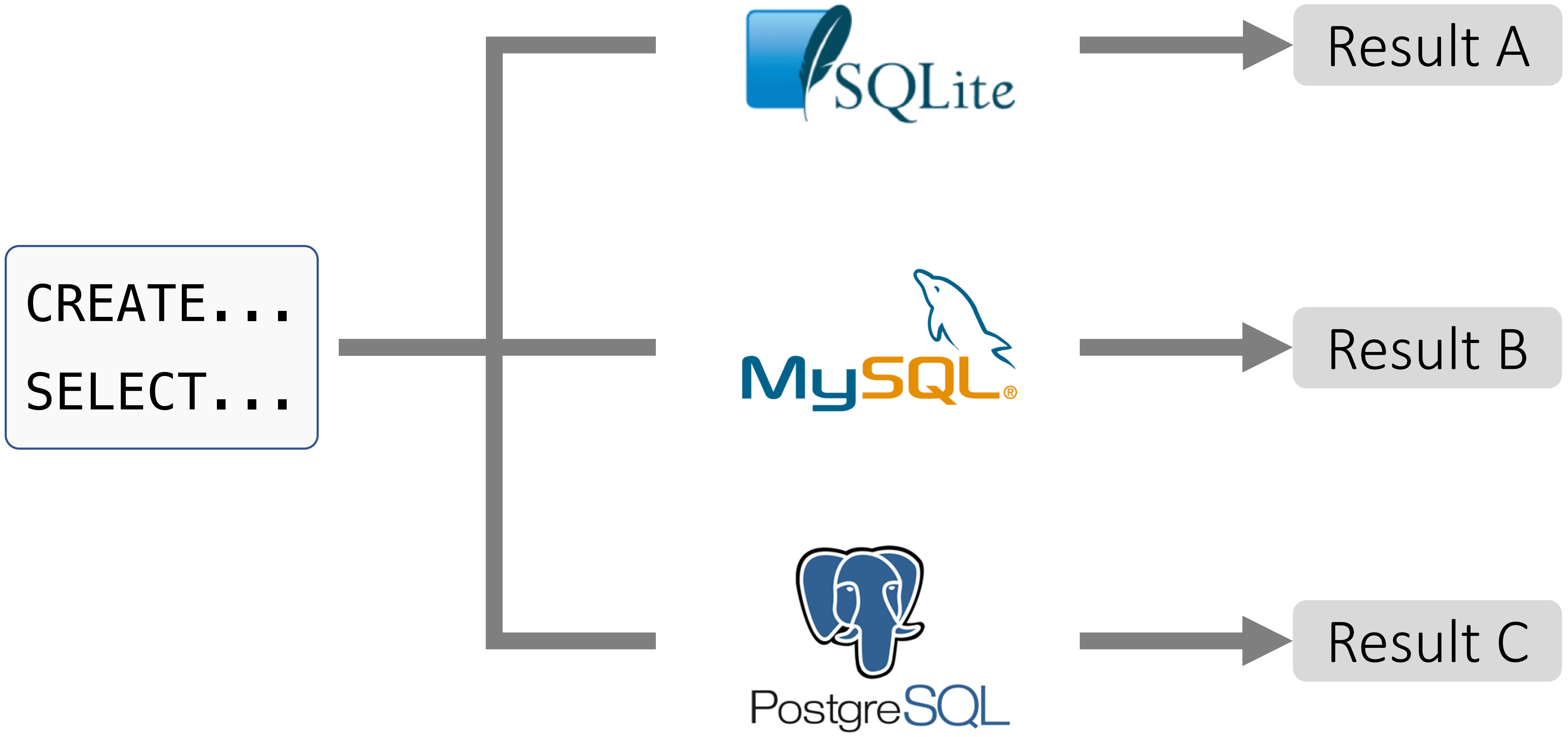
UNIQUE constraint causes a query with string comparison to omit a row in the result set

Double negation causes incorrect result #15725

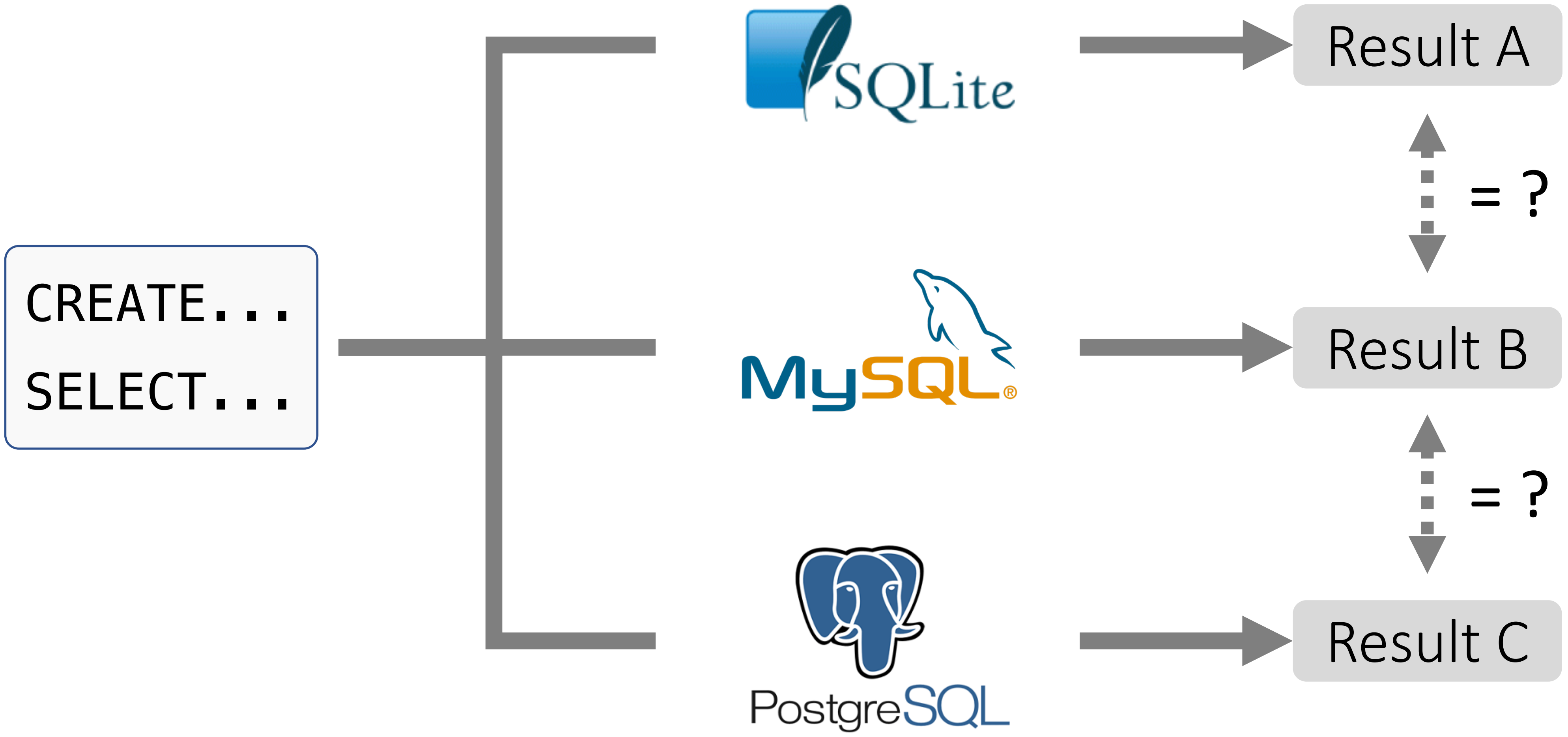
Bug #95889 Functional index seems to malfunction with UNSIGNED column

Existing Works: Differential Testing

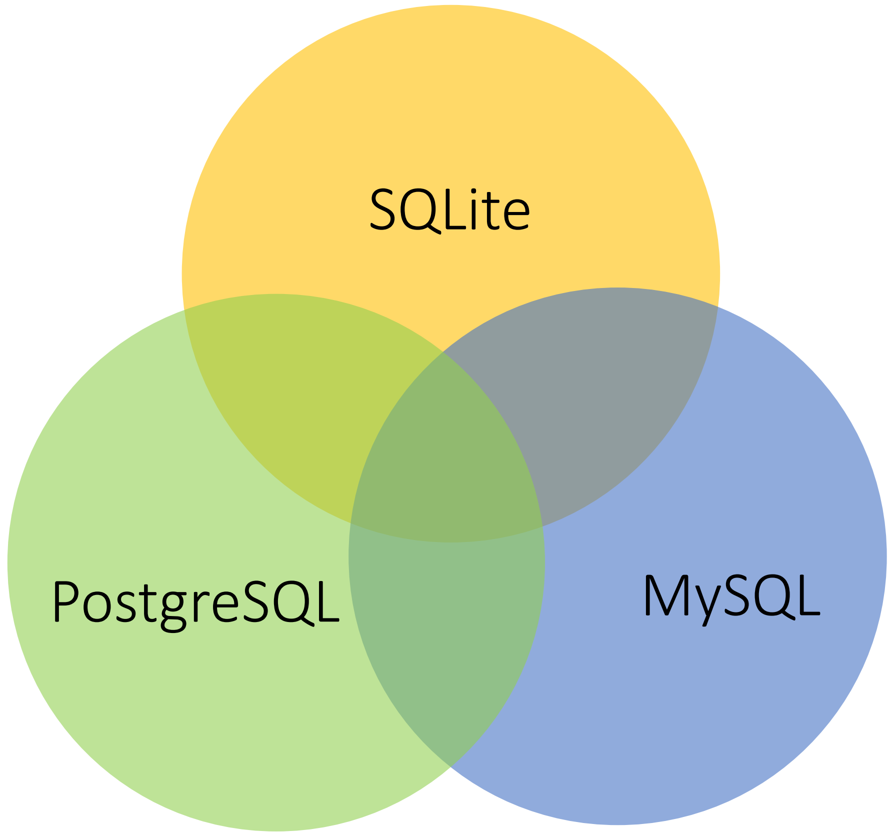
Existing Works: Differential Testing



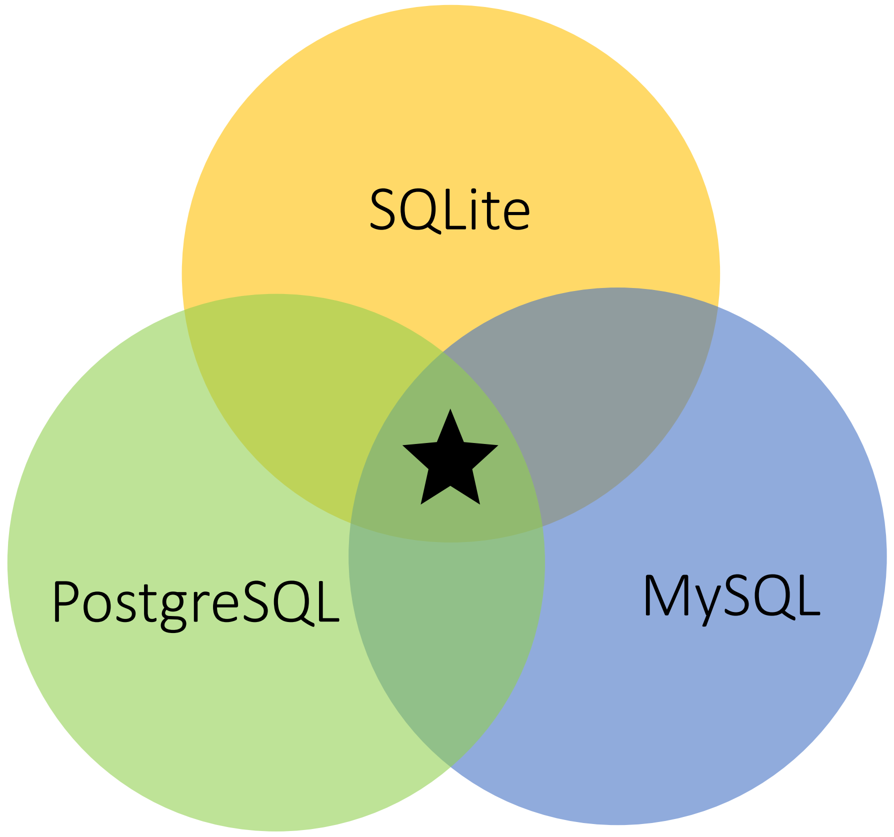
Existing Works: Differential Testing



Existing Works: Differential Testing

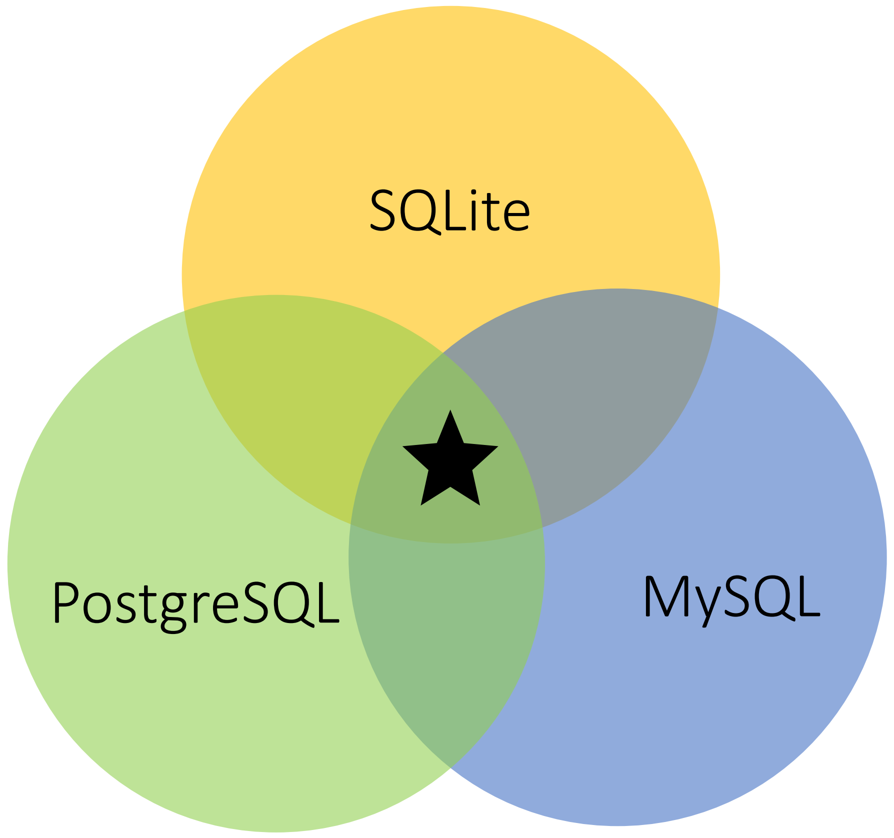


Existing Works: Differential Testing



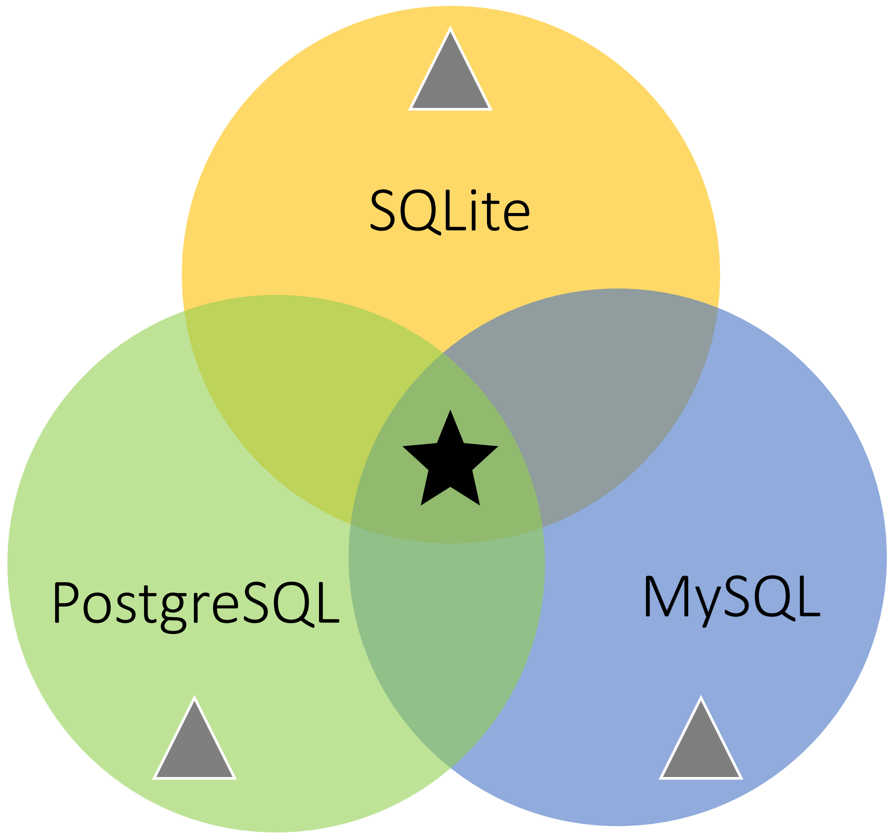
★ Limited common syntax

Existing Works: Differential Testing



★ Limited common syntax ⟶ Low coverage

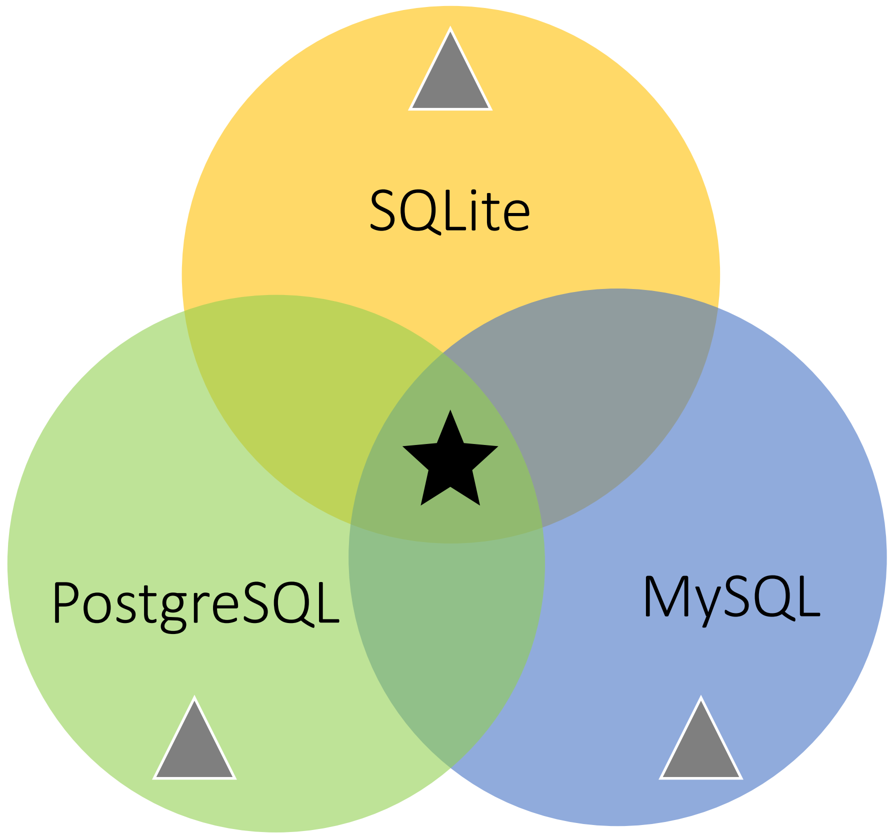
Existing Works: Differential Testing



★ Limited common syntax → Low coverage

▲ Various dialects/features

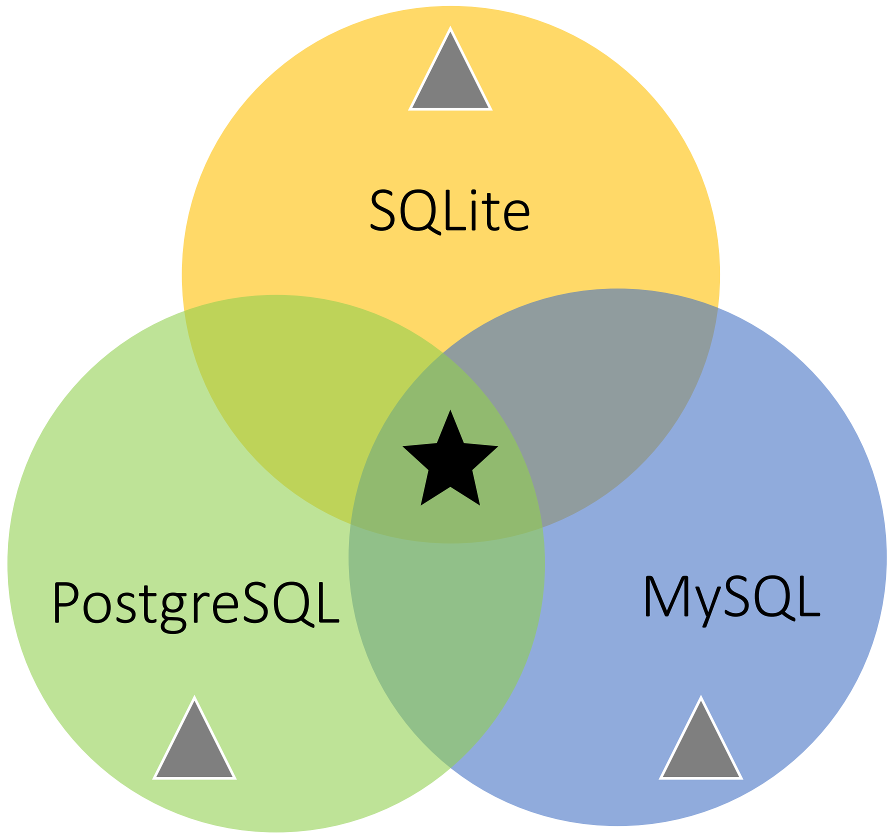
Existing Works: Differential Testing



★ Limited common syntax → Low coverage

▲ Various dialects/features → Low correctness rate (validity)

Existing Works: Differential Testing



- SQLite dialects

```
without rowid; fts5; ...
```

- PostgreSQL dialects

```
pg_catalog; integer[]; ...
```

- MySQL dialects

```
datetime; json_set(); ...
```

★ Limited common syntax ➡ Low coverage

▲ Various dialects/features ➡ Low correctness rate (validity)

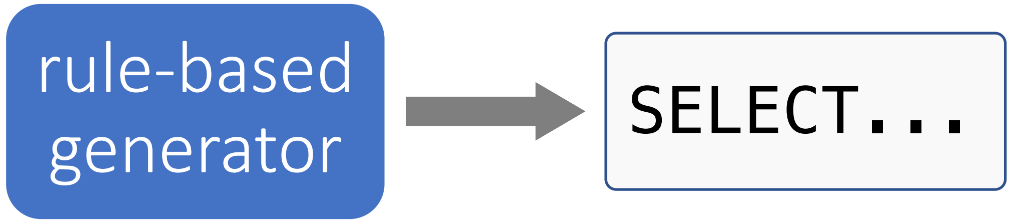
Existing Works: SQLancer

Existing Works: SQLancer

- Use oracles to find logical bugs
    - compare results from function-equivalent queries

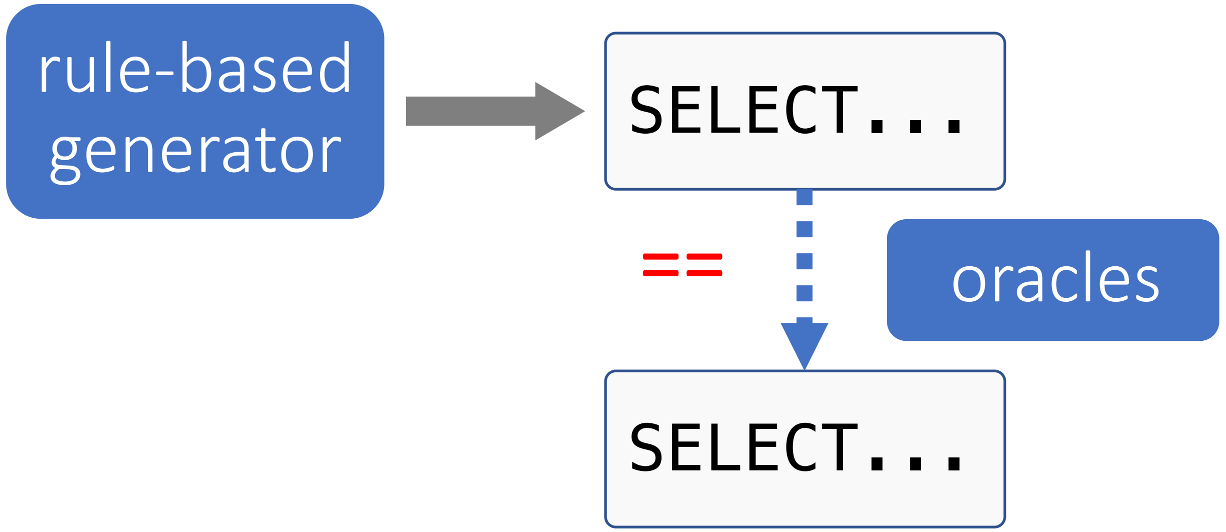
Existing Works: SQLancer

- Use oracles to find logical bugs
    - compare results from function-equivalent queries



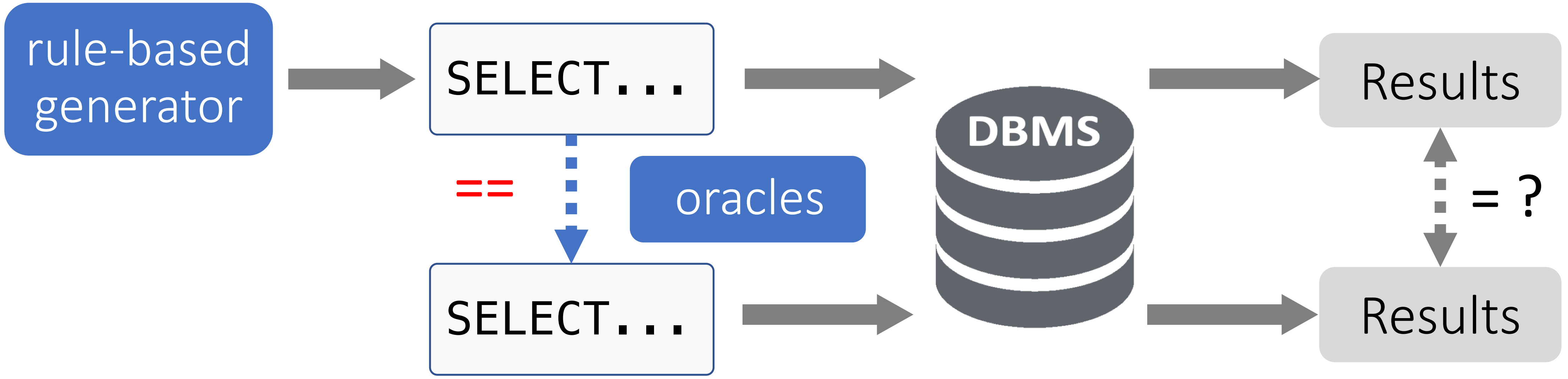
Existing Works: SQLancer

- Use oracles to find logical bugs
    - compare results from function-equivalent queries



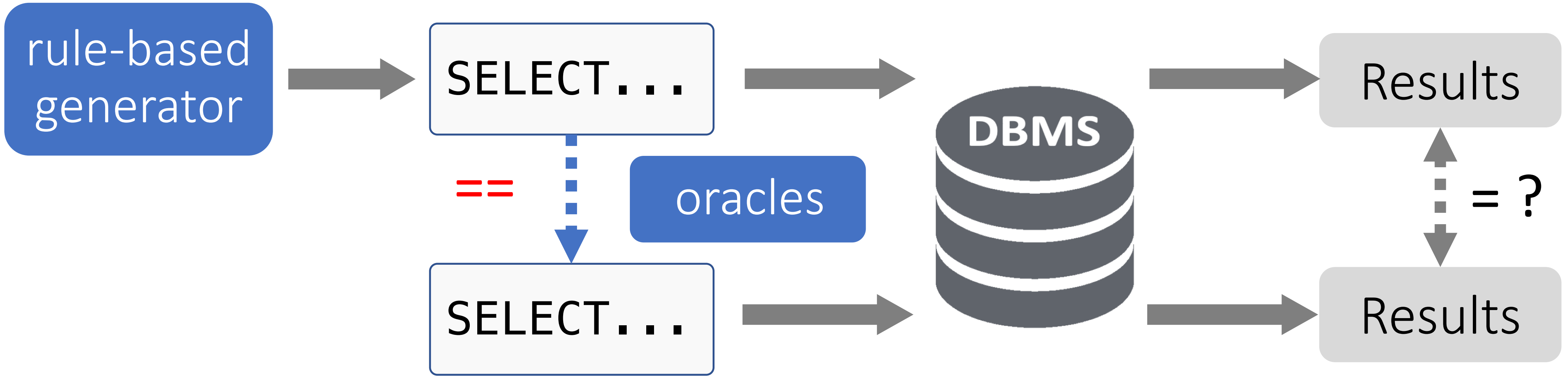
Existing Works: SQLancer

- Use oracles to find logical bugs
    - compare results from function-equivalent queries



Existing Works: SQLancer

- Use oracles to find logical bugs
  - compare results from function-equivalent queries
- Cons: rely on rule-based query generator
  - limited to explore deep program logic



Contributions

- SQLRight: a general platform to test DBMS logical bugs
  - coverage-guided fuzzing
  - validity-oriented mutation
  - general interfaces for DBMS oracles

Contributions

- SQLRight: a general platform to test DBMS logical bugs
    - coverage-guided fuzzing
    - validity-oriented mutation
    - general interfaces for DBMS oracles
- Found 18 logical bugs in SQLite and MySQL

Contributions

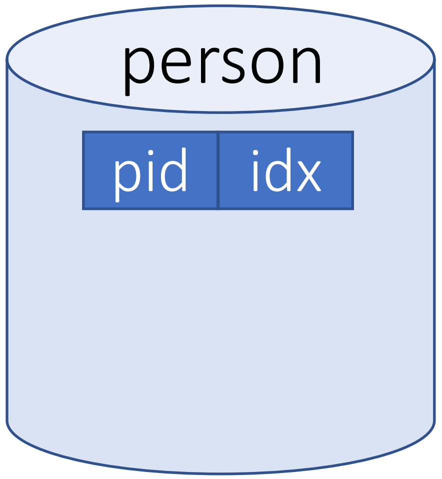
- SQLRight: a general platform to test DBMS logical bugs
  - coverage-guided fuzzing
  - validity-oriented mutation
  - general interfaces for DBMS oracles
- Found 18 logical bugs in SQLite and MySQL
- https://github.com/psu-security-universe/sqlright

Motivating Example (SQLite)

```
CREATE TABLE person (pid INT);
INSERT INTO person VALUES (1), (10), (10);
CREATE UNIQUE INDEX idx ON person (pid) WHERE pid=1;
SELECT DISTINCT pid FROM person WHERE pid=10;
```

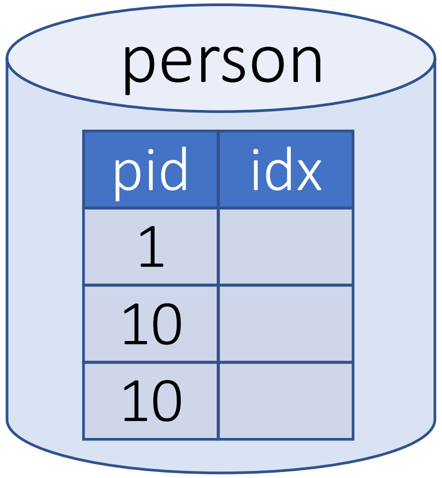
Motivating Example (SQLite)

```
CREATE TABLE person (pid INT);
INSERT INTO person VALUES (1), (10), (10);
CREATE UNIQUE INDEX idx ON person (pid) WHERE pid=1;
SELECT DISTINCT pid FROM person WHERE pid=10;
```



Motivating Example (SQLite)

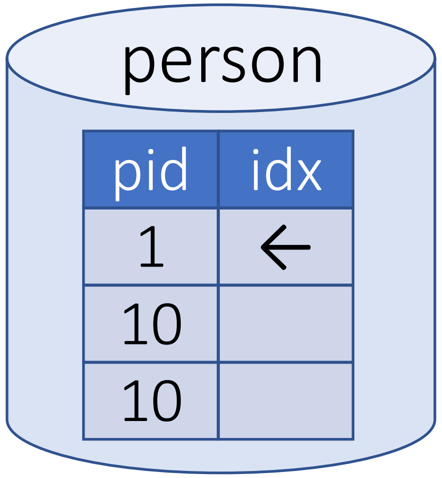
```
CREATE TABLE person (pid INT);
INSERT INTO person VALUES (1), (10), (10);
CREATE UNIQUE INDEX idx ON person (pid) WHERE pid=1;
SELECT DISTINCT pid FROM person WHERE pid=10;
```



pid	idx
1	
10	
10	

Motivating Example (SQLite)

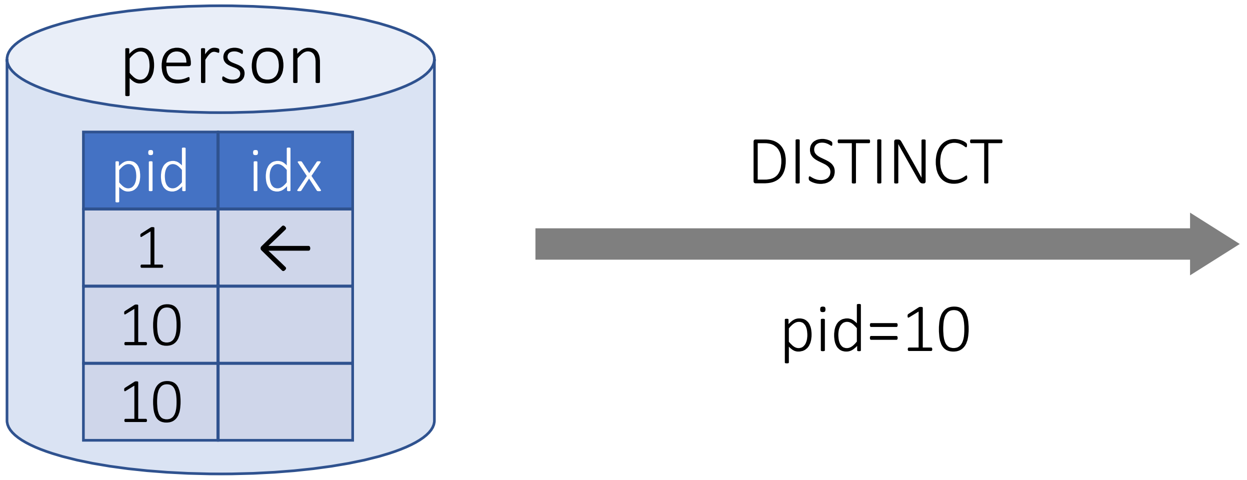
```
CREATE TABLE person (pid INT);
INSERT INTO person VALUES (1), (10), (10);
CREATE UNIQUE INDEX idx ON person (pid) WHERE pid=1;
SELECT DISTINCT pid FROM person WHERE pid=10;
```



pid	idx
1	←
10	
10	

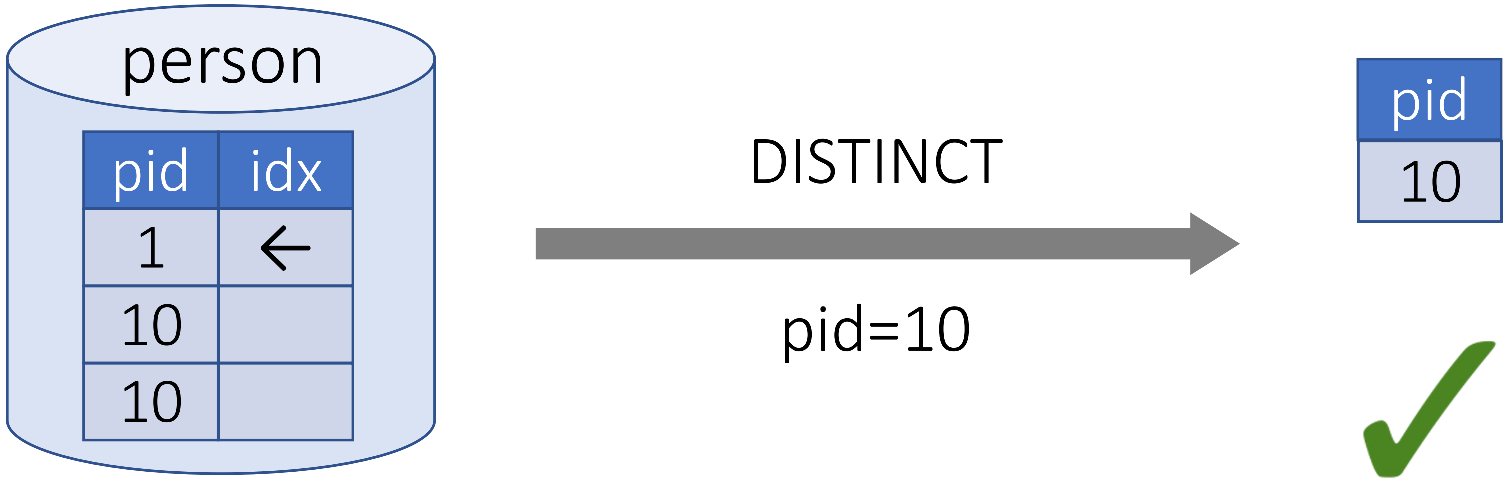
Motivating Example (SQLite)

```
CREATE TABLE person (pid INT);
INSERT INTO person VALUES (1), (10), (10);
CREATE UNIQUE INDEX idx ON person (pid) WHERE pid=1;
SELECT DISTINCT pid FROM person WHERE pid=10;
```



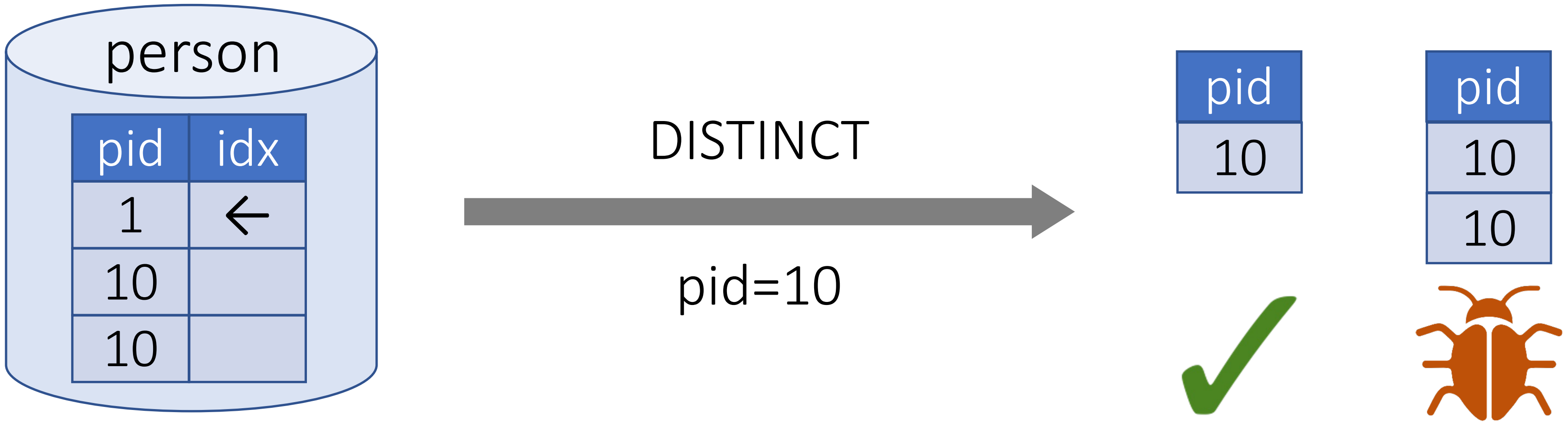
Motivating Example (SQLite)

```sql
CREATE TABLE person (pid INT);
INSERT INTO person VALUES (1), (10), (10);
CREATE UNIQUE INDEX idx ON person (pid) WHERE pid=1;
SELECT DISTINCT pid FROM person WHERE pid=10;
```



Motivating Example (SQLite)

```
CREATE TABLE person (pid INT);
INSERT INTO person VALUES (1), (10), (10);
CREATE UNIQUE INDEX idx ON person (pid) WHERE pid=1;
SELECT DISTINCT pid FROM person WHERE pid=10;
```



Challenges to Detect Logical Bugs

CHALLENGE 1

Challenges to Detect Logical Bugs

CHALLENGE 1

CHALLENGE 1

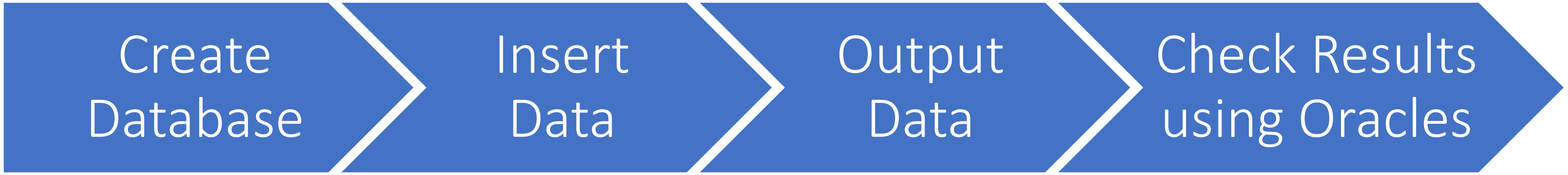
CHALLENGE 1

- Generating valid queries

Challenges to Detect Logical Bugs

CHALLENGE 1

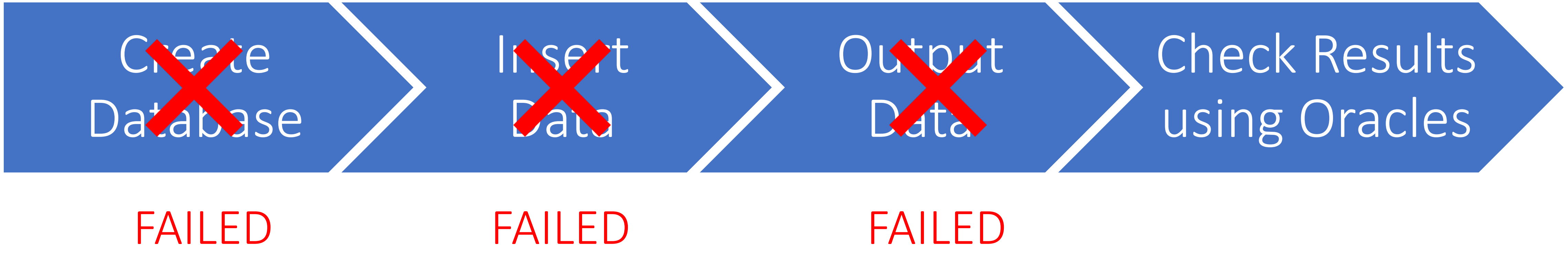
- Generating valid queries



Challenges to Detect Logical Bugs

CHALLENGE 1

- Generating valid queries
    - invalid queries cannot trigger logical bugs



Challenges to Detect Logical Bugs

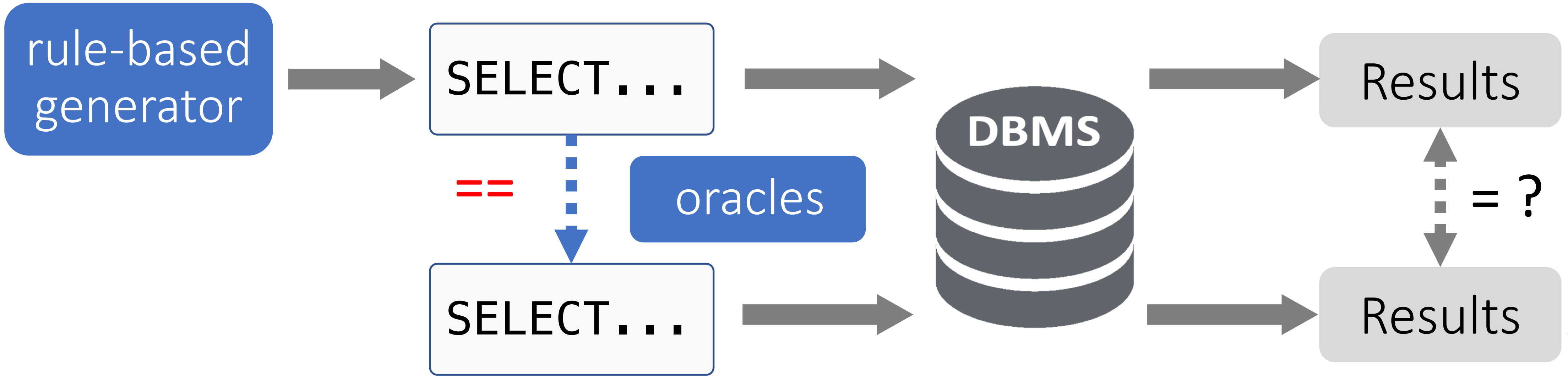
CHALLENGE 2

- Implementing DBMS oracles
  - no platform for easy oracle development
  - no easy integration with existing techniques

Challenges to Detect Logical Bugs

CHALLENGE 2

- Implementing DBMS oracles
  - no platform for easy oracle development
  - no easy integration with existing techniques
- SQLancer: non-trivial manual efforts



Validity-oriented Query Mutation

SOLUTION 1

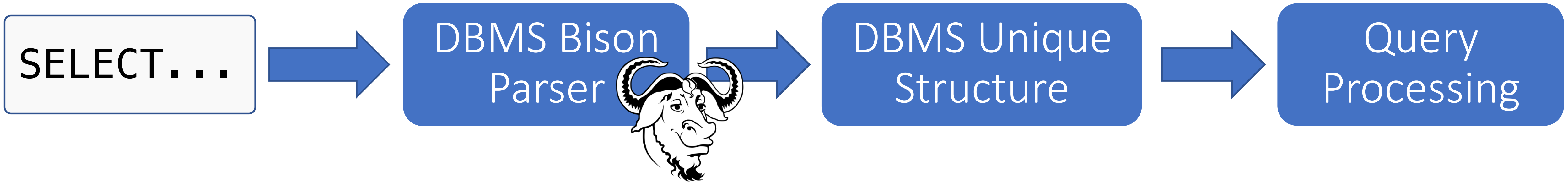
Validity-oriented Query Mutation

SOLUTION 1
SOLUTION 1
SOLUTION 1

- Dedicated Parsing

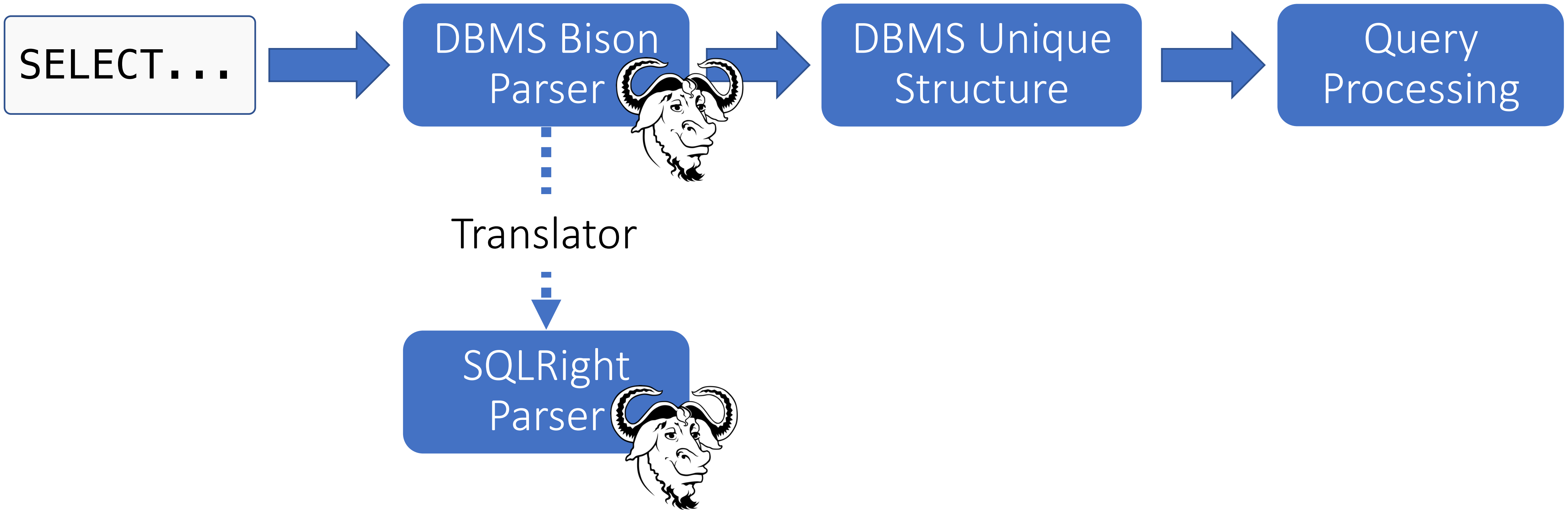
Validity-oriented Query Mutation

- Dedicated Parsing



Validity-oriented Query Mutation

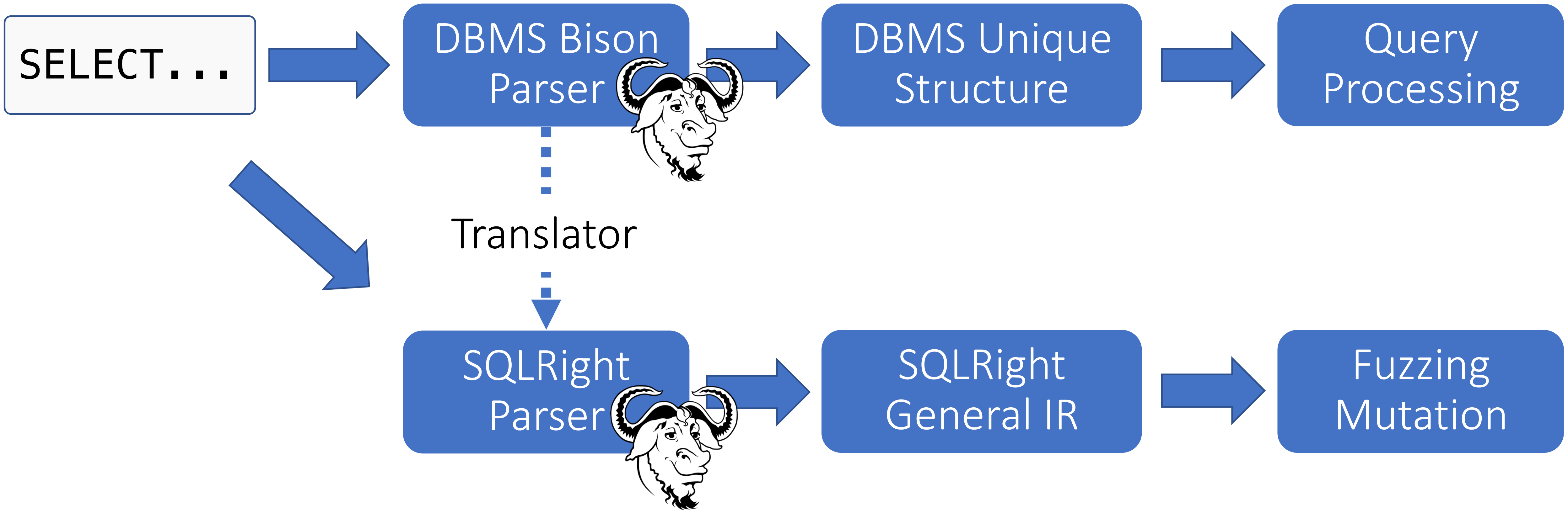
- Dedicated Parsing



Validity-oriented Query Mutation

SOLUTION 1

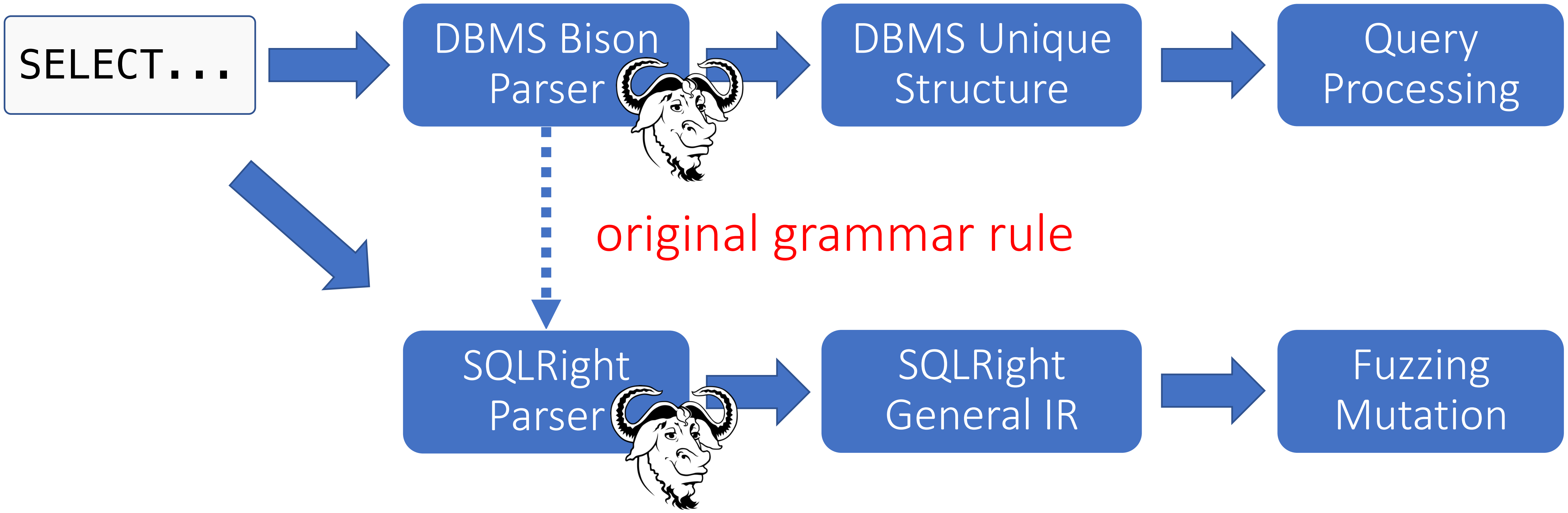
- Dedicated Parsing



Validity-oriented Query Mutation

SOLUTION 1

- Dedicated Parsing



Validity-oriented Query Mutation

SOLUTION 1

```
CREATE TABLE  x  ( x  INT,  x  INT,  x  INT);
INSERT INTO  x  VALUES (x), ( x ), ( x );
ALTER TABLE  x  RENAME  x  TO  x  ;
SELECT x FROM  x  WHERE  x  =  x  ;
```

- Context-based IR Instantiation
  - fill in concrete query operands

Validity-oriented Query Mutation

SOLUTION 1

```
CREATE TABLE v0 (c1 INT, c2 INT, c3 INT);
INSERT INTO v0 VALUES (0), (10), (10);
ALTER TABLE v0 RENAME c3 TO c4 ;
SELECT * FROM v0 WHERE c1 = c4 ;
```

- Context-based IR Instantiation
  - fill in concrete query operands

Validity-oriented Query Mutation

SOLUTION 1

```
CREATE TABLE v0 (c1 INT, c2 INT, c3 INT);
INSERT INTO v0 VALUES (0), (10), (10);
ALTER TABLE v0 RENAME c3 TO c4 ;
SELECT * FROM v0 WHERE c1 = c4 ;
```

rename c3 to c4

- Context-based IR Instantiation
  - fill in concrete query operands

Validity-oriented Query Mutation

SOLUTION 1

```
CREATE TABLE v0 ( c1 INT, c2 INT, c3 INT);
INSERT INTO v0 VALUES (0), (10), (10);
ALTER TABLE v0 RENAME c3 TO c4 ;
SELECT * FROM v0 WHERE c1 = c4 ;
```

rename c3 to c4

use c4 not c3

- Context-based IR Instantiation
  - fill in concrete query operands

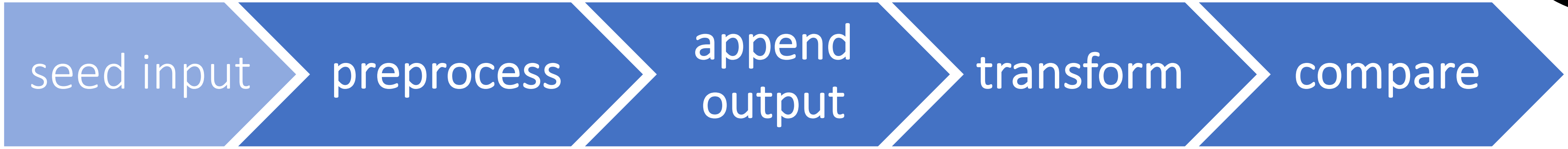
Validity-oriented Query Mutation

SOLUTION 1

- Two other techniques (details in paper)
  - cooperative mutation
  - non-deterministic behaviors removal

General Interfaces for DBMS Oracles

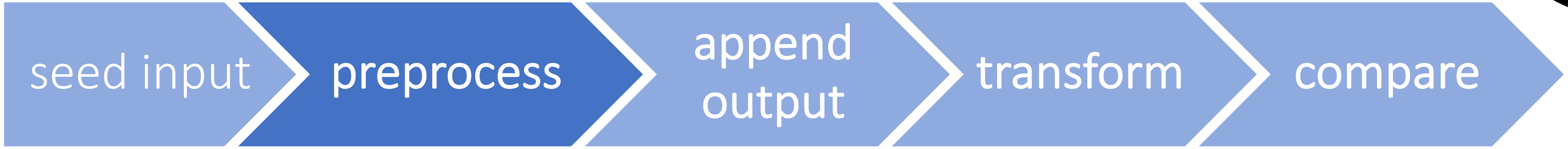
SOLUTION 2



- Easy development for new oracles
- Four general APIs

General Interfaces for DBMS Oracles

SOLUTION 2

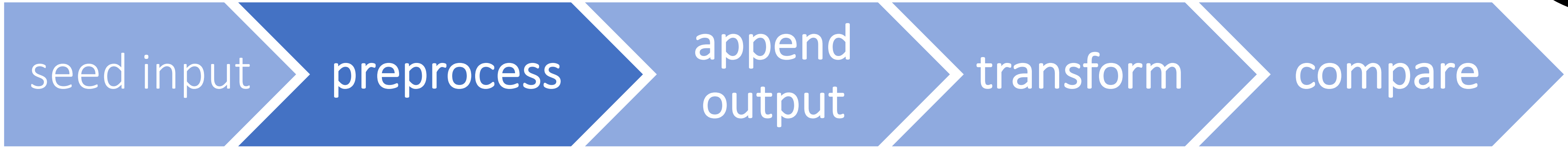


- remove improper queries

```
CREATE TABLE person (pid INT);
INSERT INTO person VALUES (1), (10), (10);
CREATE UNIQUE INDEX idx ON person (pid) WHERE pid=1;
INSERT INTO person VALUES (RANDOM());
```

General Interfaces for DBMS Oracles

SOLUTION 2



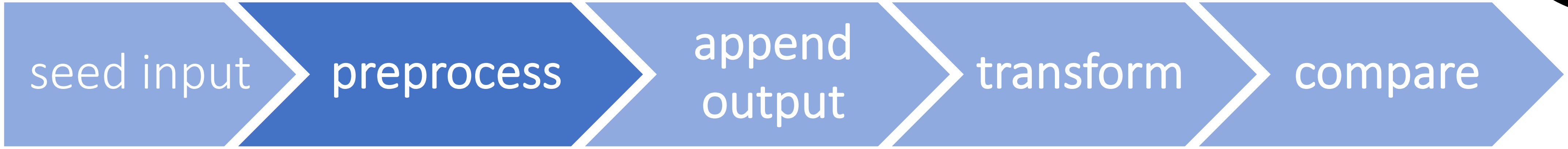
- remove improper queries

```
CREATE TABLE person (pid INT);
INSERT INTO person VALUES (1), (10), (10);
CREATE UNIQUE INDEX idx ON person (pid) WHERE pid=1;
INSERT INTO person VALUES (RANDOM());
```

random results

General Interfaces for DBMS Oracles

SOLUTION 2



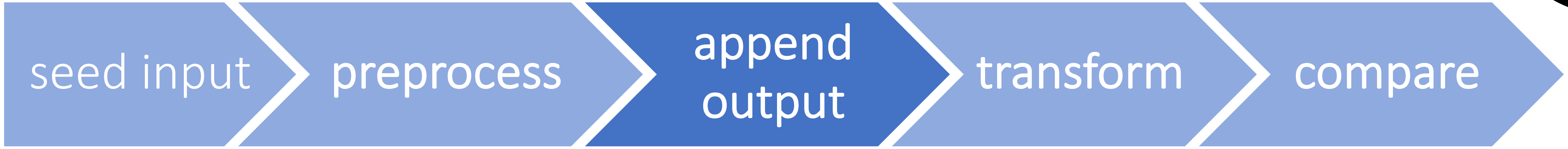
- remove improper queries

```
CREATE TABLE person (pid INT);
INSERT INTO person VALUES (1), (10), (10);
CREATE UNIQUE INDEX idx ON person (pid) WHERE pid=1;
```
– ~~INSERT INTO person VALUES (RANDOM());~~

random results

General Interfaces for DBMS Oracles

SOLUTION 2



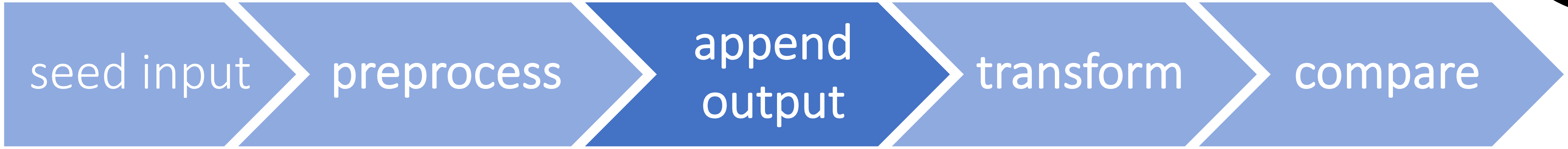
- append oracle-compatible SELECT statements

```
  CREATE TABLE person (pid INT);
  INSERT INTO person VALUES (1), (10), (10);
  CREATE UNIQUE INDEX idx ON person (pid) WHERE pid=1;
− ~~INSERT INTO person VALUES (RANDOM());~~
+ SELECT DISTINCT COUNT(*) FROM person WHERE pid=10;
```

NoREC oracle

General Interfaces for DBMS Oracles

SOLUTION 2



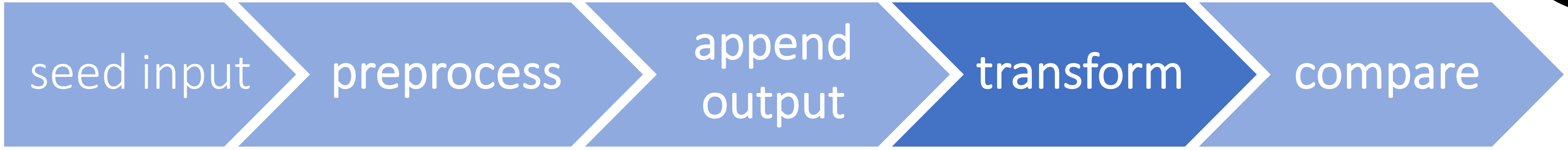
- append oracle-compatible SELECT statements

```
  CREATE TABLE person (pid INT);
  INSERT INTO person VALUES (1), (10), (10);
  CREATE UNIQUE INDEX idx ON person (pid) WHERE pid=1;
- ~~INSERT INTO person VALUES (RANDOM());~~
+ SELECT DISTINCT COUNT(*) FROM person WHERE pid=10;
```

NoREC oracle

General Interfaces for DBMS Oracles

SOLUTION 2



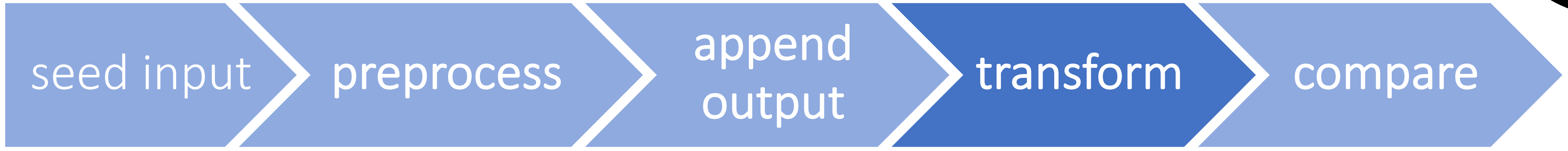
- transform query to functional equivalent forms

```
  CREATE TABLE person (pid INT);
  INSERT INTO person VALUES (1), (10), (10);
  CREATE UNIQUE INDEX idx ON person (pid) WHERE pid=1;
− ~~INSERT INTO person VALUES (RANDOM());~~
+ SELECT DISTINCT COUNT(*) FROM person WHERE pid=10;
+ SELECT DISTINCT pid=10 FROM person;
```

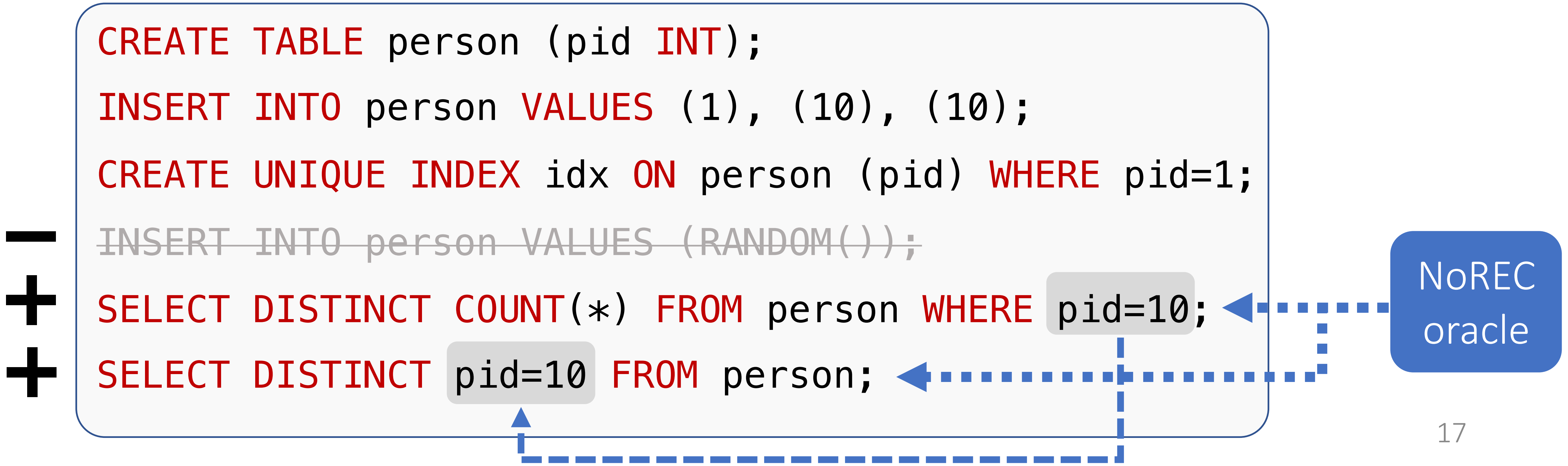
NoREC oracle

General Interfaces for DBMS Oracles

SOLUTION 2



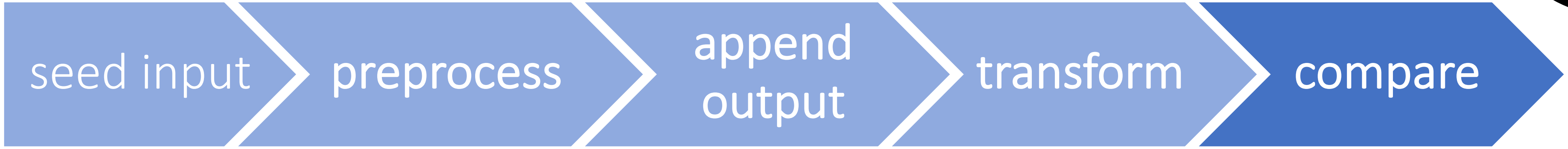
- transform query to functional equivalent forms



```
  CREATE TABLE person (pid INT);
  INSERT INTO person VALUES (1), (10), (10);
  CREATE UNIQUE INDEX idx ON person (pid) WHERE pid=1;
− INSERT INTO person VALUES (RANDOM());
+ SELECT DISTINCT COUNT(*) FROM person WHERE pid=10;
+ SELECT DISTINCT pid=10 FROM person;
```

General Interfaces for DBMS Oracles

SOLUTION 2

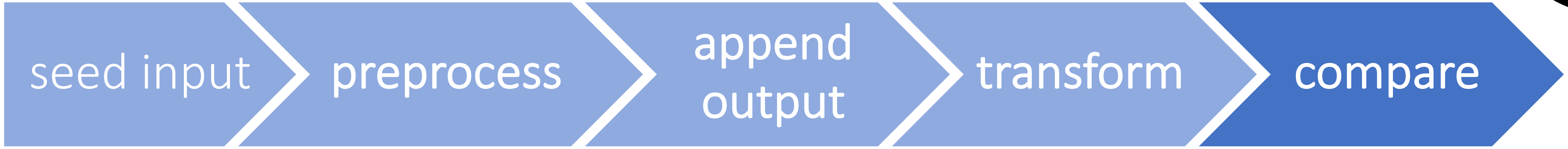


- comparison method to identify unexpected result

```
  CREATE TABLE person (pid INT);
  INSERT INTO person VALUES (1), (10), (10);
  CREATE UNIQUE INDEX idx ON person (pid) WHERE pid=1;
− ~~INSERT INTO person VALUES (RANDOM());~~
+ SELECT DISTINCT COUNT(*) FROM person WHERE pid=10;   ← res: {1}
+ SELECT DISTINCT pid=10 FROM person;                  ← res: {0, 1}
```

General Interfaces for DBMS Oracles

SOLUTION 2



- comparison method to identify unexpected result

```
  CREATE TABLE person (pid INT);
  INSERT INTO person VALUES (1), (10), (10);
  CREATE UNIQUE INDEX idx ON person (pid) WHERE pid=1;
− ~~INSERT INTO person VALUES (RANDOM());~~
+ SELECT DISTINCT COUNT(*) FROM person WHERE pid=10;
+ SELECT DISTINCT pid=10 FROM person;
```

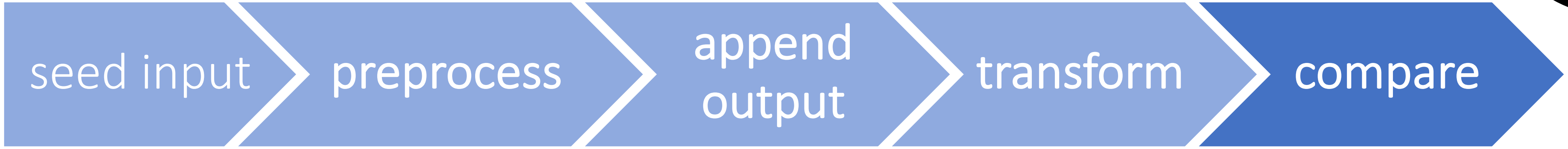
matched

res: {1}

res: {0, 1}

General Interfaces for DBMS Oracles

SOLUTION 2

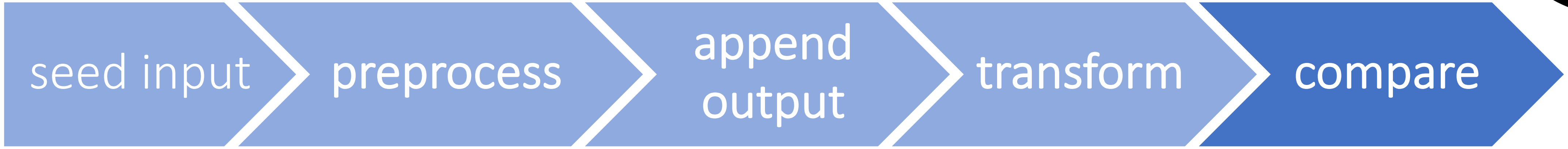


- comparison method to identify unexpected result

```
  CREATE TABLE person (pid INT);
  INSERT INTO person VALUES (1), (10), (10);
  CREATE UNIQUE INDEX idx ON person (pid) WHERE pid=1;
− ~~INSERT INTO person VALUES (RANDOM());~~
+ SELECT DISTINCT COUNT(*) FROM person WHERE pid=10;   ← res: {2}
+ SELECT DISTINCT pid=10 FROM person;                  ← res: {0, 1}
```

General Interfaces for DBMS Oracles

SOLUTION 2

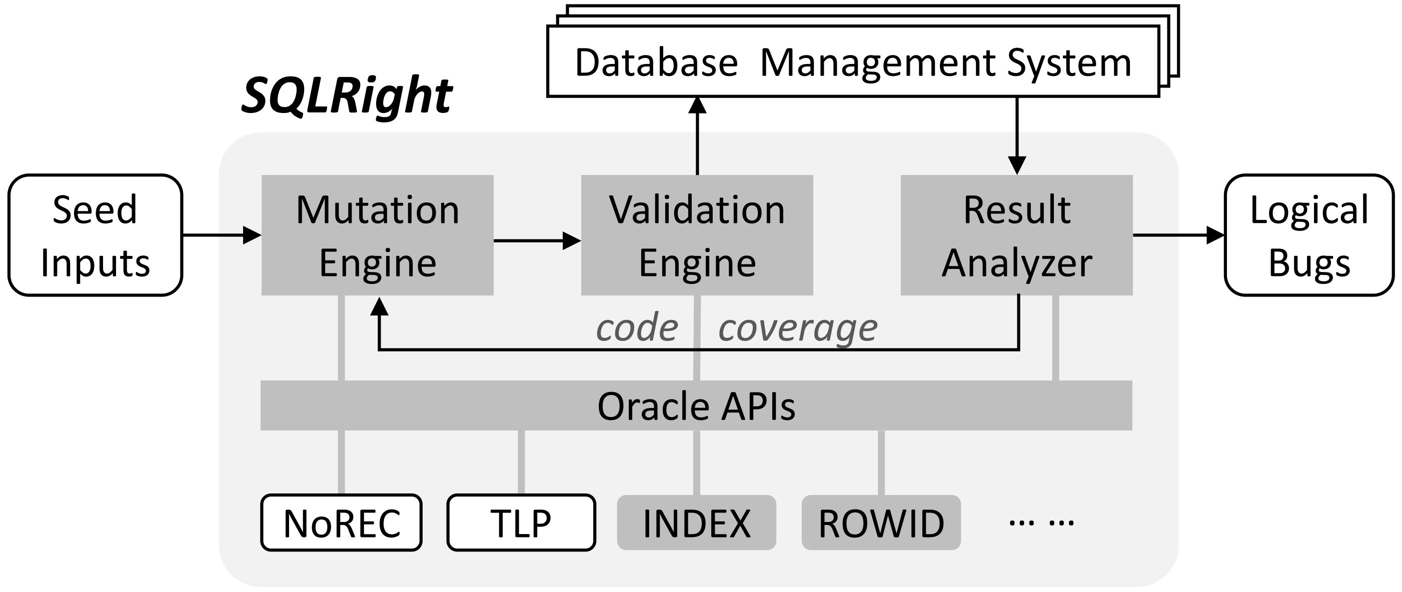


- comparison method to identify unexpected result

```
  CREATE TABLE person (pid INT);
  INSERT INTO person VALUES (1), (10), (10);
  CREATE UNIQUE INDEX idx ON person (pid) WHERE pid=1;
− ~~INSERT INTO person VALUES (RANDOM());~~
+ SELECT DISTINCT COUNT(*) FROM person WHERE pid=10;   ◀ res: {2}
+ SELECT DISTINCT pid=10 FROM person;                  ◀ res: {0, 1}
```

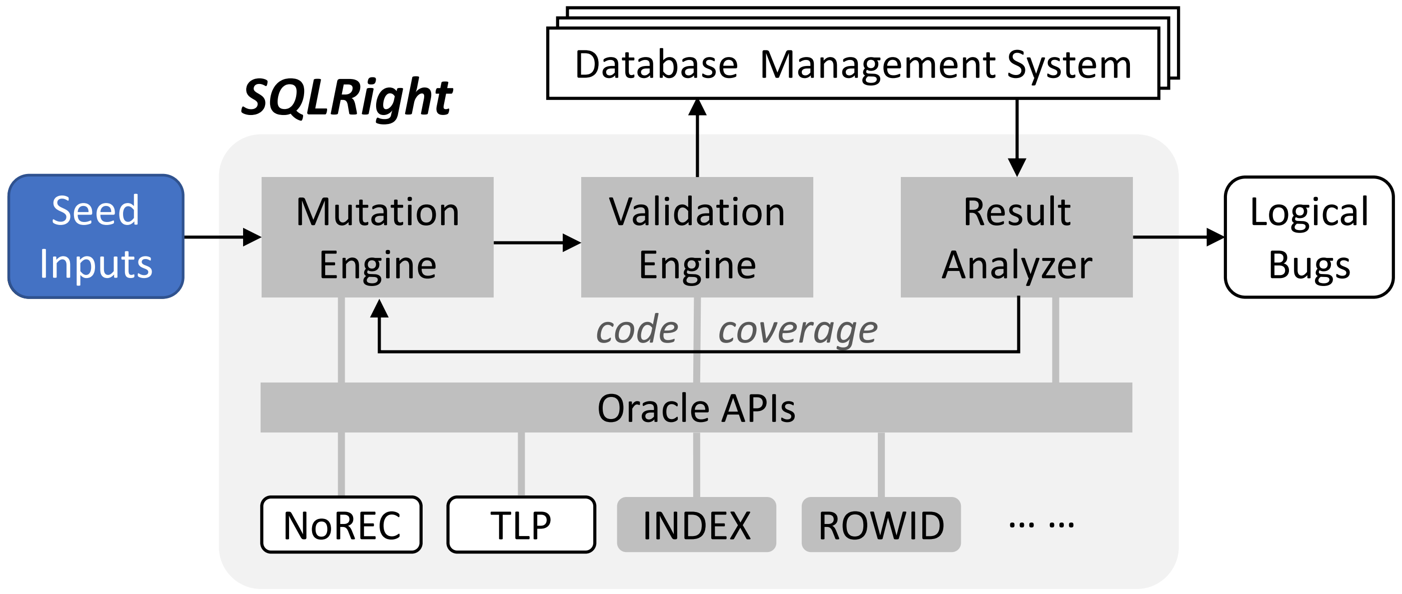
not matched

SQLRight Overview



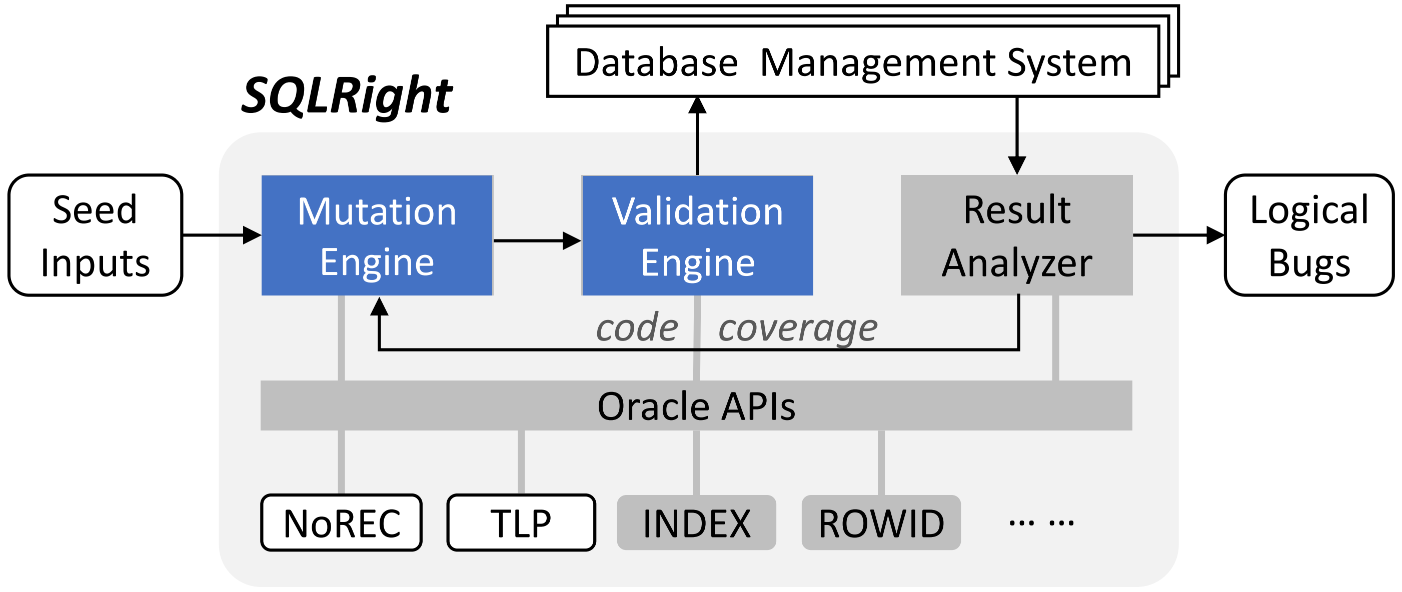
- Coverage-guided fuzzer
- Validity-oriented mutation
- General interfaces for oracles

SQLRight Overview



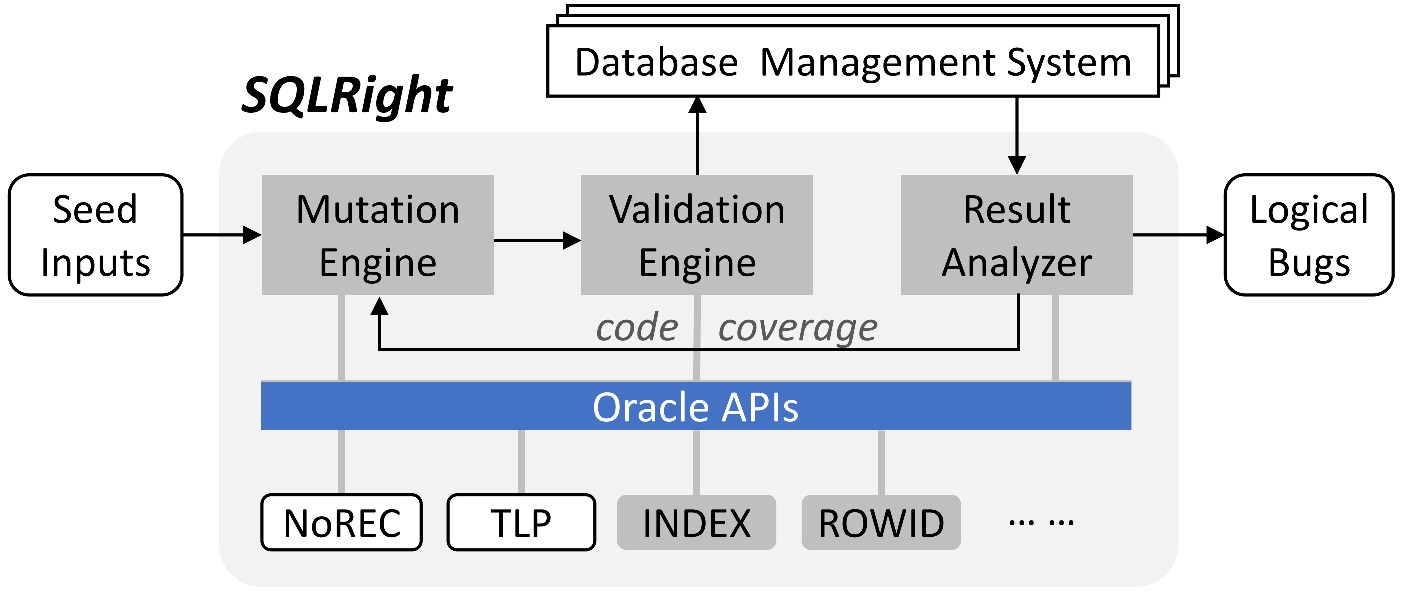
- Coverage-guided fuzzer
- Validity-oriented mutation
- General interfaces for oracles

SQLRight Overview



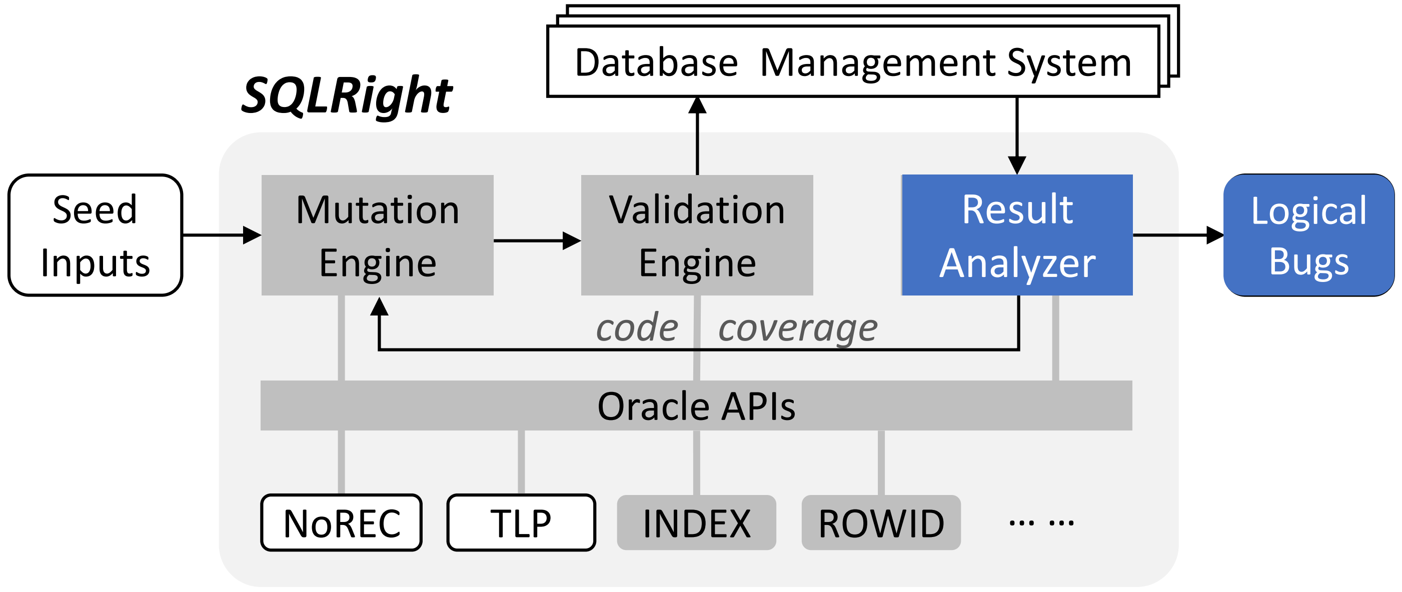
- Coverage-guided fuzzer
- Validity-oriented mutation
- General interfaces for oracles

SQLRight Overview



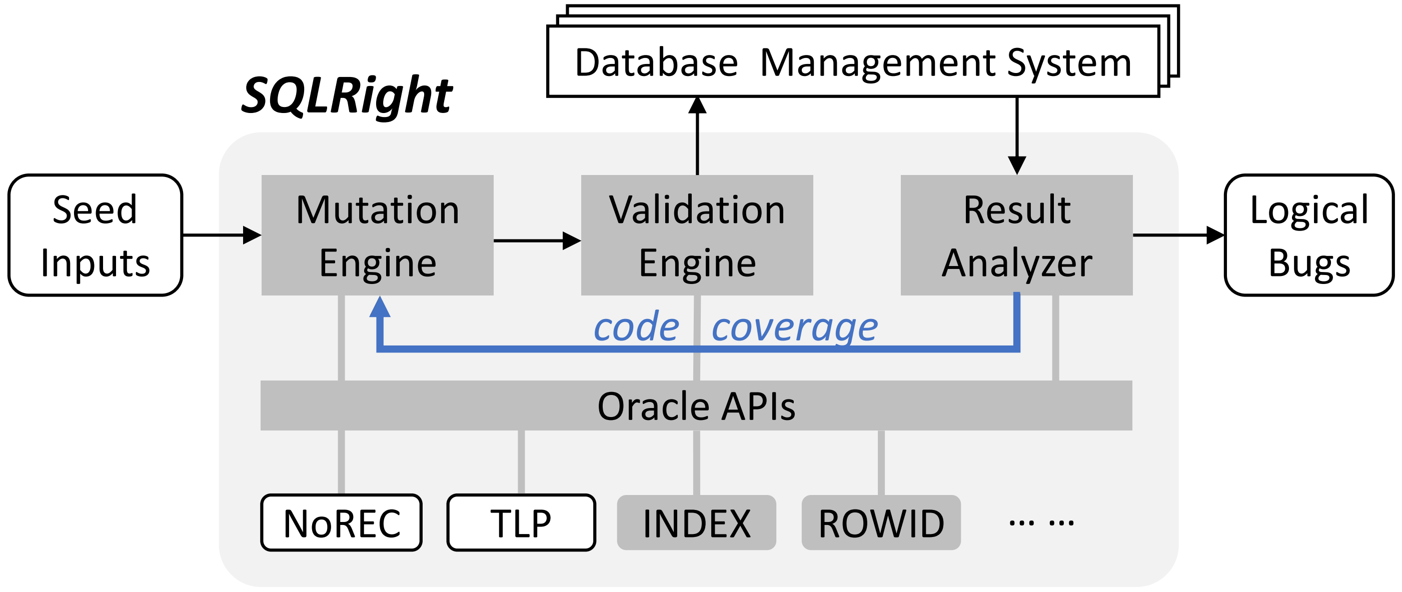
- Coverage-guided fuzzer
- Validity-oriented mutation
- General interfaces for oracles

SQLRight Overview



- Coverage-guided fuzzer
- Validity-oriented mutation
- General interfaces for oracles

SQLRight Overview



- Coverage-guided fuzzer
- Validity-oriented mutation
- General interfaces for oracles

Evaluation

Evaluation

- Can SQLRight detect real-world logical bugs?

Evaluation

- Can SQLRight detect real-world logical bugs?
- Can SQLRight find more bugs than existing tools?

Evaluation

- Can SQLRight detect real-world logical bugs?
- Can SQLRight find more bugs than existing tools?
- Contribution of different SQLRight components?

Evaluation

- Can SQLRight detect real-world logical bugs?
- Can SQLRight find more bugs than existing tools?
- Contribution of different SQLRight components?

Detect Real-world Logical bugs

Oracle \ DBMS	SQLite	MySQL	PostgreSQL	Total
NoREC	11	3	0	14
TLP	1	1	0	2
ROWID	1	0	0	1
INDEX	1	0	0	1
TOTAL	14	4	0	18

Detect Real-world Logical bugs

Oracle \ DBMS	SQLite	MySQL	PostgreSQL	Total
NoREC	11	3	0	14
TLP	1	1	0	2
ROWID	1	0	0	1
INDEX	1	0	0	1
TOTAL	14	4	0	18

- 18 logical bugs
  - 14 SQLite
  - 4 MySQL

Detect Real-world Logical bugs

DBMS / Oracle	SQLite	MySQL	PostgreSQL	Total
NoREC	11	3	0	14
TLP	1	1	0	2
ROWID	1	0	0	1
INDEX	1	0	0	1
TOTAL	14	4	0	$\boxed{18}$

- 18 logical bugs
    - 14 SQLite
    - 4 MySQL
- 15 bugs fixed

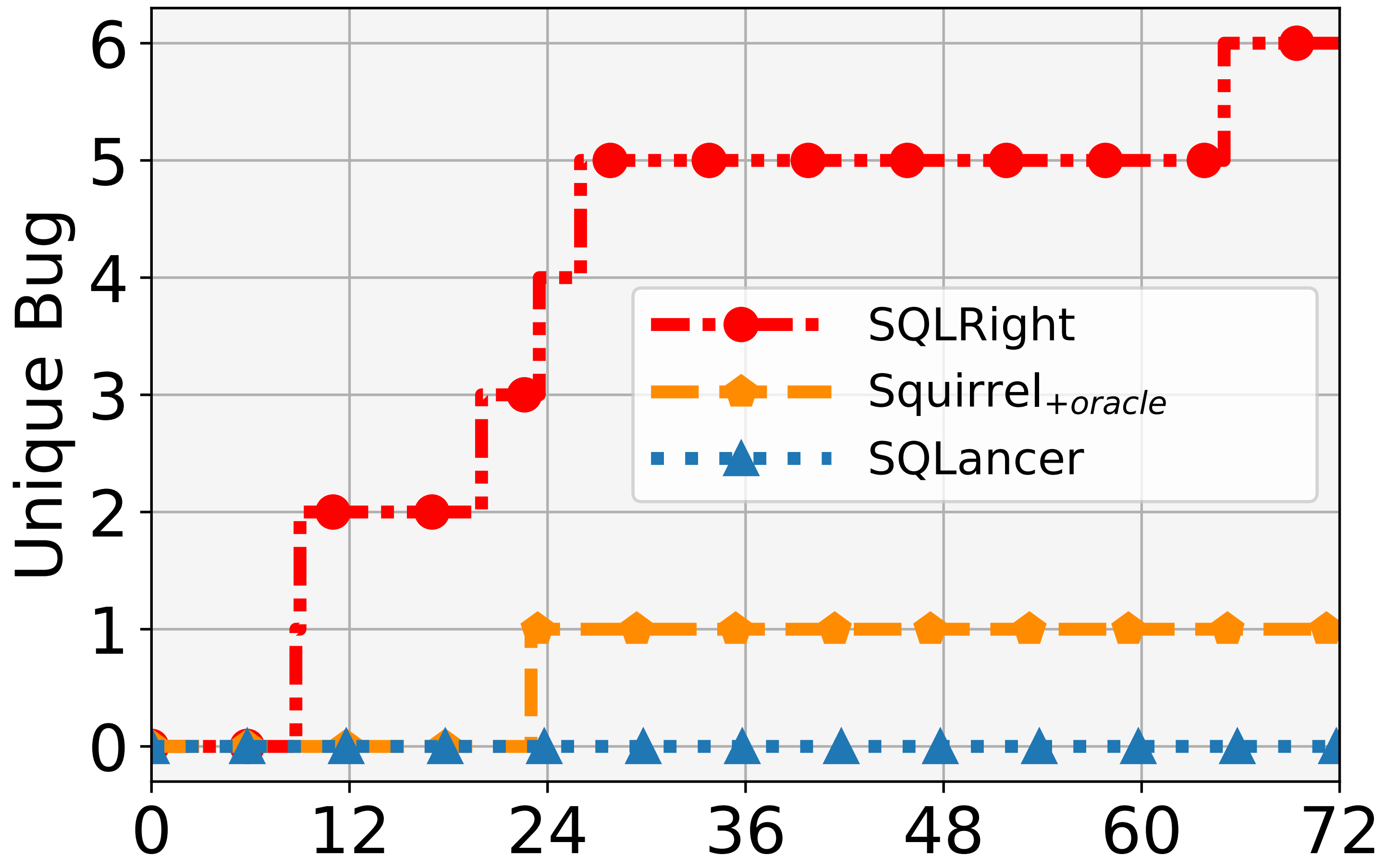
Detect Real-world Logical bugs

Oracle \ DBMS	SQLite	MySQL	PostgreSQL	Total
NoREC	11	3	0	14
TLP	1	1	0	2
ROWID	1	0	0	1
INDEX	1	0	0	1
TOTAL	14	4	0	18

- 18 logical bugs
  - 14 SQLite
  - 4 MySQL
- 15 bugs fixed
- 2 from new oracles

Comparison regarding ***Detected Bugs*** (NoREC)

- SQLite



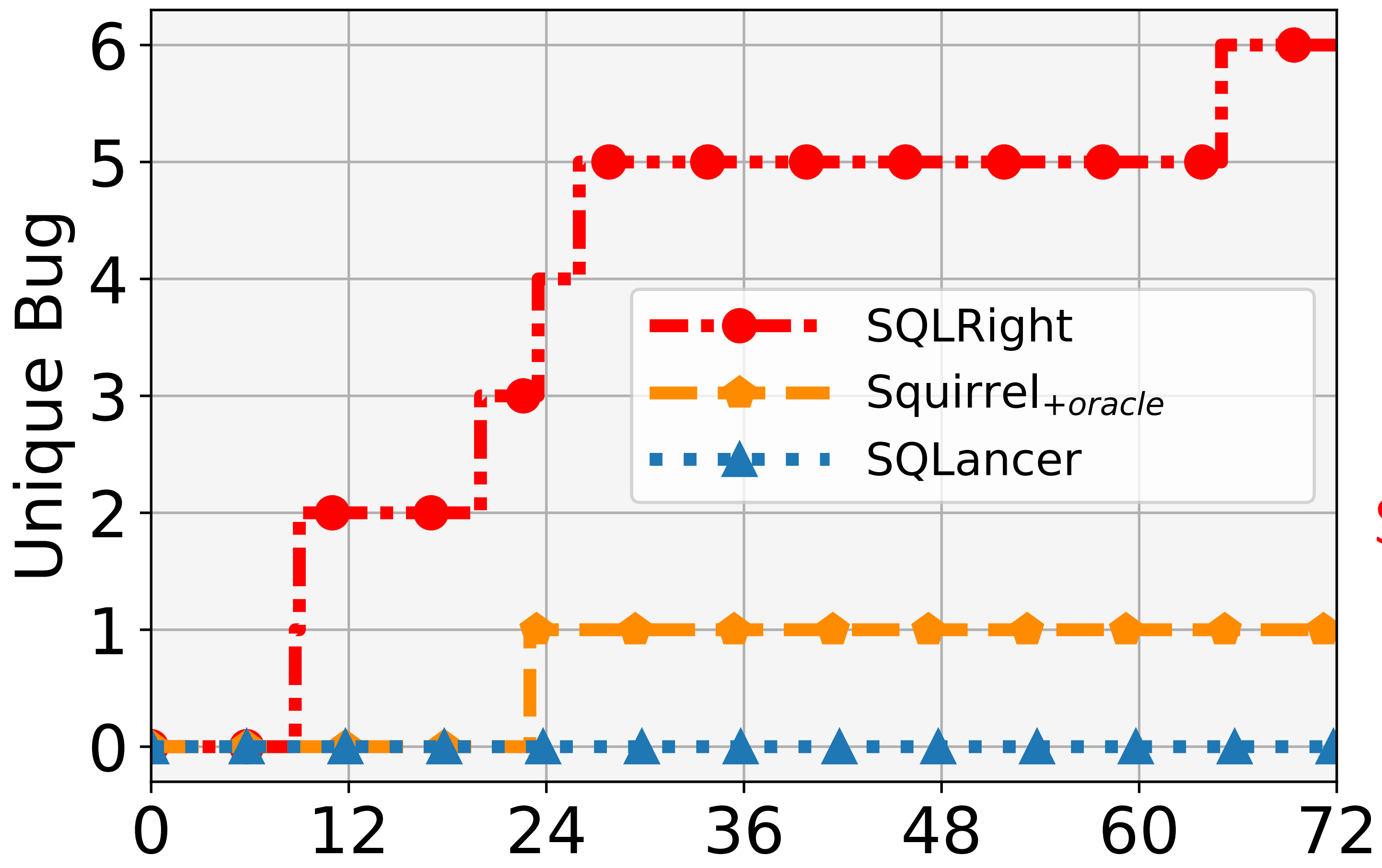
SQLRight: 6 bugs

Squirrel$_{+oracle}$: 1 bug

SQLancer: 0 bug

Comparison regarding ***Detected Bugs*** (NoREC)

- SQLite



SQLRight: 6 bugs

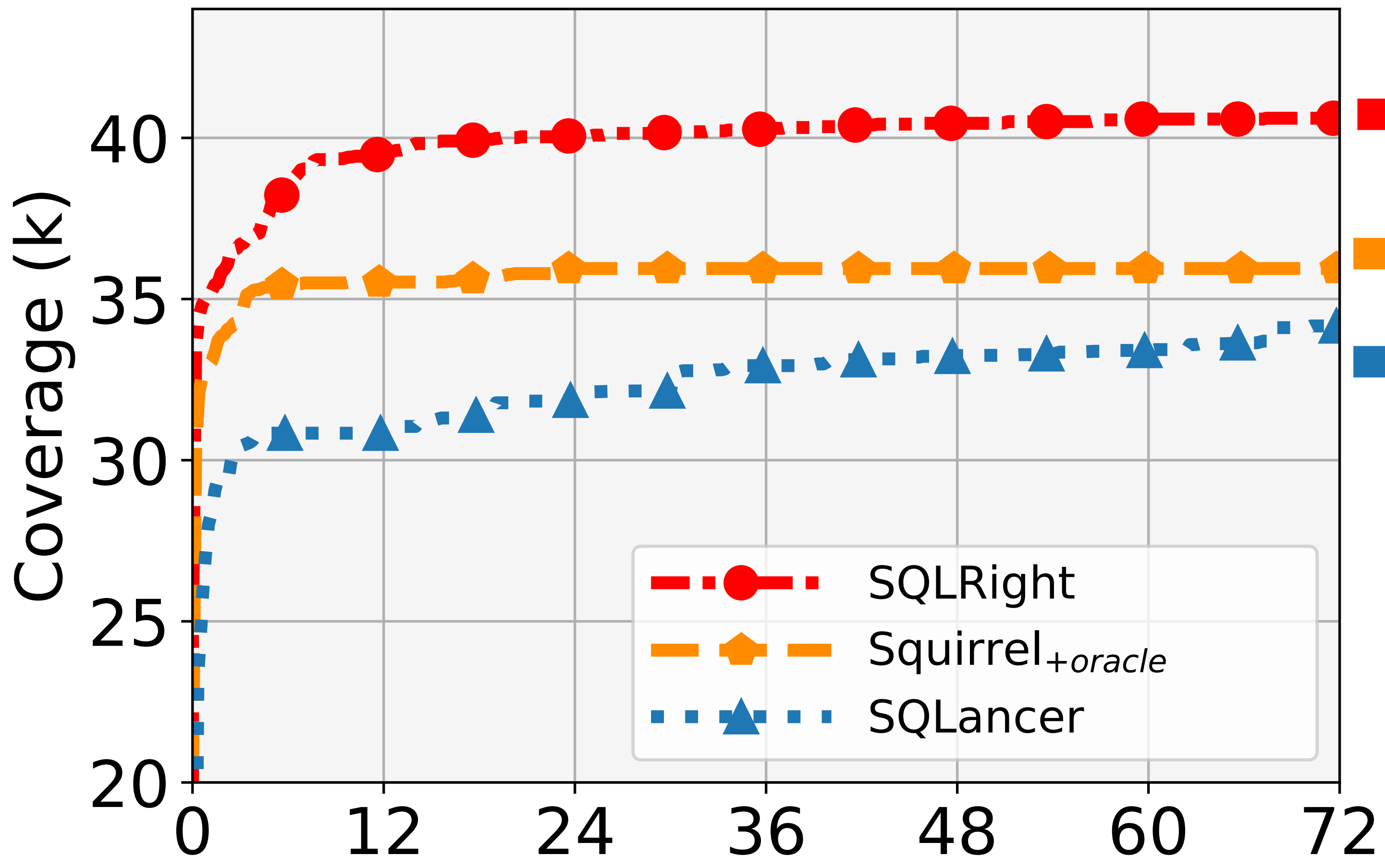
Squirrel: mem corrupt detector

Squirrel$_{+oracle}$: 1 bug

SQLancer: 0 bug

Comparison regarding ***Branch Coverage*** (NoREC)

- SQLite



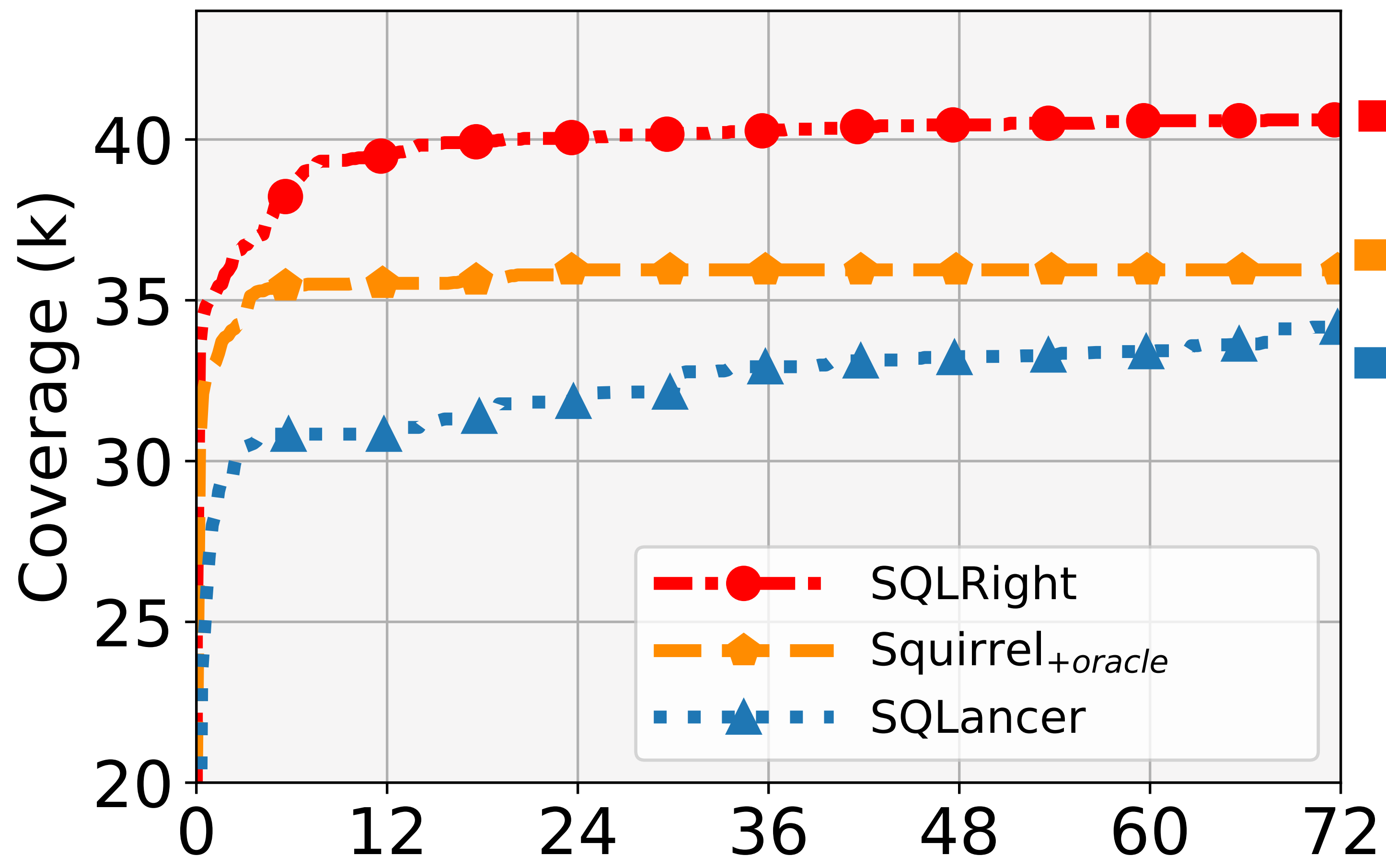
SQLRight: 40.6K

Squirrel$_{+oracle}$: 35.9K

SQLancer: 34.2K

Comparison regarding ***Branch Coverage*** (NoREC)

- SQLite



SQLRight: 40.6K

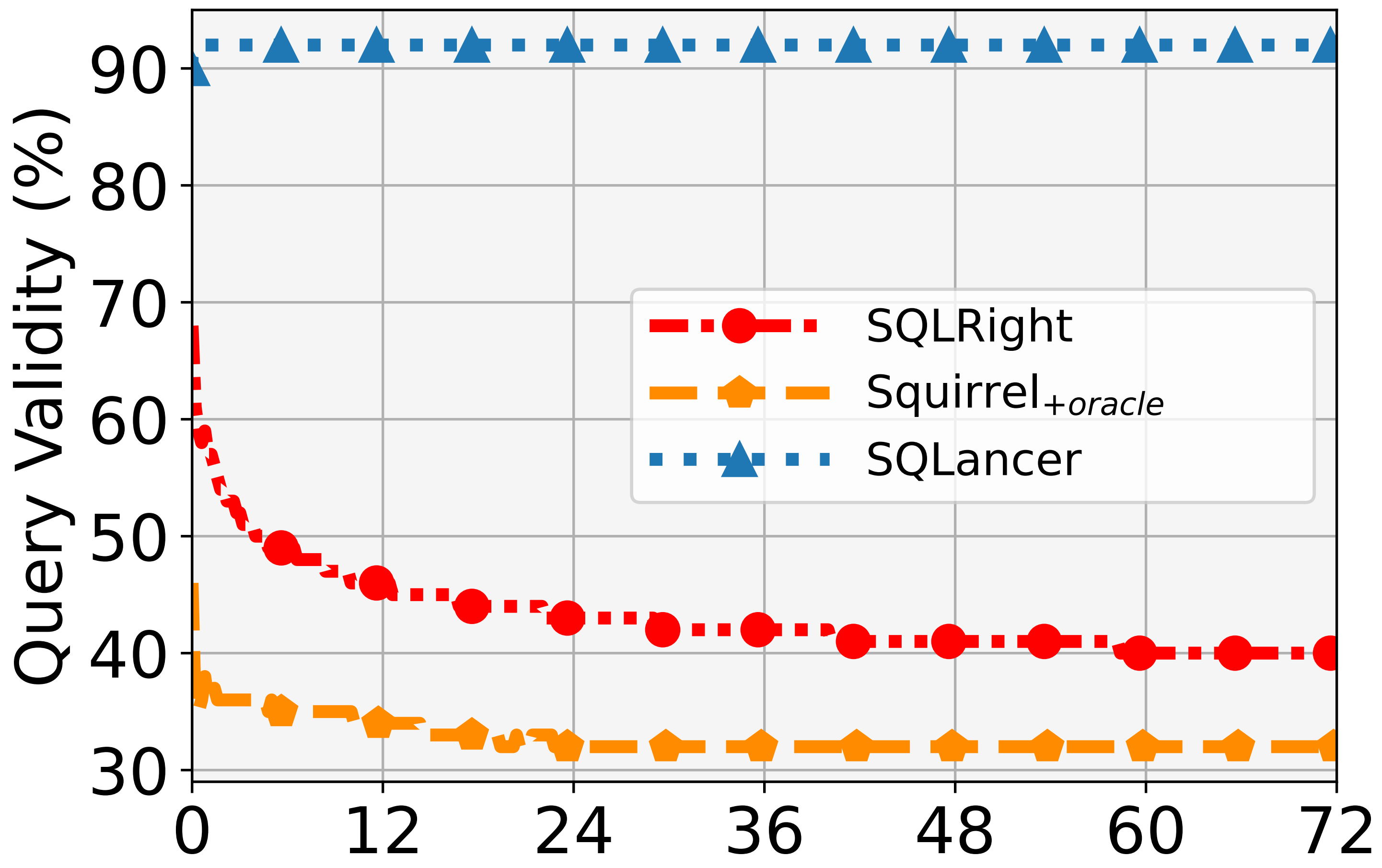
Squirrel$_{+oracle}$: 35.9K

SQLancer: 34.2K

12.97% more coverage than best baseline

Comparison regarding *Query Validity* (NoREC)

- SQLite



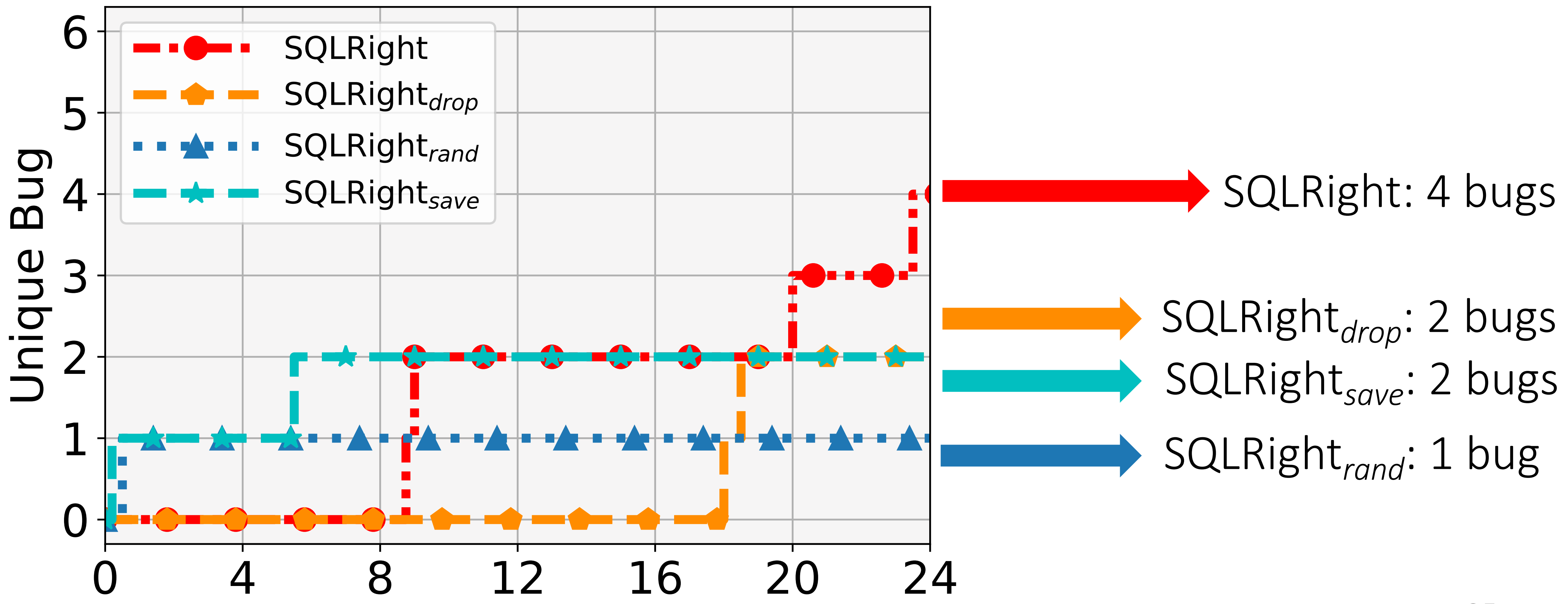
SQLancer: 92.0%
rule-based

SQLRight: 40.0%

Squirrel$_{+oracle}$: 32.0%

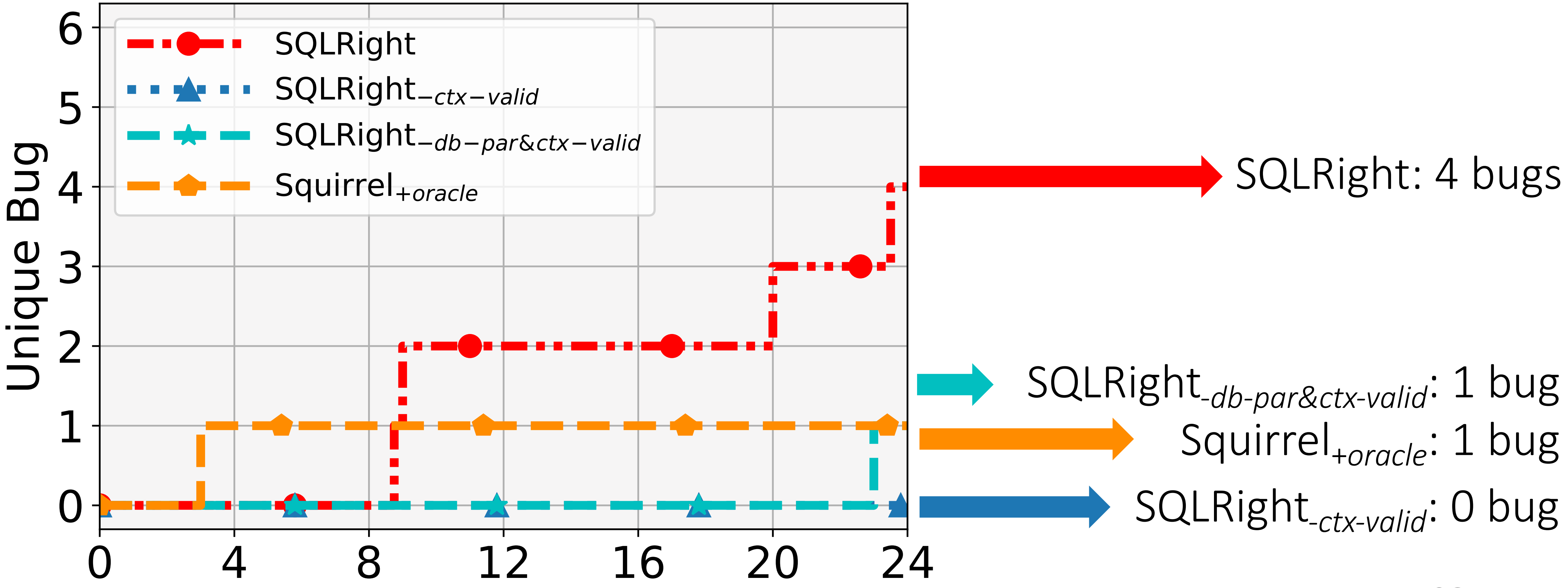
Contribution of *Coverage Feedback* (NoREC)

- SQLite

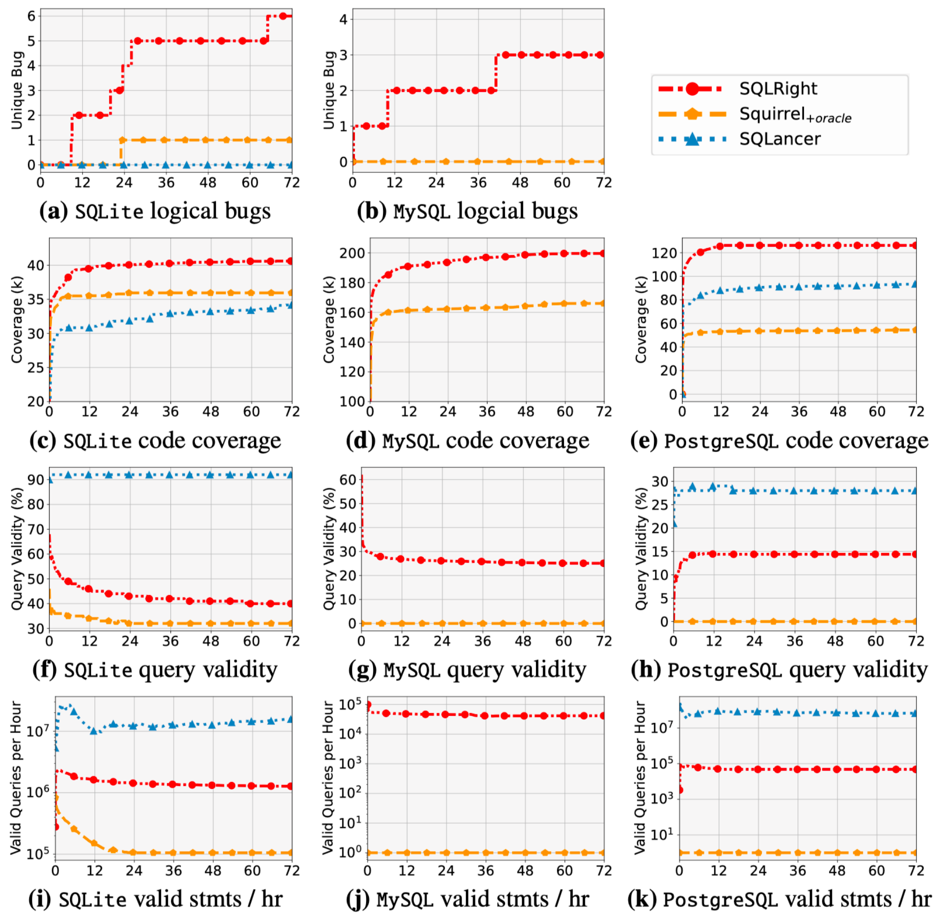


Contribution of *Validity* (NoREC)

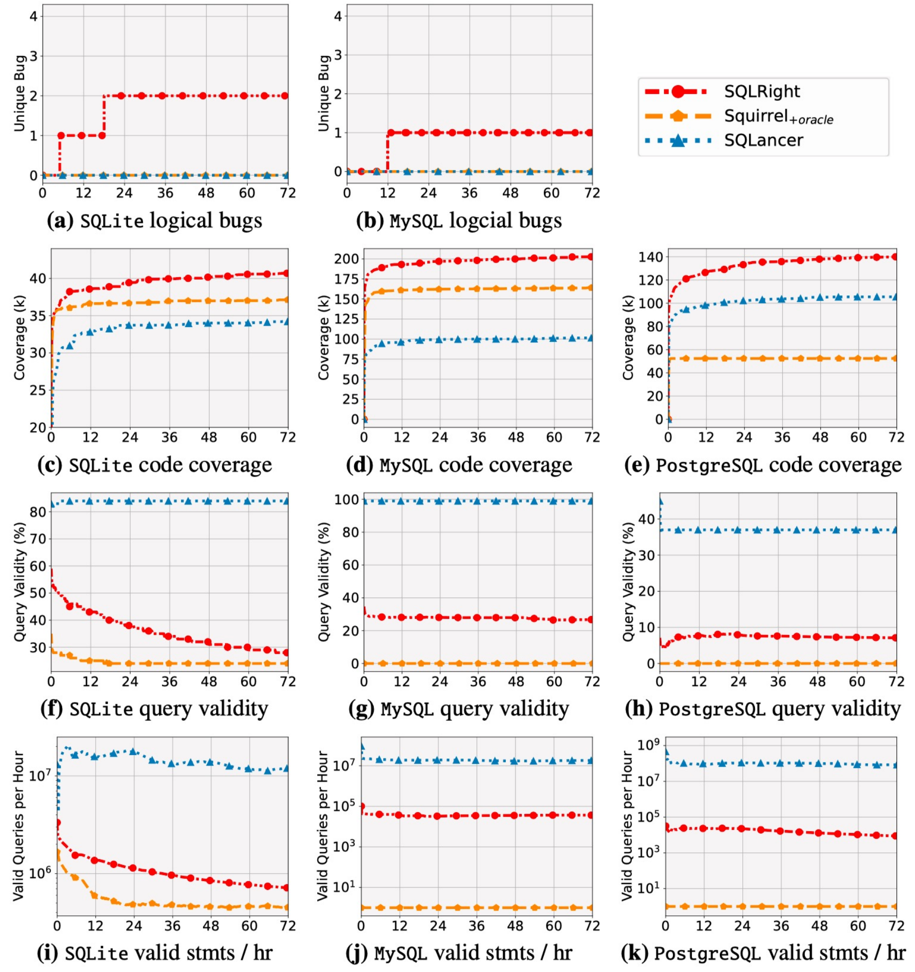
- SQLite



More Evaluations in the Paper



NoREC



TLP

Conclusion

- SQLRight: a general platform to test DBMS logical bugs
  - coverage-guided fuzzing
  - validity-oriented mutation
  - general interfaces for DBMS oracles

Conclusion

- SQLRight: a general platform to test DBMS logical bugs
  - coverage-guided fuzzing
  - validity-oriented mutation
  - general interfaces for DBMS oracles
- Found 18 logical bugs in SQLite and MySQL

Conclusion

- SQLRight: a general platform to test DBMS logical bugs
    - coverage-guided fuzzing
    - validity-oriented mutation
    - general interfaces for DBMS oracles
- Found 18 logical bugs in SQLite and MySQL
- https://github.com/psu-security-universe/sqlright

Thank You

Question?

yuliang@psu.edu