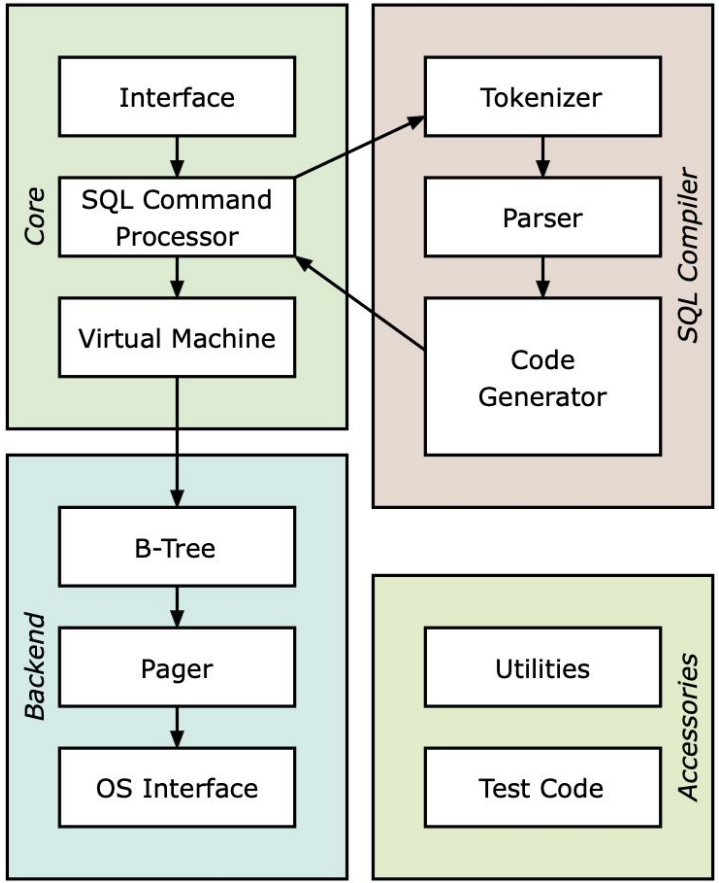
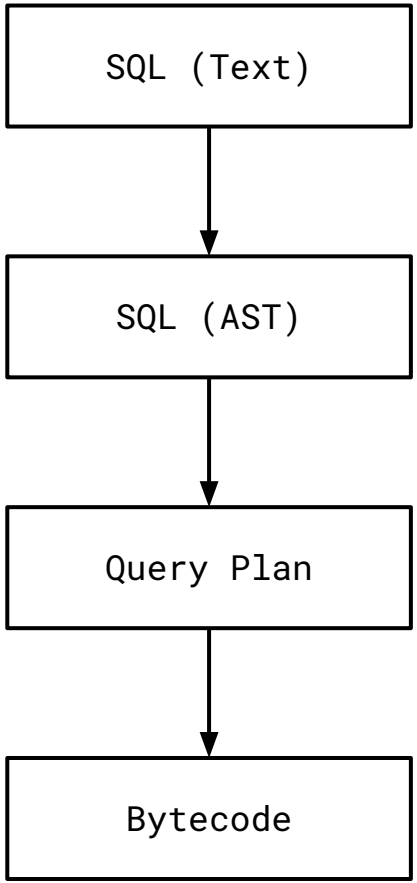
The Art of Database Testing

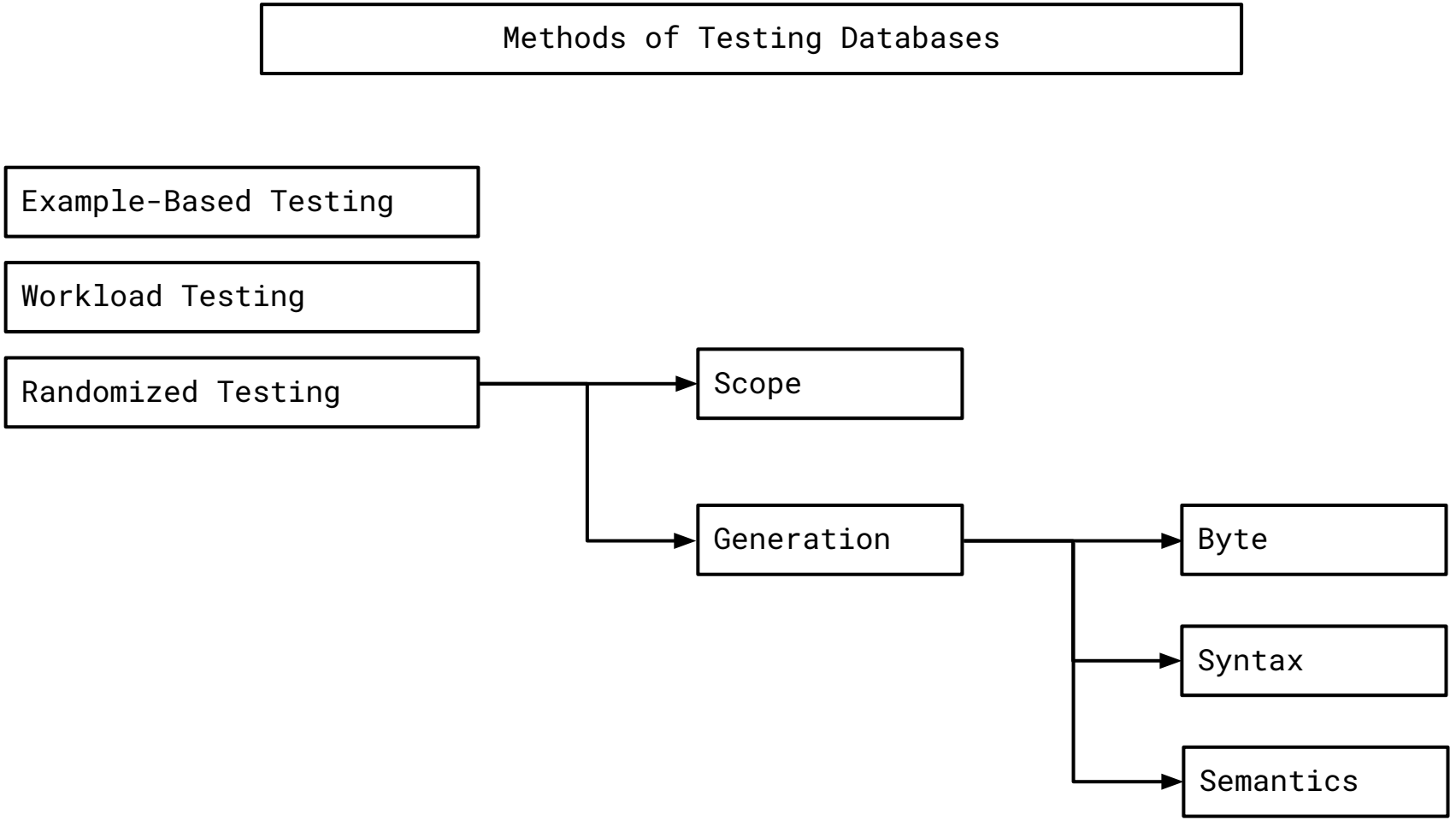
Alperen Keles - Ph.D Candidate at UMD

DC Systems — Oct 14, 2025

What's a database?






Methods of Testing Databases
Example-Based Testing
Workload Testing
Randomized Testing
Scope
Generation
Byte
Syntax
Semantics

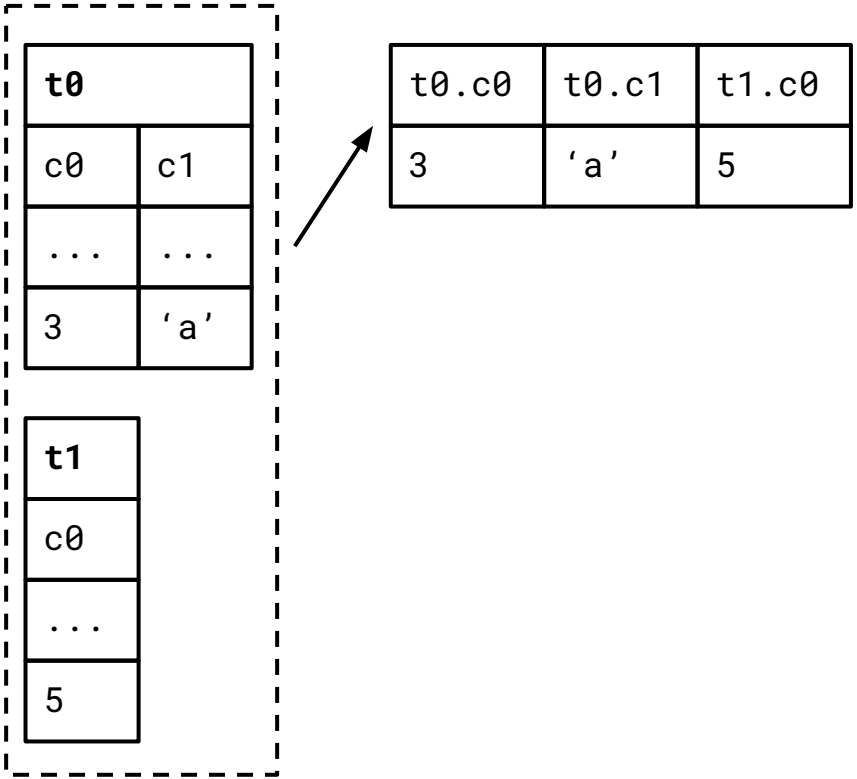
- Pivoted Query Synthesis (PQS)
- Non-optimizing Reference Engine Construction (NoREC)
- Ternary Logic Partitioning (TLP)
- Differential Query Execution (DQE)
- Query Plan Guidance (QPG)
- Cardinality Estimation Restriction Testing (CERT)
- Differential Query Plans (DQP)
- Constant Optimization Driven Database System Testing (CODDTest)

Pivoted Query Synthesis (PQS)

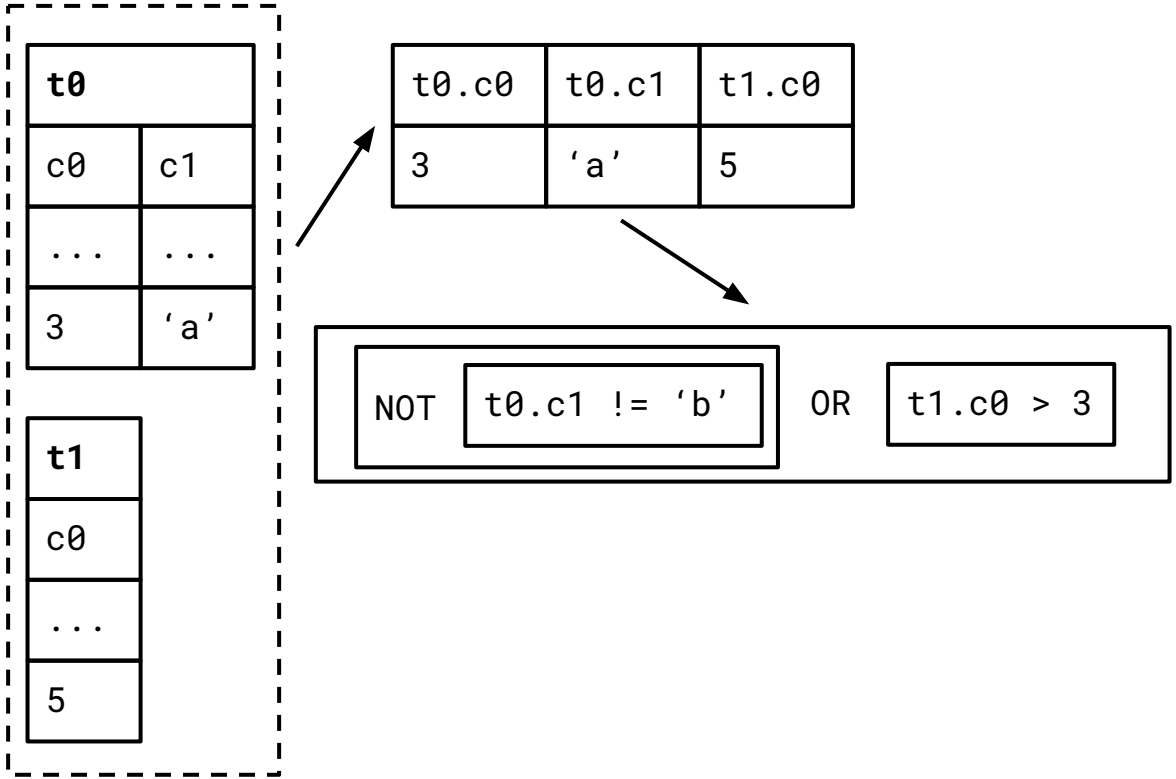
t0	
c0	c1
...	...
3	‘a’

t1
c0
...
5

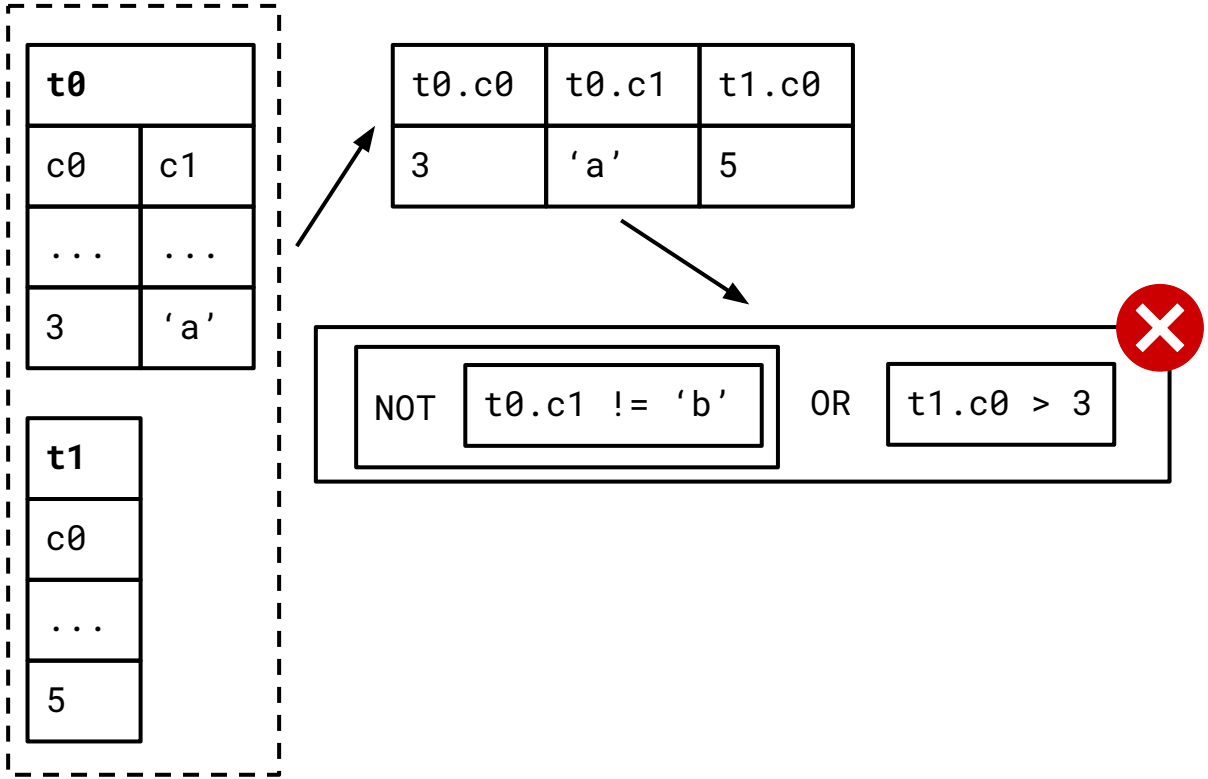
Pivoted Query Synthesis (PQS)



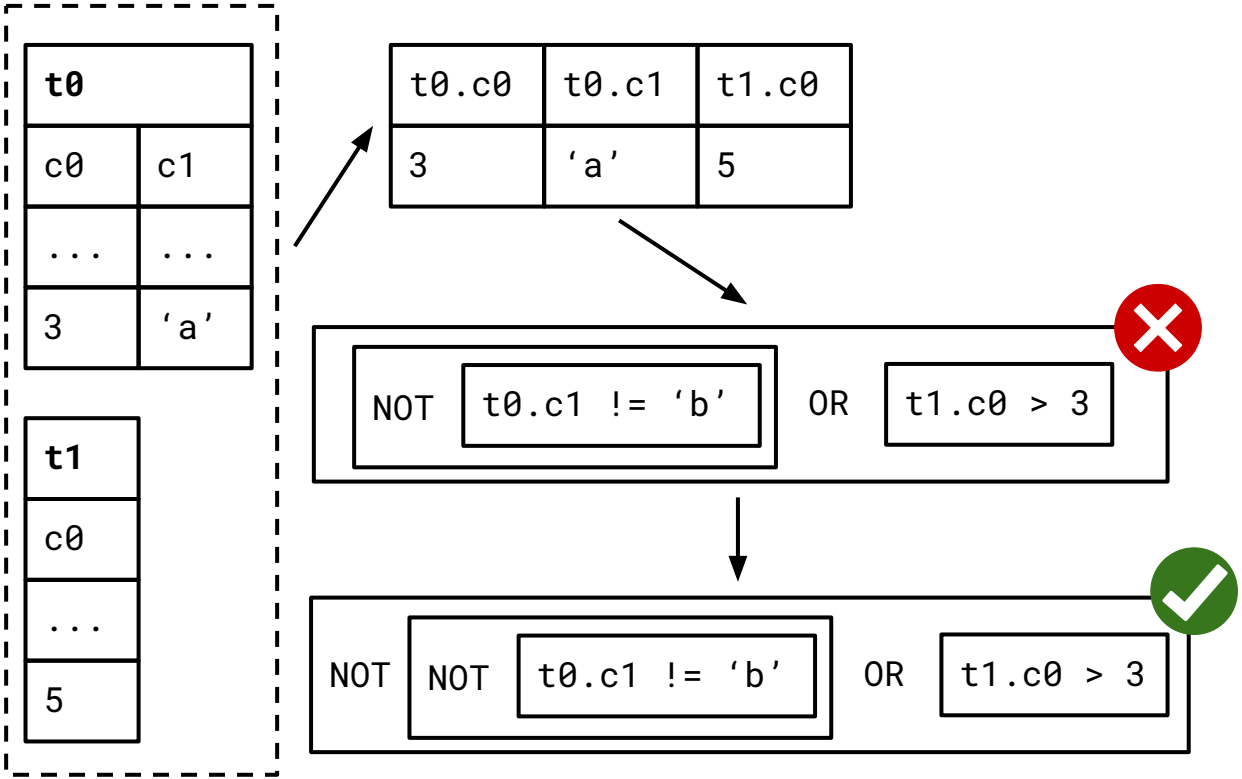
Pivoted Query Synthesis (PQS)



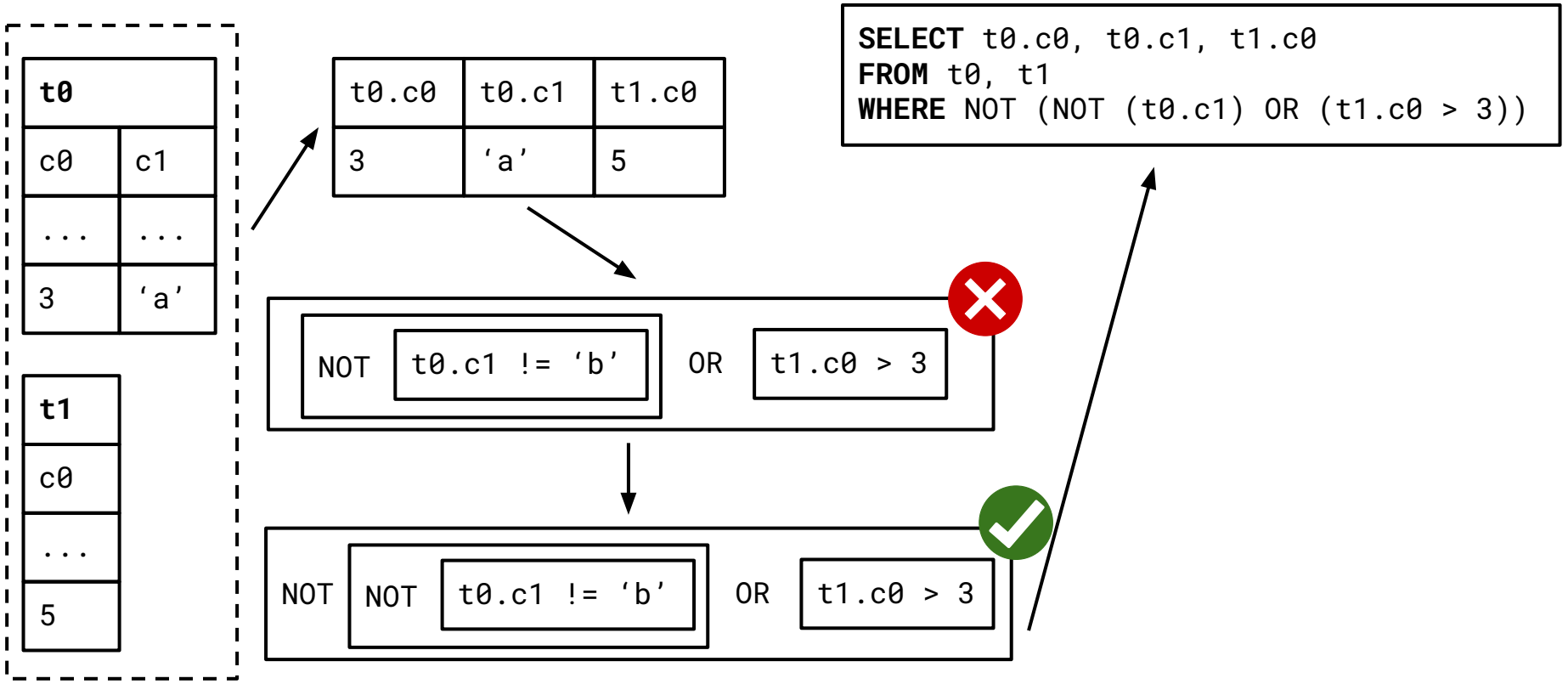
Pivoted Query Synthesis (PQS)

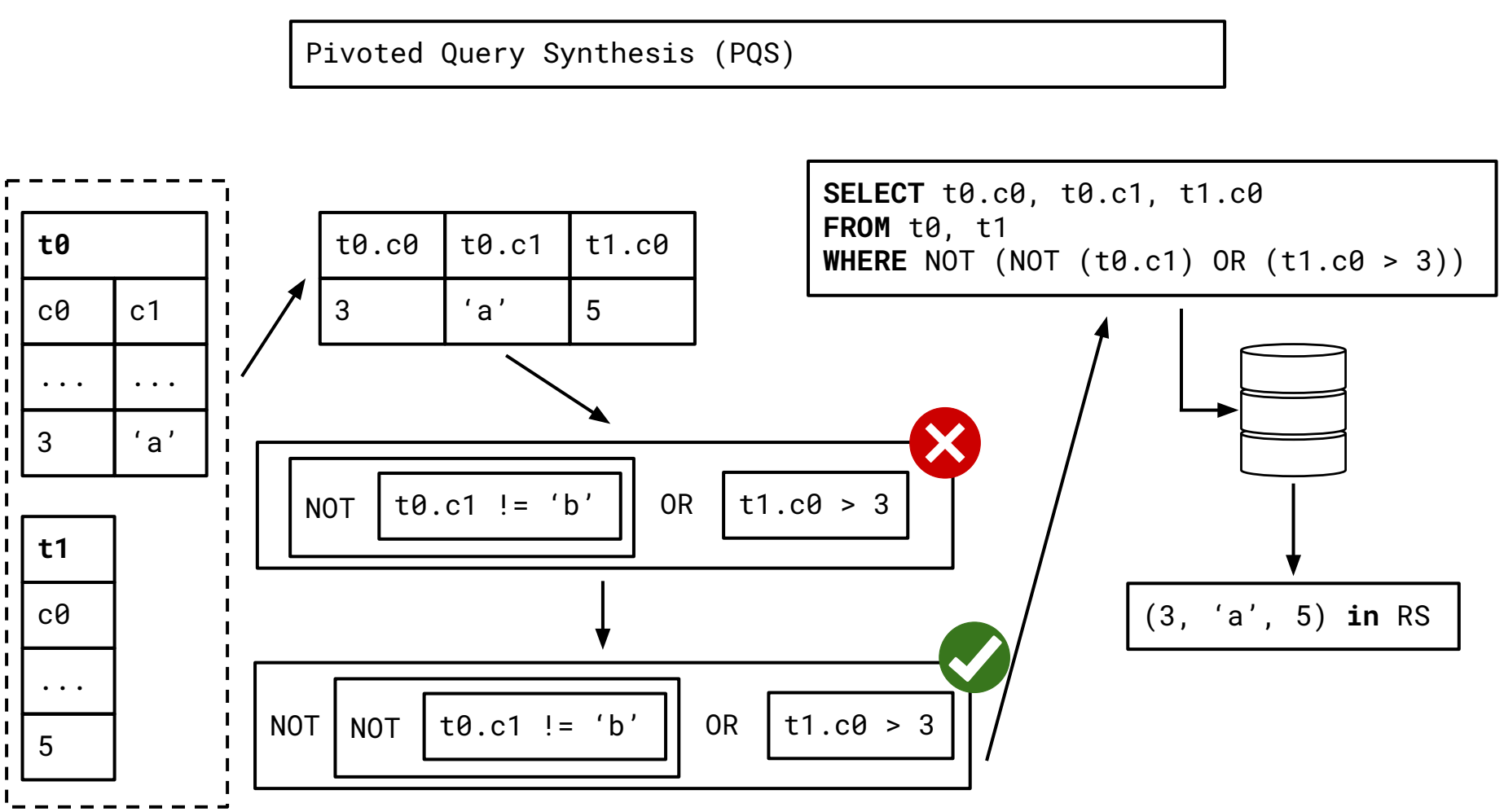


Pivoted Query Synthesis (PQS)



Pivoted Query Synthesis (PQS)




Pivoted Query Synthesis (PQS)
t0
c0
c1
...
...
3
'a'
t1
c0
...
5
t0.c0
t0.c1
t1.c0
3
'a'
5
NOT
t0.c1 != 'b'
OR
t1.c0 > 3
NOT
NOT
t0.c1 != 'b'
OR
t1.c0 > 3
SELECT t0.c0, t0.c1, t1.c0
FROM t0, t1
WHERE NOT (NOT (t0.c1) OR (t1.c0 > 3))
(3, 'a', 5) in RS

Non-optimizing Reference Engine Construction (NoREC)

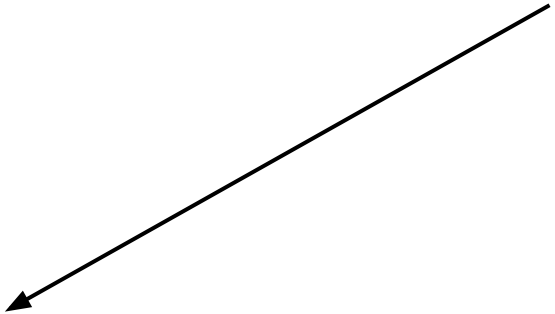
```
SELECT * FROM t0, t1 WHERE NOT (NOT (t0.c1) OR (t1.c0 > 3))
```

Non-optimizing Reference Engine Construction (NoREC)

```
SELECT * FROM t0, t1 WHERE NOT (NOT (t0.c1) OR (t1.c0 > 3))
```

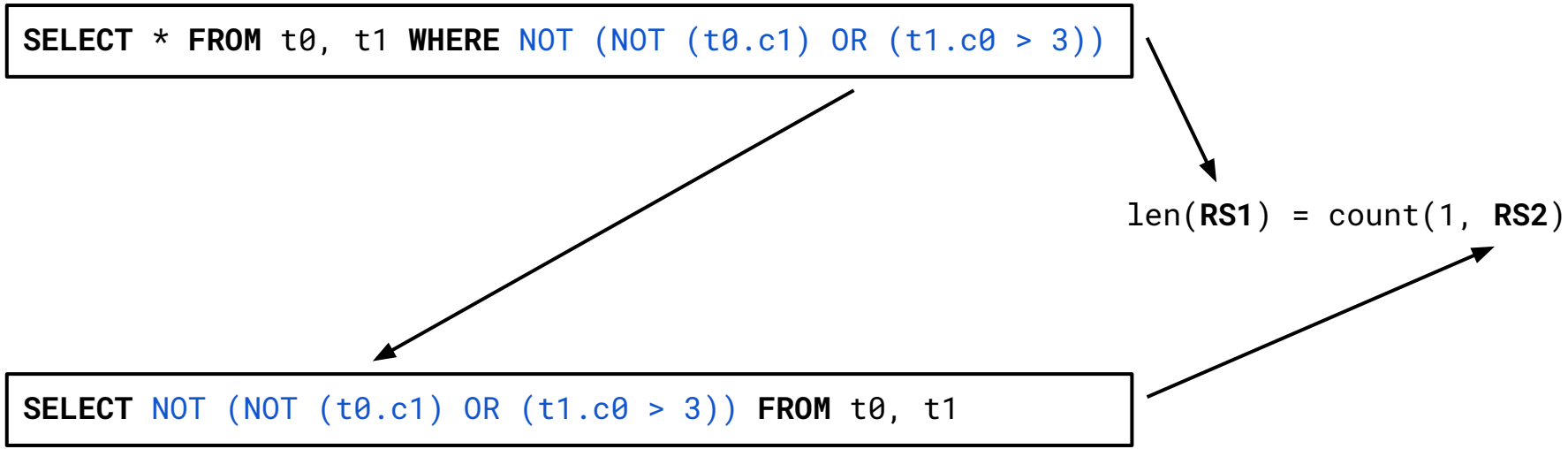
Non-optimizing Reference Engine Construction (NoREC)

```
SELECT * FROM t0, t1 WHERE NOT (NOT (t0.c1) OR (t1.c0 > 3))
```



```
SELECT NOT (NOT (t0.c1) OR (t1.c0 > 3)) FROM t0, t1
```

Non-optimizing Reference Engine Construction (NoREC)



Ternary Logic Partitioning (TLP)

$$p(r) = \begin{cases} \text{TRUE} & \text{if } \phi \\ \text{FALSE} & \text{if } \neg\phi \\ \text{NULL} & \text{if } \phi \text{ IS NULL} \end{cases}$$

Ternary Logic Partitioning (TLP)

$$p(r) = \begin{cases} \text{TRUE} & \text{if } \phi \\ \text{FALSE} & \text{if } \neg\phi \\ \text{NULL} & \text{if } \phi \text{ IS NULL} \end{cases}$$

```
SELECT * FROM t WHERE TRUE  ⟷  SELECT * FROM t WHERE p OR (NOT p) OR (P is NULL)
```

Ternary Logic Partitioning (TLP)

$$p(r) = \begin{cases} \text{TRUE} & \text{if } \phi \\ \text{FALSE} & \text{if } \neg\phi \\ \text{NULL} & \text{if } \phi \text{ IS NULL} \end{cases}$$

```
SELECT * FROM t WHERE TRUE  <-->  SELECT * FROM t WHERE p OR (NOT p) OR (P is NULL)
```

```
SELECT * FROM t WHERE TRUE  <-->  SELECT * FROM t WHERE p UNION ALL
                                  SELECT * FROM t WHERE NOT p UNION ALL
                                  SELECT * FROM t WHERE P is NULL
```

Ternary Logic Partitioning (TLP)

$$p(r) = \begin{cases} \text{TRUE} & \text{if } \phi \\ \text{FALSE} & \text{if } \neg\phi \\ \text{NULL} & \text{if } \phi \text{ IS NULL} \end{cases}$$

```
SELECT * FROM t WHERE TRUE  <-->  SELECT * FROM t WHERE p OR (NOT p) OR (P is NULL)
```

```
SELECT * FROM t WHERE TRUE  <-->  SELECT * FROM t WHERE p UNION ALL
                                  SELECT * FROM t WHERE NOT p UNION ALL
                                  SELECT * FROM t WHERE P is NULL
```

```
SELECT DISTINCT * FROM t    <-->  SELECT * FROM t WHERE p UNION
                                  SELECT * FROM t WHERE NOT p UNION
                                  SELECT * FROM t WHERE P is NULL
```

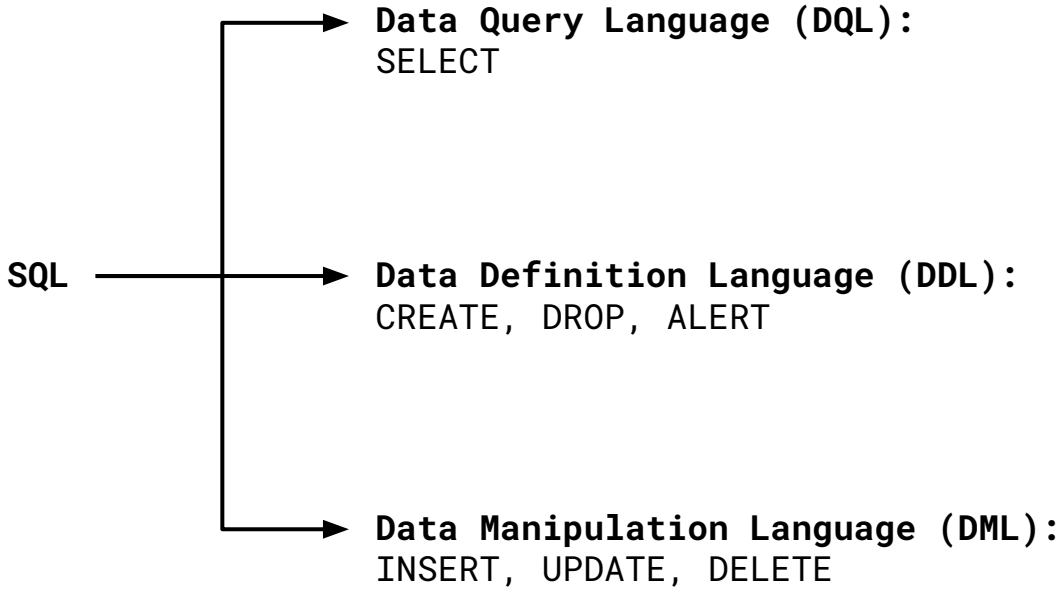
Ternary Logic Partitioning (TLP)

$$p(r) = \begin{cases} \text{TRUE} & \text{if } \phi \\ \text{FALSE} & \text{if } \neg\phi \\ \text{NULL} & \text{if } \phi \text{ IS NULL} \end{cases}$$

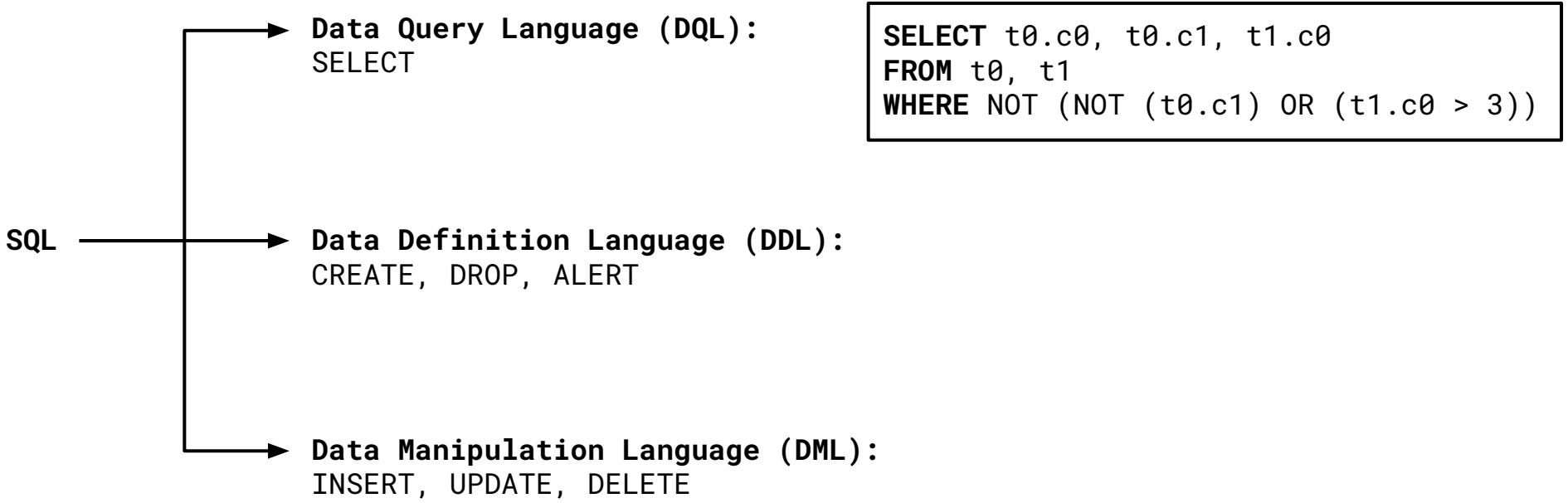
```
SELECT * FROM t WHERE TRUE  <-->  SELECT * FROM t WHERE p OR (NOT p) OR (P is NULL)

SELECT * FROM t WHERE TRUE  <-->  SELECT * FROM t WHERE p UNION ALL
                                  SELECT * FROM t WHERE NOT p UNION ALL
                                  SELECT * FROM t WHERE P is NULL

SELECT DISTINCT * FROM t    <-->  SELECT * FROM t WHERE p UNION
                                  SELECT * FROM t WHERE NOT p UNION
                                  SELECT * FROM t WHERE P is NULL

SELECT MIN*(x) FROM t       <-->  MIN (SELECT * FROM t WHERE p UNION
                                       SELECT * FROM t WHERE NOT p UNION
                                       SELECT * FROM t WHERE P is NULL)
```

* MIN, MAX, SUM, COUNT, AVG

Query Plan Guidance (QPG)

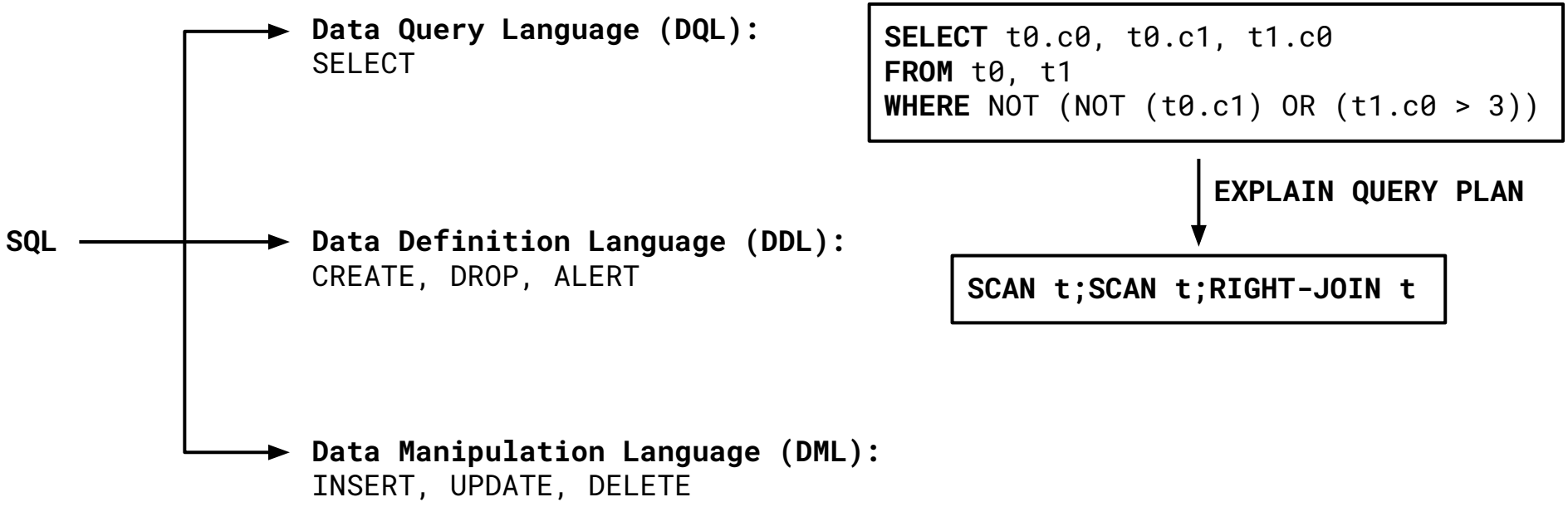


Query Plan Guidance (QPG)




```
SELECT t0.c0, t0.c1, t1.c0
FROM t0, t1
WHERE NOT (NOT (t0.c1) OR (t1.c0 > 3))
```

Query Plan Guidance (QPG)



Query Plan Guidance (QPG)

SQL

Data Query Language (DQL):
SELECT

Data Definition Language (DDL):
CREATE, DROP, ALERT

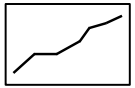
Data Manipulation Language (DML):
INSERT, UPDATE, DELETE

```
SELECT t0.c0, t0.c1, t1.c0
FROM t0, t1
WHERE NOT (NOT (t0.c1) OR (t1.c0 > 3))
```

EXPLAIN QUERY PLAN

SCAN t;SCAN t;RIGHT-JOIN t

QP Diversity



Query Plan Guidance (QPG)

SQL

Data Query Language (DQL):
SELECT

Data Definition Language (DDL):
CREATE, DROP, ALERT

Data Manipulation Language (DML):
INSERT, UPDATE, DELETE

```
SELECT t0.c0, t0.c1, t1.c0
FROM t0, t1
WHERE NOT (NOT (t0.c1) OR (t1.c0 > 3))
```

EXPLAIN QUERY PLAN

SCAN t;SCAN t;RIGHT-JOIN t

QP Diversity

RUN DQL

Query Plan Guidance (QPG)

SQL

Data Query Language (DQL):
SELECT

Data Definition Language (DDL):
CREATE, DROP, ALERT

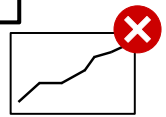
Data Manipulation Language (DML):
INSERT, UPDATE, DELETE

```
SELECT t0.c0, t0.c1, t1.c0
FROM t0, t1
WHERE NOT (NOT (t0.c1) OR (t1.c0 > 3))
```

EXPLAIN QUERY PLAN

```
SCAN t;SCAN t;RIGHT-JOIN t
```

QP Diversity



Generate DML / DDL

Cardinality Estimation Restriction Testing (CERT)

Cardinality Estimation Restriction Testing (CERT)

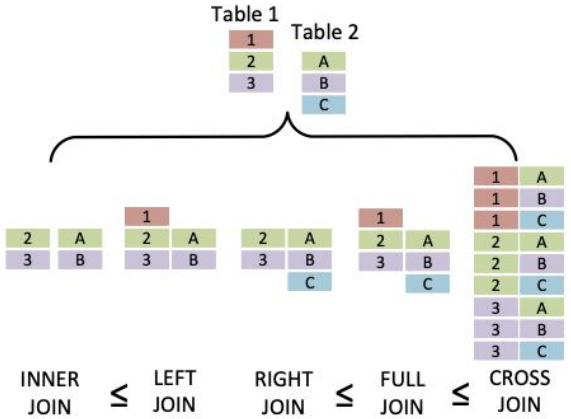


Figure 2: The inequality relationships of estimated cardinalities in the JOIN clause with an example to join two tables.

Cardinality Estimation Restriction Testing (CERT)

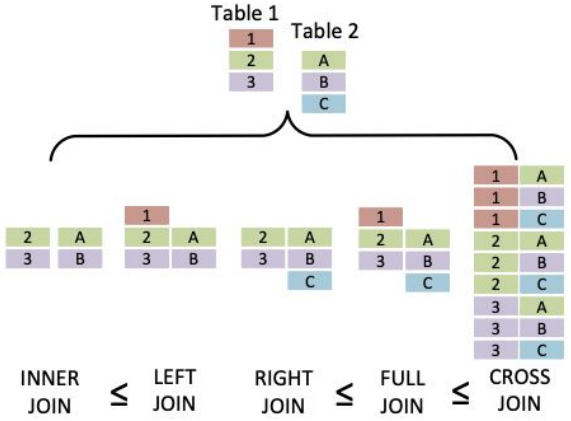


Figure 2: The inequality relationships of estimated cardinalities in the JOIN clause with an example to join two tables.

Table 2: The rules to restrict queries.

	Clause	Source	Target	Example
1	JOIN	LEFT JOIN	INNER JOIN	SELECT * FROM t0 ~~LEFT~~ INNER JOIN t1 ON ...;
2	JOIN	RIGHT JOIN	INNER JOIN	SELECT * FROM t0 ~~RIGHT~~ INNER JOIN t1 ON ...;
3	JOIN	FULL JOIN	LEFT JOIN	SELECT * FROM t0 ~~FULL~~ LEFT JOIN t1 ON ...;
4	JOIN	FULL JOIN	RIGHT JOIN	SELECT * FROM t0 ~~FULL~~ RIGHT JOIN t1 ON ...;
5†	JOIN	CROSS JOIN	FULL JOIN	SELECT * FROM t0 ~~CROSS~~ FULL JOIN t1;
6	SELECT	ALL	DISTINCT	SELECT ~~ALL~~ DISTINCT * FROM t0;
7	GROUP BY	<Empty>	<Predicate>	SELECT * FROM t0 GROUP BY c0 ;
8	HAVING	<Empty>	<Predicate>	SELECT * FROM t0 GROUP BY c0 HAVING c0>0 ;
9	WHERE	<Empty>	<Predicate>	SELECT * FROM t0 WHERE c0>0 ;
10	WHERE	<Predicate>	<Predicate> AND <Predicate>	SELECT * FROM t0 WHERE c0>0 AND c0!=8 ;
11	WHERE	<Predicate> OR <Predicate>	<Predicate>	SELECT * FROM t0 WHERE c0>0 ~~OR c0!=8~~ ;
12	LIMIT	<Natural number>	<Natural number> - <Natural number>	SELECT * FROM t0 LIMIT ~~10~~ 5 ;

† Rule 5 holds when both tables are not empty.

Cardinality Estimation Restriction Testing (CERT)

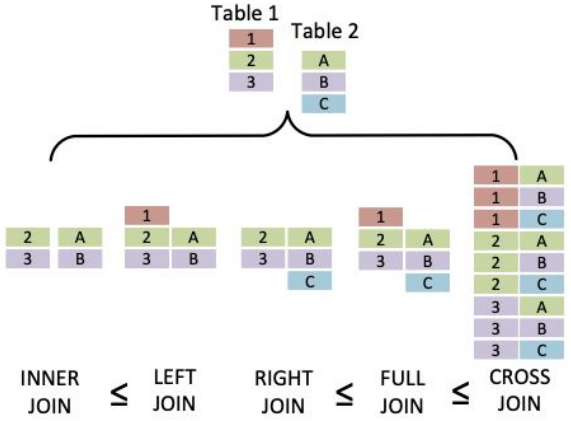


Figure 2: The inequality relationships of estimated cardinalities in the JOIN clause with an example to join two tables.

Table 2: The rules to restrict queries.

	Clause	Source	Target	Example
1	JOIN	LEFT JOIN	INNER JOIN	SELECT * FROM t0 ~~LEFT~~ INNER JOIN t1 ON ...;
2	JOIN	RIGHT JOIN	INNER JOIN	SELECT * FROM t0 ~~RIGHT~~ INNER JOIN t1 ON ...;
3	JOIN	FULL JOIN	LEFT JOIN	SELECT * FROM t0 ~~FULL~~ LEFT JOIN t1 ON ...;
4	JOIN	FULL JOIN	RIGHT JOIN	SELECT * FROM t0 ~~FULL~~ RIGHT JOIN t1 ON ...;
5†	JOIN	CROSS JOIN	FULL JOIN	SELECT * FROM t0 ~~CROSS~~ FULL JOIN t1;
6	SELECT	ALL	DISTINCT	SELECT ~~ALL~~ DISTINCT * FROM t0;
7	GROUP BY	<Empty>	<Predicate>	SELECT * FROM t0 GROUP BY c0 ;
8	HAVING	<Empty>	<Predicate>	SELECT * FROM t0 GROUP BY c0 HAVING c0>0 ;
9	WHERE	<Empty>	<Predicate>	SELECT * FROM t0 WHERE c0>0 ;
10	WHERE	<Predicate>	<Predicate> AND <Predicate>	SELECT * FROM t0 WHERE c0>0 AND c0!=8 ;
11	WHERE	<Predicate> OR <Predicate>	<Predicate>	SELECT * FROM t0 WHERE c0>0 ~~OR c0!=8~~ ;
12	LIMIT	<Natural number>	<Natural number> - <Natural number>	SELECT * FROM t0 LIMIT ~~10~~ 5 ;

† Rule 5 holds when both tables are not empty.

```
SELECT ALL * FROM t
```

Cardinality Estimation Restriction Testing (CERT)

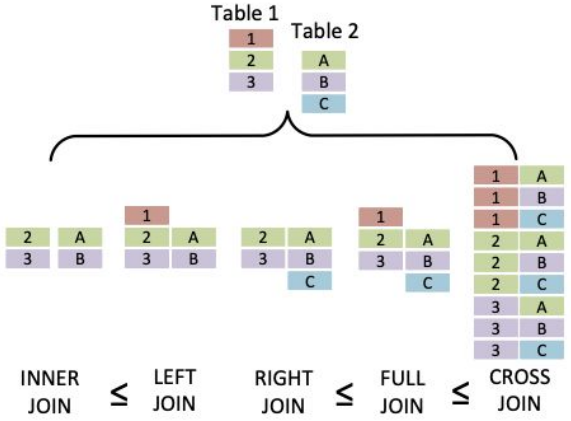


Figure 2: The inequality relationships of estimated cardinalities in the JOIN clause with an example to join two tables.

Table 2: The rules to restrict queries.

	Clause	Source	Target	Example
1	JOIN	LEFT JOIN	INNER JOIN	SELECT * FROM t0 ~~LEFT~~ INNER JOIN t1 ON ...;
2	JOIN	RIGHT JOIN	INNER JOIN	SELECT * FROM t0 ~~RIGHT~~ INNER JOIN t1 ON ...;
3	JOIN	FULL JOIN	LEFT JOIN	SELECT * FROM t0 ~~FULL~~ LEFT JOIN t1 ON ...;
4	JOIN	FULL JOIN	RIGHT JOIN	SELECT * FROM t0 ~~FULL~~ RIGHT JOIN t1 ON ...;
5†	JOIN	CROSS JOIN	FULL JOIN	SELECT * FROM t0 ~~CROSS~~ FULL JOIN t1;
6	SELECT	ALL	DISTINCT	SELECT ~~ALL~~ DISTINCT * FROM t0;
7	GROUP BY	<Empty>	<Predicate>	SELECT * FROM t0 GROUP BY c0 ;
8	HAVING	<Empty>	<Predicate>	SELECT * FROM t0 GROUP BY c0 HAVING c0>0 ;
9	WHERE	<Empty>	<Predicate>	SELECT * FROM t0 WHERE c0>0 ;
10	WHERE	<Predicate>	<Predicate> AND <Predicate>	SELECT * FROM t0 WHERE c0>0 AND c0!=8 ;
11	WHERE	<Predicate> OR <Predicate>	<Predicate>	SELECT * FROM t0 WHERE c0>0 ~~OR c0!=8~~ ;
12	LIMIT	<Natural number>	<Natural number> - <Natural number>	SELECT * FROM t0 LIMIT ~~10~~ 5 ;

† Rule 5 holds when both tables are not empty.

SELECT ALL * FROM t

Query Restriction

SELECT DISTINCT * FROM t

Cardinality Estimation Restriction Testing (CERT)

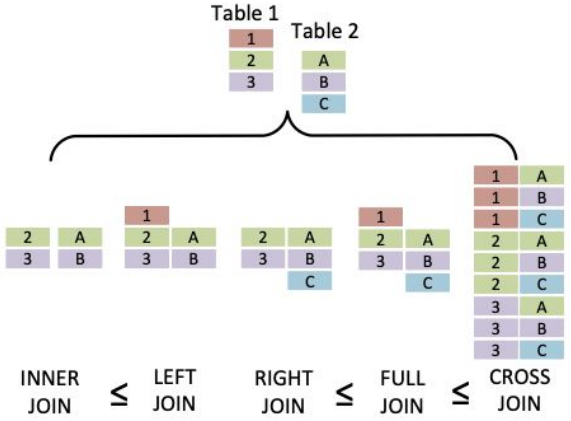
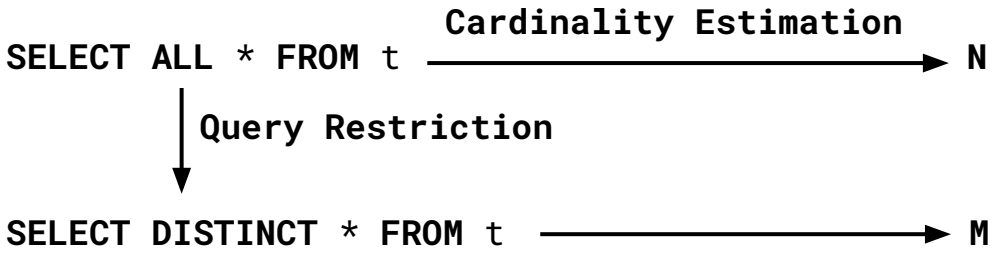


Figure 2: The inequality relationships of estimated cardinalities in the JOIN clause with an example to join two tables.

Table 2: The rules to restrict queries.

	Clause	Source	Target	Example
1	JOIN	LEFT JOIN	INNER JOIN	SELECT * FROM t0 ~~LEFT~~ INNER JOIN t1 ON ...;
2	JOIN	RIGHT JOIN	INNER JOIN	SELECT * FROM t0 ~~RIGHT~~ INNER JOIN t1 ON ...;
3	JOIN	FULL JOIN	LEFT JOIN	SELECT * FROM t0 ~~FULL~~ LEFT JOIN t1 ON ...;
4	JOIN	FULL JOIN	RIGHT JOIN	SELECT * FROM t0 ~~FULL~~ RIGHT JOIN t1 ON ...;
5†	JOIN	CROSS JOIN	FULL JOIN	SELECT * FROM t0 ~~CROSS~~ FULL JOIN t1;
6	SELECT	ALL	DISTINCT	SELECT ~~ALL~~ DISTINCT * FROM t0;
7	GROUP BY	<Empty>	<Predicate>	SELECT * FROM t0 GROUP BY c0 ;
8	HAVING	<Empty>	<Predicate>	SELECT * FROM t0 GROUP BY c0 HAVING c0>0 ;
9	WHERE	<Empty>	<Predicate>	SELECT * FROM t0 WHERE c0>0 ;
10	WHERE	<Predicate>	<Predicate> AND <Predicate>	SELECT * FROM t0 WHERE c0>0 AND c0!=8 ;
11	WHERE	<Predicate> OR <Predicate>	<Predicate>	SELECT * FROM t0 WHERE c0>0 ~~OR c0!=8~~ ;
12	LIMIT	<Natural number>	<Natural number> - <Natural number>	SELECT * FROM t0 LIMIT ~~10~~ 5 ;

† Rule 5 holds when both tables are not empty.



Cardinality Estimation Restriction Testing (CERT)

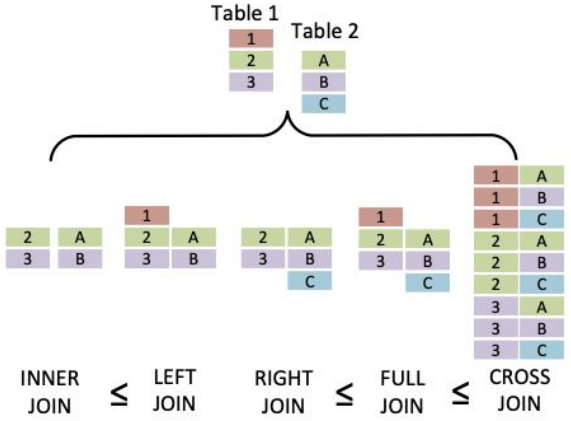
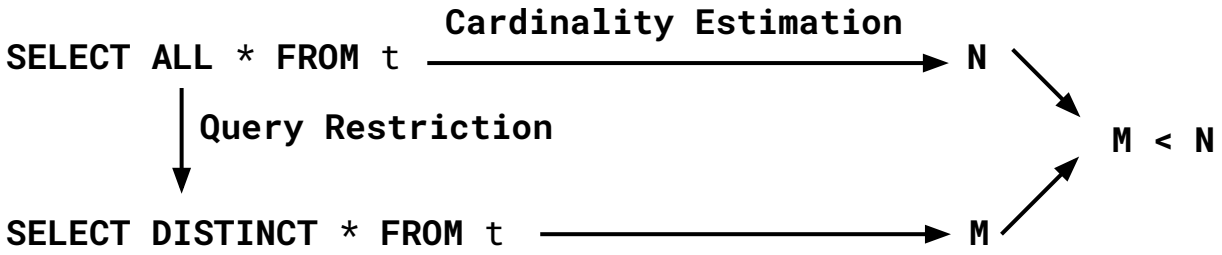


Figure 2: The inequality relationships of estimated cardinalities in the JOIN clause with an example to join two tables.

Table 2: The rules to restrict queries.

	Clause	Source	Target	Example
1	JOIN	LEFT JOIN	INNER JOIN	SELECT * FROM t0 ~~LEFT~~ INNER JOIN t1 ON ...;
2	JOIN	RIGHT JOIN	INNER JOIN	SELECT * FROM t0 ~~RIGHT~~ INNER JOIN t1 ON ...;
3	JOIN	FULL JOIN	LEFT JOIN	SELECT * FROM t0 ~~FULL~~ LEFT JOIN t1 ON ...;
4	JOIN	FULL JOIN	RIGHT JOIN	SELECT * FROM t0 ~~FULL~~ RIGHT JOIN t1 ON ...;
5†	JOIN	CROSS JOIN	FULL JOIN	SELECT * FROM t0 ~~CROSS~~ FULL JOIN t1;
6	SELECT	ALL	DISTINCT	SELECT ~~ALL~~ DISTINCT * FROM t0;
7	GROUP BY	<Empty>	<Predicate>	SELECT * FROM t0 GROUP BY c0 ;
8	HAVING	<Empty>	<Predicate>	SELECT * FROM t0 GROUP BY c0 HAVING c0>0 ;
9	WHERE	<Empty>	<Predicate>	SELECT * FROM t0 WHERE c0>0 ;
10	WHERE	<Predicate>	<Predicate> AND <Predicate>	SELECT * FROM t0 WHERE c0>0 AND c0!=8 ;
11	WHERE	<Predicate> OR <Predicate>	<Predicate>	SELECT * FROM t0 WHERE c0>0 ~~OR c0!=8~~ ;
12	LIMIT	<Natural number>	<Natural number> - <Natural number>	SELECT * FROM t0 LIMIT ~~10~~ 5 ;

† Rule 5 holds when both tables are not empty.



Differential Query Plans (DQP)

Query Hints

System Variables

Ambiguous Queries

Differential Query Plans (DQP)

Query Hints

```
CREATE TABLE t0 (c0 INT);
CREATE TABLE t1 (c0 BOOL, c1 BOOL);
INSERT INTO t1 VALUES (false, true);
INSERT INTO t1 VALUES (true, true);
CREATE VIEW v0 (c0, c1, c2) AS SELECT t1.c0, LOG10(t0.c0), t1.c0 FROM t0,t1;
INSERT INTO t0 (c0) VALUES (3);

SELECT COUNT (v0.c2) FROM v0, t0 CROSS JOIN t1 ORDER BY -v0.c1 ; -- empty set
SELECT /* + MERGE_JOIN ( t1 , t0 , v0 ) */ COUNT (v0.c2) FROM v0, t0 CROSS JOIN t1
       ORDER BY -v0.c1; -- {4}
```

System Variables

Ambiguous Queries

Differential Query Plans (DQP)

Query Hints

System Variables

```
CREATE TABLE t0 (c0 INT) ;
INSERT INTO t0 (c0) VALUES (1) ;
CREATE INDEX i0 USING HASH ON t0 (c0) INVISIBLE;

SELECT t0.c0 FROM t0 WHERE COALESCE (0.6) IN (t0.c0); -- {}
SET SESSION optimizer_switch ='use_invisible_indexes = on';
SELECT t0.c0 FROM t0 WHERE COALESCE (0.6) IN (t0.c0) ; -- {1}
```

Ambiguous Queries

Differential Query Plans (DQP)

Query Hints

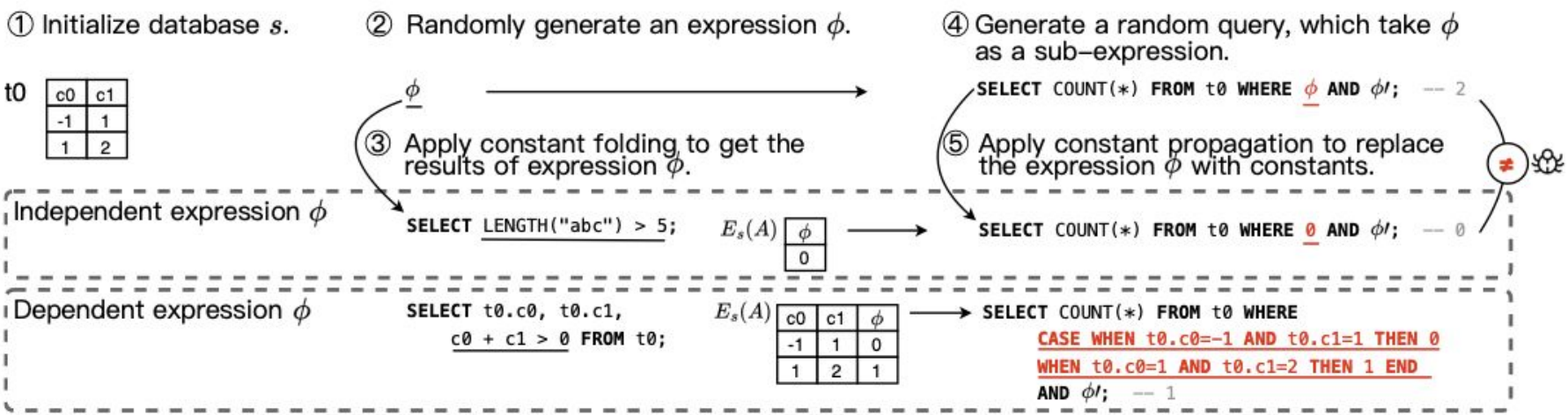
System Variables

Ambiguous Queries

```
CREATE TABLE t0 ( c0 FLOAT ) ;
INSERT INTO t0 VALUES (0.9), (0.8);
CREATE INDEX i0 ON t0 ( c0 ) ;
SET @@sql_mode =' ';

SELECT t0.c0 FROM t0 GROUP BY CAST (t0.c0 AS
DECIMAL); -- {0.8}
SELECT /* + IGNORE_INDEX (t0, i0) */ t0 . c0 FROM t0
GROUP BY CAST ( t0 . c0 AS DECIMAL ) ; -- {0.9}
```

```
CREATE TABLE t0 ( c0 FLOAT ) ;
INSERT INTO t0 VALUES (0.8), (0.9);
CREATE INDEX i0 ON t0 ( c0 ) ;
SET @@sql_mode =' ';

SELECT t0.c0 FROM t0 GROUP BY CAST (t0.c0 AS
DECIMAL); -- {0.8}
SELECT /* + IGNORE_INDEX (t0, i0) */ t0.c0 FROM t0
GROUP BY CAST (t0.c0 AS DECIMAL); -- {0.8}
```

Constant Optimization Driven Database System Testing (CODDTest)



What do I do?

- Maintainer-written more specific properties
- Interactive testing environment
- SQL with Contracts
- Fault injection
- Concurrent users

Open Problems

- Counterexample minimization
- Interesting database states
- Automated generation

Alperen Keles

akeles@umd.edu

alperenkeles.com

Slides

alperenkeles.com/slides
/the-art-of-db-testing.pdf