Generating Targeted Queries for Database Testing

Chaitanya Mishra, Nick Koudas
University of Toronto
Calisto Zuzarte
IBM Toronto

SIGMOD 2008
Performance Testing

• Standard Benchmarks (e.g. TPC)
  – May not test the new feature across a wide range of conditions.

• Microbenchmarks
  – Carefully designed tests to evaluate the performance of the new feature with controlled parameters.
Performance Testing: Examples

• New cardinality estimation technique:
  – Test across queries with varying selectivities.

• Modified join algorithm:
  – Test across queries with varying join input sizes and selectivities.
  – To test pipelining, vary intermediate relation sizes.

• New memory manager:
  – Vary sizes of intermediate blocking points.
Designing a Test Query

Select * from Part P, Supplier S, Partsupp PS
Where p_partkey = ps_partkey
and s_suppkey = ps_suppkey
Designing a Test Query

Select * from Part P, Supplier S, Partsupp PS
Where p_partkey = ps_partkey and s_suppkey = ps_suppkey

Cardinality constraints specified on intermediate relations.
Designing a Test Query

Cardinality constraints specified on intermediate relations.

Select * from Part P, Supplier S, Partsupp PS
Where p_partkey = ps_partkey
and s_suppkey = ps_suppkey
and s_acctbal < ?
and p_retailprice < ?
and ps_av ail qty < ?
and ps_supplycost < ?
Targeted Query Generation (TQG)

Given:

\[ (Q_1, N_1) \ldots (Q_m, N_m) \]
\[ |Q_i| = N_i \]
\[ m \text{ Constraints} \]

\[ D \]

\[ x_1 < c_1 \ldots x_d < c_d \]
\[ d \text{ range predicates} \]

Generate:

\[ Q^r : x_1 < c_1^r \ldots x_d < c_d^r \]

s.t \[ Q^r \] satisfies the constraints when executed on \[ D \].
Targeted Query Generation (TQG)

TPCH

Database

Constraints

Predicates

\[ P \times PS \times S \]

- \( |P| = 200K \)
- \( |P \times PS| = 120K \)
- \( |S| = 80K \)

Predicates:

- \( s_{\text{acctbal}} < 6000 \)
- \( p_{\text{retailprice}} < 2500 \)
- \( ps_{\text{availqty}} < 50 \)
- \( ps_{\text{supplycost}} < 1000 \)
Targeted Query Generation (TQG)

TPCH

\[ P \Join PS \Join S \]

| \( P \) | = 200K
| \( P \Join PS \) | = 120K
| \( S \) | = 80K

Database

Constraints

Predicates

s_acctbal < 9000
p_retailprice < 1500
ps_availqty < 30
ps_supplycost < 2000
Outline

• Base Cases
• TQGen Algorithm
• Cardinality Estimation
• Implementation and Experiments
Outline

• Base Cases
  – Single Constraint
  – Multiple Constraints with Independence
• TQGen Algorithm
• Cardinality Estimation
• Implementation and Experiments
Single Constraint

- Query $Q$, Database $D$, Constraint $(Q,N)$.
- [BCT06]: NP-complete.
Lowerbound for Exact Solution

- [BCT06]: Lowerbound of \( \binom{n+d-2}{d-1} \) calls to the cardinality estimation component to satisfy constraint \((Q,N)\) exactly.
- There may be no query that satisfies the constraint exactly!
Lowerbound for Approx. Solution

• Lowerbound of $n + d - 2 \binom{c}{d-1}$ calls to the cardinality estimation component to satisfy constraint $(Q,N)$ to within $[N-E,N+E]$ or $[N(1-\varepsilon),N(1+\varepsilon)]$.

• Approximating to within an arbitrary absolute or relative error is hard!
Binary Search

Target Card.
50K

x < 100 and y < 100
100K
Binary Search

Target Card.
50K

\(x < 80 \text{ and } y < 100\)

80K

BaseCase/SingleConstraint
Binary Search

Target Card.
50K

x < 40 and y < 100
40K
Binary Search

Target Card.  
50K

\[ x < 50 \text{ and } y < 100 \]

50321

BaseCase/SingleConstraint
Binary Search

Target Card. 50K

$x < 50 \text{ and } y < 96$

49981
Binary Search

- Cost: $O(d \log n)$
- Error: $\frac{\min_i(\maxfreq(i))}{2}$

Target Card: 50K

$x < 50$ and $y < 96$

$49981$
Binary Search

- Cost: $O(d \log n)$
- Error = $\frac{\min_i(\text{maxfreq}(i))}{2}$

Discreteness makes the problem hard.
Outline

• Base Cases
  – Single Constraint
  – Multiple Constraints with Independence
• TQGen Algorithm
• Cardinality Estimation
• Implementation and Experiments
Multiple Constraints

Select * from R, S
Where R.id = S.id
And R.x < \( C_x \)
And R.y < \( C_y \)
And S.z < \( C_z \)
Multiple Constraints: Equations

\[ X = \log(\text{sel}( R.x < C_x )) \]
\[ X + Y = \log(120K) - \log(|R|) \]
\[ X + Y + Z = \log(50K) - \log(|R \bowtie S|) \]
\[ X, Y, Z \leq 0 \]

- Assuming independence between attributes, express relationship between predicate selectivities and cardinality constraints as a system of linear equations.
Multiple Constraints: Equations

\[ X = \log(\text{sel}( R.x < C_x )) \]
\[ X + Y = \log(120K) - \log(|R|) \]
\[ X + Y + Z = \log(50K) - \log(|R \cap S|) \]
\[ X, Y, Z \leq 0 \]

- Solve for \( X, Y, Z \).
- Use Single Constraint procedure to find \( C_x C_y C_z \)

*Independence reduces the multiple constraint problem to the single constraint version.*
Outline

- Base Cases
- TQGen Algorithm
- Cardinality Estimation
- Implementation and Experiments
TQGen

Search procedure for targeted query generation given an arbitrary number of constraints
without making distributional assumptions.
TQGen

\[ x < 100 \quad \text{and} \quad y < 100 \]
x < 150 and y < 160
TQGen

*Exploration*

\( x < 150 \) and \( y < 160 \)
TQGen

Exploration

x < 150 and y < 160

x < 140 and y < 70
Outline

• Base Cases
• TQGen Algorithm
  – Bounding
  – Exploration
  – Scoring and Pruning
• Cardinality Estimation
• Implementation and Experiments
Bounding

• Restrict the search space for solutions.
• Two phase bounding
Select * from R, S
Where R.id = S.id
And R.x < 100
And R.y < 100
And S.z < 100

Original Cardinality:
|R \times S| = 70K
|R| = 80K
Bounding: Contraction

Independently Contract each predicate till you undershoot all constraints affected by it.

```
Select * from R, S
Where R.id = S.id
And R.x < 100
And R.y < 100
And S.z < 100
```

Original Cardinality:
|R \ S| = 70K
|R| = 80K
Bounding: Contraction

Independently Contract each predicate till you undershoot all constraints affected by it.

```
Select * from R, S
Where R.id = S.id
And R.x < 50
And R.y < 100
And S.z < 100
```

Current Cardinality:
|R \times S| = 50K
|R| = 65K
Bounding: Contraction

Independently Contract each predicate till you undershoot all constraints affected by it.

Select * from R, S
Where R.id = S.id
And R.x < 100
And R.y < 40
And S.z < 100

Current Cardinality:
|R \times S| = 50K
|R| = 70K
Bounding: Contraction

Independently Contract each predicate till you undershoot all constraints affected by it.

Select * from R, S
Where R.id = S.id
And R.x < 100
And R.y < 100
And S.z < 80

Current Cardinality:
|R \bowtie S| = 50K
|R| = 80K
Contracted Query

Select * from R, S
Where R.id = S.id
And R.x < 50
And R.y < 40
And S.z < 80

Contracted Query Cardinality:
|R \ S| = 20K
|R| = 55K
Bounding: Relaxation

Independently Relax each predicate till you overshoot all constraints affected by it.

Select * from R, S
Where R.id = S.id
And R.x < 50
And R.y < 40
And S.z < 80

Contracted Query Cardinality:
|R ∩ S| = 20K
|R| = 55K

TQGen/Bounding
Bounding: Relaxation

 Independently Relax each predicate till you overshoot all constraints affected by it.

Select * from R, S
Where R.id = S.id
And R.x < 150
And R.y < 40
And S.z < 80

Current Cardinality:
|R ∩ S| = 80K
|R| = 120K
Bounding: Relaxation

Independently Relax each predicate till you overshoot all constraints affected by it.

```
Select * from R, S
Where R.id = S.id
And R.x < 50
And R.y < 160
And S.z < 80
```

Current Cardinality:

|R X S| = 70K
|R| = 120K
Bounding: Relaxation

Independently Relax each predicate till you overshoot all constraints affected by it.

Select * from R, S
Where R.id = S.id
And R.x < 50
And R.y < 40
And S.z < 190

Current Cardinality:
|R ∩ S| = 50K
|R| = 55K

TQGen/Bounding
Bounding: Relaxation

Two phase bounding ensures that we consider a sufficiently large range of values for each predicate.

```
Select * from R, S
Where R.id = S.id
And R.x < 150
And R.y < 160
And S.z < 190
```

Bounding Query Cardinality:

|\(|R \bowtie S| = 150K
|\)|R| = 200K
Outline

• Base Cases
• TQGen Algorithm
  – Bounding
  – Exploration
  – Scoring and Pruning
• Cardinality Estimation
• Implementation and Experiments
Exploration

\[ x < 150 \text{ and } y < 160 \]
Exploration

\[ x < 150 \text{ and } y < 160 \]
Exploration

$x < 150 \text{ and } y < 160$
Exploration

$x < 150$ and $y < 160$
Outline

• Base Cases
• TQGen Algorithm
  – Bounding
  – Exploration
  – Scoring
• Cardinality Estimation
• Implementation and Experiments
**Scoring**

$max(A): x_h, y_h$

$min(A): x_l, y_l$

Count number of constraints $(Q_i, N_i)$ that are bounded.

$$|Q_i (max(A))| \geq N_i \geq |Q_i (min(A))|$$
Scoring: Breaking Ties

A

Q₁ (max(A)) 100K
Q₂ (max(A)) 170K

Q₁ (min(A))
20K
Q₂ (min(A))
90K

B

Q₁ (max(B)) 60K
Q₂ (max(B)) 200K

Q₁ (min(B))
20K
Q₂ (min(B))
100K

50K
120K

120K
20K
Scoring: Breaking Ties

Q₁ (max(A)) 100K
Q₂ (max(A)) 170K
Q₁ (min(A)) 50K
Q₂ (min(A)) 90K

Q₁ : 50K
Q₂ : 120K

A

Q₁ (max(B)) 60K
Q₂ (max(B)) 200K
Q₁ (min(B)) 100K
Q₂ (min(B)) 120K

B

TQGen/Scoring
Pruning

\[ A (\geq k) \]

\[ B (\leq m-k) \]
TQGen

- Space Bounding.
- Grid based exploration.
- Scoring functions to guide search, with pruning techniques for efficiency.
Outline

• Base Cases
• TQGen Algorithm
• Cardinality Estimation
• Implementation and Experiments
Cardinality Estimation

• Actual query execution can be expensive.
• Optimizer cardinality estimates are cheaper, but potentially incorrect.
• *Sampling based cardinality estimation*
Sampling

- Index Assisted Sampling.
  - Sampling multiple times may be expensive
- Sample over a superset
  - bounding query
- Store samples in a QuadTree for fast range counting.
Outline

• Base Cases
• TQGen Algorithm
• Cardinality Estimation
• Implementation and Experiments
System Design

Coordinator

$Q_1$  $Q_2$  $Q_3$  $Q_4$
Experiment Design

• Postgresql + Java.

• 2 Test Databases:
  – TPCH 1 GB Uniform data & Zipfian Skew (Z = 1)

• Experiments
  – Accuracy
    • Varying target cardinalities
    • Varying #Tables and Constraints
  – Overheads
    • Varying #Constraints and Predicates
Accuracy: Target Cardinality

Uniform

Zipfian $Z = 1$
Accuracy: #Tables/Constraints

![Bar Chart]

- **X-axis**: Number of Tables and Constraints
- **Y-axis**: Avg. Error %
- **Legend**:
  - TPCH Z=0
  - TPCH Z=1

The chart shows the average error percentage for different numbers of tables and constraints, comparing TPCH Z=0 and TPCH Z=1 conditions.
Overheads: #Constraints/Predicates

![Bar chart showing execution time for different constraints and dimensions with cold cache and warm cache.](chart.png)
Summary

• Targeted Query Generation
  – is useful.
  – is computationally hard.
  – is amenable to practical solutions.
  – presents a set of challenging and interesting problems.
Thank You!