Fast Detection of Denial Constraint Violations with FACET

Author(s): Eduardo H. M. Pena, Eduardo C. de Almeida, Felix Naumann

Presented by

Sagar Shekhargouda Patil (A20501427) **Yuvraj Nikam (A20501952)** **Marut Pandya (A20518560)**

UNDER GUIDANCE OF DR.BORIS GLAVIC

The Challenge of Constraint-Based Violations

DETECTION OF CONSTRAINT-BASED VIOLATIONS IS CRUCIAL IN DATA CLEANING.

EXISTING SOLUTIONS OFTEN FAIL DUE TO TIME OR MEMORY LIMITATIONS.

INTRODUCING FACET – LEVERAGING "COLUMN SKETCHING" FOR QUICK AND EFFICIENT DENIAL CONSTRAINT (DC) CHECKS.

FACET UTILIZES ADVANCED ALGORITHMS AND DATA STRUCTURES FOR EFFECTIVE DC DETECTION.

Drawbacks of Traditional Approaches

- Example Constraints:
  - Unique Employee IDs
  - No self-supervision (SID)
  - Years of service vs. Salary in the same department (Dept)
- Traditional SQL queries for constraint validation.
- Challenges:
  - Sorting and non-equi-joins lead to performance issues.
  - Drawbacks of existing systems: low-level representations, performance degradation, and estimation errors.
  - Need for an effective error detection system for a wide range of constraints.

```
SELECT   t.ID,u.ID
FROM     Employee t, Employee u
WHERE    t.Dept=u.Dept
AND      t.StartDate<u.StartDate
AND      t.Salary<u.Salary
```

Table 1: The Employee table.

	ID	Name	Dept	StartDate	Salary	SID
t_1	100	C. Gardner	Sales	2012	3000	100
t_2	101	R. Geller	Research	2014	8000	102
t_3	102	D. Brown	Research	2014	6000	101
t_4	103	H. McCoy	Research	2015	8000	101

FACET Framework Overview

FACET Framework:

- Mapping each DC into a refinement pipeline.
- Logical operators ensuring efficient DC evaluation.

Addressing drawbacks:

- New algorithms with hybrid data structures for optimized error detection.
- Introduction of column sketches to determine predicate evaluation order.
- FACET's flexibility in handling multiple DCs with common predicates.
- Modes of error detection and reuse for improved efficiency.

Violation Detection using Refinements

$$\varphi_1 : \neg(t.ID = t'.ID)$$

$$\varphi_2 : \neg(t.ID = t'.SID \wedge t.SID = t'.ID)$$

$$\varphi_3 : \neg(t.Dept = t'.Dept \wedge t.StartDate < t'.StartDate \wedge t.Salary < t'.Salary)$$

- **Representation of Intermediates:** Compact form of pairs of tuples $(tids_1, tids_2)$.

- **Refinement Operator:**

1. Takes pairs (tids1, tids2) and a predicate p.
2. Returns pairs $(tids'_1, tids'_2)$ representing all subsets satisfying p.

- **Refinement Pipeline :**

1. Describing the refinement of predicates p_1 and p_2.
2. Incremental consumption of pairs in the pipeline.

Design of FACET (FAst Constraint-based Error DeTector)

- Optimizing Denial Constraint (DC) Predicate Refinement
  - DC predicates (# of columns and the comparison operator) play a crucial role in refinement performance.
  - Existing algorithms face challenges. E.g. Nested loop approach
  - Author’s approach addresses limitations.
  - Predicates categorized into Equalities, Non-equalities, and Inequalities for specialized algorithms.
- Hybrid TIDs Representation
  - Goal : Optimized Data Representation
  - HYDRA and VioFinder examples using arrays and compressed bitmaps.
  - FACET's hybrid approach switches between array and bitmap representations based on computation patterns.
- Two-Phase Structure
  - Two-phase refinement structure: Fetching column values and iterating data structures.
  - Dependency on hash tables; building phase crucial for performance.

Design of FACET (Continued)

Hash-Join-Like Optimization:

Comparison with existing algorithms creating separate hash tables.

FACET's approach uses a single hash table and a hash-join-like technique, improving efficiency.

Inequality Challenges:

Inequalities pose computational challenges; reliance on a single algorithm is inefficient.

FACET introduces a new third algorithm, dynamically choosing the most efficient among the three.

Robust Evaluation Plans

Predicate Order is important in refinement pipeline.

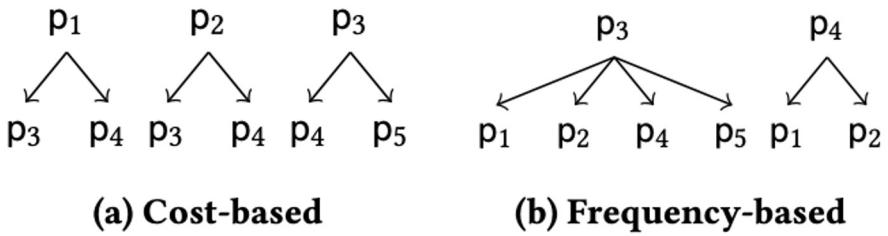
Selecting the order of predicates crucial for efficient refinement.

FACET offers three algorithms for inequalities, and their performance varies based on input.

Existing methods rely on predicate selectivity so FACET proposes a **novel algorithm** which explores column sketches for high accuracy in selecting the refinement order and algorithms.

Design of FACET (Continued)

- Multi-Constraint Execution:
  - Previous works handle error detection one constraint at a time, leading to repeated computations.
  - FACET introduces a trie-based scheme for simultaneous checking of multiple DCs.
  - Two approaches:
    - Cost based
    - Frequency based



(a) Cost-based **(b) Frequency-based**

Refinement Algorithms

- Optimization techniques for different predicates
  - Equalities
  - Non-equalities
  - Inequalities
    - IEJoin
    - Hash-Sort-Merge (HSM)
    - Binning-Hash-Sort-Merge (BHSM)
    - Optimizations

Refinement Algorithms (Continued)

Equalities:

- Definition: Equalities as predicates of the form p: t.A = t′.B.
- Build and Probe Side: Explanation of choosing build and probe sides based on cardinality estimation using HyperLogLog sketches.
- Example: Refinement of the predicate p: t.SID = t′.ID, showcasing hash table entries after the probing phase.
- Optimization: Avoiding scans in reflexive pairs of tids for efficiency.
  - Further Refinement: Explaining optimization strategies for equalities having the same column on both sides.
  - Examples: Demonstrating optimization on sequences of predicates, such as pairs of employees in the same department.

Refinement Algorithms (Continued)

Non-equality:

- Definition: Non-equalities as predicates of the form p: $t.A \neq t'.B$.
- Approach: Similar to equalities but with a difference in building the output.
- Process: Building a hash table, handling empty and non-empty tids' cases.
- Handling cases where tids' is non-empty, involving set differences.
- Hybrid TIDs in Action.
  - Leveraging functional dependencies for refinement optimizations.
  - Example DC φ : $\neg(t.\text{StartDate} = t'.\text{StartDate} \wedge t.\text{Salary} \neq t'.\text{Salary})$:
  - StartDate $\rightarrow$ Salary in functional dependency notation.
  - Refinement of Equality on StartDate.
  - Refinement of Non-equality on Salary.
  - Building output.

Refinement Algorithms (Continued)

In-equalities:

- Definition : predicates of the form p: t.A o t′.B, where operator o ∈ {<, ≤, >, ≥}.
- **IEJoin Algorithm**
  - Overview of the IEJoin algorithm for processing two inequalities.
  - Sorting phase, permutation arrays, offset arrays, and bitmap marking.
- **Hash-Sort-Merge (HSM) Algorithm**
  - Sort-merge approach, building hash tables, and interleaved linear scans.
  - Incremental output building, logical union operations, and denser tids.
- **Binning Hash-Sort-Merge (BHSM) Algorithm**
  - Introduction of Novel BHSM.
  - Extending HSM with binning technique for column value ranges.
- **Optimizations**
  - Two optimizations for single-column inequalities with reflexive pairs.

Refinement Planning

- Emphasis on the cost incurred from iterating through hash table entries
- Large hash tables are less cache-effective due to higher data movement.
- Small hash tables perform better as they fit cache more efficiently.

FACET for Single Equality

- Demonstrates how cardinality affects performance in refinement efficiently.

Prioritizing Equalities

- Equalities incur low evaluation costs and have higher selectivity.
- Prioritization based on column cardinality: focus on columns with least estimated cardinality.
- GreedyHLL approach for multiple single-column equalities considering intermediate size and evaluation costs.

Handling Inequalities

- Use of IEJoin algorithm, effective for single equality or inequality.
- Decision between HSM and BHSM for single inequality.
- Pair of refinements or single refinement with IEJoin for different scenarios.

Refinement Planning (Continued)

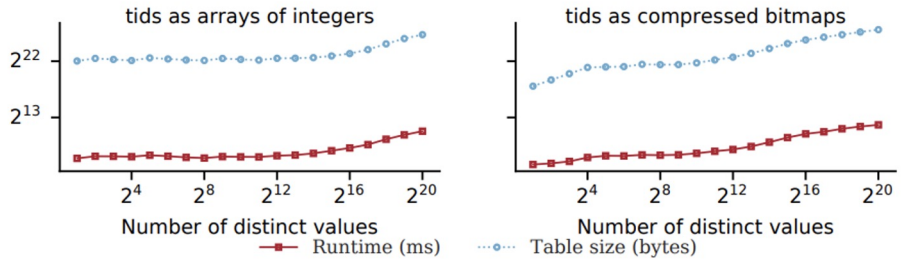


Figure 2: The impact of column cardinality on hash table sizes and equality refinement runtime.

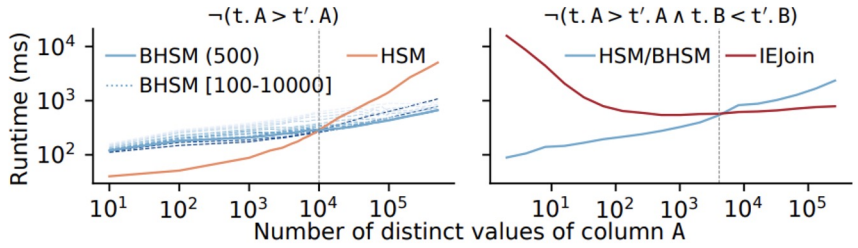


Figure 3: The different impacts of column cardinality on the runtime of `HSM`, `BHSM` and `IEJoin`.

Table 2: Datasets summary and denial constraints used in our experiments.

Dataset	Number of rows	Column Cardinalities	DC number	Denial constraint
Tax	10M	Low, High	φ_4	$\neg(t.\text{AreaCode} = t'.\text{AreaCode} \wedge t.\text{Phone} = t'.\text{Phone})$
Tax	10M	Medium, High	φ_5	$\neg(t.\text{ZipCode} = t'.\text{ZipCode} \wedge t.\text{City} \neq t'.\text{City})$
Tax	10M	Low	φ_6	$\neg(t.\text{State} = t'.\text{State} \wedge t.\text{HasChild} = t'.\text{HasChild} \wedge t.\text{ChildExemp} \neq t'.\text{ChildExemp})$
Tax	10M	Low, Medium, High	φ_7	$\neg(t.\text{State} = t'.\text{State} \wedge t.\text{Salary} > t'.\text{Salary} \wedge t.\text{Rate} < t'.\text{Rate})$
Flights	3.6M	Low, Medium	φ_8	$\neg(t.\text{Origin} = t'.\text{Dest} \wedge t.\text{Dest} = t'.\text{Origin} \wedge t.\text{Distance} \neq t'.\text{Distance})$
Flights	3.6M	Low, Medium, High	φ_9	$\neg(t.\text{Orgin} = t'.\text{Origin} \wedge t.\text{Dest} = t'.\text{Dest} \wedge t.\text{Flights} > t'.\text{Flights} \wedge t.\text{Passengers} < t'.\text{Passengers})$
TPC-H	6M	Medium, High	φ_{10}	$\neg(t.\text{Customer} = t'.\text{Supplier} \wedge t.\text{Supplier} = t'.\text{Customer})$
TPC-H	6M	Medium	φ_{11}	$\neg(t.\text{Receiptdate} \geq t'.\text{Shipdate} \wedge t.\text{Shipdate} \leq t'.\text{Receiptdate})$
TPC-H	6M	Low, High	φ_{12}	$\neg(t.\text{ExtPrice} > t'.\text{ExtPrice} \wedge t.\text{Discount} < t'.\text{Discount})$
TPC-H	6M	Low, High	φ_{13}	$\neg(t.\text{Qty} = t'.\text{Qty} \wedge t.\text{Tax} = t'.\text{Tax} \wedge t.\text{ExtPrice} > t'.\text{ExtPrice} \wedge t.\text{Discount} < t'.\text{Discount})$
IMDB	2.5M	Low, High	φ_{14}	$\neg(t.\text{Title} = t'.\text{Title} \wedge t.\text{ProductionYear} = t'.\text{ProductionYear} \wedge t.\text{Kind} \neq t'.\text{Kind})$
IMDB	5.8M	Low, High	φ_{15}	$\neg(t.\text{Title} = t'.\text{Title} \wedge t.\text{Name} = t'.\text{Name} \wedge t.\text{CharName} = t'.\text{CharName} \wedge t.\text{Role} = t'.\text{Role})$

Experimental Evaluation

- Comparison of FACET with DBMS-X and open-source DBMSs: PostgreSQL, MonteDB, and DuckDB.
- Use of 12 DCs (Data Constraints) and 4 datasets in experiments.

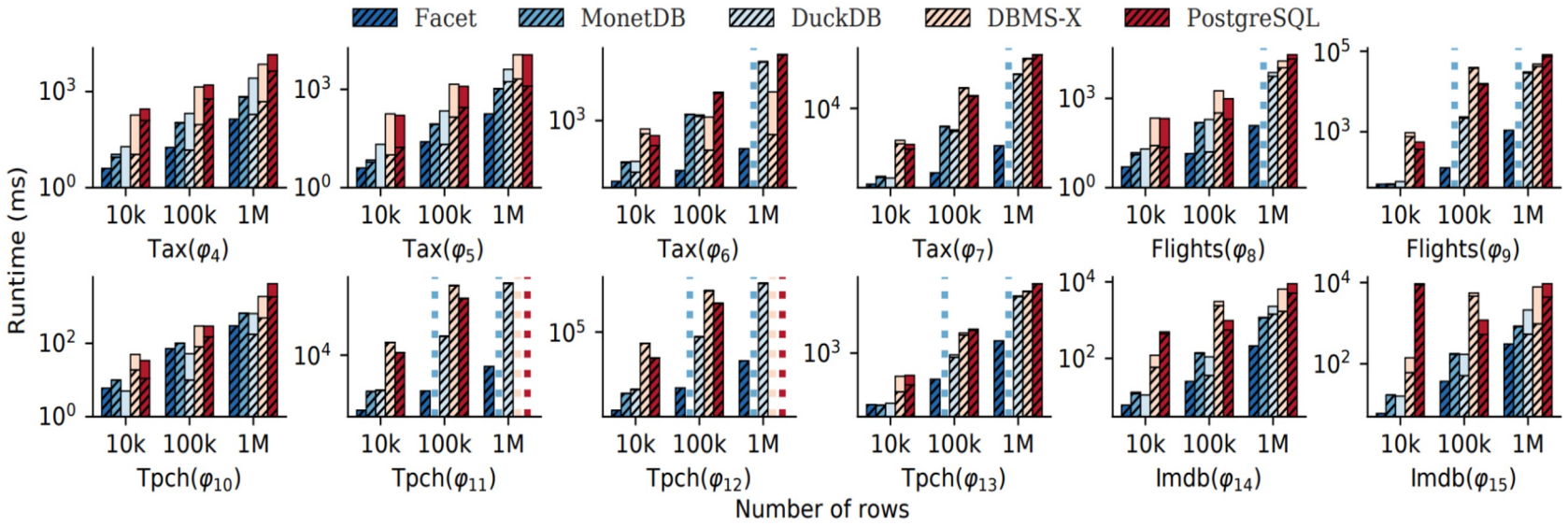


Figure 4: Runtime comparison of FACET to four distinct DBMSs using SQL queries. The dashed areas in each bar represent the violation detection time, whereas the solid areas represent the indexing construction time (only for the DBMSs). The dotted lines are cases where the respective DBMS either exceeded the time limit of four hours or ran out of memory.

Experimental Evaluation(Continued)

- Runtime comparison of FACET and DBMS approaches across different data scales.
- Results indicating FACET's superior performance and scalability and FACET completes error detection significantly faster than other DBMSs.

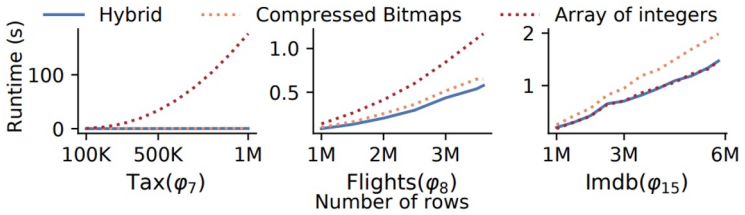


Figure 7: Impact of the types of tids storage on runtime.

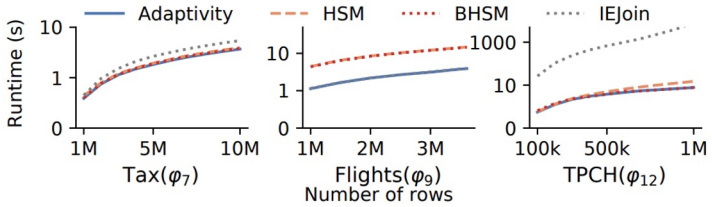


Figure 8: Impact of adaptivity vs. non-adaptivity strategies on the runtime for DCs including inequalities.

Table 3: Runtime speedups of predicate evaluations in the order shown relative to evaluations in the order by FACET.

Predicate evaluation order	Speedup	
	As shown	FACET
t.Phone = t′.Phone ∧ t.AreaCode = t′.AreaCode	1.00	**1.19**
t.Passengers < t′.Passengers ∧ t.Flights > t′.Flights	1.00	**1.22**
t.Salary > t′.Salary ∧ t.Rate < t′.Rate	1.00	**2.71**
t.ExtPrice > t′.ExtPrice ∧ t.Discount < t′.Discount	1.00	**17.02**
t.Flights > t′.Flights ∧ t.Origin = t′.Origin	1.00	**25.78**
t.Flights ≠ t′.Flights ∧ t.Origin = t′.Origin	1.00	**41.36**

- Figure 7: Performance impact of different tids (transaction IDs) storage types.
- Figure 8: Comparison of static inequality algorithms vs. FACET's adaptive approach.
- Table 3: Comparison of baseline evaluation order with FACET's reverse order approach.

Experimental Evaluation(Continued)

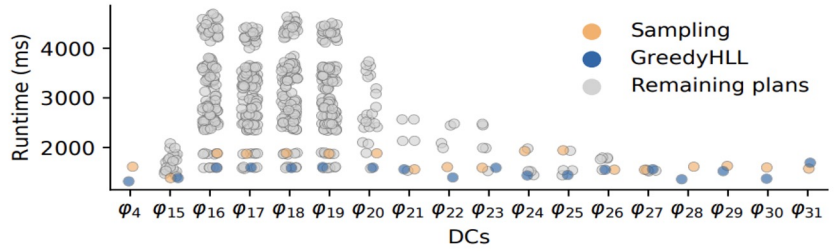


Figure 9: Runtime of FACET using the plans of GreedyHLL vs. runtime of FACET using all other possible plans.

Focus on uniqueness constraints made of single column equalities and Analysis of GreedyHLL approach effectiveness. (fig. 9)

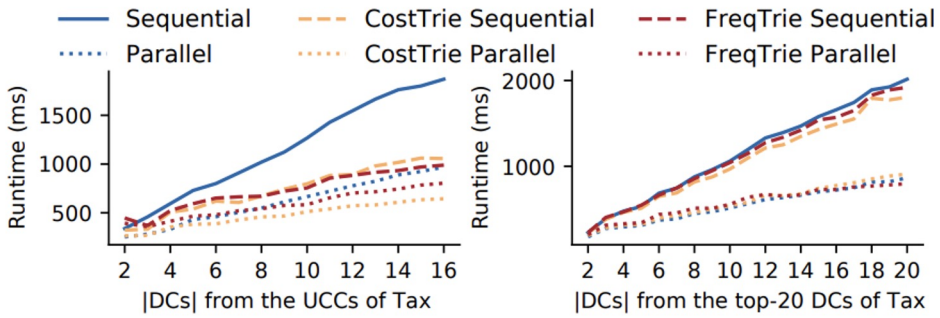


Figure 10: Performance of the multi-DC execution modes.

Examination of FACET's performance with multiple DCs. Utilization of uniqueness constraints sharing many predicates.
(fig. 10)

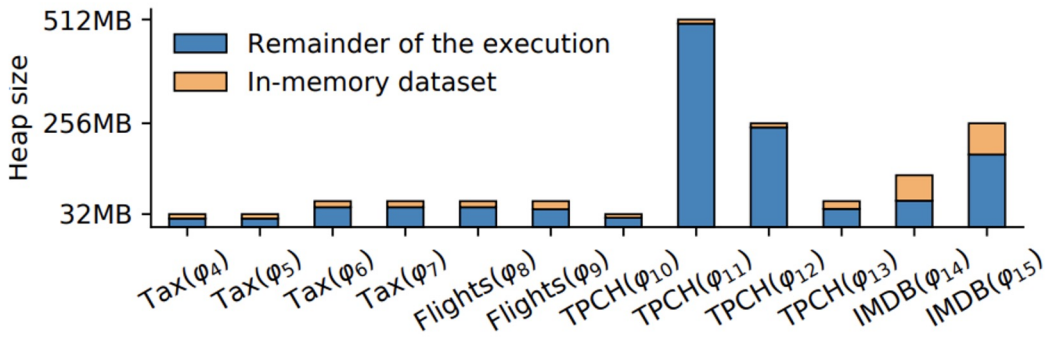


Figure 12: Memory requirement of FACET.

Final experiment assessing the heap size needed for FACET's successful execution on 1M records. (fig. 12)

Key Insights

- FACET exhibits superior performance and scalability in error detection across various datasets.
- Design decisions within FACET, such as the reverse order evaluation and adaptive inequality handling, contribute significantly to its efficiency.

Conclusion

- Emphasizes FACET's capability in detecting data errors through DC violations, addressing key challenges in data management.
- Highlights the use of column sketches for efficient predicate evaluation and resource optimization.
- Discusses the introduction of new algorithms within FACET, enhancing its performance and effectiveness.
- Details how FACET integrates seamlessly with constraint-based data cleaning pipelines, showcasing its adaptability.
- Explores potential extensions of FACET into query processing, including the integration of cleaning operators for real-time DC violation management.

Thank You