Towards Practical Differential Privacy for SQL Queries

Noah Johnson, Joseph P. Near, Dawn Song

UC Berkeley

Outline

1. **Discovering** real-world requirements
2. **Elastic sensitivity** & calculating sensitivity of SQL queries
3. **Our experience:** lessons & challenges

Part 1
Discovering Real-world Requirements

Our collaboration with Uber

- Uber's goal: **deploy differential privacy**
  - Internally (for some analysts)
  - Externally (for partners & regulators)
- Our goals
  - Explore **real-world requirements** for differential privacy
  - Build **open-source systems**

Previous work on differential privacy for analytics: insufficient for real-world applications

Previous work: either…

- Theoretical (does not explore practical applications)
- Targets specialized analytics tasks
  - Google RAPPOR: browsing statistics
  - Apple: keyboard & emoji trends

Result: little use in real-world analytics environments

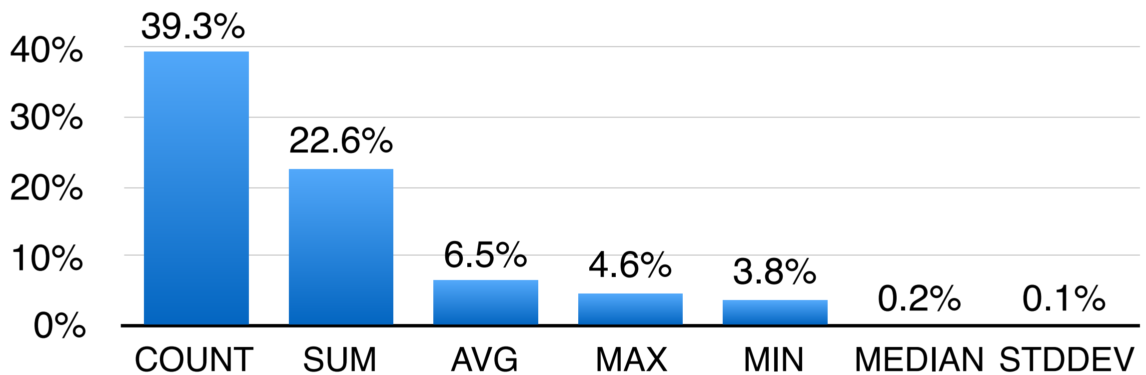
- No practical, scalable systems for DP in analytics

Empirical study: understanding real-world data analytics

- Conducted **large-scale empirical study** of real-world analytics queries
- Dataset: **8 million SQL queries** written by data analysts at Uber
  - Covers wide range of use cases: fraud detection, marketing, business metrics, etc.
- Goal: identify **DP requirements** for **real-world workload**

Empirical study results

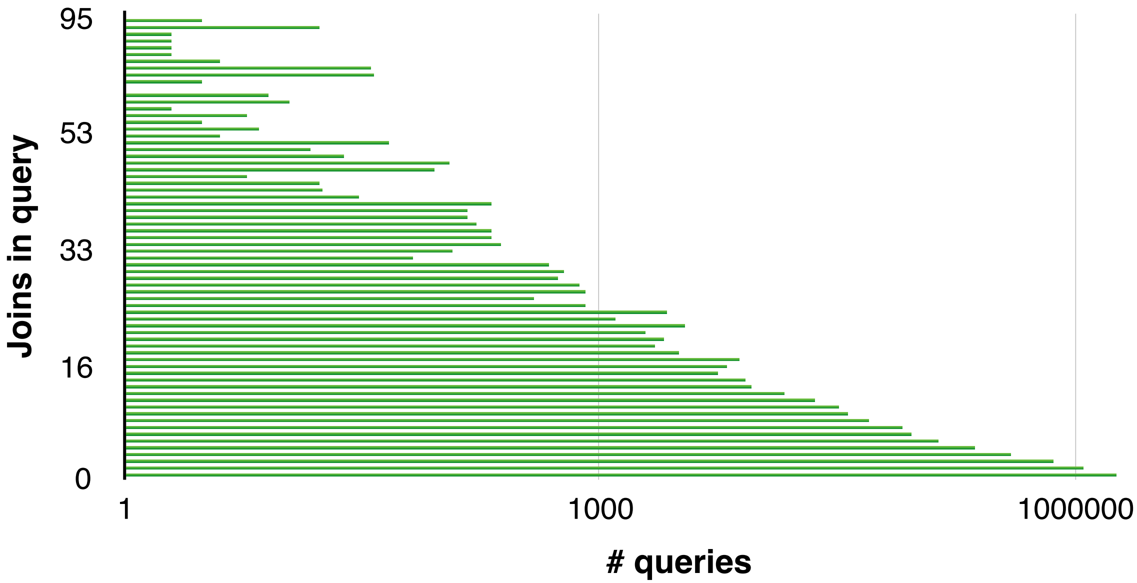
The most common aggregations are **COUNT**, **SUM, AVG, MAX,** and **MIN**:



➔ **Most existing DP mechanisms support only counting queries**

Empirical study results

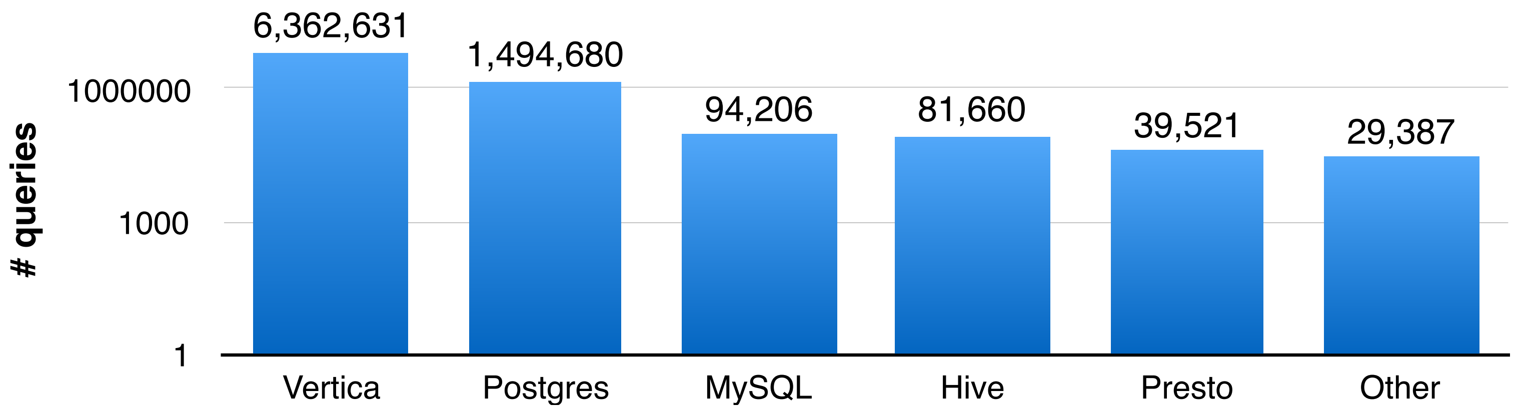
62% of queries use JOIN, and some queries use **many joins**:



➔ **Very few existing mechanisms support join**

Empirical study results

Many different databases in use



➔ **Existing approaches require modifying/replacing DB**

Part 2

Elastic Sensitivity & Analyzing SQL Queries

Global sensitivity vs. local sensitivity for joins

Global sensitivity

- **Unbounded** for queries with joins
  - Single added join key in one table could match an unbounded number of keys in another

Local sensitivity

- **Bounded** for queries with joins
  - Data in true database bounds number of possible new matches
- Computationally expensive
  - Must consider every possible change to true database

Elastic sensitivity

Upper bound on local sensitivity

- Efficient, compositional calculation from query

Supports queries with equijoins

- **Insight**: increase in size of joined relation tightly bounded by **multiplicities of join keys**
- Key multiplicities **queried from database** in advance

Supports more than just count

- Works well for COUNT
- Works less well for SUM

Example: elastic sensitivity of join

SELECT COUNT(*) FROM A **JOIN** B ON A.k = B.k

k	v
1	a

A

k
1
1

B

k	v
1	a
1	a

A JOIN B

Duplicate join key 1 causes **duplicate rows** in joined relation

k	v
1	a
1	b

A

k
1
1

B

k	v
1	a
1	a
1	b
1	b

A JOIN B

Maximum change in COUNT: add another 1 to A

Local sensitivity = 2

In general: local sensitivity bounded by **maximum multiplicities** of k in A and B

A static analysis framework for SQL queries

Built a practical framework for analyzing real-world queries

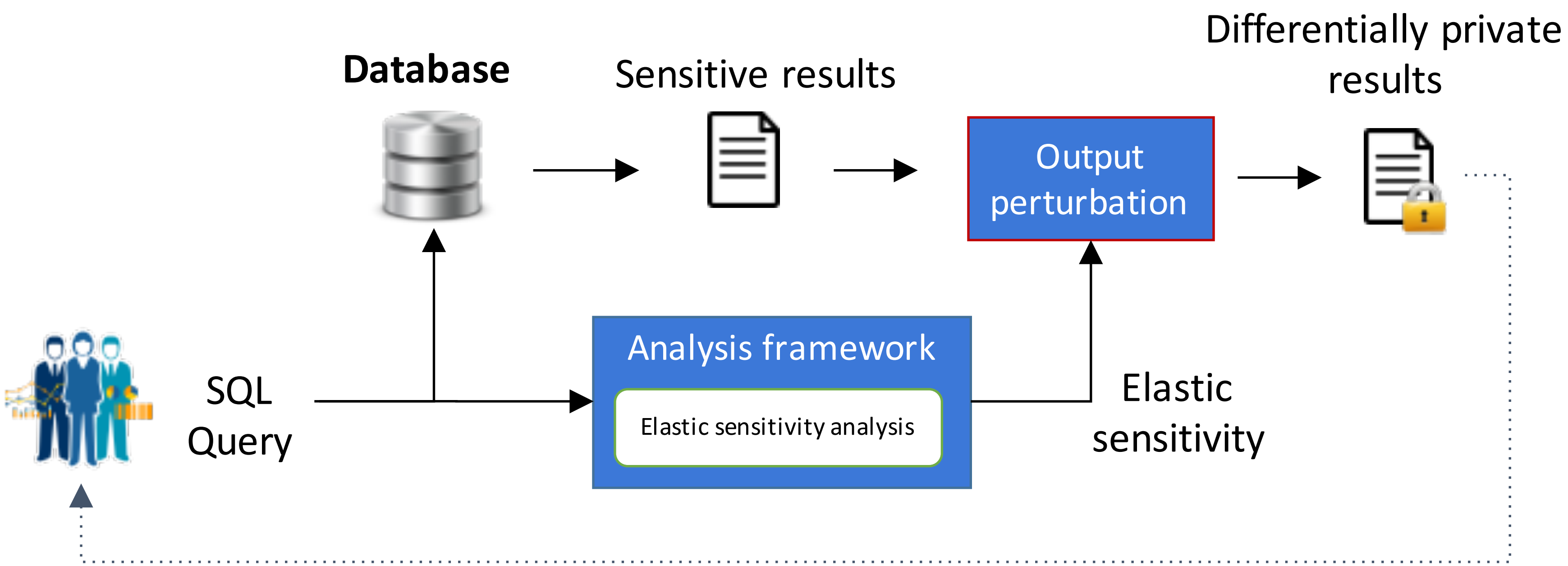
Challenge: these queries are complex

Our framework:

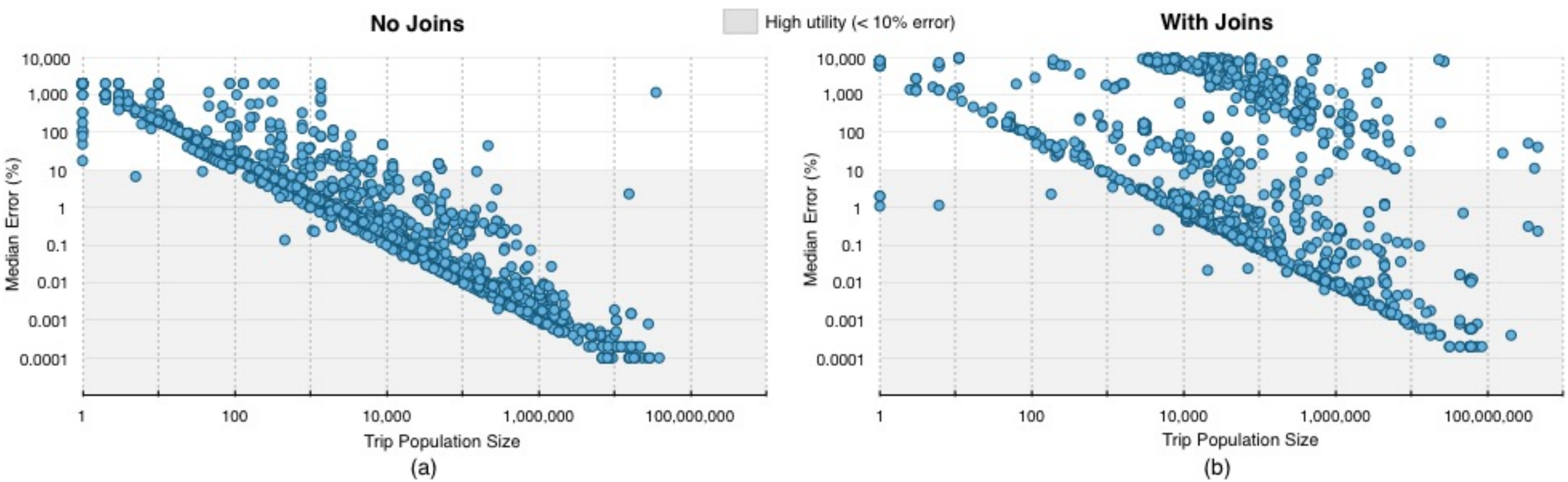
- Solve complexity **once**
- Enable **many different analyses**

```sql
--trips by day
with daily as (
select
    date_trunc('{{interval}}', request_at)::date as day
    ,city_id as city_id
    ,city_name
    ,count(distinct client_uuid) as total_eaters
    ,count(distinct case when promo_trip = 1 then client_uuid end) as total_eaters_on_promo
    ,count(distinct case when first_trip = 1 then client_uuid end) as first_eaters
    ,count(distinct case when first_trip = 1 and promo_trip = 1 then client_uuid end) as first_eaters_on_promo

from (
    select
        e.client_uuid
        ,e.request_at
        ,e.uuid
        ,e.city_id
        ,c.city_name
        ,case when e.rank = 1 then 1 else 0 end as first_trip
        ,e.promo_trip

    from (
            select
                    t.client_uuid
                    ,t.request_timestamp_local as request_at
                    ,t.uuid
                    ,t.city_id
                    ,rank() over (partition by t.client_uuid order by t.request_timestamp_local asc) as rank
                ,case when ap.trip_uuid is not null then 1 else 0 end as promo_trip

            from fact_trip t
            join dim_client cl on t.client_uuid = cl.user_uuid and cl.is_uber_email = 'False'
            left join analytics_promotion ap on t.uuid = ap.trip_uuid

            where 1=1
            and t.city_id in ({{city_ids}})
            and t.vehicle_view_id in ({{eats_vvid}})
            and t.status = 'completed'
            and t.request_timestamp_local between '2014-08-01' and '{{end}}'
            group by 1,2,3,4,6
        ) e

    join dim_city c on e.city_id = c.city_id and c.city_id in ({{city_ids}})

    where 1=1
    and e.request_at between '{{start}}' and '{{end}}'
    ) all_by_client

group by 1, 2, 3
)

--trips taken in the past 30 days

, l30_daily as (
select
    day
    ,city_id
    ,count(distinct (case when l30_trips >= 1 then client_uuid end)) as oneplus
```

Differential privacy for SQL queries using Elastic Sensitivity



Empirical evaluation results



Dataset: 9862 Uber queries, run on production database

Part 3
Lessons Learned & Future Challenges

Value of close collaboration

- Opportunity to examine **real use cases**
  - Dataset of queries: what analysts actually *did*
- Insight into **privacy goals** in the real world
  - e.g. concern about external *and* internal sharing
- Discover **requirements** & infrastructure restrictions
  - e.g. we *really can't* modify the database engine

Challenges of close collaboration

- **Analysts** skeptical about need for privacy protections
  - Concerned about utility
  - Believe privacy is already protected
  - e.g. machine learning teams believe models protect privacy
- **Privacy team** unsure of privacy goals
  - Belief that de-identification is enough, *or*
  - Differential privacy seen as a silver bullet
  - Would like to "have differential privacy" all in one go
- **Infrastructure teams** want a one-size-fits-all solution
  - Multiple solutions = more work

Conclusions

- Perfect deployment will take **time, experimentation**
  - Early versions will be limited
  - There will be bugs
- We can **accelerate** the process
  - Encouragement
  - Constructive engagement
- We should **encourage transparency**
  - Secrecy encourages bugs, discourages adoption

https://github.com/uber/sql-differential-privacy

https://arxiv.org/abs/1706.09479

jnear@berkeley.edu

Thank you!