10-30-2020

Extending APEx (Accuracy-Aware Differentially Private Data Exploration) to Multiple Table Queries

Karmanjot Singh
University of Windsor

Extending APEx (Accuracy-Aware Differentially Private Data Exploration) to Multiple Table Queries

By

Karmanjot Singh

A Thesis
Submitted to the Faculty of Graduate Studies
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science
at the University of Windsor

Windsor, Ontario, Canada

2020

Extending APEx (Accuracy-Aware Differentially Private Data Exploration) to Multiple Table Queries

by

Karmanjot Singh

APPROVED BY:

R. Razavi Far
Faculty of Engineering

D. Alhadidi
School of Computer Science

S. Samet, Advisor
School of Computer Science

July 16, 2020

DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

With the recent advances in data analytics and machine learning, organizations are becoming more and more interested in utilizing these techniques to generate insights from the data they have. But the biggest hurdle, especially for those organizations that collect private information, is that it becomes challenging to share their data with data analysts without compromising the privacy of the data. Differential privacy helps to share private data with provable guarantees of privacy for individuals. Even though differential privacy is very good at preserving privacy, it still poses a lot of burden on data analysts to understand differential privacy and its intricate algorithms. Moreover, this also doesn't give any accuracy guarantees to the data analyst. Keeping this in mind, APEx (Accuracy-Aware Differentially Private Data Exploration) was introduced in May 2019, which allows data analysts to run a sequence of queries keeping privacy and accuracy in place. APEx was implemented for only one table in the database. In this research, it is extended and evaluated on multiple table queries.

DEDICATION

*Dedicated to my **Parents** and all the **Teachers** who played a vital role throughout my life as a student.*

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisor **Dr. Saeed Samet** for his continuous support and input throughout my Masters degree. I could not have asked for a better mentor. I would also like to thank the internal reader **Dr. Dima Alhadidi** and external reader **Dr. Roozbeh Razavi-Far** for there time and efforts.

Also I would like to thank my co-op supervisor **Arman Didandeh** for always encouraging me to do things which I would have never tried myself. Special thanks to my brother **Nirvair** and my friend **Yagnik** for always being there.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

With the recent advances in data analytics and machine learning, organizations are becoming increasingly interested in utilizing these techniques to generate insights from the data they have. But the biggest hurdle, especially for those organizations that collect private information, is that it becomes challenging to share their data with data analysts without compromising the data's privacy. Moreover, the released data, when matched with the previous releases or the data released from other organizations, can reveal sensitive data about the participants of the data-set. There are a couple of examples that happened in the past, like the Netflix movie recommendation competition [23] or the re-identification of the medical record of the governor of Massachusetts [2]. Both of these incidents happened by matching the data releases of two different organizations. Because of this, various methods exist like anonymizing the data, data encryption, secure multi-party computation, but none of them are good enough or are not easy to implement on the complex datasets. In 2006, Cynthia Dwork et al. [4] came up with something called differential privacy. It helps to ensure that data privacy remains intact even after publishing the aggregate information about the data. Differential privacy does this by adding Laplacian noise to the actual result of the query. Since then, differential privacy has gained a lot of popularity and has been implemented by organizations like Google, Uber, and Apple to securely share the private data that they collect from the consumers of their products.

1.1 Differential privacy

Differential privacy is used to share data publicly without risking the confidentiality of the participants of the data. Another way to think about differential privacy is that it uses various mechanisms to add noise to the aggregate information about the statistical database. It is difficult for the adversary to infer private data from the participants of the dataset. For example, various government agencies use differentially private algorithms to publish public data by maintaining the confidentiality of survey responses. Even companies use differentially private algorithms to collect information about user behavior.

In other words, we can consider an algorithm to be differentially private if anyone seeing it's output cannot identify if some record belongs to a particular individual. It also does not adequately protect from identification and reidentification attacks; it resists such attacks. [7]

Differential privacy has its origins from cryptography, and a lot of its language is from cryptography.

Data Analysts often work on data released by organizations like Healthcare, Banking, and Consumer to help them make informed decisions and gather insights to better their processes and also to serve their customers better. But it does not come without putting private data on the risk of being used in a harmful way. Someone can easily use someone's banking data for money laundering. They can identify someone's private health data and make it public as it happened in the case of the governor of Massachusetts. This is where differential privacy[8] comes very handy to stop these kinds of attacks and financial risks for these organizations and allow them to publicly release their data for Data analysts to review and help them make decisions. This means that even if we modify a single record, it does not significantly change the result than the result that we get without making that change. For example, suppose if we have only one person with a particular characteristic present in the dataset. If we release this data publicly without differential privacy, then the adversary can easily run queries to check how many people are present who do not have that char-

acteristic and can easily get the private data of that person. But if we release the data using a differentially private algorithm, then it will be difficult for adversary to find someone's private data even if the adversary has prior knowledge of that record. We call the change of one record in two similar datasets to be bounded with ϵ, where ϵ is privacy-loss budget.

1.2 Why do we need Differential Privacy?

Government organizations have long been collecting information about individuals or establishments and sharing that data for multiple reasons. But it is not possible to share all of the data as it is, even though they share only the statistics about the data because it can lead to private data. For example, in 1790, the United States conducted a census where they collected information about people living in the United States and released statistics related to sex, age, race. Not only the U.S. but every country conducts these types of data collections to get a better understanding of their population and then publish the statistics about this data. This data is then used later on for helping governments to make policies and to decide where to project their funding, the marginalized areas of the society, and similar decisions. They share this data with private organizations as well to help them with their choices. For example, banks use this data to decide the risk levels of a particular loan and real estate companies use it to determine if it is safe to sell houses to someone. For public data distribution, government agencies have long been using data suppression to help preserve data that can be compromised. For example, a record with information about only one person or one company is removed to maintain confidentiality.

Since the 1950s and 1960s, there has been rapid adoption of electronic information systems by statistical agencies. This made the public distribution of statistical data more difficult even after data suppression. For example, there is a business whose sales numbers have been suppressed because of the risk of privacy disclosure, but if the same numbers were used in total sales of a region, then the adversary can easily estimate the actual numbers by subtracting the other sales from the total sales of that

region. Not only this, but there are various combinations of additions and subtractions that might put the privacy of the participants of the dataset. The combinations like this increase exponentially with the increasing number of publicly shared data, and this problem become more prominent when we add an interactive query system into the picture.

1.3 Randomized Response (Plausible Deniability)

To understand differential privacy, an understanding of a randomized response is fundamental. Some readers struggle with understanding differential privacy without knowing a randomized response or plausible deniability. The randomized response is a survey technique used by surveyors to collect sensitive information from the survey takers while maintaining the confidentiality of their responses. In a randomized response mechanism, n individuals answer a survey with one binary question. The truthful answer for individual i is $x_i \in \{0, 1\}$. The surveyor gives every individual an unbiased coin and tells them to answer "Yes" or one if it comes tail and to answer truthfully in case of heads. In this way, every individual gets the opportunity to potentially lie and hence the plausible deniability, which helps in keeping their responses confidential. And at the same time, as we know that there is a half-half probability of tails and heads, we can easily double the percent of "Yes" or "No" responses to get the actual picture. For example, if we ask the survey takers if they are taking a particular drug, if the number of survey takers who are taking the drug is 20%, then the actual percent of survey takers who are taking that drug would be the double of the number that we got from the survey that is 40%.

1.4 ϵ-differential privacy

Cynthia Dwork et al.[4] introduced the concept of ϵ-differential privacy in 2006. The main idea behind this work was that if some individuals did not take a survey, it would not be possible to compromise that individual's privacy. Hence, in differential

privacy, the focus is on giving each individual the amount of privacy that would result from not taking the survey. That is, the transformations and the aggregate function results should not change significantly by removing one individual's responses to the survey.

Differential Privacy mechanisms achieve this, adding noise before returning the aggregate or other transformation functions run on the survey responses. Now, as we increase the number of survey takers, we would have to add less noise and vice versa.

1.4.1 Definition of ϵ-differential privacy

Definition 1 *(ϵ-Differential Privacy) A randomized mechanism $M : D \rightarrow O$ satisfies ϵ-differential privacy if*

$$Pr[M(D)\epsilon O] \leq e^{\epsilon} Pr[M(D^{'})\epsilon O] \tag{1}$$

for any set of outputs $O \subseteq O$, and any pair of neighboring databases D,$D^{'}$ such that $|D \backslash D^{'} \cup D^{'} \backslash D| = 1$. [9]

Smaller the value of ϵ, more is the privacy, but the amount of noise added will also be high. So, the differential privacy mechanisms try to find a middle ground with the smallest possible value of ϵ. The noise added is also not too high, and the result is usable to the data analyst.

1.5 Laplace mechanism

The Laplace mechanism adds noise from the Laplace distribution, as shown in Figure 1.5.1. In Laplace distribution, the mean is zero, and the standard deviation is $\sqrt{2}\lambda$.

Let us see an example to understand why adding noise from Laplace distribution gives us differentially private results. Let us say we have a dataset that contains mental health data, and there is an attacker named Eve, and she wants to see if her target, Bob, is receiving counseling for alcoholism or not. If the query's result

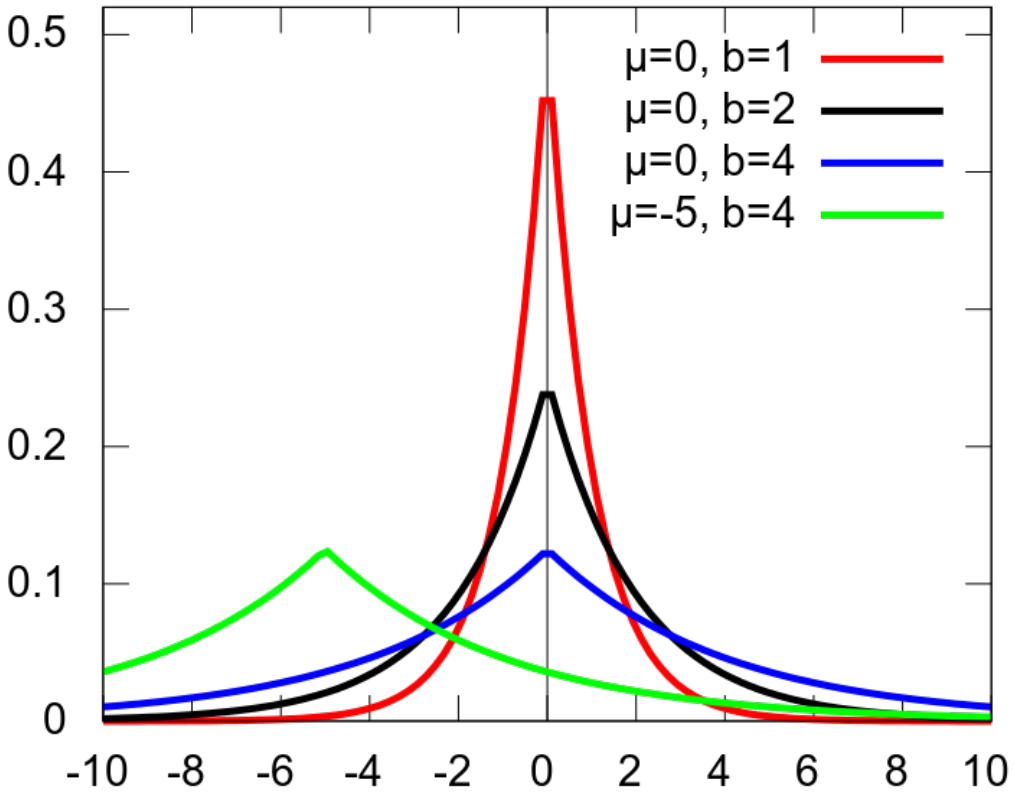

Fig. 1.5.1: Laplace Distribution

comes up as 48, then Eve will know that Bob is receiving counseling for alcoholism. Otherwise, if the query's result is 47, then he is not receiving counseling for alcoholism.

Now since we are using the Laplace mechanism, it does not matter what the actual result of the query is, the mechanism is going to add noise from Laplace Distribution. So, it is going to return results somewhat near 47 or 48. It may be 49, 46, or maybe even smaller like 44 or higher like 51. So, it is practically impossible for Eve to be very sure whether the true answer was 47 or 48. In other words, her belief about Bob (whether he is in counseling for alcoholism or not) will not meaningfully change after running the query.

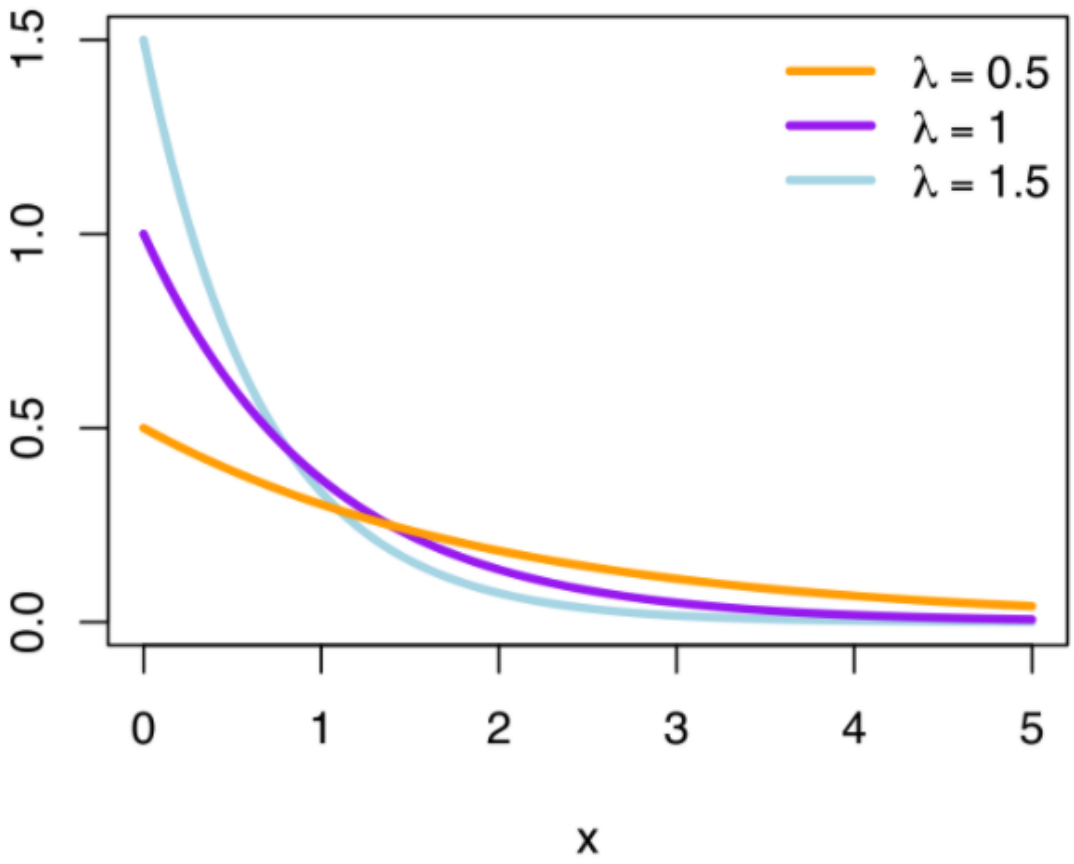

Fig. 1.6.1: Exponential Distribution

1.6 Exponential mechanism

It is used for functions that do not return a real number. For example, "What is the most common nationality in a particular room?" Chinese/Indian/Canadian.. This is also used when a small change in the output leads to invalid outputs.

Exponential mechanism is most general approach which captures all possible differential privacy mechanisms. In fact, Laplace distribution is a symmetric exponential distribution (Figure 1.6.1).

1.7 Objectives and Contribution

Even though lot of work has been done on differential privacy and it seems very promising, but it still poses a lot of burden on data analysts to understand differential privacy and manage privacy budgets accordingly. Moreover, this also does not give any accuracy guarantees to the data analyst. Keeping this in mind, Chang Ge et al.[9] introduced a novel system called APEx in May 2019, which allows data analysts to run a sequence of queries keeping privacy and accuracy in place. APEx translates queries and accuracy bounds to differentially private algorithms with the least privacy loss. By doing this, the data analyst gets the answers to its queries, keeping accuracy bounds in place, and the privacy budget set by the data owner is also not exceeded. This experiment has a lot of potential for improvements; for example, Change Ge et al.[9] did these experiments only on a relational database with only one table. In this work, I extended this to multiple tables given that in the real world, data spread across numerous related tables.

The following contributions were made in this research :

- The main idea behind this work is to extend and evaluate APEx which currently is based on Single table to multiple table database schema as mentioned by the author of the paper[9] himself :

  "We consider the sensitive dataset in the form of a single-table relational schema R(A1,A2, . . . ,Ad), where attr(R) denotes the set of attributes of R. Each attribute Ai has a domain dom(Ai). The full domain of R is dom(R) = dom(A1)x....x dom(Ad), containing all possible tuples conforming to R. An instance D of relation R is a multiset whose elements are tuples in dom(R). We let the domain of the instances be D. Extending our algorithms to schemas with multiple tables is an interesting avenue for future work."

- I have used a multiple table schema "Employees".

- Then I made code changes in APEx repository so that when the user run a WCQ, ICQ or TCQ query, instead of the data coming from only one table

(which was the case in existing work), the data is coming from multiple tables.

- Then I did a comparison study when we run the APEx on single table vs. Multiple table.
- I incorporated other error measures such as MSE (Mean Square Error), Mean Absolute Error (MAE), Mean Absolute Percentage Error.

CHAPTER 2

Related Work

Data exploration is becoming more and more important as more and more data is generated by the organizations. Since the last decade, there has been a massive explosion in digital data, also referred to as Big Data. With this vast availability of data, it gives an excellent opportunity to data analysts to better improve their analysis. Datasets consist of both public and private data, and exploring them involves a lot of operations such as summarization, building histograms, and building models for machine learning. Incorporating private data into data analytics provides a high value to the data analytics project. Still, often data owners hesitate to give access to the private data because of the risk of data leakage. It could be because of a lot of reasons such as, they do not trust the data analysts or the risk of data leakage is higher than the benefit from the data analytics. For example, Facebook recently announced that they would allow the academic study of their data to find the correlation between social media and politics, specifically in elections. But their main concern was the privacy of their data [15].

Differential privacy is something that has come up as a solution to the problem of public data distribution or at least to an outside organization. It is because of various reasons such as

- It provides a sense of data privacy even when prior knowledge about the data is available.
- It is mathematically proven method to preserve privacy of individual records when aggregate data results are released.

- Even though there is still some information leakage, it can help to keep the leakage bounded.

That is why differential privacy is gaining popularity among data owners such as the US Census Bureau [12, 20, 31], Google [10], Apple [11], and Uber [13].

2.1 Privacy Integrated Queries (PINQ)

Frank McSherry first proposed Privacy INtegrated Queries (PINQ) in his paper[21]. It is a common platform used for differentially private data analysis. It provides an interface to data that looks very much like LINQ (C#'s "language-integrated queries"). All-access through the interface to the data is guaranteed to be differentially private. In PINQ interface, data analysts who are non-privacy experts, write arbitrary LINQ code against datasets in C#.

```
var data = new PINQueryable<SearchRecord>(... ...);

var users = from record in data
            where record.Query == argv[0]
            groupby record.IPAddress;

Console.WriteLine(argv[0] + ':' + users.Count(0.1));
```

Rather than providing direct access to the underlying data, each private data source is wrapped in a **PINQueryable** object. This **PINQueryable** object is then responsible for mediating accesses to the underlying data, remembering how much privacy budget is left, deducting from the budget whenever an aggregation operator is applied to this **PINQueryable** object and denying access once the given privacy budget is exhausted.

Table 2.1.1 summarizes the main data operations supported by PINQ and their privacy implications. There are two types of operations: aggregations and transformations. Aggregations return the aggregate value after adding noise per differential

Aggregations

Count	Std. deviation of added noise is $\sqrt{2}/\epsilon$.
Sum	Std. deviation of added noise is $\sqrt{2}/\epsilon$.
Average	Std. deviation of added noise is $\sqrt{8}/\epsilon n$, where n is the number of records.
Median	The return value partitions input into sets whose sizes differ by approx. $\sqrt{2}/\epsilon$

Transformations

Where, Select Distinct	No sensitivity increase
GroupBy	Increases sensitivity by two
Join, Concat Intersect	No sensitivity increase for either input
Partition	Privacy cost equals the maximum of the resulting partitions

Table 2.1.1: Main data operations in PINQ.

privacy. Transformations return a new **PINQueryable** object that can be further operated upon. They can amplify the sensitivity of subsequent queries, so that aggregations run with one value of ϵ may deplete many multiples of ϵ from the privacy budget. PINQ ensures that any amplification is properly accounted for. Importantly, the logic within a transformation can act arbitrarily on the sensitive records.

The semantics of the transformations are similar to SQL, with two significant exceptions. First, the join operation in PINQ is not a standard equijoin, in which one record can match an unbounded number of other documents. Instead, records in both dataset are grouped by the key they are being joined on so that the Join results in a list of pairs of groups. This restricts each pair to have a limited impact on aggregates (that of a single record) despite being arbitrarily large, but it does enable differential privacy guarantees which would not otherwise exist.

A second difference is a Partition operation that can split a single protected dataset into multiple protected datasets, using an arbitrary key selection function. This operation is essential because the privacy cost to the source data set is the maximum of the costs to the multiple parts, rather than their sum. We can, for example, partition packets based on the destination port, and conduct independent analyses on each piece while costing only the maximum.

As the discussion above illustrates, the privacy cost of analysis depends not only

what the analysis aims to output but also on how it is expressed. PINQ is essentially a programming language, and the space of analyses that can be expressed is limited mainly by the analyst's creativity.

An Example

Suppose we want to count distinct hosts that send more than 1024 bytes to port 80. This computation, which involves grouping packets by source and restricting the result based on what we see in each group, can be expressed as:

```
packets = new PINQueryable<Packet>(trace, epsilon);
packets.Where(pkt => pkt.dstPort = 80)
       .GroupBy(pkt => pkt.srcIP)
       .Where(grp => grp.Sum(pkt => pkt.len) > 1024)
       .Count(epsilon_query);
```

The Packet type contains fields that we might expect, including sensitive areas such as IP addresses and payloads. The raw data lies in the trace. The total privacy budget for the trace is epsilon, and the amount to be spent on this query is epsilon_query. The analyst can run multiple queries on the data as long as the total privacy cost is less than epsilon. The expressions of the form x $\Longrightarrow$ f(x) are anonymous functions that apply f to x.

2.2 Weighted Privacy Integrated Queries (wPINQ)

Like Privacy Integrated Queries (PINQ) [21], Weighted PINQ is a declarative programming language over datasets that guarantees differential privacy for every program written in the language. I refer the reader to [21] for details on the design philosophy behind these languages, and to [27] for full technical information on Weighted PINQ's operators. Here I only provide a short overview of how queries are written in Weighted PINQ.

2.2.1 Overview

Weighted PINQ can apply two types of operators to a (secret) dataset: transformations, and noisy aggregations. Transformation operators such as Select, Where, GroupBy, SelectMany, Join, etc. transform a weighted secret dataset and then automatically rescale the resulting record weights to maintain privacy on the total disclosure of the records. After doing these transformation operations, the datasets remain secret. Before releasing the final results to the user, the results must be fed to noisy aggregation operators such as NoisyCount. NoisyCount operator aggregates the weighted confidential records, adds Laplace noise and then only exposes the final results to the end-user. After performing each transformation operator, record weights are scaled-down in such a manner that guarantees differential privacy after noisy aggregation.

2.2.2 Weighted PINQ vs. PINQ

Weighted PINQ is very similar to PINQ in terms of operators. Still, because it operates on weighted records with arbitrary weights (instead of integral weights), it differs from PINQ in few crucial ways:

First, transformations in PINQ that required either scaling up the noise or the privacy parameter ϵ, now scale down the weights associated with records. For example, the operator 'SelectMany', which produces many records (e.g., k (number of records)), scales down each record with a factor of k. The operator GroupBy collects records, results in a group with weight divided by 2, and the operator Join which produces the cross-product of records, with weights rescaled.

The other main difference is a transformation operator to manipulate weights, Shave, which takes a sequence of weights w_i and then transforms each record x with weight w into the set of records $(0, x), (1, x), ...$ with weights $w_0, w_1, ...$, for as many terms as $\sum_i w_i \leq w$. Select is the functional inverse of Shave, which can transform each w_i-weighted indexed pair from (i, x) to x whose weight re-accumulates to $\sum_i w_i = w$.

Finaly, Weighted PINQ's operator NoisyCount now rather than returning a single noisy count, returns a dictionary from records to noised weights. This means that if one looks up the value of a record which is not in the input, a weight of zero is introduced, and then the noise is added. This is in some sense generalization of PINQ's NoisyCount to weights and multi-output "histogram queries" [5]. To reproduce PINQ's NoisyCount we can first map all records to some known value, e.g., true.

2.2.3 Writing "good" queries

Now, this brings us to the conclusion that when a query is expressed using Weighted PINQ operators (i.e., transformations, followed by aggregations), it is sufficient to say that it provides differential privacy. So, what remains after this is to write a "good" Weighted PINQ query. There are two main things that we have to look into while writing a good quality query: its computational complexity and its accuracy. Writing "good" queries requires inventiveness. For example, the below query is an example of a query that provides high efficiency and performance, keeping the loss of privacy to the minimum. It is essential to note on the important thing here that it can be more challenging to write "good" queries that directly measure properties with high sensitivity [5] (e.g., graph diameter). One way to get around this is to combine indirect measurements with probabilistic inference.

```
var deqCCDF = edges.Select(edge => edge.src)
                   .Shave(1.0)
                   .Select((index, srcname) => index);
var ccdfCounts = degCCDF.NoisyCount(epsilon);
```

2.3 Differential Privacy for SQL Queries (FLEX)

Noah Johnson et al. [13] introduced Elastic Sensitivity for efficiently calculating query sensitivity without requiring changes to the database management system (DBMS)

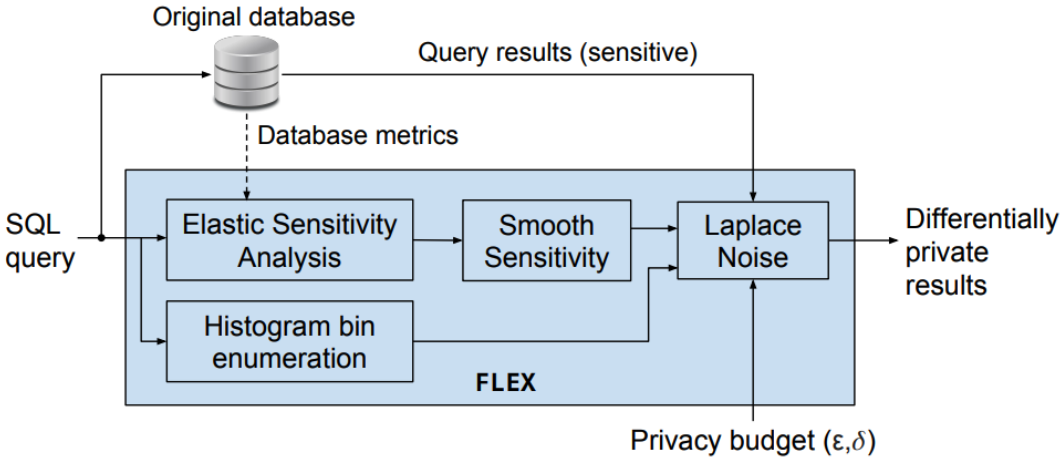

Fig. 2.3.1: Architecture of FLEX from [13].

in their paper . The techniques described in the article provide an en-to-end system to facilitate differential privacy for real-world SQL queries.

Before FLEX[13], the existing differential privacy mechanisms at that time did not support the wide variety of features and databases which were used in real-world SQL-based analytics systems. FLEX [13] was a system built at Berkeley to enforce differential privacy for SQL queries using elastic sensitivity. Noah Johnson et al. [13] discusses how FLEX is compatible with all existing databases, manages to enforce differential privacy requirements for all SQL queries and has a negligible performance overhead.

FLEX relies on the concept of elastic sensitivity. Elastic sensitivity is a novel approach for calculating an upper bound on a query's local sensitivity. Global sensitivity does not have adequate generalized support for joins in queries. Elastic sensitivity benefits from local sensitivity for queries with general equijoins. Its approach models the impact of each join that is represented in the query, using precomputed metrics about the frequency of join keys in the actual database. This allows the method to compute approximate local sensitivity without additional interactions with the database.

Elastic sensitivity supports several different aggregation functions such as sum, average, max and min. Also, calculations for elastic sensitivity can optimize for

non-sensitive information in the database, helping create tighter bounds for the approximation of local sensitivity. Due to its low computational cost, its adaptability to almost all existing database formats, the current implementation in the form of FLEX, and the general privacy guarantees provided by it, elastic sensitivity can be seen to be a very effective method for ensuring differential privacy.

2.4 ϵKTELO : A Framework for Defining Differentially-Private Computations

The paper ϵKTELO [32] by Zhang et al. talks about a framework to carry out privacy-preserving computations over data. We can say it is derived from frameworks such as PINQ [12], which extends the (non-private) LINQ framework, and Weighted PINQ. ϵKTELO [32] extends these by providing a different selection of operators with higher level of abstraction to the user who does not have knowledge about differential privacy.

2.5 APEx (Accuracy-Aware Differentially Private Data Exploration)

APEx [9] built by Change et al. helps to bridge the gap between complex differential privacy mechanisms and Data Owner/ Data Analysts. Even though we know that by adding Laplace noise to the data exploration queries provides differentially private results, the amount of noise or the Laplace distribution from which the noise is calculated depends on the data. So, APEx is built to help Data Owner/Data Analyst as shown in Figure 2.5.1 to focus on their work by just providing the privacy budget and accuracy bounds and leaving the rest to the APEx to figure out which mechanism and input factors to the mechanism would be best. APEx mainly covers three types of data exploration queries:

1. Workload Counting Query (WCQ)

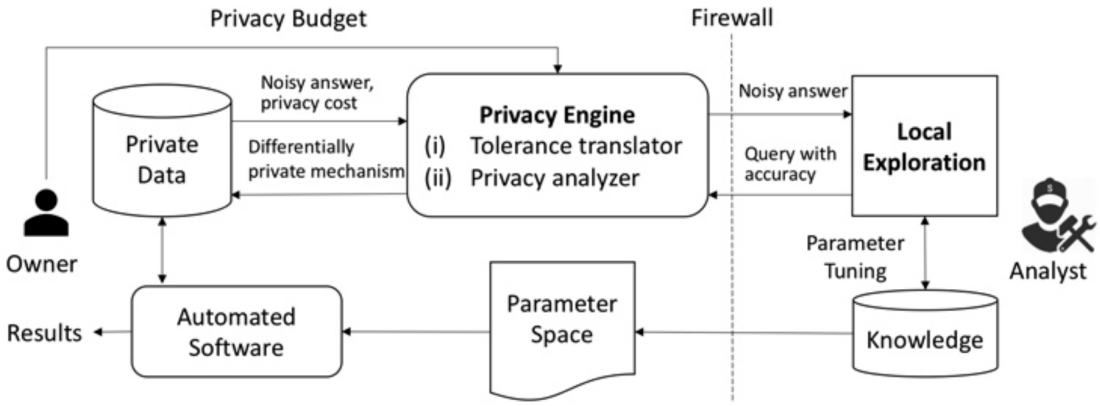

Fig. 2.5.1: Workflow of APEx [9].

2. Iceberg Counting Query (ICQ)

3. Top-k Counting Query (TCQ)

As we have discussed earlier, there exist general-purpose private query answering systems; they are not interactive and mainly lack two main aspects. Firstly, these systems expect the data analysts to have in-depth knowledge of differential privacy and differentially private algorithms, which most often do not. For example, PINQ [12] and wPINQ [28] allow users to write differentially private programs and ensure that every program expressed satisfies differential privacy. PINQ provides an SQL-like interface to the data analysts. Similarly, using a few simple operators (including a non-uniformly scaling Join operator), wPINQ can reproduce (and improve) several recent results on graph analysis and introduce new generalizations (e.g., counting triangles with given degrees). However, to achieve high efficacy of the system, the analyst has to be familiar with the intricate literature of data privacy to understand how the differentially private algorithms add noise and know if the desired accuracy level can be achieved in the first place. *ϵktelo* [32] tried to mitigate this problem by giving access to high-level operators to the data analyst that can be composed to create accurate differentially private programs to answer counting queries. However, the analyst is still expected to know how to distribute privacy budgets across different operators optimally. Similarly, FLEX [13] provides users with an interface to answer

one SQL query under differential privacy but has the same issue of distributing privacy budget when tested across a sequence of queries. Secondly, and somewhat ironically, these systems do not provide any assurance to the data analyst on the quality they care about, namely the accuracy of query answers. Most of these systems depend on the privacy level (ϵ) as input to figure out the most feasible differentially private algorithm without considering the accuracy of the result of the query.

This was the primary purpose behind APEx, to design a system that does not only allow data analysts to explore a sensitive dataset D held by a data owner by running a sequence of queries with high accuracy but also keeping the query results under the privacy budget set by the data owner. The system mainly focused on these two main functions: (1) as the data is sensitive, the data owner is assured that any information leakage is bounded under the privacy budget of any individual record in dataset D; and (2) also as the addition of noise affects the accuracy of the query result, it is still under the accuracy bounds set by the data analyst. The main features of APEx involved:

- To support declaratively specified aggregate queries that capture a wide variety of data exploration tasks.
- To allow analysts to specify accuracy bounds on queries.
- To translate an analyst's query into a differentially private mechanism with minimal privacy loss ϵ so that it can answer the query set, keeping the accuracy bounds checked.
- To prove that for any interactively specified sequence of queries, the analyst's view of the entire data exploration process satisfies B-differential privacy, where B is an owner specified privacy budget.

Ligett et al. [19] also worked on similar work as APEx, which also considered accuracy constraints specified by data analysts. While their work focused on finding the smallest privacy cost for a given differentially private mechanism and accuracy bound, APEx focus was on a more general problem: for a given query, find a mechanism and

a minimal privacy cost to achieve the given accuracy bound. Moreover, APEx was tested on a single table MYSQL database. As we know that in real-life, data spread across multiple related database tables. In this work, APEx is extended/tested on multiple-table database. This helped to get a better picture of the applicability of this niche concept on a relational multi-table database.

2.6 Internal structure of APEx

This section gives an outline of how APEx translates queries set by the data analyst along with the accuracy budget into differentially private mechanisms, and how it ensures that the privacy budget B specified by the data owner is also not violated. The main functionality of the extended APEx remains the same as the actual functionality of APEx. The only main difference is that the result of the queries is from multiple database tables instead of only one table. APEx consists of two parts:

1. Accuracy Translator
2. Privacy Analyzer

2.6.1 Accuracy Translator

This section covers the accuracy-to-privacy translation mechanisms supported by APEx and the corresponding run and translate functions. In the original APEx paper, the author has discussed two types of transformations for all three types of queries: (1) Baseline Transformation and (2) Special Transformation. In this work, I have focussed my study on Baseline Transformation only, which is discussed in more detail later.

Given an analyst's query (q, α, β), APEx first uses the accuracy translator to choose a mechanism M that can (1) answer q under the specified accuracy bounds set by the data analyst, and also with (2) minimal privacy loss and under the privacy budget set by the data owner. To achieve this, APEx supports a set of differentially private mechanisms that can be used to answer each query type (WCQ, ICQ, TCQ).

Multiple mechanisms are supported for each query type as different mechanisms result in the least privacy loss depending on the query and the dataset.

For each query of the user, APEx has to make sure that the answer is between the accuracy and the privacy bounds. To achieve this, it contains various differential privacy mechanisms, and each of these mechanisms is most effective depending on the query and the dataset. Thus, each mechanism M has two functions:

1. M.TRANSLATE
2. M.RUN

When a mechanism M is executed, M.TRANSLATE is responsible for translating a query and accuracy requirement into lower and upper bound (ϵ^l, ϵ^u) on the privacy loss. After that, M.RUN runs the differentially private algorithm and returns an approximate answer ω for the query. The answer ω is guaranteed to satisfy the specified accuracy requirement. Moreover, M also fulfills ϵ^u differential privacy.

Algorithm 2.6.1 APEx Overview [9]

Require: Dataset D, privacy budget B
1: Initialize privacy loss $B_0 \Leftarrow 0$, index $i \Leftarrow 1$
2: **repeat**
3: Receive (q_i, α_i, β_i) from analyst
4: $\mathcal{M} \Leftarrow$ mechanisms applicable to $q_i^{'}s$ type
5: $\mathcal{M}^* \Leftarrow \{M \in \mathcal{M} | M.\text{TRANSLATE}(q_i, \alpha_i, \beta_i).\epsilon^u \leq B - B_{i-1}\}$
6: **if** $\mathcal{M}^* \neq \emptyset$ **then**
7: $//Pessimistic\,Mode$
8: $M_i \Leftarrow argmin_{M \in \mathcal{M}^*} \text{M}.TRANSLATE(q_i, \alpha_i, \beta_i).\epsilon^u$
9: $//Optimistic\,Mode$
10: $M_i \Leftarrow argmin_{M \in \mathcal{M}^*} \text{M}.TRANSLATE(q_i, \alpha_i, \beta_i).\epsilon^l$
11: $(\omega_i, \epsilon_i) \Leftarrow M_i.\text{RUN}(q_i, \alpha_i, \beta_i, D)$
12: $B_i \Leftarrow B_{i-1} + \epsilon_i$, i++ **return** ω_i
13: **else**
14: $B_i = B_{i-1}$, i++ **return** 'Query Denied'
15: **end if**
16: **until** No more queries sent by local exploration

As described in Algorithm 2.6.1, APEx first identifies the mechanisms M that are applicable for the type of the query q_i (Line 4). Next, it runs M.translate to get

conservative estimates on privacy loss ϵ^u for all these mechanisms (Line 5). APEx picks one of the mechanisms M from those that can be safely run using the remaining privacy budget, executes M.run, and returns the output to the analyst. As we will see, there exist mechanisms where the privacy loss can vary based on the data in a range between $[\epsilon^l, \epsilon^u]$, and the actual privacy loss is unknown before running the mechanism. In such cases, APEx can choose to be pessimistic and pick the mechanism with the least ϵ^u (Line 8), or choose to be optimistic and pick the mechanism with the least ϵ^l (Line 10).

2.6.1.1 Matrix Transform

The workload in (WCQ,ICQ,TCQ) queries is represented in a matrix form, like the prior work for WCQ [17, 18, 20]. A workload can be transformed in many possible ways. Given a query with L predicates, the number of domain partitions can be as large as 2^L . In this work, the following transformation is considered to reduce complexity. Given a workload counting query q_W with the set of predicates $W = \{\phi_1, ..., \phi_L\}$, the full domain of the relation $dom(R)$ is partitioned based on W to form the new discretized domain $dom_W(R)$ such that any predicate $\phi_i \in W$ can be expressed as a union of partitions in the new domain $dom_W(R)$ and the number of partitions are minimized. For example, given $W = \{Age > 50 \wedge State = AL, ..., Age > 50 \wedge State = WY\}$, one possible partition is $dom_W(R) = \{Age > 50 \wedge State = AL, ..., Age > 50 \wedge State = WY, Age \leq 50\}$.

Let x represents the histogram of the table D over $dom_W(R)$. The set of the corresponding counting queries $\{c\phi_1, ..., c\phi_L\}$ for q_W can be represented by a matrix $W = [w_1, ..., w_L]^T$ of size $L \times |dom_W(R)|$. Hence, we can say that the answer to each counting query is simply $c_{\phi_i}(D) = w_i.x$, and the answer to the workload counting query is simply Wx. This transformation denoted by $W \leftarrow \mathcal{T}(W), x \leftarrow \mathcal{T}_W(D)$ is used throughout later. Prior work such as [17, 18, 20] were based on to bound the total expected error on single query. APEx differs from them by bounding the maximum error per query with high probability which is more instinctive in the process of data exploration.

2.6.1.2 Baseline Translation

For the baseline translation for all three query types in APEx, the Laplace mechanism is used [6, 8]. Laplace mechanism is widely used and accepted as a standard differentially private mechanism.

Definition 2 *(Laplace Mechanism (Vector Form)[6, 8]). Given an* $L \times |dom_W(R)|$ *query matrix W, the randomized algorithm LM that outputs the following vector is* ϵ*-differentially private:* $LM(W, x) = Wx + Lap(b_w)^L$ *where* $b_w = \frac{||W||_1}{\epsilon}$*, and* $Lap(b)^L$ *denote a vector of L independent samples* η_i *from a Laplace distribution with mean 0 and variance* $2b^2$*, i.e.,* $Pr[\eta_i = z] \propto e^{-z/b}$ *for i=1,...,L. [9]*

The sensitivity of queries set defined by the workload W [17, 18] is equal to the constant $||W_1||$. This constant calculates the maximum difference in the answers to the queries in W when these queries are run on any two databases that differ only by a single record. Mathematically, it is the maximum of the L1 norm of a column of W.

Algorithm 2.6.2 presents the run and translate of Laplace mechanism for all three query types. In this algorithm we can see that first the query q_W and the data D are converted into matrix representation W and x, respectively. The translate generates a lower and upper bound (ϵ^l, ϵ^u) for each query type with a given accuracy requirement and since Laplace mchanism is data independent, these two bounds are same. However, these bounds vary among query types. The run takes the privacy budget computed by TRANSLATE(q, α, β) (Line 3) and adds the corresponding Laplace noise $[\tilde{x_1}, ..., \tilde{x_L}]$ to the true workload counts Wx. In case of when q is a Workload Counting Query (WCQ), the noisy counts are returned directly at the end of running the Laplace mechanism. When q is an Iceberg Counting Query (ICQ), the bin ids (the predicates) that have noisy counts $\geq$ c are returned. And, when q is a Top-k Counting Query (TCQ), the bin ids (the predicates) that have the largest k noisy counts are returned. Along with the noisy output, the privacy budget consumed by this mechanism is also returned. The following theorem provide a summarization of the properties of the two functions run and translate.

Algorithm 2.6.2 Laplace Mechanism (LM) (q, α, β, D) [9]

Initialize $W \leftarrow \mathcal{T}(W = \{\phi_1, \cdots, \phi_L\}), x \leftarrow \mathcal{T}_W(D), \alpha, \beta$
function RUN(q, α, β, D)
    $\epsilon \leftarrow TRANSLATE(q_W, \alpha, \beta, D).\epsilon^u$
    $[\tilde{x}_1, \cdots, \tilde{x}_L] \leftarrow Wx + Lap(b)^L$, where $b = ||W||_1/\epsilon$
    if q.type==WCQ (i.e., q_W) **then**
      return $([\tilde{x}_1, \cdots, \tilde{x}_L],\epsilon)$
    else if q.type==ICQ (i.e., $q_{W,>c}$) **then**
      return $(\phi_i \in W|\tilde{x}_i > c,\epsilon)$
    else if q.type==TCQ (i.e., $q_{W,k}$) **then**
      return $(argmax^k_{\phi_1,\cdots,\phi_L}\tilde{x}_i,\epsilon)$
    end if
end function
function TRANSLATE(q,α,β)
    if q.type==WCQ (i.e., q_W) **then**
      return $(\epsilon^u = \frac{||W||_1 \ln 1/(1-(1-\beta)^{1/L}))}{\alpha},\epsilon^l = \epsilon^u)$
    else if q.type==ICQ (i.e., $q_{W,>c}$) **then**
      return $(\epsilon^u = \frac{||W||_1(\ln 1/(1-(1-\beta)^{1/L})) - \ln 2)}{\alpha},\epsilon^l = \epsilon^u)$
    else if q.type==TCQ (i.e., $q_{W,k}$) **then**
      return $(\epsilon^u = \frac{||W||_1 2(\ln (L/(2\beta)))}{\alpha},\epsilon^l = \epsilon^u)$
    end if
end function

Theorem 1 *Given a query q where q.type* $\in \{WCQ, ICQ, TCQ\}$*, Laplace mechanism (Algorithm 2.6.2) denoted by M can achieve* $(\alpha, \beta)-$*q.type accuracy by executing the function* $RUN(q, \alpha, \beta, D)$ *for any* $D \in \mathcal{D}$*, and satisfy differential privacy with a minimal cost of* $TRANSLATE(q, \alpha, \beta).\epsilon^u$*. [9]*

The accuracy and privacy proof is mainly based on the noise property of Laplace mechanism.

2.6.2 Privacy Analyzer

Given a sequence of queries $(M_1, ..., M_i)$ that has already been executed by the privacy engine and that satisfy an overall B_{i-1}-differential privacy. If a new query (q_i, α_i, β_i) is run, APEx first identifies a set of mechanisms M^* that all will have a worst-case privacy loss smaller than $B - B_{i-1}$ (Line 5 in Algorithm 2.6.1). That is, running any mechanism in M^* will not result in exceeding the privacy budget in the worst case. If $M^* = \emptyset$, then APEx returns 'Query Denied' to the analyst (Line 16 in Algorithm 2.6.1). Otherwise, APEx runs one of the mechanisms M_i from M^* by executing M_i.RUN() and the output ω_i will be returned to the analyst. APEx then increments B_{i-1} by the actual privacy loss ϵ_i rather than the upper bound ϵ^u (Line 12 in Algorithm 2.6.1).

The privacy analyzer ensures that every sequence of queries answered by APEx results in a B-differentially private execution, where B is the privacy budget specified by the data owner. The formal proof of privacy primarily follows from the well-known composition theorems. According to sequential composition, the privacy loss of a set of differentially private mechanisms (that use independent random coins) is the sum of the privacy losses of each of these mechanisms. Moreover, postprocessing the outputs of a differentially private algorithm does not degrade privacy.

The main critical part of the privacy proof (described in Section 3.2.2.1) arises since (1) the ϵ parameter for a mechanism is chosen based on the analyst's query and accuracy requirement, which in turn are adaptively chosen by the analyst based on previous queries and answers, and (2) some mechanisms may have an actual privacy

loss dependent on the data. APEx accounts for privacy based on the actual privacy loss (and not the worst case privacy loss) (see Line 12, Algorithm 2.6.1).

2.6.2.1 Overall Privacy Guarantee

Privacy guarantee means that given any sequence of interactions between the data analyst and APEx, APEx satisfies B-differential privacy, where B is the privacy budget specified by the data owner. To state this guarantee formally, we first need to understand a record of the interaction between APEx and the data analyst.

We define the transcript of interaction $\mathbb{T}$ as an alternating sequence of queries (with accuracy requirements) set to APEx by the data analyst and answers returned by APEx. $\mathbb{T}$ depicts the analyst's view of the private database. More formally,

- The transcript $\mathbb{T}_i$ after i interactions is a sequence $[(q_1, \alpha_1, \beta_1), (\omega_1, \epsilon_1), \cdots, (q_i, \alpha_i, \beta_i), (\omega_i, \epsilon_i)]$, where (q_i, α_i, β_i) are queries with accuracy requirements, and ω_i is the answer returned by APEx and ϵ_i the actual privacy loss.

- Given $\mathbb{T}_{i-1}$, analyst choses the next query $(q_{i+1}, \alpha_{i+1}, \beta_{i+1})$ adaptively. APEx model this using a (possibly randomized) algorithm $\mathbb{C}$ that maps a transcript $\mathbb{T}_{i-1}$ to (q_i, α_i, β_i); i.e., $\mathbb{C}(\mathbb{T}_{i-1}) = (q_i, \alpha_i, \beta_i)$. Note that the analyst's algorithm $\mathbb{C}$ does not access the private database D.

- Given (q_i, α_i, β_i), APEx select a subset of mechanisms $\mathcal{M}^*$ such that $\forall M \in \mathcal{M}^*$, M.TRANSLATE$(q_i, \alpha_i, \beta_i).\epsilon^u \leq B - B_i$. Furthermore, if $\mathcal{M}^*$ is not empty, APEx choses one mechanism $M_i \in \mathcal{M}^*$ deterministically (either based on ϵ^l or ϵ^u) to run. The selection of M_i is deterministic and independent of D.

- If APEx find no mechanism to run ($\mathcal{M}^* = \emptyset$), then the query is declined by APEx. In this case, $\omega_i = \perp$ and $\epsilon_i = 0$.

- If the APEx chosen algorithm M_i is LM, WCQ-SM, ICQ-SM or TCQ-LTM, $\epsilon_i = \epsilon_i^u$, where ϵ_i is the upperbound on the privacy loss returned by M_i.TRANSLATE. For ICQ-MPM, the actual privacy loss can be smaller; i.e. $\epsilon_i \leq \epsilon_i^u$.

- Let $Pr[\mathbb{T}_i|D]$ denote the probability that the transcript of interaction is $\mathbb{T}_i$ given input database D. The probability is over the randomness in the analyst's choices $\mathbb{C}$ and the randomness in the mechanisms $M_1, \cdots, M_i$ executed by APEx.

Not all transcripts of interactions are realizable under APEx. Given a privacy budget B, the set of valid transcripts is defined as:

Definition 3 *(Valid Transcripts [9]). A transcript of interaction $\mathbb{T}_i$ is a valid APEx transcript generated by Algorithm 2.6.1 if given a privacy budget B the following conditions hold:*

- $B_{i-1} = \sum_{j=1}^{i-1} \epsilon_j \leq B$, *and*
- *Either* $\omega_i = \perp$, *or* $B_{i-1} + \epsilon_i^u \leq B$.

We are now ready to state the privacy guarantee:

Theorem 2 *(APEx PRIVACY GUARANTEE[9]). Given a privacy budget B, and valid APEx transcript $\mathbb{T}_i$,and any pair of databases D, D' that differ in one row (i.e., $|D \setminus D' \cup D' \setminus D| = 1$), we have:*

(1) $B_i = \sum_{j=1}^{i} \epsilon_i \leq B$, *and*

(2) $Pr[\mathbb{T}_i|D] \leq e^{B_i} Pr[\mathbb{T}_i|D']$.

More details regarding APEx can be found in APEx[9] research paper.

2.7 Accuracy Measure

The answers to the exploration queries are typically noisy to ensure Differential Privacy. To allow the data analyst to explore data with bounded error, the queries are extended to incorporate an accuracy requirement. The syntax for accuracy is similar to that in BlinkDB [1]:

BIN D on $f(\cdot)$ WHERE $W = \{\phi_1, ..., \phi_L\}$ [HAVING $f(\cdot)$>c]
[ORDER BY $f(\cdot)$ LIMIT k] ERROR α;

The accuracy requirement for a WCQ q_w is defined as a bound on the maximum error across queries in the workload W.

Definition 4 *($(\alpha, \beta) - WCQ$ accuracy [9]) Given a workload counting query $q_w : D^{'} \rightarrow R^L$, where $W = \{\phi_1, \cdots, \phi_L\}$. Let $M : D^{'} \rightarrow R^L$ be a mechanism that outputs a vector of answers y on D. Then, M satisfies $(\alpha, \beta) - W$ accuracy, if $\forall D \in D^{'}$,*

$$Pr[||y - q_w(D)||_\infty \geq \alpha] \leq \beta, \tag{1}$$

where $||y - q_w(D)||_\infty = max_j|y[i] - c_{\phi_i}(D)|$.

The output of iceberg counting queries ICQ and top-k counting queries TCQ are not numeric, but a subset of the given workload predicates. Their accuracy measures are different from WCQ, and depend on their corresponding workload counting query q_W.

Definition 5 *($(\alpha, \beta) - ICQ$ accuracy [9]) Given an iceberg counting query $q_{w,>c} : D^{'} \rightarrow O$, where $W = \{\phi_1, \cdots, \phi_L\}$, and O is a power set of W. Let $M : D^{'} \rightarrow O$ be a mechanism that outputs a subset of W. Then, M satisfies $(\alpha, \beta) - ICQ$ accuracy for $q_{w,>c}$, if for D,*

$$Pr[|\{\phi \in M(D)|c_\phi(D) < c - \alpha\}| > 0] \leq \beta \tag{2}$$

$$Pr[|\{\phi \in (W - M(D))|c_\phi(D) > c + \alpha\}| > 0] \leq \beta \tag{3}$$

A mechanism for ICQ can make two kinds of errors: label predicates with true counts greater than c as $< c$ (red dots in 2.7.1) and label predicates with true counts less than c as $> c$ (blue dots in 2.7.1).

We say a mechanism satisfies $(\alpha, \beta) - ICQ$ accuracy if with high probability, all the predicates with true counts greater than $c + \alpha$ are correctly labeled as >c and

all the predicates with true counts less than $c - \alpha$ are correctly labeled as $< c$. The mechanism may make arbitrary mistakes within the range $[c - \alpha, c + \alpha]$.

Definition 6 *($(\alpha, \beta) - TCQ$ accuracy [9]) Given a top-k counting query $q_{w,k} : D^{'} \rightarrow O$, where $W = \{\phi_1, \cdots, \phi_L\}$, and O is a power set of W. Let $M : D^{'} \rightarrow O$ be a mechanism that outputs a subset of W. Then, M satisfies $(\alpha, \beta) - TCQ$ accuracy if for $D \in D^{'}$,*

$$Pr[|\{\phi \in M(D) | c_\phi(D) < c_k - \alpha\}| > 0] \leq \beta \tag{4}$$

$$Pr[|\{\phi \in (\Phi - M(D)) | c_\phi(D) > c_k + \alpha\}| > 0] \leq \beta \tag{5}$$

where c_k is the k^{th} largest counting value among all the bins, and Φ is the true top-k bins.

The intuition behind Definition 4 is similar to that of ICQ and is explained in Figure 2.7.1: predicates with count greater than $c_k + \alpha$ are included and predicates with count less than $c_k - \alpha$ do not enter the top-k with high probability.

The advantages of the accuracy definitions defined above are that they are intuitive (when α increases, noisier answers are expected) and we can design privacy-preserving mechanisms that introduce noise while satisfying these accuracy guarantees. On the other hand, this measure is not equivalent to other bounds on the accuracy like relative error and precision/recall which can be very sensitive to small amounts of noise (when the counts are small, or when lie within a small range). For example, if the counts of all the predicates in ICQ lie outside $[c - \alpha, c + \alpha]$, a mechanism M that perturbs counts within $\pm\alpha$ and then answers an ICQ will have precision and recall of 1.0 with high probability as it makes no mistakes. However, if all the query answers lie within $[c - \alpha, c + \alpha]$, then the precision and recall of the output of M could be 0.

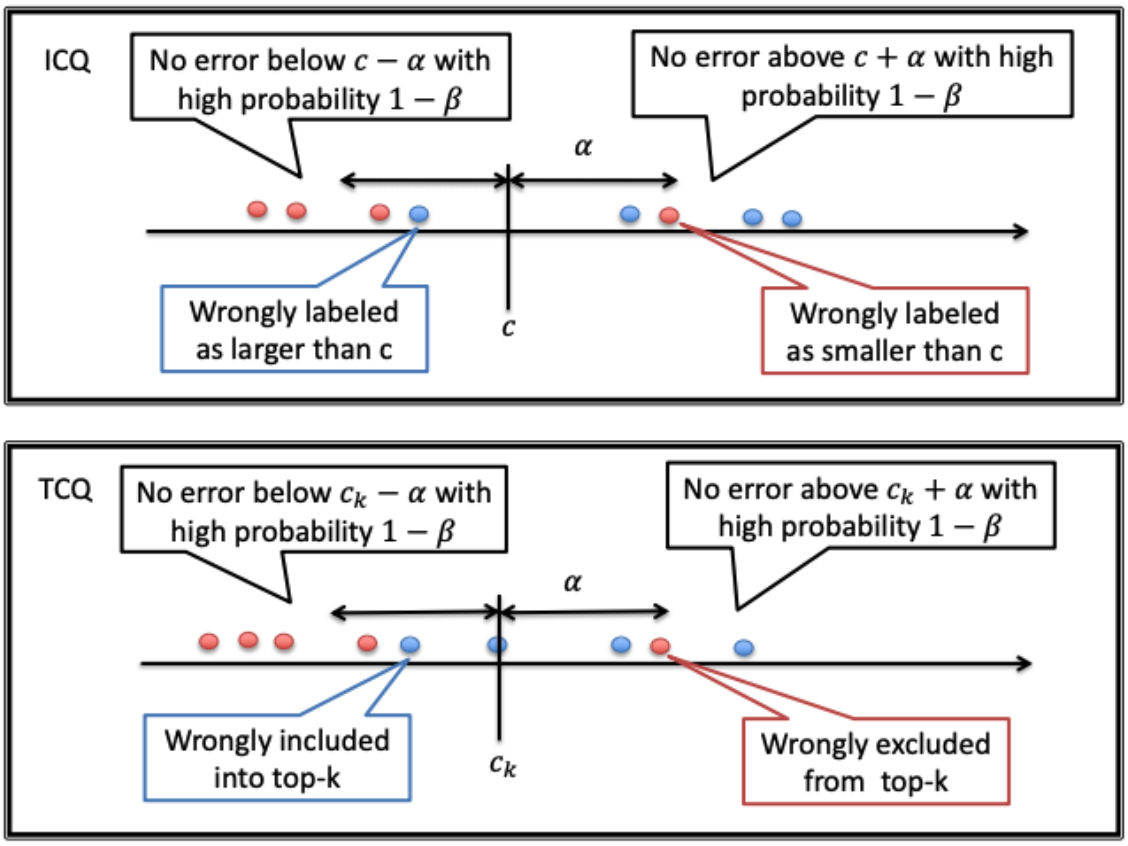

Fig. 2.7.1: Accuracy Requirement for ICQ and TCQ

CHAPTER 3

Methodology used and Extensions to APEx

In this chapter, the extensions to the APEx are covered in more detail.There are some similarities to the APEx, as this is the extension to the original paper. In this work, it is tested on multiple-table dataset. A dataset with multiple tables is used, which is also discussed in detail later in section 3.1. Then the changes to the queries structure were made, which are more evident in the code used to test the concept. The code is attached in the Appendix.

3.1 Dataset

In this research, I have taken multi-table relational schema $R_1(A_1, A_2, ..., A_d)$,

$R_2(B_1, B_2, ..., B_d)$ where $attr(R_1)$ and $attr(R_2)$ denotes the set of attributes of $R_1 \bigcup R_2$. Each attribute A_i has a domain $dom(A_i)$. The full domain of R is $dom(R) = dom(A_1) \times \cdots \times dom(A_d) \times dom(B_1) \times \cdots \times dom(B_d)$, containing all possible tuples conforming to R. An instance D of relation R is a multiset whose elements are tuples in $dom(R)$. Let us denote the domain of the instances be D.

Based on the above multiple-table schema, I found Employee sample database from MYSQL website. Patrick Crews and Giuseppe Maxia [26] developed the Employee sample database and provide a combination of a broad base of data (approximately 160MB), which spread over six separate tables and consists of 4 million records in total. There are about 300,000 employee records with 2.8 million salary entries

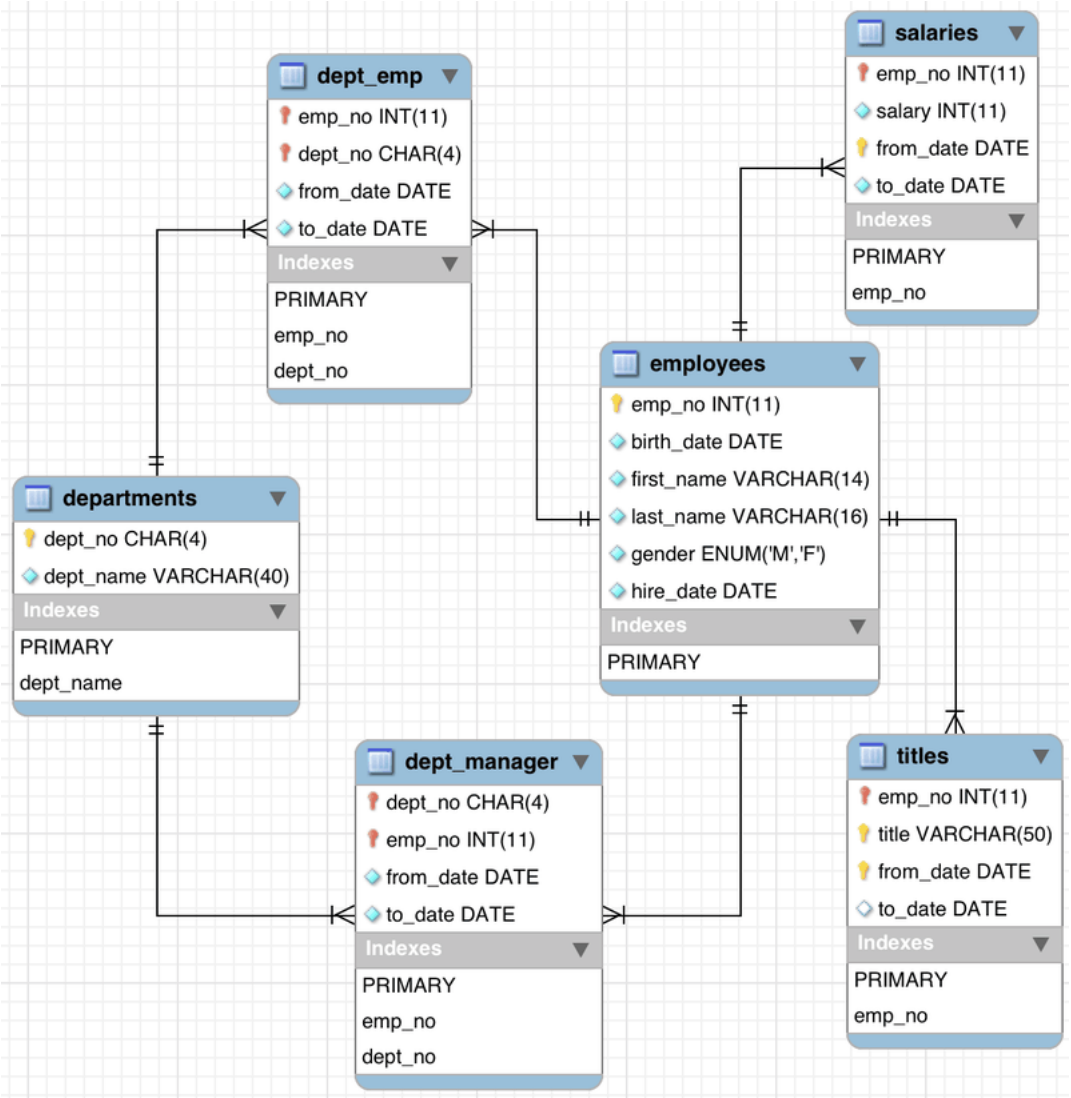

Fig. 3.1.1: The Employees Schema

in the database. The diagram 3.1.1 provides an overview of the structure of the Employees sample database.

3.2 Queries and their types

This section describes the different types of queries used in APEx. It is believed that most of the data exploration done by data analysts can be done by writing queries that can be categorized into one of these query types. These queries are commonly referred to as Exploration Queries.

3.2.1 Exploration Queries

There are mainly three types of exploration queries:

1. Workload Counting Query (WCQ)
2. Iceberg Counting Query (ICQ)
3. Top-k Counting Query (TCQ)

3.2.1.1 Workload Counting Query (WCQ)

Workload counting queries cover a large part of Linear Counting queries. They are very similar to SQL SELECT ... GROUP BY queries. Another excellent example of workload counting queries is histogram queries. For example, let's take a table D with attribute state having domain {AL, AK, . . . , WI, WY} and an attribute Age with domain $[0, \infty)$. Then, a query to return the number of people with age above 50 for each state can be expressed using WCQ as:

BIN D on COUNT(*) WHERE W = $\{Age > 50 \wedge State = AL, ..., Age > 50 \wedge State = WY\}$;

3.2.1.2 Iceberg Counting Query (ICQ)

The main difference between the iceberg counting query and linear counting query is that the answer to the query is a subset of the predicates in W.

BIN D on COUNT(*) WHERE W = $\{\phi_1, ..., \phi_L\}$ HAVING COUNT(*) > c;

An iceberg counting query returns bin identifiers if the aggregated value for that bin is higher than a threshold. For example, a query which returns the states in the US which have a population of at least 5 million can be expressed as:

```
BIN D on COUNT(*)
WHERE W = {State=AL, ... , State=WY}
HAVING COUNT(*) > 5 million;
```

Note that since the answer to the query is a subset of the predicates in W (i.e. a subset of bin identifiers) but not the aggregate values for these bins, an ICQ is not a linear counting query.

3.2.1.3 Top-k Counting Query (TCQ)

BIN D on COUNT(*) WHERE $W = \{\phi_1, ..., \phi_L\}$
ORDER BY COUNT(*) LIMIT k;

This query firsts sorts all the bins based on a threshold and returns the top k bin identifiers. For example, a query to return the top three US states with the highest population can be written as:

BIN D on COUNT(*) WHERE W = $\{State = AL, ..., State = WY\}$
ORDER BY COUNT(*) LIMIT k;

3.3 Code changes on APEx

In order to make APEx compatible with multiple-table queries, there were a couple of changes that were required at the code level. But before that, we need to understand how APEx works at the code level. When we run the program, there are two threads that run simultaneously. First, based on the given values of α, β and the differential privacy mechanism, the estimated cost of differential privacy mechanism and differential privacy mechanism itself is calculated. This can be seen in the following code snippet for mechanisms LM and LM_SM.

Listing 3.1: Estimated cost and calculation of mechanism [9]

```
# estimate cost using sequential composition
    def estimate_loss(self, m):
        # estimate the cost based on the query type
        q = m.query
        if q.query_type == Type.QueryType.WCQ:
            if q.m_type == Type.MechanismType.LM:
```

```
            # estimate cost
            est_cost = lm_est_cost(m)
            m.set_est_cost(est_cost)

            # set the laplace b
            m.set_lap_b(q.get_sensitivity() / est_cost)

        elif q.m_type == Type.MechanismType.LM_SM:
            # estimate cost
            est_cost = lm_sm_est_cost(m)
            m.set_est_cost(est_cost)

            # set the laplace b
            m.set_lap_b(m.strategy_sens / est_cost)

    return self.total_privacy_cost + m.est_cost
```

As we can see from the above code snippet that first it calculates cost estimation. It is done using this formula.

Listing 3.2: Cost estimation [9]

```
def lm_est_cost(m):
    q = m.query
    est_cost = q.get_sensitivity() * np.log(1.0 / (1.0 - (1.0 - m.beta)
    ** (1.0 / len(q.cond_list)))) / m.alpha
    return est_cost
```

Sensitivity in the above function is just the sensitivity of the matrix. After calculating the estimated cost of the mechanism, the mechanism itself is calculated as can be seen in Listing 3.1.

Second, the query is converted into a matrix form. This can be seen in the following code snippet.

Listing 3.3: Matrix Representation of query [9]

```
# construct the matrix representation of the query
    def to_matrix(self):
```

```
# for 1D and 2D histogram, just count(*)
# get all the counts for each predicate
self.domain_hist = self.get_cond_counts()

# for i in range(0, len(self.domain_hist)):
if self.index.__name__ in ['qw_1', 'qwm_1',
'qi_3', 'qt_1', 'qtm_1', 'qw_4', 'qi_2', 'qim_2', 'qt_3']:
# 1D/2D histogram
    is_hist = True
else:
    is_hist = False

if is_hist:
    # for histogram query
    count_row_by_predicates = sum(self.domain_hist)

self.query_matrix =
np.zeros((len(self.cond_list), len(self.domain_hist)))

if is_hist:
    for i in range(0, len(self.cond_list)):
        self.query_matrix[i][i] = 1
```

In order to convert a histogram query to matrix form, first we calculate the domain of the histogram. This is the condition counts. It contains the value corresponding to each condition. Using this and the length of condition list, we get x $\times$ y matrix where x is the length of condition list and y is the length of domain of histogram.

These two threads give us the foundation to calculate the final answers to the query. As can be seen from the code snippet of matrix transformation (Listing 3.3) that it calculates the condition counts, which is the primary thing in the type of queries that we are trying to execute. The condition counts are the ones that are shown as bars in a histogram. To calculate these condition counts, the previous version of APEx used to calculate these on a single table, as we can see from this

code snippet.

Listing 3.4: Condition counts for single table schema

```
crnt_query = select count(*) from
+ table_name +  where  + cond
```

This is changed to get the condition counts from multiple tables based on the query type as can be seen from the following code snippet.

Listing 3.5: Condition counts for multiple table schema

```
# prepare the query to query ids

    if(self.index.__name__ == 'qim_2'):

    crnt_query = SELECT COUNT(*) FROM
    (SELECT A.emp_no,  A.gender, B.salary
    FROM employees.employees as A INNER JOIN
    employees.salaries as B ON
    A.emp_no = B.emp_no where  + cond + ) C

# get the total count from the table
    if(self.index.__name__ == 'qim_2'):
    crnt_query = SELECT COUNT(*) FROM (SELECT A.emp_no,
    A.gender, B.salary FROM employees.employees as
    A INNER JOIN employees.salaries as B ON A.emp_no = B.emp_no) C
```

3.4 Incorporating other measures

Extended APEx is evaluated on various error measures like Mean Square Error (MSE), Mean Absolute Error (MAE) and Mean Absolute Percentage Error. These error measures are widely used in Machine Learning models. The motivation behind using these as error measures for APEx is the existence of an excellent analogy behind a machine learning model and APEx. A machine learning model predicts a value, and we use these measures to see how close that prediction is to the actual value.

Similarly, APEx returns noisy values to the data analyst. These error measures can help us evaluate how close or how accurate these noisy values are to the true values. The more precise these noisy values are, the more beneficial they can be to the data analyst. Since we have used Differentially Private mechanism to calculate these values, the privacy constraint set by the data owner is also kept.

Mean Square Error (MSE): In statistics, Mean Squared Error is widely used measure to determine the performance of an estimator. In simple words, it is the square root of the average of squared differences between the true value and the predicted value.

$$MSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

Mean Absolute Error (MAE): The main difference in Mean Absolute Error and Mean Square Error is that it measures average magnitude of the errors in a set of noisy answers, without considering their direction. It's the average over the sample of the absolute differences between the noisy answer and the true answer where all individual differences have equal weight.

$$MAE = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

Mean Absolute Percentage Error (MAPE): Mean Absolute Percentage Error (MAPE) measures the size of error in terms of percentage. It is mostly used to give perspective to how much error is there on scale of 1-100.

$$MAPE = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j| * 100$$

The error values of these measures are provided in the next chapter. The one important observation to notice is that when we compare the error values from single table queries and multiple table queries, there is not much difference. For example, the error values of WCQ for single table query are 83.4 and 13894.5 for MAE and MSE respectively. Whereas for WCQ for multiple table query, the error values are

87.10, and 9927.95 for MAE and MSE respectively. This re-assures us that APEx can easily be scaled to data which multi-dimensional, multi-table and also with large volume without affecting its error measures.

CHAPTER 4

Experiments and Results

In this research, I have evaluated extended APEx on real-world datasets, out of which two were single-table dataset, and one was multiple-table dataset. The details, along with the entity-relationship diagram about the numerous table dataset, can be found in Chapter 3 (Methodology). Extended APEx was evaluated using a set of benchmark queries to see if :

- APEx can effectively translate queries associated with accuracy bounds into differentially private mechanisms. These mechanisms accurately answer a wide variety of new data exploration queries with moderate to low privacy loss.
- The set of query benchmarks show that no single mechanism can dominate the rest and APEx picks the mechanism with the least privacy loss for all the queries.

4.1 Setup

4.1.1 Datasets

The experiment uses three real-world datasets. The first data set Adult was extracted from the 1994 US Census release [3]. This dataset includes 15 attributes (6 continuous and nine categorical), such as "capital gain", "country", and a binary "label" indicating whether an individual earns more than 5000 or not, for a total of 32, 561 individuals. The second dataset, referred to as NYTaxi, includes 9, 710, 124 NYC's yellow taxi trip records [30]. Each record consists of 17 attributes, such

as categorical attributes (e.g., "pick-up-location"), and continuous attributes (e.g., "trip distance"). The third dataset, Employees sample database [25], is taken from the MYSQL website. It was developed by Patrick Crews and Giuseppe Maxia and provides a combination of a large base of data (approximately 160MB) spread over six separate tables and consisting of 4 million records in total.

4.1.2 Query Benchmarks

In this research, ten meaningful exploration queries on Adult and Employees datasets, summarized in Table 4.1.1 are tested. These ten queries cover the three types of exploration queries defined in Section 3.3, QW1, QW2, QWM1, QWM2, QI1, QI2, QIM1, QIM2, QT1 and QT2 corresponds to WCQ (Workload Counting Query), ICQ (Iceberg Counting Query) and TCQ (Top-K Counting Query) respectively. Queries with numbers 1 and 2 are for Adult, with numbers M1 and M2 are for Employees dataset. The predicate workload W cover 1D histogram, 1D prefix, 2D histogram and count over multiple dimensions. $\beta = 0.0005$ and vary $\alpha \in$
$\{0.02, 0.04, 0.08, 0.16, 0.32, 0.64\}$.

4.1.3 Metrics

For each query (q, α, β), APEx outputs (ϵ, ω) after running a differentially private mechanism, where ϵ is the actual privacy loss and ω is the noisy answer. The empirical error of a WCQ qW (D) is measured as $||\omega - q_w(D)||_\infty/|D|$, the scaled maximum error of the counts. The empirical errors of $ICQ_{q_w,>c}$(D) and $TCQ_{q_w,k}$(D) are measured as $||\alpha||_\infty/|D|$, the scaled maximum distance of mislabeled predicates.

4.1.4 Experimental Setup

APEx is implemented using python-3.4 and is run on a machine with 64 cores and 256 GB memory. APEx is run with optimistic mode. For strategy mechanism, H_2 (a hierarchical set of counts [17, 18, 20]) strategy is used for all queries. The code for this research was taken from APEx [9].

Name	D	Query workload W	Query output
QW1	Adult	"capital gain"∈ [0, 50), "capital gain"∈ [50, 100), ...,"capital gain"∈ [4950, 5000)	bin counts
QW2	Adult	"capital gain"∈ [0, 50), "capital gain"∈ [0, 100), ..., "capital gain"∈ [0, 5000)	bin counts
QWM1	Employees	"salary"∈ [0, 1500), "salary"∈ [1500, 3000), ...,"capital gain"∈ [148500, 150000)	bin counts
QWM2	Employees	"salary"∈ [0, 1500), "salary"∈ [0, 3000), ..., "salary"∈ [0, 150000)	bin counts
QI1	Adult	"capital gain"<50, "capital gain"<100,..., "capital gain"<5000	bin ids having counts >0.1|D|
QI2	Adult	(0 ≤"capital gain"<100, "sex"='M'),...(4500 ≤"capital gain"<5000, "sex"='F')	bin ids having counts >0.1|D|
QIM1	Employees	"salary"<1500, "salary"<3000,..., "salary"<150000	bin ids having counts >0.1|D|
QIM2	Employees	(0 ≤"salary"<1500, "gender"='M'),...(145000 ≤"capital gain"<150000, "gender"='F')	bin ids having counts >0.1|D|
QT1	Adult	"age"= 0,"age"= 1,...,"age"= 99	top 10 bins with highest counts
QTM1	Employees	"gender"= 0,"gender"= 1,...,"gender"= 99	top 10 bins with highest counts

Table 4.1.1: Query benchmarks include three types of exploration queries on three datasets.

4.2 Experiment Results

Ten queries as shown in Table 4.1.1 were run with different accuracy requirements from $0.01|D|$ to $0.64|D|$ and $\beta = 0.0005$. After running these queries for different configurations, I found that for all the queries the mechanism chosen for each α incurs an actual privacy cost at $\epsilon = \epsilon^u$.

As we can see from Table 4.2.1, the privacy cost of WCQ, ICQ, and TCQ queries is almost the same for both single and multiple table databases. Privacy cost is the privacy that we loose by running these queries. It's important to mention here that even if we are using differential privacy mechanisms here, we still loose privacy and the execution of any new query should be bound by the privacy bound set by the data owner. This tells us that the concept of APEx can easily be scaled to big databases with multiple tables without the loss of privacy. I checked the privacy cost with α-values $0.2|D|$ and $0.8|D|$. In both of these α values, either there is no difference or very similar values.

But on the contrary, if we see Table 4.2.2 , which tells the time taken in seconds to run WCQ, ICQ, and TCQ queries with different mechanisms on both single and multiple table queries, we can see that there is a vast difference. The single table queries take much less time than the time taken to run queries on multiple table queries.

Mechanism	Query-α							
	QW1-0.02$\|D\|$	QW1-0.08$\|D\|$	QW2-0.02$\|D\|$	QW2-0.08$\|D\|$	QWM1-0.02$\|D\|$	QWM1-0.08$\|D\|$	QWM2-0.02$\|D\|$	QWM2-0.08$\|D\|$
WCQ-LM	0.01874	0.00469	1.87430	0.46858	0.01874	0.00468	1.87430	0.46858
WCQ-SM	0.09809	0.02367	0.09201	0.02523	0.09785	0.02385	0.09548	0.02655
	QI1-0.02$\|D\|$	QI1-0.08$\|D\|$	QI2-0.02$\|D\|$	QI2-0.08$\|D\|$	QIM1-0.02$\|D\|$	QIM1-0.08$\|D\|$	QIM2-0.02$\|D\|$	QIM2-0.08$\|D\|$
ICQ-LM	1.76786	0.44197	0.01768	0.00442	1.76786	0.44196	0.01768	0.0041
ICQ-SM	0.09526	0.02379	0.0999	0.02359	0.11330	0.02298	0.09090	0.0251
ICQ-MPM	2.12148	0.53037	0.0212	0.00530	2.12148	0.53037	0.0212	0.00530
	QT1-0.02$\|D\|$	QT1-0.08$\|D\|$	–	–	QTM1-0.02$\|D\|$	QTM1-0.08$\|D\|$	–	–
TCQ-LM	0.03536	0.00884	–	–	0.03536	0.00884	–	–
TCQ-LTM	0.03535	0.07071	–	–	0.03536	0.00884	–	–

Table 4.2.1: Comparison of privacy cost values of WCQ, ICQ and TCQ queries on single table and multiple table databases.

Mechanism	Time taken (in seconds)							
	QW1-0.02$\|D\|$	QW1-0.08$\|D\|$	QW2-0.02$\|D\|$	QW2-0.08$\|D\|$	QWM1-0.02$\|D\|$	QWM1-0.08$\|D\|$	QWM2-0.02$\|D\|$	QWM2-0.08$\|D\|$
WCQ-LM	2.44245	2.00211	1.99205	1.87870	94.97264	107.16471	96.53642	94.57371
WCQ-SM	2.54281	2.11564	1.92897	1.92770	84.52438	96.99313	93.88398	92.78742
	QI1-0.02$\|D\|$	QI1-0.08$\|D\|$	QI2-0.02$\|D\|$	QI2-0.08$\|D\|$	QIM1-0.02$\|D\|$	QIM1-0.08$\|D\|$	QIM2-0.02$\|D\|$	QIM2-0.08$\|D\|$
ICQ-LM	1.96315	1.92326	2.24353	2.22824	94.52726	100.41878	85.16131	120.01252
ICQ-SM	3.63365	1.95576	2.58680	2.46656	107.29946	108.99569	118.37343	144.70835
ICQ-MPM	2.31195	2.01719	3.47117	2.25693	94.54965	97.82684	83.80026	82.20479
	QT1-0.02$\|D\|$	QT1-0.08$\|D\|$	–	–	QTM1-0.02$\|D\|$	QTM1-0.08$\|D\|$	–	–
TCQ-LM	1.90949	1.64939	–	–	7.27493	7.19846	–	–
TCQ-LTM	1.88965	1.66203	–	–	7.07214	7.19349	–	–

Table 4.2.2: Comparison of time taken to run WCQ, ICQ and TCQ queries on single table and multiple table databases.

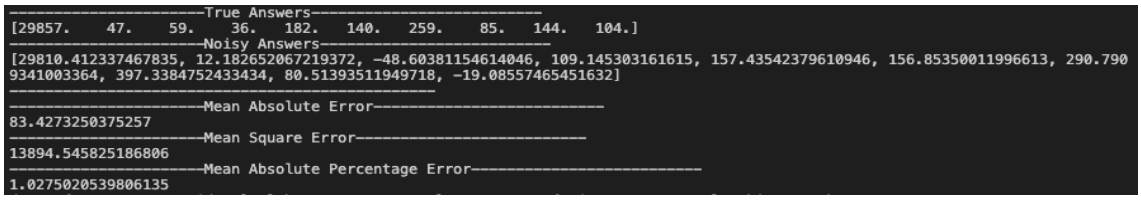

Fig. 4.2.1: True values, Noisy values and Error metrics for WCQ query for single table database.

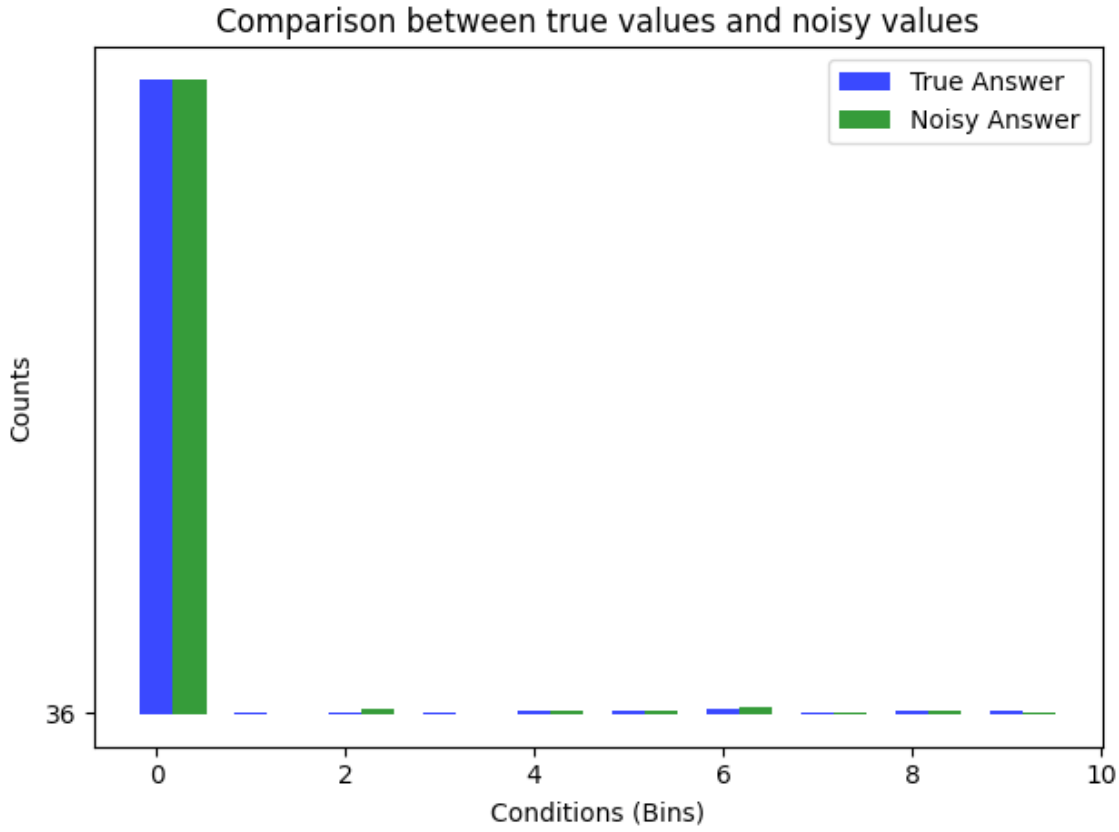

Fig. 4.2.2: Bar graph for WCQ query for single table database.

```
----------True Answers----------
[0.00000e+00 0.00000e+00 3.59064e+05 9.88968e+05 8.19016e+05 4.42715e+05
 1.78063e+05 4.79240e+04 7.60100e+03 6.60000e+02]
----------Noisy Answers----------
[62.67652369754887, -79.86218183540004, 358958.18996474904, 989063.6673204562, 818943.0689134566, 442620.8421793979, 178145.76
904606033, 47886.878919433104, 7387.706192463484, 686.7887742184602]
--------------------------------------------------
----------Mean Absolute Error----------
87.10776767678468
----------Mean Square Error----------
9927.95294040881
```

Fig. 4.2.3: True values, Noisy values and Error metrics for WCQ query for multiple table database.

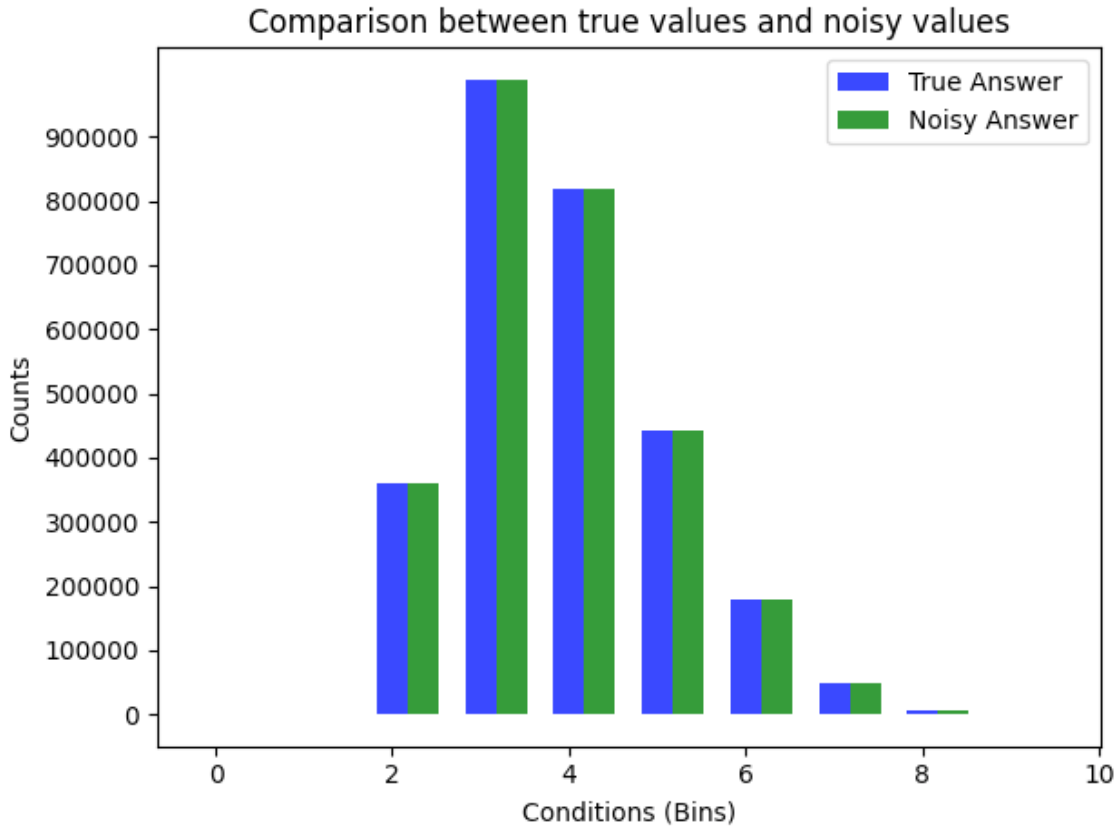

Fig. 4.2.4: Bar graph for WCQ query for multiple table database.

```
--------------------True Answers--------------------------
[   8.   55.  114.  150.  332.  472.  731.  816.  960. 1064.]
--------------------Noisy Answers-------------------------
[-150.27306522067838, 54.28191157282093, 124.40426397963618, 219.35375481876332, 353.0308765175972, 561.150635312464, 750.5543
394305987, 809.1736476302913, 974.1818004963768, 1134.8442676131986]
----------------------------------------------
--------------------Mean Absolute Error-------------------------
46.0337444186201
--------------------Mean Square Error---------------------------
4400.8209381683855
--------------------Mean Absolute Percentage Error--------------------------
2.0719509040400115
```

Fig. 4.2.5: True values, Noisy values and Error metrics for ICQ query for single table database.

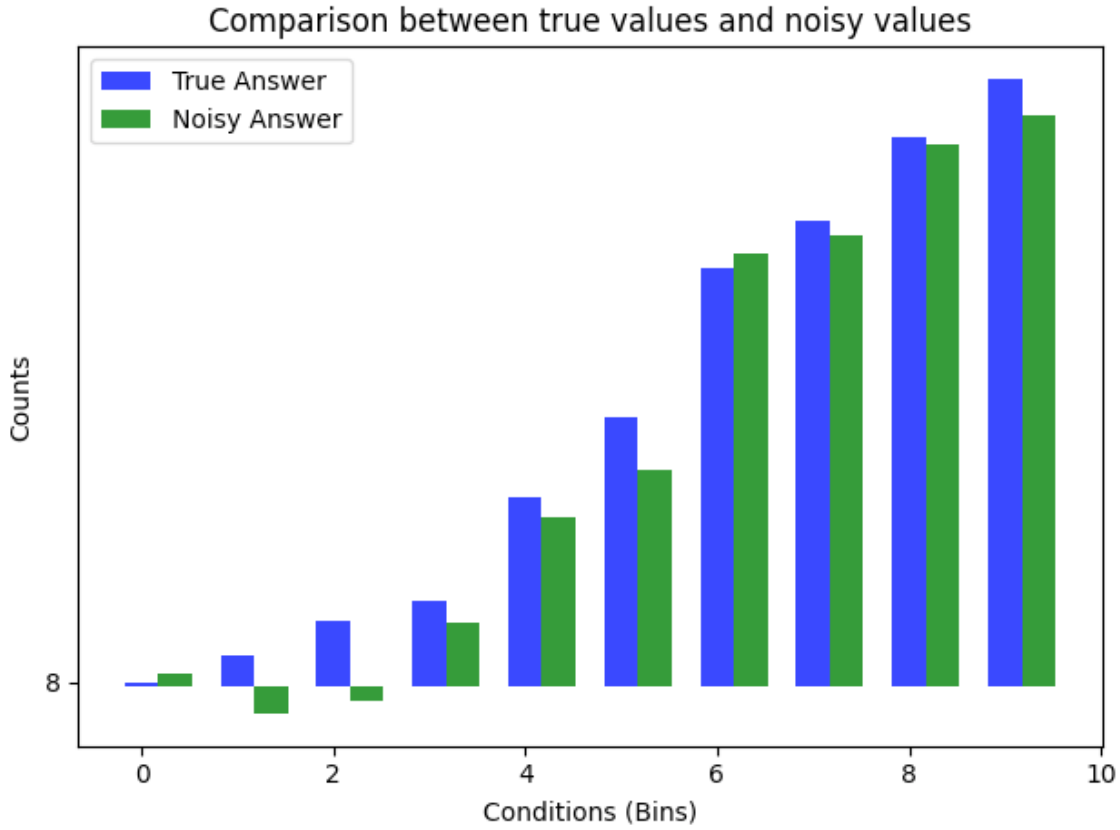

Fig. 4.2.6: Bar graph for ICQ query for single table database.

```
--------------------True Answers--------------------------
[      0.       0.  359105. 1348114. 2167087. 2609780. 2787827. 2835753.
 2843351. 2844011.]
--------------------Noisy Answers-------------------------
[36.9039549221696, 106.8533521361176 5, 359103.99655027746, 1348106.939154995, 2167101.224510345, 2609699.08419539, 2787858.705
0111457, 2835773.4925399777, 2843537.4926960752, 2844098.6750400662]
----------------------------------------------
--------------------Mean Absolute Error-------------------------
57.33272040059963
--------------------Mean Square Error--------------------------
6347.1697591927095
```

Fig. 4.2.7: True values, Noisy values and Error metrics for ICQ query for multiple table database.

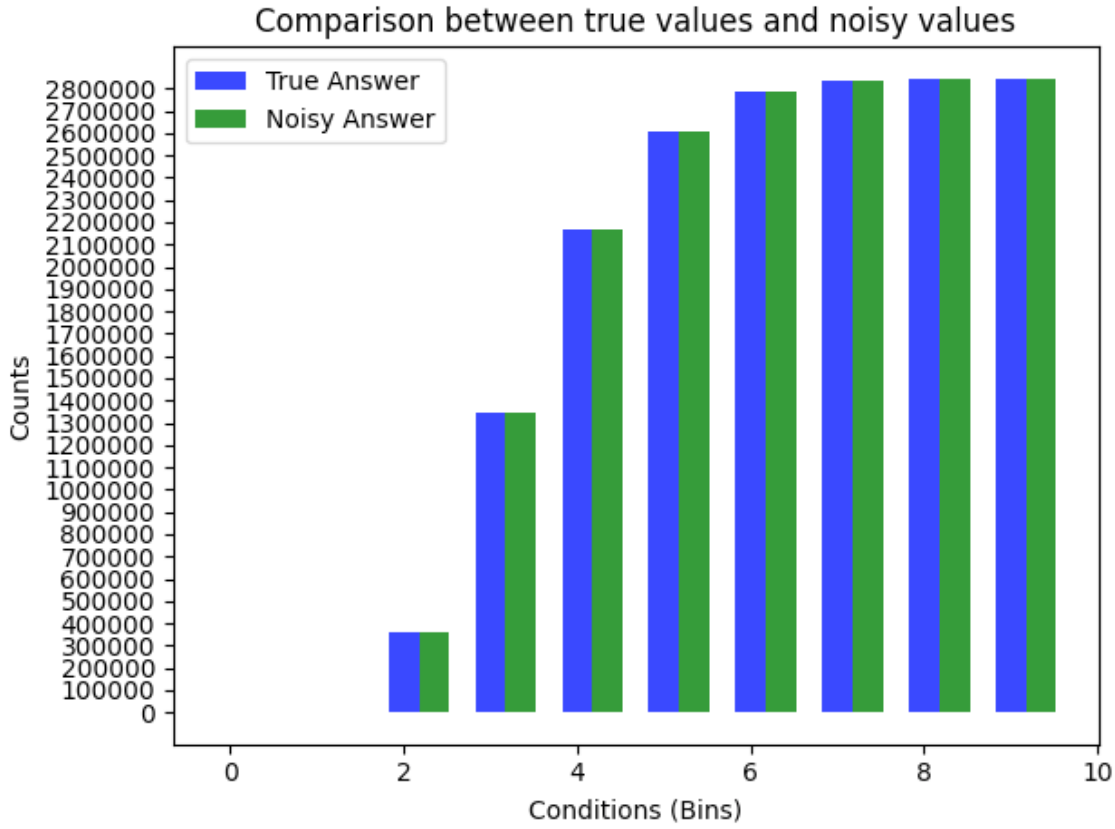

Fig. 4.2.8: Bar graph for ICQ query for multiple table database.

```
----------True Answers----------
[  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0. 395. 550. 712. 753. 720. 765. 877. 798. 841. 785. 835.
 867. 813. 861. 888. 828. 875. 886. 876. 898. 858. 827. 816. 794. 808.
 780. 770. 724. 734. 737. 708. 543. 577. 602. 595. 478. 464. 415. 419.
 366. 358. 366. 355. 312. 300. 258. 230. 208. 178. 150. 151. 120. 108.
  89.  72.  67.  64.  51.  45.  46.  29.  23.  22.  22.  20.  12.   6.
  10.   3.   1.   1.   3.   0.  43.   0.   0.   0.   0.   0.   0.   0.
   0.   0.]
----------Noisy Answers----------
[-80.28264038873536, -22.140149204897643, 27.33114080991357, 24.771371419553585, 8.729081227399723, 90.08508403367577, 36.7203
2154242542, 47.14474151753749, -6.420453773674793, -68.08990164963136, -2.2769913666372337, -2.462627656030588, -6.07575616185
4232, 0.3882237737582841, -5.736707114699666, 5.177085900098763, 14.757686621950223, 391.94790694884693, 600.742051358542, 697
.6023254195838, 736.2446305447071, 624.059156019226, 768.3798257096478, 820.5749146986059, 789.6915510206281, 846.432066217302
1, 778.3601666125545, 860.6803762726827, 820.5249689707347, 828.156444309186, 858.8791751607948, 857.0399999391393, 846.626931
5846416, 876.1980355069736, 894.423687442396, 792.9043429772521, 941.2010501472356, 883.7563985098002, 840.7755504461496, 767.
6384797688495, 750.8010116043542, 723.849814734075, 778.1846138282508, 787.9442772303721, 743.2223945026615, 670.8907131709091
, 693.8924207606825, 720.2619147163815, 530.5047091317172, 568.756477745576, 598.0823940821265, 599.1987757383012, 510.6276559
6900215, 451.44127248564865, 430.2085444470303, 416.513182760572, 368.7286975864676, 319.97687322945933, 385.70157349557775, 3
68.3268426663973, 376.52737397649406, 237.61413375338978, 257.29539936128384, 184.16209098590812, 211.6299985168244, 311.12519
567845754, 173.09630147071132, 137.2593594035295, 61.03274507990611, 115.6941515634462, 31.5300999211663382, 51.5019772265388,
42.446015186590614, 45.082712453129034, 52.01947979797548, 13.49449466787172, 75.23772166954575, 32.70916814653399, 29.5726990
31606984, 33.77676975222998, 84.30789182786322, 24.213828068151543, -7.582925630736884, 18.370331358963067, 50.08924690034885,
 -23.84341163372665, 2.381259472340557, -52.962649957677066, -5.890572274484862, 25.09551531865554, 32.7677180603979, 20.24655
05821227, 8.978167809093879, -44.64556043286753, -5.817557118148207, -8.085044893884788, 25.919066866804332, 0.505369098477149
1, 27.59986401292179, -11.413904472765141]
------------------------------------------------
----------Mean Absolute Error----------
25.59972787639078
----------Mean Square Error----------
1299.586367623812
```

Fig. 4.2.9: True values, Noisy values and Error metrics for TCQ query for single table database.

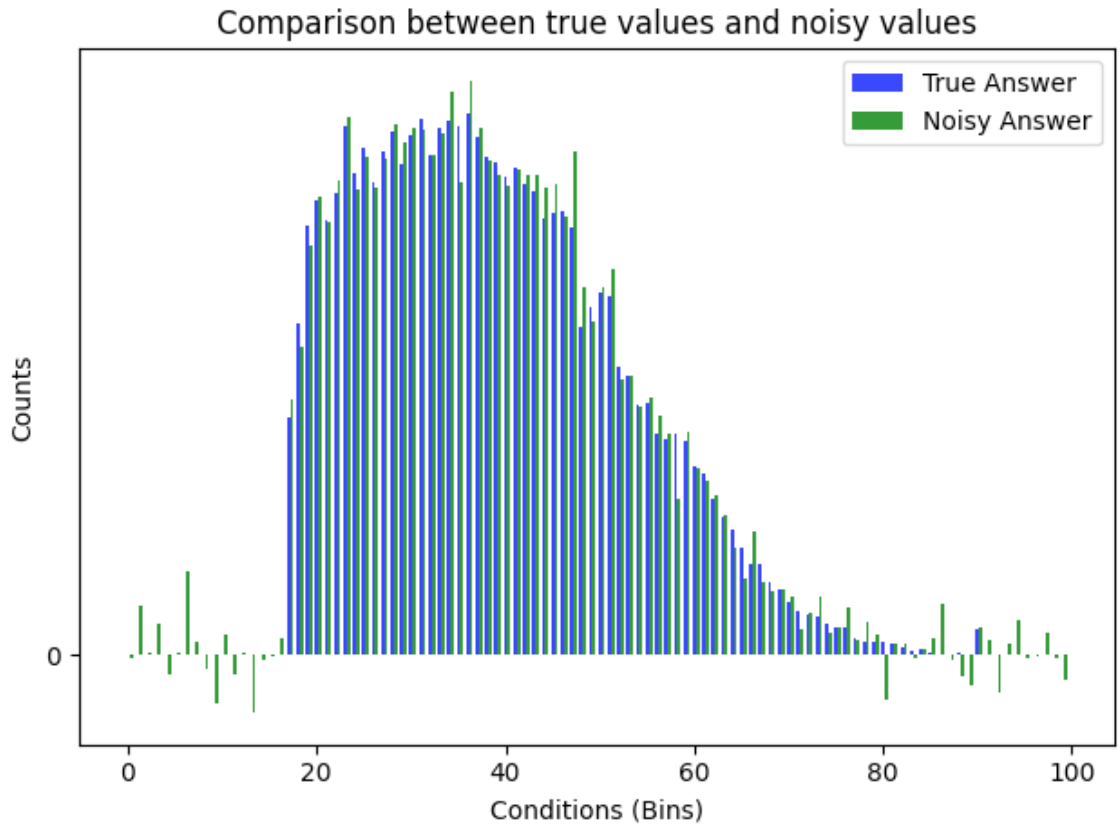

Fig. 4.2.10: Bar graph for TCQ query for single table database.

```
----------True Answers----------
[    0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
     0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
     0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
     0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
     0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
     0.     0.     0.     0.     0.  1940. 22913. 23080. 23014. 23065.
 23126. 23311. 23276. 22850. 23051. 23104. 23228. 22857. 21209.     0.
     0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
     0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
     0.     0.     0.     0.     0.     0.     0.     0.     0.     0.]
----------Noisy Answers----------
[42.98167505138099, -17.206162046264467, 4.331742742561997, 9.33237563243022, 14.44484867761132, -4.510339010637339, -18.28365
5651373618, -12.384960475352992, -14.275430728318986, -22.635916753755723, -25.82999180678911, 5.302689290675692, 28.231220425
907278, -3.6603534995068916, 129.37977919479323, -10.491569918707945, -15.492215722911201, 16.86744930818949, -17.615604163284
242, 21.72628548829819, -19.638679419009208, -24.172498148325428, 3.412948937566769, 111.0625794092368, 68.33650280441822, 3.0
544752523474554, 1.8967471495676516, -3.4849096279723812, -154.47363896308008, -8.947779740786912, -65.12683618352744, 26.8821
00354711575, -38.16367578770714, -54.78009580129042, 0.7569228137039792, -22.315403895535635, -89.04331738898364, -6.353291576
4359466, -34.29336499602939, 21.83133979380894, 43.62035122529943, -42.976703758877605, 3.938401024932643 8, 33.744686357348506
, 4.185639652082563, 13.677091839316736, -12.381033873200598, 6.111641328893772, -30.702287728695694, 0.031776470683975405, 62
.05236891064015, -132.13763122226695, 4.590736612930226, -27.380105164112734, -17.016427874764368, 2010.8602842091345, 22911.4
57709427592, 23083.10167202211, 22952.218614763027, 23035.601933303496, 23122.216692149676, 23313.787323388224, 23157.70906023
9253, 22865.901818073762, 23013.85767666153, 23073.827928578256, 23295.12243550507, 22902.028868537647, 21215.43569240928, -6.
861215975670616, 60.942812306283344, -13.39887913773507, -9.028789945803412, 8.0162015867376, 3.851208696052648, 43.1396445948
5865, -117.1950099067946, -4.86140061393156, 26.285035859647415, 25.15782212585498, -54.44950676728479, 10.475253497189074, -0
.13843282983699293, 8.365114811007718, 16.348220011646113, -8.647090301215709, -101.54127857458566, -5.739311666929754, 17.126
598523170795, -16.576493606165442, 41.638608193147654, -168.05383312322914, 90.4336425008676, 54.98203904126026, 2.77344278212
42176, 39.60505399934113, 8.35764123694081, 27.139261196316742, 5.368845369699646, -7.811753248039254]
------------------------------------------------
----------Mean Absolute Error----------
31.25216177728604
----------Mean Square Error----------
2253.909087881329
```

Fig. 4.2.11: True values, Noisy values and Error metrics for TCQ query for multiple table database.

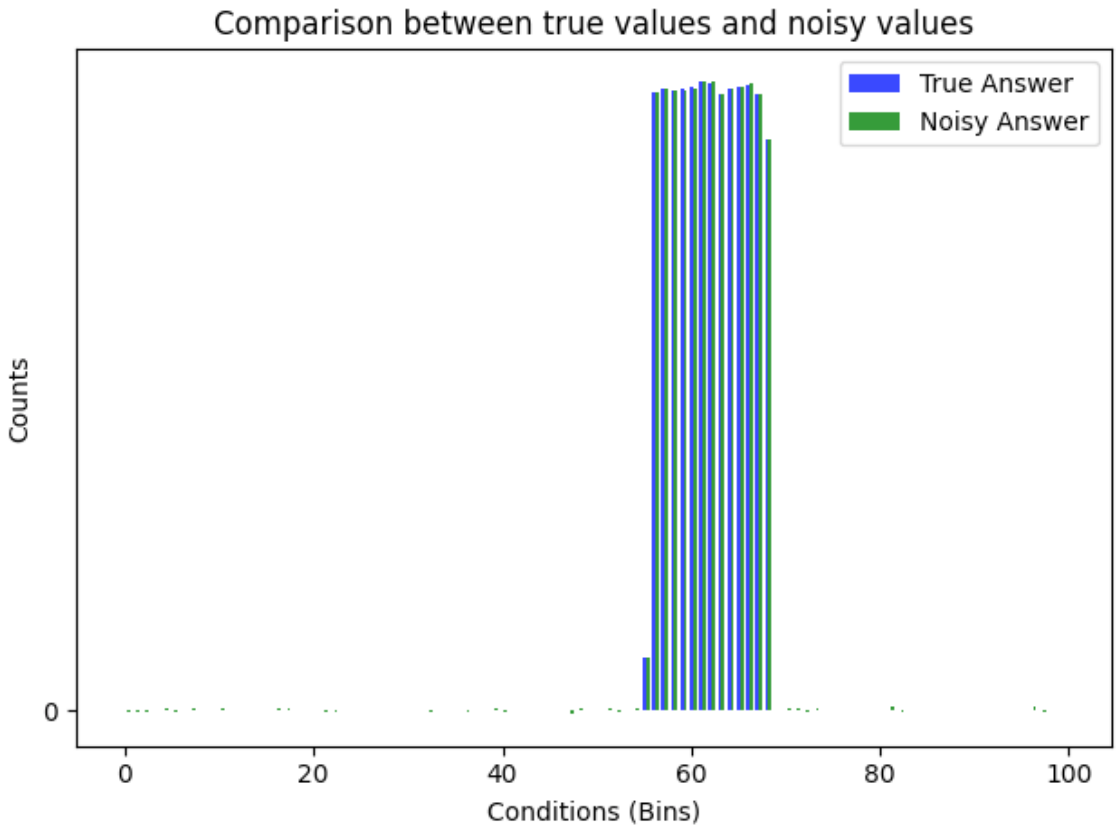

Fig. 4.2.12: Bar graph for TCQ query for multiple table database.

Figures from 4.2.1 to 4.2.12 are screenshots of when I ran extended APEx on different parameters as depicted in Table 4.1.1.

4.3 Limitations and Assumptions

While experimenting with APEx on a multiple-table database, I have assumed that the execution time difference due to the difference in the size of the dataset is negligible. It's a limitation of this research. It can be an exciting phenomenon to try to divide the same single-table database into multiple tables and then test APEx. In this research I have taken different database hich was already divided among multiple tables. By doing this, the size of the dataset will be identical, and the execution time difference will not include the time taken to run the queries on the database.

CHAPTER 5

Conclusion and Future Work

In this research, not only I studied APEx in depth but I also tried to implement one of the extensions mentioned in the original paper. According to my observation, APEx can easily be scaled to Big Data databases with multiple table. The privacy cost of running queries on single table databases and multiple table databases did not differ that much. The only difference that I noted was that the time taken to run queries on multiple table queries was almost ten folds. Now, this could be because of the difference in the size of the databases. Using experiments with query benchmarks and entity resolution application, I established that APEx allows high exploration quality with a reasonable privacy loss.

This research opens up various interesting future expansions along with those mentioned in the original paper. A few of those:

- Evaluate APEx on NoSQL or graph databases such as MongoDB, DynamoDB, GraphDB.
- Integrating it with data visualization tools.
- Integrating it with Machine Learning algorithms like Neural Networks.

REFERENCES

[1] Agarwal, S., Mozafari, B., Panda, A., Milner, H., Madden, S., and Stoica, I. (2013). Blinkdb: Queries with bounded errors and bounded response times on very large data. In *Proceedings of the 8th ACM European Conference on Computer Systems*, EuroSys '13, page 29–42, New York, NY, USA. Association for Computing Machinery.

[2] Barth-Jones, D. (2012). The 're-identification' of governor william weld's medical information: A critical re-examination of health data identification risks and privacy protections, then and now. *SSRN Electronic Journal*.

[3] Dheeru, D. (2017). Karra taniskidou e. *UCI machine learning repository*, 12.

[4] Dwork, C. (2006). Differential privacy. In *Proceedings of the 33rd International Conference on Automata, Languages and Programming - Volume Part II*, ICALP'06, page 1–12, Berlin, Heidelberg. Springer-Verlag.

[5] Dwork, C., McSherry, F., Nissim, K., and Smith, A. (2006a). Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography*, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), pages 265–284. 3rd Theory of Cryptography Conference, TCC 2006 ; Conference date: 04-03-2006 Through 07-03-2006.

[6] Dwork, C., McSherry, F., Nissim, K., and Smith, A. (2006b). Calibrating noise to sensitivity in private data analysis. In *Proceedings of the Third Conference on Theory of Cryptography*, TCC'06, page 265–284, Berlin, Heidelberg. Springer-Verlag.

[7] Dwork, C., McSherry, F., Nissim, K., and Smith, A. (2017). Calibrating noise to sensitivity in private data analysis. *Journal of Privacy and Confidentiality*, 7(3):17–51.

[8] Dwork, C. and Roth, A. (2014). The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3–4):211–407.

[9] Ge, C., He, X., Ilyas, I. F., and Machanavajjhala, A. (2019). Apex: Accuracy-aware differentially private data exploration. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, page 177–194, New York, NY, USA. Association for Computing Machinery.

[10] Ge, C., Ilyas, I., He, X., and Machanavajjhala, A. (2017). Private exploration primitives for data cleaning. *arXiv preprint arXiv:1712.10266*.

[11] Greenberg, A. (2016). Apple's' differential privacy'is about collecting your data—but not your data.(2016). *URL www. wired. com/2016/06/apples-differential-privacy-collecting-data*.

[12] Haney, S., Machanavajjhala, A., Abowd, J. M., Graham, M., Kutzbach, M., and Vilhuber, L. (2017). Utility cost of formal privacy for releasing national employer-employee statistics. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, page 1339–1354, New York, NY, USA. Association for Computing Machinery.

[13] Johnson, N., Near, J. P., and Song, D. (2017a). Practical differential privacy for sql queries using elastic sensitivity. *arXiv preprint arXiv:1706.09479*.

[14] Johnson, N. M., Near, J. P., and Song, D. X. (2017b). Practical differential privacy for SQL queries using elastic sensitivity. *CoRR*, abs/1706.09479.

[15] King, G. and Persily, N. (2018). A new model for industry–academic partnerships. *PS: Political Science & Politics*, pages 1–7.

[16] Konda, P., Das, S., Suganthan G. C., P., Doan, A., Ardalan, A., Ballard, J. R., Li, H., Panahi, F., Zhang, H., Naughton, J., Prasad, S., Krishnan, G., Deep, R., and Raghavendra, V. (2016). Magellan: Toward building entity matching management systems. *Proc. VLDB Endow.*, 9(12):1197–1208.

[17] Li, C., Hay, M., Rastogi, V., Miklau, G., and McGregor, A. (2010). Optimizing linear counting queries under differential privacy. In *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '10, page 123–134, New York, NY, USA. Association for Computing Machinery.

[18] Li, C., Miklau, G., Hay, M., Mcgregor, A., and Rastogi, V. (2015). The matrix mechanism: Optimizing linear counting queries under differential privacy. *The VLDB Journal*, 24(6):757–781.

[19] Ligett, K., Neel, S., Roth, A., Waggoner, B., and Wu, Z. S. (2017). Accuracy first: Selecting a differential privacy level for accuracy-constrained erm. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 2563–2573, Red Hook, NY, USA. Curran Associates Inc.

[20] McKenna, R., Miklau, G., Hay, M., and Machanavajjhala, A. (2018). Optimizing error of high-dimensional statistical queries under differential privacy. *Proc. VLDB Endow.*, 11(10):1206–1219.

[21] McSherry, F. D. (2009a). Privacy integrated queries: An extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD '09, page 19–30, New York, NY, USA. Association for Computing Machinery.

[22] McSherry, F. D. (2009b). Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 19–30.

[23] Narayanan, A. and Shmatikov, V. (2006). How to break anonymity of the netflix prize dataset.

[24] Parameswarany, A., Sarma, A. D., Venkataramani, V., Papaemmanouil, O., Diao, Y., Dimitriadou, K., Peng, L., Eichmann, P., Zgraggen, E., Zhao, Z., et al. (2018). Data engineering. *ACM*.

[25] Patrick crews, G. M. (2004). Mysql: Employees sample database.

[26] Patrick Crews, G. M. (2008). Employee sample database. `https://dev.mysql.com/doc/employee/en/employees-introduction.html`.

[27] Proserpio, D., Goldberg, S., and McSherry, F. (2012). A workflow for differentially-private graph synthesis. In *Proceedings of the 2012 ACM Workshop on Workshop on Online Social Networks*, WOSN '12, page 13–18, New York, NY, USA. Association for Computing Machinery.

[28] Proserpio, D., Goldberg, S., and McSherry, F. (2014). Calibrating data to sensitivity in private data analysis: A platform for differentially-private analysis of weighted datasets. *Proc. VLDB Endow.*, 7(8):637–648.

[29] Stonebraker, M., Bruckner, D., Ilyas, I. F., Beskales, G., Cherniack, M., Zdonik, S. B., Pagan, A., and Xu, S. (2013). Data curation at scale: the data tamer system. In *Cidr*.

[30] TLC, N. (2017). Nyc taxi and limousine commission (tlc) trip record data. *URL http://www. nyc. gov/html/tlc/html/about/trip_record_data. shtml*.

[31] Vilhuber, L. and Schmutte, I. M. (2016). Proceedings from the 2016 nsf–sloan workshop on practical privacy. In *Proceedings from the*.

[32] Zhang, D., McKenna, R., Kotsogiannis, I., Hay, M., Machanavajjhala, A., and Miklau, G. (2018). Ektelo: A framework for defining differentially-private computations. In *Proceedings of the 2018 International Conference on Management of*

Data, SIGMOD '18, page 115–130, New York, NY, USA. Association for Computing Machinery.

APPENDIX

Code changes on APEx

Query.py

Listing 1: Condition counts for multiple table schema

```
from conn import DB
import numpy as np
from query import Type
from sklearn.metrics import f1_score


class QueryBag:
    """intermediate structure when
     formalizing query matrix representation"""
    def __init__(self):
        self.id_set = set()
        self.query_list = []

    def add_q(self, new_q):
        if type(new_q) is list:
            self.query_list.extend(new_q)
        else:
            self.query_list.append(new_q)

    def add_ids(self, ids):
        self.id_set.update(set(ids))
```

```
class Query:
    """the query class"""

    query_type = Type.QueryType.WCQ

    def __init__(self):
        self.index = -1
        self.cond_list = []

        self.query_type = -1

        # intermediate holder for each condition/predicate
        self.query_bags = []
        # W
        self.query_matrix = []
        # x
        self.domain_hist = []
        # the true counting of queries using matrix mechanism
        self.true_answer = []
        self.noisy_answer = []  # noisy counting
        self.lap_noise = []  # a list of laplace noise
        # cardinality of the table, will assign when get_cond_list
        self.cardinality = -1

        # WCQ related
        self.answer_diff = []

        # ICQ related
        self.icq_c = -1
        self.selected_cond_index = []
        self.count_poking = -1
        # a list of true poking times per predicate
        self.real_count_poking = []
```

```
        # TCQ related
        self.tcq_k = -1
        self.selected_arg_list = []

        # data set related
        self.table_name = ""
        self.data_size = 1
        self.db = ""

        self.m_type = ""
        self.query_key = -1

        self.Wx_cache_W = dict()
        self.Wx_cache_x = dict()
        self.Wx_cache_N = dict()
        self.eps_cache = dict()

    # set the query index
    def set_index(self, i):
        self.index = i

    # set query predicates
    def set_cond_list(self, p):
        self.cond_list = p

    # set the table and db connection
    def set_data_and_size(self, ds, s):
        if ds == 'census':
            self.table_name = 'census.income'
        elif ds == 'location':
            self.table_name = 'location.trip'
        elif ds == 'employees':
            self.table_name = 'employees.salaries'
        else:
            print("ERROR: not supported data set ", ds)
        # set the db connection and size
```

```
        self.db = DB.DB(ds)
        self.data_size = s

    def set_query_type(self, qt, icq_c=-1, mpm_p=-1, tcq_k=-1):
        self.query_type = qt
        if qt == Type.QueryType.ICQ:
            assert icq_c != -1
            self.icq_c = icq_c
            self.count_poking = mpm_p
            self.real_count_poking = 0
        elif qt == Type.QueryType.TCQ:
            assert tcq_k != -1
            self.tcq_k = tcq_k

    def set_mechanism_type(self, m):
        self.m_type = m

    def set_cache(self, w, x, n, e):
        self.Wx_cache_W = w
        self.Wx_cache_x = x
        self.Wx_cache_N = n
        self.eps_cache = e

    # collect the ids for each condition to build histogram
    def get_cond_ids(self):
        counter = 0
        table_name = self.table_name + str(self.data_size)

        for cond in self.cond_list:
# initialize an object to hold the ids and its belonging queries
            crnt_query_bag = QueryBag()
            crnt_query_bag.add_q(counter)

            # prepare the query to query ids
            crnt_query = 'select id from '\
                + table_name + ' where ' + cond
```

```
            crnt_query_rs = self.db.run(crnt_query)

            # store the row ids
            crnt_query_bag.add_ids([i[0] for i in crnt_query_rs])
            self.query_bags.append(crnt_query_bag)
            # print(len(crnt_query_rs))
            counter = counter + 1

        # get the total count from the table
        crnt_query = 'select count(*) from ' + table_name
        self.cardinality = self.db.run(crnt_query)[0][0]
        self.db.close_conn()

    def get_cond_counts(self):
        table_name = self.table_name + str(self.data_size)
        if table_name == 'employees.salaries1':
            table_name = 'employees.salaries'
        counts_list = []

        for cond in self.cond_list:
            # prepare the query to query ids
            # print(self.index.__name__)
            if(self.index.__name__ == 'qim_2'):
                # print("Here")
                crnt_query = 'SELECT COUNT(*) FROM \
                (SELECT A.emp_no, A.gender, B.salary \
                FROM employees.employees as A INNER JOIN \
                employees.salaries as B ON \
                A.emp_no = B.emp_no where ' + cond + ') C'
            elif(self.index.__name__ == 'qtm_1'):
                crnt_query = 'Select count(*) from \
                employees.employees where \
                year(curdate()) - year(birth_date) = ' + cond
            else:
                crnt_query = 'select count(*) from ' \
                + table_name + ' where ' + cond
```

```
            counts_list.append(self.db.run(crnt_query)[0][0])

        # get the total count from the table
        if(self.index.__name__ == 'qim_2'):
            crnt_query = 'SELECT COUNT(*) FROM (SELECT A.emp_no,\
                A.gender, B.salary FROM employees.employees as \
                A INNER JOIN employees.salaries as B ON A.emp_no = B.emp_no) C'
        elif(self.index.__name__ == 'qtm_1'):
            crnt_query = 'select count(*) from employees.employees'
        else:
            crnt_query = 'select count(*) from ' + table_name
        self.cardinality = self.db.run(crnt_query)[0][0]
        self.db.close_conn()

        return counts_list

    # construct the matrix representation of the query
    def to_matrix(self):

        # check if the results was previously cached
        # key qw1_100_1
        self.query_key = self.index.__name__ + \
        str(len(self.cond_list)) + str(self.data_size)

        # if self.query_key in self.Wx_cache_W:
        #     self.domain_hist = self.Wx_cache_W.get(self.query_key)
        #     self.query_matrix = self.Wx_cache_x.get(self.query_key)
        #     self.cardinality = self.Wx_cache_N.get(self.query_key)
        #     print("DEBUG: reuse cache query_key=", self.query_key)
        #     return
        # else:
        #     print("DEBUG: no cache query_key=", self.query_key)

        # if the query is qt_2 or qt_4, otherwise just count
        if self.index.__name__ in ['qt_2', 'qt_4']:  # HD
            # first of all, collect the ids for each condition
```

```
            self.get_cond_ids()

            disjoint_query_bags = list()
            print("len(self.query_bags)= ", len(self.query_bags))
            disjoint_query_bags.append(self.query_bags[0])

            for i in range(1, len(self.cond_list)):
# use the current query bag to iterate the existing disjoint bags
                crnt_query_bag = self.query_bags[i]
                disjoint_query_bags_size = len(disjoint_query_bags)

                for j in range(0, disjoint_query_bags_size):
                    existing_query_bag = disjoint_query_bags[j]
                    # find the intersection
                    set_intersect = existing_query_bag.id_set\
                     & crnt_query_bag.id_set
                    if len(set_intersect) > 0:
                        # intersection is not empty
                        intersect_query_bag = QueryBag()

                        # add the intersected ids
                        # into new query bag
                        intersect_query_bag.add_ids(set_intersect)

                        # set the queries from both
                        # current and previous query bags
                        intersect_query_bag.
                        add_q(existing_query_bag.query_list)
                        intersect_query_bag.
                        add_q(crnt_query_bag.query_list)

                        # extract the intersection
                        # from previous two bags
                        existing_query_bag.id_set =
                        existing_query_bag.id_set - set_intersect
                        crnt_query_bag.id_set =
```

```
                crnt_query_bag.id_set - set_intersect

                # append the intersected
                # query bag into disjoint bag list
                disjoint_query_bags.append(intersect_query_bag)

        # add the remaining
        # crnt_query_bag into disjoint bag list
        if len(crnt_query_bag.id_set) > 0:
            disjoint_query_bags.append(crnt_query_bag)

    # clean the disjoin query bags
    caught_count = 0
    for crnt_bag in disjoint_query_bags:
        if len(crnt_bag.id_set) == 0:
            disjoint_query_bags.remove(crnt_bag)
        caught_count += len(crnt_bag.id_set)

    # add the uncaught ids into the last position
    if caught_count < self.cardinality:
        last_query_bag = QueryBag()
        last_query_bag.add_ids(range(0,
        self.cardinality - caught_count))
        disjoint_query_bags.append(last_query_bag)

    disjoint_query_bags_size = len(disjoint_query_bags)

    # generate the domain histogram
    self.domain_hist = [len(crnt_bag.id_set)
    for crnt_bag in disjoint_query_bags]
    print("domain_hist: \n", self.domain_hist)
    print("domain_hist_len: \n", len(self.domain_hist))

    # test disjoint query bags
    # for i in range(0, len(disjoint_query_bags)):
    #     crnt_bag = disjoint_query_bags[i]
```

```
            #     print("query_list= ", crnt_bag.query_list)
            #     print("id_set= ", crnt_bag.id_set)

            # initialize a matrix
            self.query_matrix = np.zeros((len(self.cond_list),
            disjoint_query_bags_size))
            for i in range(0, disjoint_query_bags_size):
                crnt_query_list = disjoint_query_bags[i].query_list
                for j in range(0, len(crnt_query_list)):
                    self.query_matrix[crnt_query_list[j]][i] = 1

            print("query matrix: \n", self.query_matrix)

        else:
            # for 1D and 2D histogram, just count(*)

            # get all the counts for each predicate
            self.domain_hist = self.get_cond_counts()

            # for i in range(0, len(self.domain_hist)):
            #     print(i, "\t", self.domain_hist[i])

            if self.index.__name__ in ['qw_1', 'qwm_1',
            'qi_3', 'qt_1', 'qtm_1', 'qw_4', 'qi_2',
            'qim_2', 'qt_3']:  # 1D/2D histogram
                is_hist = True
            else:
                is_hist = False
                assert self.index.__name__ in ['qw_2', 'qwm_2',
                 'qw_3', 'qi_1', 'qim_1', 'qi_4']  # 1D prefix

            if is_hist:
                # for histogram query
                count_row_by_predicates = sum(self.domain_hist)
            else:
                # for prefix query
```

```
                count_list = list(self.domain_hist)
                count_row_by_predicates = self.domain_hist[-1]
                for i in range(1, len(self.domain_hist)):
                    self.domain_hist[i] = count_list[i]
                    - count_list[i - 1]

                    assert self.domain_hist[i] >= 0

            # now the domain_hist represents non-overlapping sets
            count_remaining = self.cardinality -
            count_row_by_predicates
            if count_remaining > 0:
                self.domain_hist.append(count_remaining)
            self.query_matrix = np.zeros((len(self.cond_list),
             len(self.domain_hist)))

            if is_hist:
                for i in range(0, len(self.cond_list)):
                    self.query_matrix[i][i] = 1
            else:
                for i in range(0, len(self.cond_list)):
                    for j in range(0, i+1):
                        self.query_matrix[i][j] = 1

        # cache the results
        self.Wx_cache_W[self.query_key] = self.domain_hist
        self.Wx_cache_x[self.query_key] = self.query_matrix
        self.Wx_cache_N[self.query_key] = self.cardinality

    # return the sensitivity of matrix
    def get_sensitivity(self):
        return max([sum(a) for a in self.query_matrix])

    # to count f1 for ICQ and TCQ, max error for WCQ
    def get_accuracy(self):
```

```
if self.query_type == Type.QueryType.WCQ:
    # print("DEBUG: max_error",
    # max([abs(crnt_answer_diff) for
    # crnt_answer_diff in self.answer_diff]))
    max_errpr = 1.0 - max([abs(crnt_answer_diff)
    for crnt_answer_diff
    in self.answer_diff]) / self.cardinality
    # print(max_errpr)
    return [max_errpr, 'N/A']

true_list = [0] * len(self.cond_list)
pred_list = [0] * len(self.cond_list)

c = 0
if self.query_type == Type.QueryType.ICQ:
    c = self.icq_c
    for i in self.selected_cond_index:
        pred_list[i] = 1
    for i in range(0, len(self.cond_list)):
        if self.true_answer[i] > self.icq_c:
            true_list[i] = 1

elif self.query_type == Type.QueryType.TCQ:
    for i in self.selected_arg_list:
        pred_list[i] = 1

    cp_true = list(self.true_answer)
    cp_true.sort(reverse=True)
    c = cp_true[self.tcq_k - 1]

    for i in range(0, len(self.cond_list)):
        if self.true_answer[i] >= c:
            true_list[i] = 1

# return sum(true_list), sum(pred_list),
#  f1_score(true_list, pred_list)
```

```
        f1 = f1_score(true_list, pred_list)


        alpha_hat = 0
        for i in range(0, len(self.cond_list)):
            if true_list[i] != pred_list[i]:
                alpha_hat = max(alpha_hat, abs(c -
                self.true_answer[i]))

        error = alpha_hat / self.cardinality
        # print("Error: "+error+" F1 Score: "+f1)
        return [error, f1]
```

Query_Gen.py

Listing 2: Condition counts for multiple table schema

```
import math



trip_amount_bound = 10.0
trip_amount_step = 0.1
# trip_amount_bound = 1000000
# trip_amount_step = 10000

trip_distance_bound = 10.0
trip_distance_step = 0.1

income_capgain_bound = 5000.0
income_capgain_step = 50.0
# income_capgain_bound = 100000
# income_capgain_step = 1000
salary_capgain_bound = 150000.0

# generate qw1, given the workload size l
```

```
def qw_1(l):
    bin_size = income_capgain_bound / l
    predicates = []
    for i in range(0, l):
        p = str(i * bin_size) +
        "<=capgain and capgain<" + str((i + 1) * bin_size)
        predicates.append(p)

    return predicates[:l]

def qwm_1(l):
    bin_size = salary_capgain_bound /l
    predicates = []
    for i in range(0,l):
        p = str(i * bin_size) +
        "<=salary and salary<" + str((i+1) * bin_size)
        predicates.append(p)

    return predicates[:l]

# generate qw2, given the workload size l
def qw_2(l):
    bin_size = income_capgain_bound / l
    predicates = []
    for i in range(0, l):
        p = "capgain>0 and capgain<=" + str((i + 1) * bin_size)
        predicates.append(p)

    return predicates[:l]

# generate qwm2, given the workload size l
def qwm_2(l):
    bin_size = salary_capgain_bound / l
    predicates = []
    for i in range(0, l):
        p = "salary>0 and salary<=" + str((i + 1) * bin_size)
```

```
        predicates.append(p)

    return predicates[:l]

# generate qw3, given workload size l
def qw_3(l):
    bin_size = trip_distance_bound / l
    predicates = []
    for i in range(0, l):
        p = "trip_distance<=" + str((i + 1) * bin_size)
        predicates.append(p)

    return predicates[:l]



# qw4
def qw_4(l):
    domain_passenger_count = 10
    count_total_amount = int(l / domain_passenger_count)
    total_amount_bin_size =
    trip_amount_bound / count_total_amount
    predicates = []

    for i in range(0, count_total_amount):
        l_total_amount = i * total_amount_bin_size
        r_total_amount = (i + 1) * total_amount_bin_size
        for j in range(0, domain_passenger_count):
            p = str(l_total_amount) +
            "<=total_amount and total_amount<" +
            str(r_total_amount) + \
                " and passenger_count=" + str(j)
            predicates.append(p)

    return predicates[:l]
```

```
# qi1
def qi_1(l):
    return qw_2(l)

# qim1
def qim_1(l):
    return qwm_2(l)

# qi2
def qi_2(l):
    if l == 1:
        l = 2
    count_capgain = int(l / 2.0)
    # two values on second dimension
    total_capgain_bin_size =
    int(income_capgain_bound / count_capgain)
    predicates = []
    for i in range(0, count_capgain):
        l_capgain = i * total_capgain_bin_size
        r_capgain = (i + 1) * total_capgain_bin_size

        p1 = str(l_capgain) +
        "<=capgain and capgain<" + str(r_capgain) + " and sex='Male'"
        p2 = str(l_capgain) +
        "<=capgain and capgain<" + str(r_capgain) + " and sex='Female'"

        predicates.append(p1)
        predicates.append(p2)

    return predicates[:l]

# qim2
def qim_2(l):
    if l == 1:
        l = 2
    count_capgain = int(l / 2.0)
```

```
    # two values on second dimension
    total_capgain_bin_size =
    int(salary_capgain_bound / count_capgain)
    predicates = []
    for i in range(0, count_capgain):
        l_capgain = i * total_capgain_bin_size
        r_capgain = (i + 1) * total_capgain_bin_size

        p1 = str(l_capgain) +
        "<=salary and salary<" + str(r_capgain) + " and gender='M'"
        p2 = str(l_capgain) +
        "<=salary and salary<" + str(r_capgain) + " and gender='F'"

        predicates.append(p1)
        predicates.append(p2)

    return predicates[:l]

# qi3
def qi_3(l):
    bin_size = trip_amount_bound / l
    predicates = []
    for i in range(0, l):
        p = str(i * bin_size) +
        "<=fare_amount and fare_amount<" + str((i + 1) * bin_size)
        predicates.append(p)

    return predicates[:l]


# qi4
def qi_4(l):
    domain = trip_amount_bound
    bin_size = domain / l
    predicates = []
    for i in range(0, l):
```

```
        p = "total_amount<=" + str((i + 1) * bin_size)
        predicates.append(p)

    return predicates[:l]


# qt1
def qt_1(l):
    bin_size = salary_capgain_bound / l
    predicates = []
    for i in range(0, l):
         p = "age=" + str(i)
        predicates.append(p)

    return predicates[:l]

# qtm1
def qtm_1(l):
    bin_size = income_capgain_bound / l
    predicates = []
    for i in range(0, l):
        p = str(i)
        predicates.append(p)

    return predicates[:l]


# qt2
def qt_2(l):
    predicates = []

    age = range(1, 101)
    workclass = ['Private', 'Self-emp-not-inc',
    'Self-emp-inc', 'Federal-gov',
    'Local-gov', 'State-gov', 'Without-pay', 'Never-worked']
    education = ['Bachelors', 'Some-college',
```

```
'11th', 'HS-grad', 'Prof-school', 'Assoc-acdm', 'Assoc-voc',
'9th', '7th-8th', '12th', 'Masters', '1st-4th', '10th',
 'Doctorate', '5th-6th', 'Preschool']
edunum = range(1, 17)
marital = ['Married-civ-spouse', 'Divorced', 'Never-married',
 'Separated', 'Widowed', 'Married-spouse-absent', 'Married-AF-spouse']
occupation = ['Tech-support', 'Craft-repair', 'Other-service',
 'Sales', 'Exec-managerial', 'Prof-specialty',
  'Handlers-cleaners', 'Machine-op-inspct',
   'Adm-clerical', 'Farming-fishing', 'Transport-moving',
    'Priv-house-serv', 'Protective-serv', 'Armed-Forces']
relationship = ['Wife', 'Own-child', 'Husband', 'Not-in-family',
 'Other-relative', 'Unmarried']
race = ['White', 'Asian-Pac-Islander',
 'Amer-Indian-Eskimo', 'Other', 'Black']
sex = ['Male', 'Female']
capgain = frange(0, income_capgain_bound, income_capgain_step)
caploss = range(0, 100, 1)
hourweek = range(1, 101)
country = ['United-States', 'Cambodia', 'England', 'Puerto-Rico',
 'Canada', 'Germany', 'Outlying-US(Guam-USVI-etc)',
 'India', 'Japan', 'Greece', 'South', 'China', 'Cuba',
  'Iran', 'Honduras', 'Philippines', 'Italy',
  'Poland', 'Jamaica', 'Vietnam', 'Mexico',
   'Portugal', 'Ireland', 'France', 'Dominican-Republic',
  'Laos', 'Ecuador', 'Taiwan', 'Haiti',
   'Columbia', 'Hungary', 'Guatemala', 'Nicaragua',
  'Scotland', 'Thailand', 'Yugoslavia', 'El-Salvador',
   'Trinadad&Tobago', 'Peru',
  'Hong', 'Holand-Netherlands']


idx = 0
while len(predicates) < l:

    start_wl = len(predicates)
```

```
# age
if idx < len(age):
    p = "age=" + str(age[idx])
    predicates.append(p)

# work class
if idx < len(workclass):
    p = "workclass='" + str(workclass[idx]) + "'"
    predicates.append(p)

# education
if idx < len(education):
    p = "education='" + str(education[idx]) + "'"
    predicates.append(p)

if idx < len(edunum):
    p = "edunum=" + str(edunum[idx])
    predicates.append(p)

if idx < len(marital):
    p = "marital='" + str(marital[idx]) + "'"
    predicates.append(p)

if idx < len(occupation):
    p = "occupation='" + str(occupation[idx]) + "'"
    predicates.append(p)

if idx < len(relationship):
    p = "relationship='" + str(relationship[idx]) + "'"
    predicates.append(p)

if idx < len(race):
    p = "race='" + str(race[idx]) + "'"
    predicates.append(p)

if idx < len(sex):
```

```
            p = "sex='" + str(race[idx]) + "'"
            predicates.append(p)

        if idx < len(capgain):
            p = str(capgain[idx]) +
             "<capgain and capgain<" + str(capgain[idx + 1])
            predicates.append(p)

        if idx < len(caploss):
            p = str(caploss[idx]) +
             "<caploss and caploss<" + str(capgain[idx + 1])
            predicates.append(p)

        if idx < len(hourweek):
            p = "hourweek=" + str(hourweek[idx])
            predicates.append(p)

        if idx < len(country):
            p = "country='" + country[idx] + "'"
            predicates.append(p)

        end_wl = len(predicates)
        if start_wl == end_wl:
            break

        idx += 1

    return predicates[:l]


# qt3
def qt_3(l):

    predicates = []
    domain = 266
    count_range = int(math.sqrt(l))
```

```
    step = int(domain / count_range) + 1

    for i in range(0, count_range):
        left_pickup = 1 + i * step
        right_pickup = 1 + (i + 1) * step
        for j in range(0, count_range):
            left_dropoff = 1 + j * step
            right_dropoff = 1 + (j + 1) * step

            p = str(left_pickup) +
             "<=PULocationID and PULocationID<" +
              str(right_pickup) + " and " \
                + str(left_dropoff) +
                 "<=DOLocationID and DOLocationID< "
                  + str(right_dropoff)

            predicates.append(p)

    return predicates[:l]


# qt4
def qt_4(l):

    predicates = []

    pickup_date = range(1, 31)
    pickup_time = range(1, 25)
    dropoff_date = range(1, 31)
    dropoff_time = range(1, 25)

    passenger_count = range(1, 11)
    trip_distance = frange(1.0,
    trip_distance_bound, trip_distance_step)

    PULocationID = range(1, 266)
```

```
DOLocationID = range(1, 266)
fare_amount = frange(0,
trip_amount_bound, trip_amount_step)
tip_amount = frange(0,
trip_amount_bound, trip_amount_step)
tolls_amount = frange(0,
trip_amount_bound, trip_amount_step)
total_amount = frange(0,
trip_amount_bound, trip_amount_step)

idx = 0
while len(predicates) < l:
    start_wl = len(predicates)

    if idx < len(pickup_date):
        p = "date(tpep_pickup_datetime)=" + str(pickup_date[idx])
        predicates.append(p)

    if idx < len(pickup_time):
        p = "hour(tpep_pickup_datetime)=" + str(pickup_time[idx])
        predicates.append(p)

    if idx < len(dropoff_date):
        p = "date(tpep_dropoff_datetime)=" + str(dropoff_date[idx])
        predicates.append(p)

    if idx < len(dropoff_time):
        p = "hour(tpep_dropoff_datetime)=" + str(dropoff_time[idx])
        predicates.append(p)

    if idx < len(passenger_count):
        p = "passenger_count=" + str(passenger_count[idx])
        predicates.append(p)

    if idx < len(trip_distance):
        p = str(trip_distance[idx]) +
```

```
            "<=trip_distance and trip_distance<"
             + str(trip_distance[idx + 1])
        predicates.append(p)

    if idx < len(PULocationID):
        p = "PULocationID=" + str(PULocationID[idx])
        predicates.append(p)

    if idx < len(DOLocationID):
        p = "DOLocationID=" + str(DOLocationID[idx])
        predicates.append(p)

    if idx < len(fare_amount):
        p = str(fare_amount[idx]) +
         "<=fare_amount and fare_amount<" +
         str(fare_amount[idx + 1])
        predicates.append(p)

    if idx < len(tip_amount):
        p = str(tip_amount[idx]) +
         "<=tip_amount and tip_amount<" +
          str(tip_amount[idx + 1])
        predicates.append(p)

    if idx < len(tolls_amount):
        p = str(tolls_amount[idx]) +
         "<=tolls_amount and tolls_amount<" +
          str(tolls_amount[idx + 1])
        predicates.append(p)

    if idx < len(total_amount):
        p = str(total_amount[idx]) +
         "<=total_amount and total_amount<" +
          str(total_amount[idx + 1])
        predicates.append(p)
```

```
        end_wl = len(predicates)
        if start_wl == end_wl:
            break

        idx += 1

    return predicates[:l]


def frange(x, y, jump):
    re_list = []
    while x <= y:
        re_list.append(x)
        x += jump
    return re_list

# wl = 100
# pp = gen_qt_4(wl)
# for i in range(0, wl):
#     print(pp[i])
#
# print(len(pp))
```

Type.py

Listing 3: Condition counts for multiple table schema

```
from enum import Enum


class QueryType(Enum):
    """enum type for different query type"""
    WCQ = 1
    ICQ = 2
    TCQ = 3
```

```
class MechanismType(Enum):
    """enum type for different mechanism"""
    LM = 10
    LM_SM = 11
    LCM = 20
    LCM_SM = 21
    LCM_OM = 22
    LCM_MP = 23
    LCT = 30
    LCT_NM = 31


class ReturnMsgType(Enum):
    QD = 'Denied lack of budget'
    SUCCESS = 'Success'


mechanism_name_dict = {MechanismType.LM: 'WCQ-LM',
                       MechanismType.LM_SM: 'WCQ-SM',
                       MechanismType.LCM: 'ICQ-LM',
                       MechanismType.LCM_SM: 'ICQ-SM',
                       MechanismType.LCM_OM: 'ICQ-OM',
                       MechanismType.LCM_MP: 'ICQ-MPM',
                       MechanismType.LCT: 'TCQ-LM',
                       MechanismType.LCT_NM: 'TCQ-TM'
                       }

wcq_mechanisms = [MechanismType.LM, MechanismType.LM_SM]
icq_mechanisms = [MechanismType.LCM, MechanismType.LCM_SM, MechanismType.LCM_MP]
tcq_mechanisms = [MechanismType.LCT, MechanismType.LCT_NM]
```

VITA AUCTORIS

NAME:	Karmanjot Singh
PLACE OF BIRTH:	Ludhiana, Punjab
YEAR OF BIRTH:	1992
EDUCATION:	Guru Nanak Dev University, B.Tech in Computer Science, Amritsar, Punjab, 2014
	University of Windsor, M.Sc in Computer Science, Windsor, Ontario, 2020