On Scalable Query Pricing in Data Marketplaces

Huanhuan Peng[*1] Xiaoye Miao[*2] Yicheng Fu[*3] Jinshan Zhang[‡4] Shuiguang Deng[†5] Jianwei Yin[*†6]
[*]Center for Data Science, Zhejiang University, Hangzhou, China
[†]College of Computer Science, Zhejiang University, Hangzhou, China
[‡]School of Software and Technology, Zhejiang University, Hangzhou, China
Email: {hhpeng[1], miaoxy[2], fuycc[3], zhangjinshan[4], dengsg[5]}@zju.edu.cn zjuyjw[6]@cs.zju.edu.cn

***Abstract*—Query-based pricing enables personalized data acquisition for data buyers, exhibiting potential in data markets. The state-of-the-art SQL query pricing strategy tackles the *#P-hard arbitrage-free* pricing task with the *quadratic* computational complexity, far from promptly fulfilling customer demands. In this paper, we propose a novel arbitrage-free and scalable pricing framework ARIA to calculate the prices for *various* query types in *linear* time, including select-project-join and simple aggregate (SPJA) queries. For the first time, we model what the query answer *tells* about the value of each tuple and formulate the *tuple-level information* of selection, projection, and simple aggregation queries. We develop several price functions based on the total *information gain* of all tuples. The containing relationship between the query information prevents possible arbitrage arising from query determinacy. We present efficient price computation algorithms to derive the prices of different types of queries with *linear* time complexity, which scan the common *possible value set* of tuples one time. In ARIA, the join query is decomposed as multiple single-relation queries for pricing in linear time. Extensive experiments on real and synthetic datasets demonstrate that, ARIA performs 3x faster than the state of the arts while enjoying desirable pricing characteristics.**

***Index Terms*—Data pricing, Query pricing, Information gain**

I. Introduction

Data serves as the fuel in numerous data-driven applications, driving huge economic value [1], [2]. The global big data market, valued at $327.26 billion in 2023, is projected to experience a compound annual growth rate of 14.9% from 2024 to 2030, according to Grand View Research [3]. Consequently, there emerges a large number of data markets to trade data, e.g., AggData [4] and AWS data marketplace [5], facilitating the data usage for different individuals and parties.

To facilitate the data trade, query-based pricing [6], [7] becomes promising in data markets. It enables customers to express the data need via the *ad hoc query* and pay for the desirable data without unnecessary extras. As exemplified in Fig. 1, each customer could be interested in different aspects of a dataset and purchase desirable data via queries. For instance, an analyst interested in movies in a certain period can purchase the query `select * from Movie where year between 1970 and 2010`. In query-based pricing, the customers pay only for the exact information they need, not the entire dataset. Hence, query-based pricing not only provides high customization and flexibility but also significantly saves costs for data customers. It thereby attracts more buyers and raises the profitability of sellers, fostering a more effective and customer-centric data market.

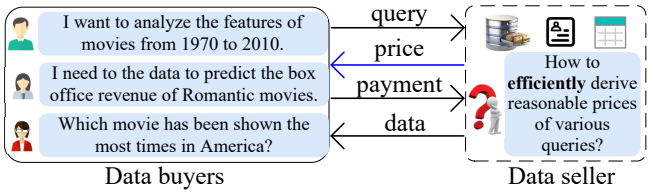


Fig. 1. The example of query pricing in data markets

Existing query pricing research can be summarized into two categories, i.e., usage-based pricing (UBP) and information-based pricing (IBP). UBP prices each query as the minimal sum-up price of tuples or views that can *produce* the query answer [7]–[11]. It works for select-project-join (SPJ) queries but *not* for aggregate queries. In comparison, IBP considers the database as a *variable* with many *possible values*, where the query price depends on the information it reveals about the *true database instance* [12]–[16]. It is feasible for SPJ and aggregate queries as it measures the information, enabling flexible pricing regardless of query type. Besides, IBP can capture the intrinsic value beyond the outputted result while UBP fails. For instance, an empty query result can provide meaningful information (e.g., indicating no purchases of certain products above a specified price), yet it is valued as zero under UBP.

However, these works suffer from the efficiency issue on large-scale databases and complex queries. In UBP, selection and join (SJ) queries can be priced in linear time by considering fine-grained *tuple-level* usage. But it has the *exponential* time complexity in pricing projection queries. In practice, UBP *cannot* finish in *7,200 seconds* when pricing a multi-table projection query on *TPC-H* benchmark [17]. Meanwhile, as it is #P-hard to accurately compute the database information in IBP, the exemplary work QIRANA [14]–[16] employs limited M databases for pricing queries with *quadratic* time complexity $O(M \cdot |D|)$ where $|D|$ is the database size. The high latency impacts subsequent data acquisition and analysis from data customers [18], [19], leading to revenue loss and customer churn. As a result, it is essential to design efficient query pricing algorithms to support more query trade.

Inspired by the scalability of information-based modeling and the efficiency of tuple-level usage in SJ queries, we attempt to measure the query information at the *tuple* level to enhance pricing efficiency for various query types. Specifically, we model each tuple (instead of the whole database) as a variable with many possible values, while the tuple-level information corresponds to the tuple value range narrowed

by the query. For example, querying one attribute of tuple t in relation $R(x, y)$ can narrow the two-dimensional possible value set into the space with one certain dimension (i.e., tuple-level information), reducing the uncertainty on t. For selection queries, the information of all tuples can be derived by scanning all possible values in *linear* time, since tuples in a relation share a *common possible value set*. In contrast, QIRANA needs to scan the M databases, which is more costly.

While it is promising to compute tuple-level information in linear time, there are two remaining open problems for query pricing. The first one is to derive the tuple information under projection, aggregation, and join queries. Each result under these queries may represent multiple tuples and it is hard to recognize the information of each tuple separately in some cases. When tuple information of various queries is ready, the second problem is how to design price functions with desirable characteristics, such as *arbitrage-freeness* and *positivity*, which are the basis to promote healthy data marketplaces [7], [8], [12]. Arbitrage-free pricing ensures that a data buyer cannot acquire the desired result of a query at a lower price by combining the results of other queries. Positivity means that queries with non-empty answers should not be free.

In this work, we present a novel query pricing framework ARIA, which efficiently determines the Arbitrage-fRee price based on the *tuple-level* Information gAin for SPJA queries. We first theoretically derive the tuple-level information for selection, projection, and simple aggregation (including `count`, `max`, and `min`) queries, where each tuple can be considered separately and the tuple information beyond the output can be analyzed. Then, we develop several pricing functions based on the *tuple-level information gain*. The *arbitrage-freeness* is ensured through the *containing relationship between the information*, i.e., the information of a query is contained/inferred by the information from the queries that determine it. The price computation algorithms with *linear* complexity are presented, which traverse the common possible sets once to calculate the query price. For join queries where the tuple information is mixed and cannot be obtained, we decompose each join query as multiple single-relation queries for pricing, which prevents arbitrage based on the query determinacy relationship. Overall, ARIA is scalable to price SPJ and simple aggregation queries in the linear time, while necessary pricing characteristics (such as arbitrage-freeness and positivity) are ensured. The contributions of this paper are as follows.

- We propose a novel arbitrage-free and scalable query pricing framework ARIA for SPJA queries based on the tuple-level information gain.
- We formulate and deduce the tuple information of selection, projection, and aggregation queries and measure information gain to design arbitrage-free price functions.
- Benefiting from the tuple-level granularity information, we propose efficient algorithms to compute the prices of various queries in linear time.
- Extensive experiments on four datasets confirm the efficiency of ARIA, compared to the state of the arts. We also verify the desirable pricing characteristics of ARIA.

TABLE I
SYMBOLS AND DESCRIPTION

Notation	Description
D	the on-sale database with relations $(R_1, \cdots, R_m)$
N_j	the size of the relation R_j
$p(Q)$	the price function that outputs the price of query Q
$\mathcal{Q}$	the query bundle with a set of queries $\{Q_1, \cdots, Q_k\}$
$\mathcal{S}_j$	the set containing all possible values of tuples in R_j
$E_t(Q)$	the value range of the tuple t under Q
$S_j \subseteq \mathcal{S}_j$	the support set on relation R_j in pricing

The rest of the paper is organized as follows. Section II formulates the studied problem. We describe how to model tuple-level information in Section III. We propose the arbitrage-free pricing framework ARIA with an efficient price computation algorithm in Section IV. Section V elaborates ARIA to price projection and aggregation queries. We describe how ARIA handles join queries in Section VI. The experimental results and our findings are reported in Section VII. Section VIII reviews the related work. We conclude this paper in Section IX.

II. PROBLEM FORMULATION

In this section, we formulate pricing characteristics and the studied problem. Table I describes frequently used symbols.

In the query-based data market, the data seller has the on-sale database D with the relations $(R_1, R_2, \cdots, R_m)$. The database schema, the data size N_j of each relation R_j, and the domain of each attribute in R_j are known as public. Then, the seller charges the data buyer according to the query Q rather than the full dataset. The price of query Q is $p(Q)$ given a price function $p(\cdot)$. We focus on pricing select-project-join-aggregate queries formed in SQL language.

Pricing characteristics. Existing studies have explored various features of on-sale data and different scenarios [6], [20]–[23], identifying key characteristics that are essential for effective data pricing. In this work, we focus on the following principles in SQL query pricing to ensure the practicality of the pricing strategy in general scenarios where the intended use of the purchased data is unknown.

First, arbitrage-free pricing has long been a fundamental principle in the design of data markets and has been widely explored in data pricing studies [6]–[16]. A query pricing function $p(\cdot)$ exhibits k-arbitrage on the dataset D if there exists a set of queries $Q, Q_1, \cdots, Q_k$ such that $p(Q) > \sum_{i=1}^{k} p(Q_i)$ and the queries $Q_1, \cdots, Q_k$ determine Q on D. The determinacy relationship means that $Q(D)$ can be derived from $Q_1(D), \cdots, Q_k(D)$ without accessing the database D [24].

Definition 1: **Arbitrage-free property**. A price function $p(Q)$ is arbitrage-free on D iff it does not exhibit k-arbitrage for any $k \in \mathbb{N}^+$.

The arbitrage-free property ensures that, data buyers cannot acquire the desired query for a *lower* price by combining other query results. However, designing an effective query pricing framework requires more than just considering the arbitrage-free property. On the one hand, pricing some queries (with meaningful answers) *free* may not bring arbitrage but is *unreasonable* in practice. On the other hand, Definition 1 builds on the *broad* concept of query determinacy, which does

mid	name	country	rating
1	Inception	USA	8.8
2	Three idiots	India	8.4
3	Your name	Japen	8.4
4	E.T.	USA	7.9

(a) Movie relation

mid	week	country	grosses
1	1	USA	62,785,337
1	2	USA	42,725,012
2	1	India	8,665,678
3	1	USA	1,813,781

(b) Gross relation

Fig. 2. The on-sale database

mid	name	country	rating
?	?	?	?
?	?	?	?
?	?	?	?
?	?	?	?

(a) Uncertain relation

mid	name	country	rating
?	?	USA	≥8.5
?	?	?	< 8.5
?	?	?	< 8.5
?	?	?	< 8.5

(b) Information of Q_σ

Fig. 3. Uncertain relation and the information of query Q_σ

not consider the possible arbitrage due to *public* functional dependency, e.g., the "'ZipCode" determines the "State".

Therefore, for the first time, we incorporate the *positive price* and *dependency-preserving* properties into consideration. Existing query-based pricing works have not formally incorporated these critical properties, significantly undermining their practicality. For instance, the zero price of the valuable query (answer) directly decreases the revenue of data sellers, while meaningful data can be derived freely given functional dependency [14]. While positive price is easy to understand, the dependency-preserving property is stated in Definition 2.

Definition 2: **Dependency-preserving property**. Given any functional dependency $\alpha \longrightarrow \beta$ on the relation R_j and two queries Q_α and $Q_{\alpha \bigcup \beta}$ on R_j that have the same predicates and output attributes in α and $\alpha \bigcup \beta$ respectively, a pricing function is dependency-preserving iff $p(Q_\alpha) = p(Q_{\alpha \bigcup \beta})$.

In addition, the query price should be computed as quickly as possible, in order to meet the customer's pricing request in time. Considering all these pricing characteristics together, the studied problem in this paper is formulated in Definition 3.

Definition 3: **Problem statement.** Suppose there is an on-sale dataset D with relations $(R_1, R_2, \cdots, R_m)$. The studied problem is to design a pricing function $p(\cdot)$ for the query Q on D that achieves four pricing characteristics, including

- *Efficiency*. The query price $p(Q)$ should be computed with the linear time complexity.
- *Arbitrage-freeness*. For a set of queries $Q_1, \cdots, Q_k$ determining Q, there is $p(Q) \leq \sum_{i=1}^{k} p(Q_i)$.
- *Positive price*. The query Q that exactly outputs the values of some tuples should have $p(Q) > 0$.
- *Dependency-preserving*. $p(Q_\alpha) = p(Q_{\alpha \bigcup \beta})$ holds given any functional dependency $\alpha \longrightarrow \beta$ and two queries Q_α and $Q_{\alpha \bigcup \beta}$ in Definition 2.

Existing query pricing approaches [7]–[16] have exponential computation complexity (w.r.t. usage based pricing) or #P-hardness (w.r.t. information based pricing). They cannot support efficient price calculation for queries over large-scale databases, e.g., projection and aggregation. Also, the database-level information-based pricing does not ensure the positivity and dependency-preserving property, as shown later.

It faces a series of challenges to address the problem. How to model the tuple information of various queries and compute it efficiently? How to design a query pricing framework with desirable pricing properties based on the tuple information? What if the tuple information of some queries is unavailable?

In the following, we illustrate how to model and derive the tuple information in the linear time in Section III. Based on tuple-level information modeling, we propose the scalable query pricing framework ARIA with desirable properties in Section IV. In these two sections, we focus on selection queries for better understanding the main idea. The tuple information and prices of other queries (i.e., projection and count, max/min aggregation) under ARIA are detailed in Section V. For join queries whose tuple information is unavailable, ARIA leverages query determinacy for pricing in Section VI.

III. Modeling Tuple-level Information

In this section, we model the tuple information to illustrate how the query answer tells about outputted and non-outputted tuples, which is the basis for pricing queries in *linear* time. We focus on the tuple information of selection queries in this section for clear clarification, while the information of other query types is presented in Section V. For better understanding, we first formulate the tuple information of any selection query where the database has one tuple and then generalize the formulation to the database with multiple tuples.

Specifically, when there is only one tuple t in the database D, any selection query Q asks the value of the tuple t (on some attributes) if t satisfies query predicates. Given the query answer $Q(D)$ (i.e., $Q(t)$), it tells the useful information about t, no matter whether $Q(t)$ is empty or not. In particular, when $Q(t)$ is non-empty, it tells that t not only takes the value of $Q(t)$ but also satisfies the query predicates. In contrast, the empty $Q(t)$ means that t breaks query predicates. These two aspects form the tuple information that the query answer tells.

Actually, the above information defines the *value range* of the tuple t under the query answer $Q(t)$. Formally, let $\mathcal{S}$ be the set containing all possible values of the tuple t that satisfy the domain constraints of attributes in t's relation, i.e., the value domain of t. The value range of t under $Q(t)$ can be formulated as $E_t(Q) = \{v | v \in \mathcal{S} \text{ and } Q(v) = Q(t)\}$. The tuple t can never take the values out of $E_t(Q)$, since these values cannot produce the same query answer $Q(t)$. Note that, $Q(t) = \varnothing$ when t breaks query predicates. It thus ensures the *completeness* of the formulation of query information. What's more, the tuple information $E_t(Q)$ can be easily obtained with linear computational complexity by traversing the set $\mathcal{S}$.

Example 1: For a relation Gross(country, grosses) simplified from Fig. 2(b) with one tuple t = (USA, 8×10^7), its value domain $\mathcal{S}$ contains all tuples with valid country names and grosses greater than 0. For a selection query Q : `select country from Gross where gross` $> 10^7$ and $Q(t) = \{\text{USA}\}$, its value range $E_t(Q)$ includes all tuples that are from USA with grosses exceeding 10^7.

The above information formulation should be *generalized* when the database consists of numerous tuples. Previous information-based studies regard the database as a *whole* (like

a tuple) and model the database-level information as the value range of the database, i.e., $E_D(Q) = \{v|v \in \mathcal{S}_D \text{ and } Q(v) = Q(D)\}$ [14]–[16]. When the set $\mathcal{S}_D$ contains M possible databases, the database-level information computation requires $O(M \cdot |D|)$ complexity, which is time-consuming on large-scale datasets. Different from them, we consider each tuple *separately* to formulate the tuple-level information for pricing. The previous formulation can be directly applied to any selection query since all tuples are checked separately and each query result comes from *one tuple*. Definition 4 formulates the information of the selection query.

Definition 4: **Tuple information of the selection query**. For a selection query Q on the relation R_j, the query answer $Q(D)$ indicates that the information of each tuple $t \in R_j$ under Q is the value range $E_t(Q) = \{v|v \in \mathcal{S}_j \text{ and } Q(v) = Q(t)\}$, where $\mathcal{S}_j$ is the set containing all possible values of data in R_j and $Q(v)$ (resp. $Q(t)$) is the answer of Q on a relation having only a tuple v (resp. t).

Example 2: Considering the Movie relation in Fig. 2(a), the data buyer Bob is uncertain about the four tuples in the beginning, as shown in Fig. 3(a). Then, he purchases a query Q_σ : `select country from Movie where rating` $\geq$ `8.5` and the answer $Q_\sigma(D)$ is {`USA`}. Based on $Q_\sigma(D)$, one can know that the Movie relation contains one tuple (country = USA, rating ≥ 8.5) and three tuples whose rating is less than 8.5, as demonstrated in Fig. 3(b).

It is noteworthy that, within the framework of tuple-level information modeling, the information of *all* tuples for any selection query can be acquired in *linear* time. More precisely, given that all tuples within the same relation R_j share a *common* possible value set $\mathcal{S}_j$, their information can be obtained by traversing and grouping the query results $Q(\mathcal{S}_j)$. Consequently, the computational complexity of deriving tuple-level information for any selection query is $O(|\mathcal{S}_j|)$, providing opportunities to boost the efficiency of query pricing.

On top of the computational efficiency under tuple-level information modeling, the next goal is to build the connection between tuple information and query pricing. We propose the arbitrage-free framework ARIA to price SPJA queries in Section IV, taking selection queries as an example. We describe how ARIA handles projection and aggregation in Section V, while pricing join queries is detailed in Section VI.

Note that when the database consists of one tuple, our tuple-level information equals the database-level information in [14]–[16]. Database-level information for various queries exhibits a uniform structure in theory. However, it also makes the price calculation computationally expensive due to the coarse-grained information modeling. Conversely, the fine-grained tuple-level information varies among different queries. In this manner, the possible value sets of tuples can be organized as a *common* one. All tuples' information can be computed *simultaneously*, thereby significantly enhancing efficiency. The high computational efficiency of tuple-level information serves as the motivation of our paper.

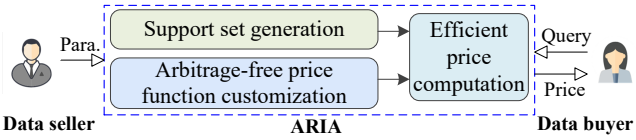


Fig. 4. The ARIA framework

IV. Pricing Framework ARIA

In this section, we present ARIA, an Arbitrage-fRee pricing framework based on tuple-level Information gAin, taking selection queries as an example. ARIA also supports pricing projections, simple aggregations, and joins, as detailed later.

A. Framework Overview

Given the tuple information of any query, the main challenge of pricing lies in the design of arbitrage-free price functions and efficient price computation algorithms. Fortunately, the special property of tuple information provides the basis to present arbitrage-free price functions. Further, to make the price computable, we leverage a *discrete support set* S_j on each relation R_j to model the continuous and infinite value domain $\mathcal{S}_j$, i.e., the possible values of tuples in R_j. Finally, the arbitrage-free price of any select-project-join and simple aggregation (SPJA) query is obtained in linear time.

Fig. 4 overviews the ARIA framework that prices the SPJA queries based on the tuple-level information gain. It consists of three parts, i.e., (i) the price function customization that allows sellers to choose from various price functions, (ii) the support set generation that generates a discrete possible value set S_j for each relation R_j, and (iii) the price computation to price each query from buyers efficiently.

Note that, the price computation varies across query types due to different forms of tuple information. For better clarification, we focus on selection queries to elaborate ARIA in this section. We leave the extension to projection, join, and simple aggregation queries in Sections V and VI.

B. Arbitrage-free Price Function Design

Recall that, our main target is to overcome the bottleneck of price computation efficiency based on tuple-level information while ensuring *no arbitrage*. To realize this, we first describe in Section III that, the information of each tuple t under the selection query Q is measured by the value range $E_t(Q) = \{v|v \in \mathcal{S} \text{ and } Q(v) = Q(t)\}$. So the next step lies in: *How to design arbitrage-free price functions based on fine-grained tuple information?* Fortunately, the containing relationship between information (i.e., Lemma 1) builds the connection between the tuple information (i.e., Definition 4) and query determinacy relationship [24]. It thus provides the basis for designing arbitrage-free price functions.

Lemma 1: If the query bundle $\mathcal{Q} = \{Q_1, \cdots, Q_k\}$ determines the query Q and each query $Q_i \in \mathcal{Q}/Q$ is any selection query on the single relation R_j, there is $\bigcap_{i=1}^{k} E_t(Q_i) \subseteq E_t(Q)$ for each tuple $t \in D$.

Proof. Let D_t^v replace the tuple t in D with the value v. Then, for each tuple $t \in R_j$ and each value $v \in E_t(Q)$, there

is $Q(D_t^v) = Q(D)$. Also, for any $t \in R_j$ and $v \notin E_t(Q)$, $Q(D_t^v) \neq Q(D)$ holds. The two statements are obvious based on the definition of $E_t(Q)$. Suppose there is $\bigcap_{i=1}^k E_t(Q_i) \nsubseteq E_t(Q)$ for a tuple $t \in R_j$, and there must exist one value $v' \in \bigcap_{i=1}^k E_t(Q_i)$ and $v' \notin E_t(Q)$. For such v', there is a database D' replacing the tuple t in D with v, satisfying $Q_1(D') = Q_1(D), \cdots, Q_k(D') = Q_k(D)$. Moreover, we have $Q(D') \neq Q(D)$ as $v' \notin E_t(Q)$. It contradicts the fact of query determinacy [24], i.e., $Q(D) = Q(D')$ holds when $Q_1(D) = Q_1(D'), \cdots, Q_k(D) = Q_k(D')$ for any database D'. Therefore, v' does not exist. The proof is complete. □

Lemma 1 tells that, the value range of each tuple under $\mathcal{Q}$ is contained by that under Q if the query bundle $\mathcal{Q}$ determines Q. Further, define $C_t(Q) = \mathcal{S}_j - E_t(Q)$, i.e., *impossible value set* of tuple t under Q, such that $C_t(Q) \subseteq \bigcup_{i=1}^k C_t(Q_i)$. It means that, once $\mathcal{Q}$ determines the query Q, more impossible values under $\mathcal{Q}$ will be removed than those under Q.

To ensure arbitrage-free pricing in Definition 1, the price of the tuple t under Q should be no higher than the sum-up price under $\mathcal{Q}$, i.e., $p(t;Q) \leq \sum_{i=1}^k p(t;Q_i)$ where $Q_i \in \mathcal{Q}$. Inspired by the pioneering work [14], the price function meets the arbitrage-free requirement once the price of each tuple t to be *monotone and subadditive* on the set $C_t(Q)$. In light of this, ARIA measures the *information gain* of each tuple t as $|\mathcal{S}_j - E_t(Q)|$, which satisfies the monotone and subadditive properties. Then, the arbitrage-free price function for selection queries is presented based on tuple-level information gain.

Definition 5: **Tuple-level information gain-based price function**. For a selection query Q on relation R_j, the query price is $p_b(Q) = \sum_{t \in R_j} (|\mathcal{S}_j| - |E_t(Q)|)$, where $\mathcal{S}_j$ contains all possible values of data in R_j and $E_t(Q)$ is the information of the tuple t under Q.

Theorem 1 formally proves that $p_b(Q)$ is arbitrage-free. Meanwhile, data sellers can also specify/construct other price functions $f(p_b(Q))$. $f(\cdot)$ can be any function that is increasing and subadditive on $\mathbb{N}$ with $f(0) = 0$. For example, $f(x) = a \cdot x + b$, $f(x) = \sqrt{x}$, and $f(x) = \log(x+1)$ are satisfying ones. The arbitrage-freeness of these price functions can be proved similarly. Besides, data sellers can customize the price of each relation by scaling the price function.

Theorem 1: The price function $p_b(Q)$ is arbitrage-free on pricing selection queries, i.e., $p_b(Q) \leq \sum_{i=1}^k p_b(Q_i)$ holds for any the selection query bundle $\mathcal{Q} = \{Q_1, \cdots, Q_k\}$ determining the selection query Q.

Proof. Suppose all queries in $\mathcal{Q}$ and Q are on the relation R_j. First, the price of each tuple t is $|C_t(Q)|$ where $C_t(Q) = \mathcal{S}_j - E_t(Q)$. Based on Lemma 1, $|C_t(Q)| \leq |\bigcup_{i=1}^k C_t(Q_i)| \leq \sum_{i=1}^k |C_t(Q_i)|$ holds for each tuple t. Finally, there is $p_b(Q) = \sum_{t \in R_j} |C_t(Q)| \leq \sum_{t \in R_j} \sum_{i=1}^k |C_t(Q_i)| = \sum_{i=1}^k p_b(Q_i)$. The proof is complete. □

Please note that, ARIA does not employ the reduced entropy $H(\cdot)$ to measure the information gain, since it is not subadditive on $C_t(Q) = \mathcal{S}_j - E_t(Q)$. In the information theory [25], the entropy of the tuple t is $H(O) = \log|O|$ if O contains all possible values of t and each value has the same probability $1/|O|$. Then, the entropy of t reduced by the query Q is $H(\mathcal{S}_j) - H(E_t(Q)) = \log|\mathcal{S}_j| - \log|E_t(Q)|$, which is not subadditive on $C_t(Q)$ and may bring arbitrage.

C. Support Set Generation

The core operation of pricing queries in ARIA is to derive the value of $|E_t(Q)|$. However, it is unreachable to compute it on the continuous domain $\mathcal{S}_j$ with infinite values. Thus, ARIA considers a discrete set (i.e., support set) S_j to model $\mathcal{S}_j$ and the real price is derived from S_j, preserving the arbitrage-freeness and high efficiency. S_j is generated as follows.

First, ARIA initializes S_j as the *distinct* tuples in R_j to reduce the generation cost, since these tuples must obey the database constraints. Then, when the data seller specifies the positive extra generated size W_j on each relation R_j, ARIA randomly generates W_j tuples following the database constraints. It can be implemented by storing each support set as a relation in the underlying database, where the constraints can be *automatically* ensured by the database management system (DBMS). In this way, the support set size $|S_j|$ is the summation of W_j and the number of distinct tuples in R_j.

Overall, ARIA leverages both the original data and generated data adhering to database constraints to form support sets. Not only reduces the generation cost by including original data, but it also accelerates the price computation by using query results $Q(R_j)$ to construct $Q(S_j)$. The considered database constraints include primary/foreign key, column constraints, and functional dependencies, which are critical to ensure the dependency-preserving property in Lemma 2. In practice, we recommend using distinct tuples in R_j to form S_j, since generated tuples may break some implicit constraints.

D. Price Computation for Selection Queries

Based on the generated support set S_j and the arbitrage-free price function in Definition 5, the price of any selection query Q on R_j could be derived as

$$p_b(Q) = \sum_{v_i \in Q(D)} (|S_j| - n_i) + (N_j - |Q(D)|) \cdot |Q(S_j)| \quad (1)$$

n_i is the frequency of the element v_i in $Q(S_j)$, N_j is the size of the relation R_j, and $|S_j|$ is the size of the support set S_j. In particular, $(|S_j| - n_i)$ is the price of the tuple t_i satisfying query predicates. For $(N_j - |Q(D)|)$ tuples that break query predicates, they have the same price $(|S_j| - |E_\varnothing|)$ where $E_\varnothing = \{v | v \in S_j \text{ and } Q(v) = \varnothing\}$ and $|E_\varnothing| = (|S_j| - |Q(S_j)|)$. The query price in Eq. 1 is the total price of all tuples.

Algorithm 1 depicts the pseudo-code of ARIA-Base to compute the price of the selection query Q on the relation R_j. It takes the query answers $Q(D)$ and $Q(S_j)$, the size N_j of the relation R_j, the support set size $|S_j|$ as inputs, and outputs the query price p. It first initializes an empty dictionary T to record the frequency of each element in $Q(S_j)$ (line 1). Then, it traverses $Q(S_j)$ to update T (lines 2-3). Finally, ARIA-Base computes and returns the query price p (lines 4-5).

The time complexity of ARIA-Base is $O(|S_j|)$ due to the traversal on $Q(S_j)$. Theorem 2 ensures that ARIA-Base derives arbitrage-free prices of selection queries.

Algorithm 1: The ARIA-Base Algorithm

Input: the query answers $Q(D)$ and $Q(S_j)$; the relation size N_j; the support set size $|S_j|$;
Output: the query price p

initialize an empty dictionary T with default value 0
foreach $v \in Q(S_j)$ **do**
  $T[v] \longleftarrow T[v] + 1$
$p \longleftarrow \sum_{v \in Q(D)}(|S_j| - T[v]) + (N_j - |Q(D)|) \cdot |Q(S_j)|$
return p

Theorem 2: ARIA-Base derives the *arbitrage-free* prices.

Proof. First, $C_t(Q) \subseteq \bigcup_{i=1}^{k} C_t(Q_i)$ holds for each tuple t if queries $Q_1, \cdots, Q_k$ determines the query Q, where $C_t(Q) = \mathcal{S}_j - E_t(Q)$. Then, $(C_t(Q) \bigcap S_j) \subseteq \bigcup_{i=1}^{k}(C_t(Q_i) \bigcap S_j)$ holds for any set S_j. The price of each tuple t in ARIA-Base is $|C_t(Q) \bigcap S_j|$ with $|C_t(Q) \bigcap S_j| \leq \sum_{i=1}^{k} |C_t(Q_i) \bigcap S_j|$. Overall, $p_b(Q; S_j) = \sum_{t \in R_j} |C_t(Q) \bigcap S_j| \leq \sum_{t \in R_j} \sum_{i=1}^{k} |C_t(Q_i) \bigcap S_j| = \sum_{i=1}^{k} p_b(Q_i; S_j)$. The proof is complete. □

Example 3: Consider the query Q_σ : `select country from Movie where rating` $\geq$ `8.5` with $Q_\sigma(D) = \{\text{USA}\}$. Suppose the support set S_1 on Movie relation is the relation itself, there are $|S_1| = N_1 = 4$ and $Q_\sigma(S_1) = Q_\sigma(D)$. ARIA-Base traverses $Q_\sigma(S_1)$ and gets the frequency n_i of each element $v_i \in Q_\sigma(S_1)$, i.e., $n_1 = 1$ for the element USA. Then, the price of Q_σ is $(|S_1| - n_1) + 3 \cdot |Q_\sigma(S_1)| = 6$.

E. Pricing Characteristics and Scope of ARIA

Efficiency. ARIA considers the query information at the tuple level to boost efficiency. It achieves *linear* time complexity for selection queries, which is applicable for projection, join, and simple aggregation queries as shown in Sections V and VI. ARIA is the *fastest* among all pricing frameworks theoretically, and also experimentally (verified in Section VII).

Arbitrage-freeness. ARIA guarantees *no-arbitrage* across all SPJ queries as well as a set of aggregation queries. Specifically, Theorem 2 holds on queries whose information can be theoretically derived, as illustrated in Section V. Meanwhile, ARIA leverages query determinacy to price join queries without arbitrage in Section VI.

Dependency-preserving. In ARIA, the database constraints are explicitly considered in support set generation and thus all tuples in the support set obey the functional dependencies. In light of this, ARIA can capture the *information consistency* due to the functional dependencies, as described in Lemma 2.

Lemma 2: Given the functional dependency $\alpha \longrightarrow \beta$ on the relation R_j and two selection queries Q_α and $Q_{\alpha \bigcup \beta}$ that have the same predicates on R_j and output attributes in α and $\alpha \bigcup \beta$ respectively, the information of each tuple under the two queries are same, i.e., $E_t(Q_\alpha) = E_t(Q_{\alpha \bigcup \beta})$ for $t \in R_j$. Furthermore, these two queries have the same price.

Proof. As stated in Definition 4, the tuple information under Q_α is $E_t(Q_\alpha) = \{v | v \in S_j \text{ and } Q_\alpha(v) = Q_\alpha(t)\}$, while that under $Q_{\alpha \bigcup \beta}$ is $E_t(Q_{\alpha \bigcup \beta}) = \{v | v \in S_j \text{ and } Q_{\alpha \bigcup \beta}(v) = Q_{\alpha \bigcup \beta}(t)\}$. As $Q_{\alpha \bigcup \beta}$ obviously determines Q_α, there is $E_t(Q_{\alpha \bigcup \beta}) \subseteq E_t(Q_\alpha)$ for any $t \in R_j$ based on Lemma 1.

Moreover, each value v in $E_t(Q_\alpha)$ must be in $E_t(Q_{\alpha \bigcup \beta})$, i.e., $E_t(Q_\alpha) \subseteq E_t(Q_{\alpha \bigcup \beta})$. Otherwise, there exists at least one tuple t taking the value v on attributes in α but a different value v' on attributes in $\alpha \bigcup \beta$. It contradicts the fact that each value $v \in S_j$ obeys the dependency $\alpha \longrightarrow \beta$. Finally, each tuple has the same information under two queries, resulting in the same prices of two queries. The proof is complete. □

Positive price. Besides, ARIA ensures positive prices for selective queries with the non-empty answer based on Lemma 3.

Lemma 3: For any selection query Q with the non-empty answer (i.e., some tuples satisfy the predicates of Q and some do not), there is $p_b(Q) > 0$.

Proof. Suppose the query Q is on the relation R_j. Let t be the tuples t satisfying the predicates of Q and t' be the tuple breaking the predicates of Q. Based on Definition 4, the value range $E_t(Q)$ of t must not contain t' and thus $|S_j| - |E_t(Q)| \geq 1$. Hence, the price of the tuple t under Q is positive and the query price is positive. The proof is complete. □

In comparison, QIRANA considers no database constraints when generating possible databases, which is not dependency-preserving. Meanwhile, it may assign zero prices to numerous queries, as shown in their and our experiments. This is because, QIRANA employs M possible databases for pricing, which have one or two tuples differing from the real database D. As M is much smaller than the database size $|D|$, numerous tuples take the same value as those in D, yielding zero prices of queries outputting these tuples.

Discussion. In this work, we illustrate how ARIA leverages tuple information to price a set of SPJA queries on data with functional dependencies. The query predicates can be any pure function $F(t)$ on the tuple t allowing various operations, while the inner and outer joins are feasible in Section VI. Further, ARIA can be extended to handle complex queries whose information can be fully captured at the *tuple* level, e.g., `limit` query and the median number query. For queries with unavailable tuple information (e.g., some nested queries), data buyers can acquire the same results by *combining* multiple queries supported in ARIA. Overall, ARIA is practical to cover a *broad* spectrum of data retrieval and analysis needs in real-world applications [26], [27], while achieving four desirable pricing characteristics for the first time.

The reliance on tuple information limits ARIA in handling data with tuple-generating dependencies [28], [29], where some tuples' values or existence can be inferred from others. The join dependency (JD) holds on a relation R that equals the lossless natural join of its projections on $R_1, R_2, \ldots, R_n$, i.e., $R = \pi_{R_1}(R) \bowtie \pi_{R_2}(R) \bowtie \cdots \bowtie \pi_{R_n}(R)$. The inclusive dependency (IND) on relations R_1 and R_2 means that values appearing in $R_1[X_1, \cdots, X_l]$ must appear in $R_2[X_1, \cdots, X_l]$. In most cases, these dependencies can be eliminated via database normalization [29], [30], where data is normalized into multiple relations with different relationships. However, there is still a potential issue under IND after normalization. When the attribute $R_1.A$ is the foreign key referring to $R_2.A$,

mid	name	country	rating
?	?	USA	≥8.5
?	?	(USA, ≥8.5), (?, < 8.5)	
?	?	(USA, ≥8.5), (?, < 8.5)	
?	?	(USA, ≥8.5), (?, < 8.5)	

(a) Projection query Q_π

mid	name	country	rating
?	?	?	≥8.5
?	?	?	< 8.5
?	?	?	< 8.5
?	?	?	< 8.5

(b) Count query Q_c

mid	name	country	rating
?	?	?	8.8
?	?	?	≤ 8.8
?	?	?	≤ 8.8
?	?	?	≤ 8.8

(c) Max query Q_m

Fig. 5. Information of different projection and aggregation queries

queries on $R_1.A$ with *non-empty* answer tell that these values are also in $R_2.A$. ARIA and previous works [7]–[11], [14]–[16] are unable to capture such information, while the foreign keys (that are used as identifiers) always tell little about the actual values of tuples. This issue remains for future studies.

V. ARIA: Pricing Projection & Aggregation

This section illustrates how ARIA efficiently prices the projection and simple aggregation queries (i.e., count and max/min queries). For these queries, each tuple can be considered separately, and the tuple information can be derived from the connotation of query answers. Hence, ARIA can leverage the previous idea to price these queries in *linear* time with desirable properties. We analyze the tuple information for these queries below and present price computation algorithms.

Different from the selection queries, the projection query tells the *distinct* values of qualified tuples. Thus, for each item $v_i \in Q(D)$, the value range of the corresponding tuple t is $E_t(Q) = \{v | v \in \mathcal{S}_j \text{ and } Q'(v) = v_i\}$, where Q' removes the distinct keyword from Q. For other tuples, they either satisfy the query predicates and take one value $v_i \in Q(D)$, or break the query conditions. Therefore, these tuples have the same value range $E_t(Q) = \{v | v \in \mathcal{S}_j \text{ and } (Q'(v) \in Q(D) \text{ or } Q'(v) = \varnothing)\}$. In light of this, the price of the projection query Q on relation R_j is

$$p_b(Q) = \sum_{v_i \in Q(D)} (|S_j| - n_i) + (N_j - |Q(D)|) \cdot \sum_{\substack{v_i \notin Q(D) \text{ and} \\ v_i \in Q'(S_j)}} n_i \quad (2)$$

where Q' removes the distinct keyword from Q, n_i is the frequency of the element v_i in $Q'(S_j)$, N_j is the size of the relation R_j, and $|S_j|$ is the size of the support set S_j. Compared with Eq. 1, the $(N_j - |Q(D)|)$ tuples cannot take the values out of $Q(D)$ and the $\varnothing$, which have the lower price.

Moreover, ARIA-Base can be modified to price projection queries. The only difference is to input the $Q(D)$ and $Q'(S_j)$ to ARIA-Base and compute the query price based on Eq. 2, where Q' removes the distinct keyword from Q. Then, the prices of projection queries can be computed in linear time.

Example 4: Suppose the support set S_1 on Movie relation is the relation itself, there are $|S_1| = N_1 = 4$. Fig. 5(a) demonstrates the revealed information of the projection query Q_π : `select distinct country from Movie where rating ≥ 8.5`. Specifically, $Q_\pi(D) = \{\text{USA}\}$ indicates the following information. First, there is one tuple taking the values of (country = USA, rating $\geq$ 8.5). For the rest of the three tuples, they can take the values of (country = USA, rating $\geq$ 8.5) or (country = ?, rating $<$ 8.5). Moreover, as $Q'_\pi(S_j) = \{\text{USA}\}$, the query price is $p_b(Q_\pi) = (4-1) + (4-0) + (4-0) + (4-0) = 3$.

Count query equals the one Q' : `select 1 from` R_j `where predicates`. Also, the group-by-count query equals the query Q' that outputs the values of attributes in the group-by clause. Therefore, the price of the count query can be obtained as $p_b(Q')$ in Eq. 1.

Example 5: Fig. 5(b) shows the tuple information of the count query Q_c : `select count(*) from Movie where rating ≥ 8.5` with $Q_c(D) = \{1\}$. It indicates that, one tuple takes the values of (rating $\geq$ 8.5), while other three tuple satisfies (rating $<$ 8.5). With $Q'_c(S_j) = \{1\}$, the price is $p_b(Q_c) = (4-1) + (4-3) + (4-3) + (4-3) = 6$.

For any **group-by-max/min** query Q over the single relation R_j, the query answer $Q(D) = \{(g_1, v_1), \cdots, (g_k, v_k)\}$ indicates the information of all tuples as follows. Here, v_i corresponds to the max/min value of the group g_i. Let Q' be the query removing the `max/min` keyword and `group-by` clause from Q. First, each item $(g_i, v_i) \in Q(D)$ corresponds a tuple t with the value domain $E_i = \{v | v \in S_j \text{ and } Q'(v) = (g_i, v_i)\}$. Meanwhile, the rest of $(N_j - k)$ tuples can never take the values higher/lower than the maximum/minimum of each group. For instance, for the group-by-max query, they have the same value range $E_0 = O_1 \cup \cdots \cup O_k \cup O_0$, where $O_i = \{v | v \in S_j \text{ and } Q'(v) = (g_i, v_c) \text{ and } v_c \leq v_i\}$ and $O_0 = \{v | v \in S_j \text{ and } Q'(v) = \varnothing\}$. Thus, ARIA can traverse $Q'(S_j)$ once to obtain the sizes of the above sets. Let n_i be the size of the set E_i and o_i be the size of the set O_i, the tuple-level information gain-based price of Q is

$$p_b(Q) = \sum_{i=1}^{k} (|S_j| - n_i) + (N_j - k) \cdot (|S_j| - \sum_{i=0}^{k} o_i) \quad (3)$$

Algorithm 2 shows the pseudo-code of ARIA-MM algorithm, which derives the prices of group-by-max/min queries on the relation R_j. It takes the query answers $Q(D) = \{(g_1, v_1), (g_2, v_2) \cdots\}$ and $Q'(S_j) = \{(g_1, v_1), (g_2, v_2) \cdots\}$, and the relation size N_j of R_j, the support set size $|S_j|$ as inputs, and outputs the query price p. Q' is the selection query that outputs the values of attributes (i.e., all selected attributes in Q) and has the same predicates as Q. ARIA-MM first initializes the query result size k and the size of O_0, i.e., o_0, while n_i and o_i are set as zero for each group in $Q(D)$ (lines 1-2). n_i records the number of elements in $Q'(S_j)$ that equals to the extreme value of the i-th group in $Q(D)$. o_i is the number of elements in $Q'(S_j)$ that does not exceed the extreme value of the i-th group. Then, ARIA-MM traverses $Q'(S_j)$ to update the values of n_i and o_i (lines 3-8). Finally, the query price is derived and returned (lines 9-10).

The time complexity of Algorithm 2 is $O(|S_j|)$, as there is a traversal on $Q'(S_j)$ in ARIA-MM. Meanwhile, finding the group of the element (in line 4) can be achieved with $O(1)$ complexity when using the dictionary to store $Q(D)$.

Example 6: Suppose the data buyer wants to purchase the query Q_m : `select max(rating) from Movie` with $Q_m(D) = \{8.8\}$. Fig. 5(c) depicts the information of each tuple under Q_m. To compute the price, one first executes the query Q'_m : `select rating from Movie` and obtains $Q'_m(S_1) = \{8.8, 8.4, 8.4, 7.9\}$. Then, ARIA-MM traverses

Algorithm 2: The ARIA-MM Algorithm

Input: the query answers $Q(D)$ and $Q'(S_j)$; the relation size N_j; the support set size $|S_j|$
Output: the query price p

$k \longleftarrow |Q(D)|, o_0 \longleftarrow (|S_j| - |Q'(S_j)|)$
initialize n_i, o_i as zero for $i = 1, \cdots, k$
foreach $(g, v) \in Q'(S_j)$ **do**
  if g corresponds to one group g_i in $Q(D)$ **then**
    if $v = v_i$ **then** // v_i is the extreme value of g_i
      $n_i \longleftarrow n_i + 1$
    if $v \leq v_i$ **then** // $v \geq v_i$ for the min query Q
      $o_i \longleftarrow o_i + 1$
compute the query price p based on Eq. 3
return p

Algorithm 3: The ARIA-SPJ Algorithm

Input: the query Q on relations R_Q; the sizes of all relations $N_1, \cdots, N_m$; the support sets $S_1, \cdots, S_m$
Output: the query price p

$Q' \longleftarrow$ remove the distinct keyword and add the ID column of each relation $R_j \in R_Q$ into Q
obtain $Q'(D)$ and initialize the price p as zero
foreach $R_j \in R_Q$ **do**
  $Q_j(D) \leftarrow$ extract the results of $Q'(D)$ on relation R_j, remove repetitive results, and remove the ID column
  $Q_j(S_j) \leftarrow$ extract the results of $Q'(D - R_j + S_j)$ on S_j, remove repetitive results and the ID column
  if Q is not a projection query **then**
    $p_j \longleftarrow$ ARIA-Base$(Q_j(D), Q_j(S_j), N_j, |S_j|)$
  else
    $Q_j(D) \leftarrow$ remove repetitive results from $Q_j(D)$
    use Eq. 2 to compute p_j
  $p \longleftarrow p + p_j$
return p

$Q'_m(S_1)$ and obtains $n_1 = 1$ and $o_1 = 4$. Finally, the query price is derived as $(|S_1| - n_1) + (N_1 - |Q_m(D)|) \cdot (|Q'_m(S_1)| - o_1) = (4-1) + (4-1) \cdot (4-4) = 3$.

Besides, the proposed algorithms achieve the desirable pricing properties in Section II, since previous theoretical findings (e.g., Theorem 1 and Lemma 1) are applicable to these queries.

VI. ARIA: Pricing Join Queries

In this section, we present how ARIA prices the join query that can be decomposed as multiple single-relation queries, by utilizing the query determinacy relationship.

Unlike single-relation queries, ARIA prices the join query as the total price of multiple queries that can determine it, rather than directly deriving tuple information. This is because each tuple in a join query can produce zero or multiple results, making it impossible to identify the information of each tuple separately. For example, the joined results from two relations of size 20 could be produced by various combinations of tuples, such as four tuples from one relation joining with five tuples from another, or one tuple joining with twenty tuples. Therefore, ARIA leverages the query determinacy relationship to price join queries without arbitrage, as detailed below.

The join queries Q supported in ARIA have the following features. First, Q has no cross-relation predicate except for the join predicates. Second, the joined attributes must be outputted as query results. The join type can be inner, full outer, left outer, and right outer joins. Such join queries can be decomposed into multiple single-relation queries as follows.

In particular, the inner join predicate $R_i.A = R_j.A$ can be transformed into two *single-relation predicates*, i.e., $R_i.A \in H_j$ and $R_j.A \in H_i$. Here, the set H_j (resp. H_i) contains the values of tuples in the relation R_j (resp. R_i) on the attribute A that meet the non-join predicates of Q on R_j (resp. R_i). The outer join predicate can be transformed into zero or one such predicate. For instance, there is no need to transform the join predicate for the left outer join on R_i. Then, one can construct the query Q_j for each relation R_j involved in Q, i.e., Q_j : `select expressions from` R_j `where predicates`. Each predicate in Q_j is either the original single-relation predicate of Q on R_j or the predicate transformed from the join predicate. Actually, Q_j returns the raw data of the join results $Q(D)$ on R_j.

For example, the join query $Q_{\bowtie}$: `select * from Movie m, Gross g where rating` $\geq$ `8.5 and m.mid = g.mid` can be decomposed as Q_1: `select * from Movie where rating` $\geq$ `8.5 and mid in` $\{$`1,2,3`$\}$ and Q_2: `select * from Gross where mid in` $\{$`1`$\}$. The original join predicate in $Q_{\bowtie}$ is transformed as two predicates, i.e., "mid in {1, 2, 3}" in Q_1 and "mid in $\{1\}$" in Q_2. Specifically, as the mids of tuples in Gross are $\{1, 2, 3\}$, the join predicate corresponds to "mid in {1, 2, 3}" in Q_1. The join predicate is regarded as "mid in $\{1\}$" in Q_2, where only the tuple with mid = 1 has rating $\geq$ 8.5.

Let $\mathcal{Q} = \{Q_1, \cdots, Q_k\}$ be the set containing all decomposed single-relation queries, ARIA prices the join query Q as $p_b(\mathcal{Q}) = \sum_{Q_j \in \mathcal{Q}} p_b(Q_j)$. In fact, the join query is priced via the query determinacy where $\mathcal{Q}$ determines Q.

It is worthwhile noting that, other decomposition strategies can find a set of single-relation queries that determine the join query. A natural decomposition is to consider each single-relation query with filter predicates and ignore all join predicates. Also, one can construct each single-relation one by one where the filter predicates rely on the previous results. However, the first one cannot effectively avoid arbitrage since unused data are included in the final join results, while the other has to evaluate $|R_Q|!$ possible decompositions to find the minimal arbitrage-free price. Here, R_Q contains all involved relations in the join query. In contrast, each decomposed query in ARIA only outputs the data used in the join operation. Formally, Theorem 3 confirms that it is impossible for buyers to purchase Q at a price lower than $p_b(\mathcal{Q}) = \sum_{Q_j \in \mathcal{Q}} p_b(Q_j)$, ensuring arbitrage-freeness and efficiency.

Theorem 3: Given the join query Q and the query bundle $\mathcal{Q} = \{Q_1, \cdots, Q_k\}$ containing all decomposed single-relation queries of Q, $\mathcal{Q}$ determines Q. Also, buyers cannot construct the cheaper query bundle $\mathcal{Q}' = \{Q'_1, \cdots, Q'_l\}$ to produce the query answer $Q(D)$.

Proof. First, each query $Q_j \in \mathcal{Q}$ outputs the raw data of the join results $Q(\cdot)$ over R_j. Then, the join results $Q(D)$ can

TABLE II
DATASET STATISTICS

Dataset	# Relations	# Tuples	# Attributes
Walmart	1	4636	12
Employment	1	2624	8
TPC-H	8	SF = 1	61
SSB	5	SF = 1	56

TABLE III
QUERY QUANTITY OF DIFFERENT TYPES

Dataset	S	SP	SA	SJ	SPJ	SAJ
Walmart	20	20	20	—	—	—
Employment	20	20	20	—	—	—
TPC-H	5	5	5	6	6	6
SSB	5	5	6	12	12	12

be derived by manually joining $Q_1(D), \cdots, Q_k(D)$. In other words, $\mathcal{Q}$ determines Q. Second, as each transformed predicate $R_j.A \in H_i$ is the *black box* for buyers, they cannot directly construct Q_j in reality. When buyers want to purchase a set of single-relation queries to produce $Q(D)$, they must purchase some additional queries, e.g., Q_i^1 : `select distinct` A `from` R_i `where predicates_on_Ri`, to obtain the set H_i. Therefore, they have at least to buy the queries Q_i^1 and Q_j for constructing Q_j, and there must hold $p_b(Q_i^1) + p_b(Q_j) \geq p_b(Q_j)$. As a result, buyers cannot construct the cheaper $\mathcal{Q}'$ to purchase Q. The proof is complete. □

Moreover, it is costly to obtain the specific form of each decomposed query Q_j in practice. On the one hand, one needs to issue extra queries to derive the set H_j for each involved relation R_j. On the other hand, the `IN` (i.e., $\in$) operator in the database necessitates a series of value comparisons, leading to significant computational overhead on large datasets. Recalling that, Q_j returns the raw data of the join results on R_j, while ARIA prices each single-relation query based on $Q_j(D)$ and $Q_j(S_j)$. Hence, we focus on deriving $Q_j(D)$ and $Q_j(S_j)$ rather than constructing Q_j. The core idea is that $Q_j(D)$ can be obtained by extracting the results $Q(D)$ on the relation R_j and removing the repetitive elements. In a similar way, $Q_j(S_j)$ can be obtained without the time-consuming `IN` operation.

Algorithm 3 depicts the procedure of ARIA-SPJ to compute the price of the join query. It takes the query Q on multiple relations R_Q, the sizes of all relations (i.e., $N_1, \cdots, N_m$), and all support sets (i.e., $S_1, \cdots, S_m$) as inputs. It outputs the query price p. It first constructs the query Q' based on Q, which removes the distinct keyword and adds the ID column of each relation $R_j \in R_Q$ into the selection clause of Q (line 1). Next, it executes $Q'(D)$ to derive $Q_j(D)$ and initializes the query price p as zero (line 2). Then, ARIA-SPJ starts to derive the price of each query Q_j (lines 3-10). It first derives $Q_j(D)$ by manually extracting the results of $Q'(D)$ on R_j, removing the repetitive results due to the join operation, and dropping the ID column. Similarly, ARIA-SPJ can obtain the query answer $Q_j(S_j)$ (line 5). Q' is executed on the relation S_j and other relations (except for R_j) in D, while the query answer is denoted as $Q'(D - R_j + S_j)$. Based on $Q_j(D)$ and $Q_j(S_j)$, ARIA-SPJ employs ARIA-Base to compute the price p_j of Q_j and accumulates p_j to the price p (lines 6-11). Note that, for the projection query Q, ARIA-SPJ needs to remove the repetitive elements in $Q_j(D)$ and compute the price with Eq. 2 (lines 9-10). When all involved relations are considered, ARIA-SPJ returns the query price p.

Example 7: Suppose the data buyer wants to purchase the query $Q_\bowtie$: `select m.mid, g.mid, name, grosses from Movie m, Gross g where rating` $\geq$ `8.5 and m.mid = g.mid`. The query answer is $Q_\bowtie(D) = \{$(1, 1, Inception, 62,785,337), (1, 1, Inception, 42,725,012)$\}$. To derive the query price, ARIA-SPJ constructs $Q'_\bowtie$ to output the ID column of each relation. Furthermore, based on $Q'_\bowtie(D)$, one can get the raw data of the join results $Q(D)$ on each relation, i.e., $Q_1(D) = \{$(1, Inception)$\}$ on Movie and $Q_2(D) = \{$(1, 62,785,337), (1, 42,725,012)$\}$ on Gross. As the support set on each relation is the relation itself, the query answer on the support set is the same as that on the relation. Thus, the price of Q_1 is $p_1 = (4-1)+(4-1)\cdot 1 = 6$, while that of Q_2 is $p_2 = (4-1)+(4-1)+(4-2)\cdot 2 = 10$. Finally, $Q_\bowtie$ is priced as $p_1 + p_2 = 16$.

ARIA-SPJ needs executing ARIA-Base for each involved $R_j \in R_Q$ with $O(\sum_{R_j \in R_Q} |S_j|)$ time complexity. It is feasible to price SAJ queries with the same complexity, where the select-aggregate-join (SAJ) query is decomposed into multiple select-aggregate (SA) queries and selection queries. The relation of the SA query depends on where the aggregated attribute is located. Theorem 3 also holds for the SAJ query.

VII. EXPERIMENTS

In this section, we evaluate the performance of our proposed ARIA. All algorithms were conducted on an Intel Core 2.90GHz Server with 512GB RAM and Ubuntu 20.04 system.

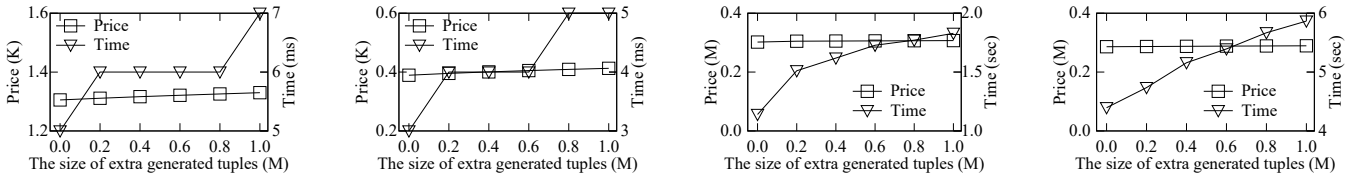
We use two real-world datasets *Walmart* [31], *Employment* [32], and two benchmarks *TPC-H* [17], *SSB* [33] for evaluation, as descried in Table II. Specifically, *Walmart* and *Employment* are from AWS data marketplace [5]. *Walmart* has a single table that encapsulates information about Walmart within the United States. *Employment* includes a table that delineates employment statistics across various industries in US. *TPC-H* and *SSB* are popular benchmarks for performance metrics on large-scale systems. The scale factor (SF) serves as a critical parameter in these benchmarks, quantifying the memory size of the database. *TPC-H* and *SSB* databases both occupy 1GB of memory, corresponding to an SF of 1.

Table III depicts the number of SQL queries in each type evaluated on four datasets. In particular, the 'S', 'P', 'J', and 'A' represent the selection, projection, join, and simple aggregation queries, respectively. The other query types are combinations of these four types. The *Walmart* and *Employment* datasets contain only one table and therefore cannot generate join queries. Besides, we also conduct experiments on specific queries to study the effect of query parameters.

We compare the proposed ARIA with the state-of-the-art query pricing methods, including provenance-based pricing (PBP) and QIRANA [10], [11], [15], [16]. PBP sets the query price as the minimal total price of *tuples* contributing to the query [10], [11]. QIRANA [15], [16] is an information-based

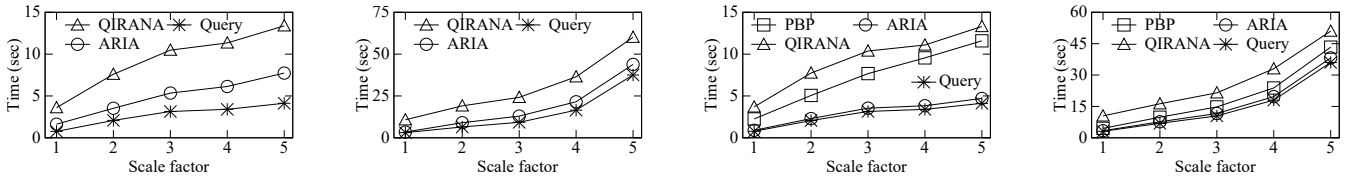
TABLE IV
EFFICIENCY COMPARISON ON PRICING DIFFERENT TYPES OF QUERIES (SECOND)

Method	*Walmart*			*Employment*			*TPC-H*			*SSB*		
	S	SP	SA	S	SP	SA	S + SJ	SP + SPJ	SA + SAJ	S + SJ	SP + SPJ	SA + SAJ
Query execution	0.003	0.002	0.003	0.002	0.002	0.002	0.808	0.801	0.798	3.098	3.007	3.080
PBP	0.007	0.071	—	0.003	0.027	—	2.165	—	—	4.532	—	—
QIRANA	0.010	0.010	0.008	0.007	0.007	0.006	3.691	3.789	3.634	13.022	12.453	12.854
Our ARIA	**0.005**	**0.004**	**0.006**	**0.003**	**0.003**	**0.004**	**0.917**	**0.876**	**1.626**	**4.332**	**4.061**	**4.765**



(a) *Walmart* (b) *Employment* (c) *TPC-H* (d) *SSB*

Fig. 6. Performance of ARIA vs. the size of extra generated tuples in support sets



(a) SPJA queries on *TPC-H* (b) SPJA queries on *SSB* (c) SJ queries on *TPC-H* (d) SJ queries on *SSB*

Fig. 7. Time cost of query pricing vs. the scale factor

method that prices the SPJ and aggregation queries based on database-level information. All methods are implemented in Python by ourselves and MySQL (with version 8.0) is used as the underlying DBMS. In ARIA, we employ the *distinct* tuples of each relation to form the support set on this relation, so that each support set size $|S_j|$ equals the relation size N_j. The settings on QIRANA follow the original paper, with 1K possible databases on *Walmart* and *Employment* and 1M possible databases over the rest of the datasets.

We use the CPU time to evaluate the efficiency, i.e., the total time of query execution and price computation. We employ the price ratio and free query rate to evaluate the pricing characteristics. The meaning of these metrics are detailed in Section VII-B. Each set of experiments is repeated 5 times and the average results are reported.

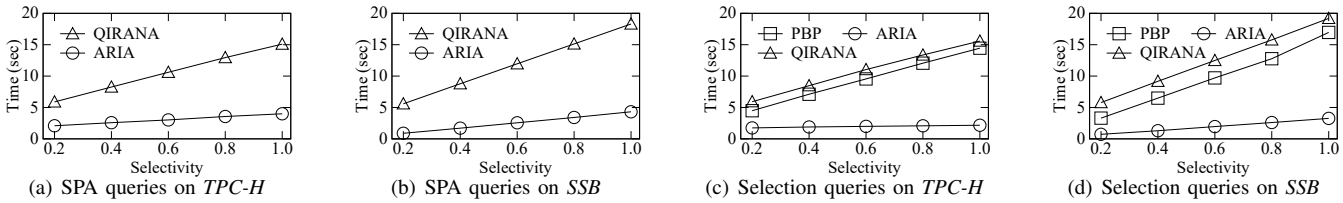
A. Scalability Evaluation

In this set of experiments, we first study the effect of the support set size on the performance of our framework ARIA. Then, we evaluate the scalability of ARIA under different query and database settings, compared to the state of the arts.

The first set of experiments explores the effect of the parameter $\sum |S_j|$, i.e., the total size of the support sets on all relations. Fig. 6 depicts the average query price and time cost of each dataset. We use the size of extra generated tuples in all support sets as the x-axis in Fig. 6, where the support set size is the summation of the dataset size and the extra tuple size. One can observe that, with the growth of the support set size, the price derived by ARIA is stable. This is because query information does not change, as does tuple-level information gain. Moreover, the larger the support sets, the higher the time cost. The reason is that, ARIA needs to process more possible values for each tuple to derive its price. In addition, ARIA also enables fast query pricing on large-scale benchmarks (i.e., *TPC-H* and *SSB*). In the following, we utilize the distinct tuples in each relation as the support set on that relation.

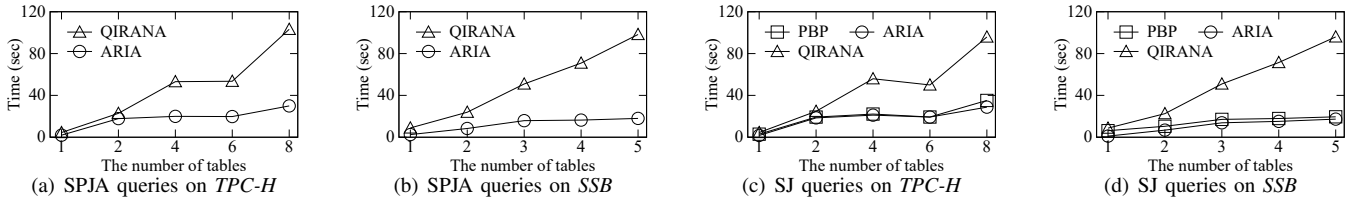
Table IV shows the comparison between ARIA and baselines, evaluated on four datasets with different types of queries. The execution time of the query itself is also reported, while "—" means that the method cannot price the type of queries or cannot complete within 2 hours. As all methods rely on query results to assign prices, they have the longer running time compared to the mere query execution. As observed, ARIA significantly outperforms all other methods on efficiency. It is 3x faster than QIRANA, which confirms the superiority of fine-grained information modeling. Meanwhile, PBP can efficiently price selection and join queries with linear complexity, while the exponential complexity of pricing projection queries is unacceptable on large-scale datasets.

Fig. 7 depicts the experimental results of varying the scale factor from 1 to 5 over *TPC-H* and *SSB* datasets. The results of pricing selection and join (SJ) queries are reported separately to illustrate the performance of the baseline PBP. We also show the execution time of priced queries to better evaluate the efficiency. Obviously, the time cost of query pricing increases with the scale factor, since all methods must evaluate more data to get the query price. In addition to the query execution time, ARIA spends the least time among all pricing methods. Moreover, the efficiency gap between ARIA and QIRANA becomes larger with the larger scale factor, since ARIA has lower complexity on price computation. Besides, the superiority of ARIA becomes less pronounced on *SSB*, since the query execution process takes up a lot of time in the query pricing. With the linear time complexity of pricing selection and join queries, PBP is faster than QIRANA in



(a) SPA queries on *TPC-H* (b) SPA queries on *SSB* (c) Selection queries on *TPC-H* (d) Selection queries on *SSB*

Fig. 8. Time cost of query pricing vs. the selectivity



(a) SPJA queries on *TPC-H* (b) SPJA queries on *SSB* (c) SJ queries on *TPC-H* (d) SJ queries on *SSB*

Fig. 9. Time cost of query pricing vs. the number of involved tables in the query

Figs. 7(c) and 7(d). PBP is slower than ARIA since PBP has extra operations to process the query results. Overall, ARIA is the most scalable approach to price SPJA queries.

We study the impact of the selectivity of the query on pricing. Fig. 8 depicts the experimental results. The selectivity is the ratio between the number of tuples satisfying the query predicates and the relation size. We control the predicates of the select-project-aggregate (SPA) queries to vary the selectivity. The results of pricing the selection query are depicted separately to show the performance of PBP, which can only handle the SJ queries in polynomial time. As observed, the running time of all methods grows with the selectivity, since they need to evaluate more query results to compute the price. ARIA is the fastest and less sensitive to selectivity, as the fine-grained tuple-level information significantly boosts efficiency.

Fig. 9 illustrates the experimental results of varying the number of involved tables in the query. The results of pricing the select-join (SJ) queries are shown separately to illustrate the performance of PBP. Obviously, the execution time over *SSB* increases with the number of tables in the query. This is because that the size of the query results on *SSB* expands with the number of tables. In contrast, all methods spend less time when the number of tables involved in queries on *TPC-H* shifts from 4 to 6. This phenomenon occurs as queries joining six relations have fewer quantified tuples, resulting in faster price computation. On top of fine-grained tuple-level information, ARIA outperforms QIRANA significantly, achieving a 3x speedup on *TPC-H* and a 5x speedup on *SSB*.

B. Pricing Characteristic Evaluation

In this part, we evaluate whether ARIA and the baselines meet pricing characteristics in Section II. As each pricing method formally proves the arbitrage-freeness, we focus on the characteristics of dependency-preserving and positive price.

Fist, we employ the price ratio to illustrate the dependency-preserving property, where the price ratio between queries Q_α and $Q_{\alpha \bigcup \beta}$ should be one if the pricing function is dependency-preserving. Fig. 10 illustrates the price ratios of queries under different public functional dependencies on *Walmart*, while other datasets have no such dependencies.

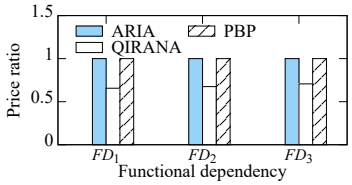


Fig. 10. Price ratio vs. functional dependency

The three functional dependencies are (latitude, longitude) $\longrightarrow$ (zipcode), (zipcode)$\longrightarrow$ (state), and (latitude, longitude) $\longrightarrow$ (city). As observed, the prices in ARIA can capture these functional dependencies, since all tuples in the support set obey these constraints. PBP has the 1 price ratio since the price of any attribute(s) is the same as that of all attributes. Actually, they do not capture the underlying consistent information. QIRANA fails to perceive the same information and prices the queries differently, as it uses the randomly generated possible databases that break these dependencies. In this case, the price of Q_α is undervalued, causing unnecessary loss of data sellers.

We study whether the selective queries are priced positively. In particular, we generate one selection query for each tuple in the database, where only this tuple satisfies the predicates and it is fully returned. Obviously, each query reveals the value of one tuple and should have a positive price. We accumulate the number of free queries for each method and utilize the *free query rate* to measure the frequency of unreasonable pricing behavior. Table V depicts the free query rates of all pricing frameworks. ARIA and PBP price all queries with positive prices, since they consider each query result (i.e., the corresponding tuple) for pricing. In contrast, QIRANA employs limited possible databases with one or two tuples different from the real ones. It can yield zero query price since all possible databases can have the same query answer as the real database instance. Hence, a large percentage of queries with non-empty and meaningful results are free in QIRANA, which is always unexpected.

C. Ablation Study

In this part, we conduct the ablation study on ARIA to confirm the effectiveness of the support set generation and the decomposition strategy used for pricing join queries.

TABLE V
FREE QUERY RATES OF PRICING FRAMEWORKS (%)

Method	*Walmart*	*Employment*	*TPC-H*	*SSB*
PBP	0	0	0	0
QIRANA	52.67	53.89	52.91	34.47
Our ARIA	0	0	0	0

TABLE VI
COMPARISON OF SUPPORT SET GENERATION STRATEGIES

Method	*Walmart*		*Employment*	*TPC-H*	*SSB*
	Time (s)	Price ratio	Time (s)	Time (s)	Time (s)
ARIA-UG	0.013	0.4	0.012	2.430	8.932
ARIA-UCG	0.013	1	0.011	2.387	8.901
ARIA	0.005	1	0.003	1.140	4.386

TABLE VII
COMPARISON OF DIFFERENT QUERY DECOMPOSITION STRATEGIES

Method	*TPC-H*		*SSB*	
	Time (s)	Price	Time (s)	Price
ARIA-Natural	5.19	3.52e+7	0.73	267681.2
ARIA-Sequential	—	—	—	—
ARIA	1.16	7.13e+4	5.68	113467.5

Table VI depicts the performance of ARIA under different support set generation strategies. UG randomly generates the support sets, while UCG considers database constraints in the generation. The default strategy leverages both original data and database constraints for generation, which is the fastest since it can utilize query results on original data for pricing. Moreover, the price ratio is evaluated following the settings in Section VII-B. The price ratio of UG is not one as it does not consider functional dependencies. Besides, the free query rates (following settings in Section VII-B) of all methods are zero due to the fine-grained tuple information, which are not illustrated in the table due to the space constraint.

We evaluate the performance of ARIA under different decomposition strategies on pricing the select-join queries, as illustrated in Table VII. ARIA-Natural ignores join predicates to obtain multiple single-relation queries, while ARIA-Sequential considers the $k!$ possible join orders for decomposition and employs the minimum total price as the final price. k is the number of tables involved in the join query. As observed, ARIA-Sequential cannot finish in two hours due to high time complexity. Meanwhile, ARIA-Natural can be faster than ARIA since the burden of the join operation is eliminated. However, the query under ARIA-Natural is much more expensive than that under ARIA, leading to high arbitrage risks. ARIA is more efficient and effective on pricing join queries with an arbitrage-free guarantee.

In summary, benefiting from the fine-grained tuple information modeling, ARIA is scalable on large-scale datasets and arbitrage-free for various SPJA queries. It not only performs 3x faster than previous pricing works but also maintains more desirable pricing properties. The effectiveness of the strategies in ARIA is also verified. Overall, ARIA is superior to the state of the arts in terms of both pricing efficiency and effectiveness.

VIII. RELATED WORK

SQL query pricing was introduced for the first time in database community in [6], where the arbitrage-free property is identified. View-based approaches (VBP) price each query as the total price of views that determine the query [7]–[9]. Then, the provenance-based method (PBP) is presented as the special case of VBP where each view corresponds to one tuple, which is faster than VBP [10], [11]. Some efficient heuristics are presented in [11] but have no arbitrage-freeness guarantee. VBP and PBP belong to the category of usage-based pricing and have exponential computation complexity when multiple view/tuple sets can produce each query result, which is unacceptable in practice. Moreover, the information-based method QIRANA [15], [16] turns the theoretical exploration of the arbitrage-free pricing [12]–[14] into practice. It measures the *database-level* information to price SPJA queries with quadratic complexity. In addition, the work [34] maximizes the revenue of query pricing on the top of QIRANA. Overall, existing SQL pricing approaches perform slowly on real-world benchmarks, far from promptly fulfilling customer demands.

Data pricing beyond SQL queries. Prior research has also explored pricing various other data products. Some studies focus on pricing linear aggregate queries over personal data in an arbitrage-free manner [20], [35]–[37]. However, these techniques are *not* applicable to SQL query pricing, as SPJ queries cannot be expressed as linear aggregations. Recent works extend the idea of pricing tabular data to graph queries, i.e., querying matched subgraphs and statistics [21], [38], [39]. Furthermore, the pricing of data, models, and services for machine learning has been investigated [22], [23], [40], [41]. Several comprehensive surveys provide an overview of the field of data pricing [42]–[45], summarizing key challenges, techniques, and future directions.

IX. CONCLUSION

In this paper, we propose an arbitrage-free and scalable pricing framework ARIA for SPJA queries. The tuple-level information is formulated and derived to capture what the query tells about each tuple. We propose arbitrage-free pricing functions based on information gain and efficient price computation algorithms with linear complexity. ARIA is 3x faster than the state of the arts while enjoying desirable pricing properties. Extensive experiments on real-world and large-scale datasets confirm the pricing effectiveness and efficiency of ARIA. In the future, we intend to model the information of complex aggregate queries (e.g., the average) and spatial queries (e.g., skyline queries) for pricing a broad set of queries.

ACKNOWLEDGMENTS

This work is partly supported by the National Key RD Program under Grant No. 2024YFB3908401, the National NSFC under Grants No. 62372404 and No. 62125206, the Fundamental Research Funds for the Central Universities under Grant No. 226-2024-00030, and the Leading Goose RD Program of Zhejiang under Grant No. 2024C01109. Jinshan Zhang was supported by National NSFC under Grant No. 62472376 and the Key RD Program of Zhejiang under Grant No. 2023C01002. Xiaoye Miao is the corresponding author.

REFERENCES

[1] C. I. Jones and C. Tonetti, "Nonrivalry and the economics of data," *Am. Econ. Rev.*, vol. 110, no. 9, pp. 2819–2858, 2020.

[2] M. Ghasemaghaei and G. Calic, "Assessing the impact of big data on firm innovation performance: Big data is not always better data," *J. Bus. Res.*, vol. 108, pp. 147–162, 2020.

[3] G. view research. (2024) Big data market size, share & trends analysis report. [Online]. Available: https://www.grandviewresearch.com/industry-analysis/big-data-industry

[4] AggData. (2024) Aggdata market. [Online]. Available: https://www.aggdata.com

[5] Amazon. (2023) Aws marketplace. [Online]. Available: https://aws.amazon.com/marketplace

[6] M. Balazinska, B. Howe, and D. Suciu, "Data markets in the cloud: An opportunity for the database community," *Proc. VLDB Endow.*, vol. 4, no. 12, pp. 1482–1485, 2011.

[7] P. Koutris, P. Upadhyaya, M. Balazinska, B. Howe, and D. Suciu, "Query-based data pricing," in *PODS*, 2012, pp. 167–178.

[8] ——, "Toward practical query pricing with query market," in *SIGMOD*, 2013, pp. 613–624.

[9] ——, "Query-based data pricing," *J. ACM*, vol. 62, no. 5, p. 43, 2015.

[10] X. Miao, Y. Gao, L. Chen, H. Peng, J. Yin, and Q. Li, "Towards query pricing on incomplete data," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 8, pp. 4024–4036, 2022.

[11] R. Tang, H. Wu, Z. Bao, S. Bressan, and P. Valduriez, "The price is right: Models and algorithms for pricing data," in *DEXA*, 2013, pp. 380–394.

[12] B. Lin and D. Kifer, "On arbitrage-free pricing for general data queries," *Proc. VLDB Endow.*, vol. 7, no. 9, pp. 757–768, 2014.

[13] ——, "Information measures in statistical privacy and data processing applications," *ACM Trans. Knowl. Discov. Data*, vol. 9, no. 4, pp. 28:1–28:29, 2015.

[14] S. Deep and P. Koutris, "The design of arbitrage-free data pricing schemes," in *ICDT*, 2017, pp. 12:1–12:18.

[15] ——, "QIRANA: A framework for scalable query pricing," in *SIGMOD*, 2017, pp. 699–713.

[16] S. Deep, P. Koutris, and Y. Bidasaria, "QIRANA demonstration: Real time scalable query pricing," *Proc. VLDB Endow.*, vol. 10, no. 12, pp. 1949–1952, 2017.

[17] TPC. (2024) TPC-H dataset. [Online]. Available: http://www.tpc.org/tpch

[18] Y. Li, H. Sun, B. Dong, and W. H. Wang, "Cost-efficient data acquisition on online data marketplaces for correlation analysis," *Proc. VLDB Endow.*, vol. 12, no. 4, pp. 362–375, 2018.

[19] S. Hu, S. Yao, H. Jin, Y. Zhao, Y. Hu, X. Liu, N. Naghibolhosseini, S. Li, A. Kapoor, W. Dron, L. Su, A. Bar-Noy, P. A. Szekely, R. Govindan, R. L. Hobbs, and T. F. Abdelzaher, "Data acquisition for real-time decision-making under freshness constraints," in *RTSS*, 2015, pp. 185–194.

[20] C. Li, D. Y. Li, G. Miklau, and D. Suciu, "A theory of pricing private data," *ACM Trans. Database Syst.*, vol. 39, no. 4, pp. 34:1–34:28, 2014.

[21] C. Chen, Y. Yuan, Z. Wen, G. Wang, and A. Li, "GQP: A framework for scalable and effective graph query-based pricing," in *ICDE*, 2022, pp. 1573–1585.

[22] J. Liu, J. Lou, J. Liu, L. Xiong, J. Pei, and J. Sun, "Dealer: An end-to-end model marketplace with differential privacy," *Proc. VLDB Endow.*, vol. 14, no. 6, 2021.

[23] H. Peng, X. Miao, L. Chen, Y. Gao, and J. Yin, "Pricing prediction services for profit maximization with incomplete information," in *ICDE*, 2023, pp. 1353–1365.

[24] A. Nash, L. Segoufin, and V. Vianu, "Views and queries: Determinacy and rewriting," *ACM Trans. Database Syst.*, vol. 35, no. 3, pp. 1–41, 2010.

[25] Wiki. (2024) Entropy (information theory). [Online]. Available: https://en.wikipedia.org/wiki/Entropy_(information_theory)

[26] B. Hilprecht, A. Schmidt, M. Kulessa, A. Molina, K. Kersting, and C. Binnig, "Deepdb: Learn from data, not from queries!" *Proc. VLDB Endow.*, vol. 13, no. 7, pp. 992–1005, 2020.

[27] M.-S. Chen, J. Han, and P. S. Yu, "Data mining: An overview from a database perspective," *IEEE Trans. Knowl. Data Eng.*, vol. 8, no. 6, pp. 866–883, 1996.

[28] Wiki. (2024) Dependency theory. [Online]. Available: https://en.wikipedia.org/wiki/Dependency_theory_(database_theory)

[29] A. Silberschatz, H. F. Korth, and S. Sudarshan, "Database system concepts," 2011.

[30] C. Beeri, P. A. Bernstein, and N. Goodman, "A sophisticate's introduction to database normalization theory," in *VLDB*, 1978, pp. 113–124.

[31] Foursquare. (2024) Fsq places data: Walmart in us. [Online]. Available: https://aws.amazon.com/marketplace/pp/prodview-zaejml2253r7k

[32] S. Data. (2024) Us employment numbers by industry. [Online]. Available: https://aws.amazon.com/marketplace/pp/prodview-yp5x2esst5dji#offers

[33] P. E. O'Neil, E. J. O'Neil, X. Chen, and S. Revilak, "The star schema benchmark and augmented fact table indexing," in *TPCTC*, vol. 5895, 2009, pp. 237–252.

[34] S. Chawla, S. Deep, P. Koutris, and Y. Teng, "Revenue maximization for query pricing," *Proc. VLDB Endow.*, vol. 13, no. 1, pp. 1–14, 2019.

[35] C. Niu, Z. Zheng, F. Wu, S. Tang, X. Gao, and G. Chen, "Unlocking the value of privacy: Trading aggregate statistics over private correlated data," in *SIGKDD*, 2018, pp. 2031–2040.

[36] H. Cai, F. Ye, Y. Yang, Y. Zhu, J. Li, and F. Xiao, "Online pricing and trading of private data in correlated queries," *IEEE Trans. Parallel Distributed Syst.*, vol. 33, no. 3, pp. 569–585, 2022.

[37] Y. Fu, X. Miao, H. Peng, C. Na, S. Deng, and J. Yin, "Online query-based data pricing with time-discounting valuations," in *ICDE*, 2024, pp. 3449–3461.

[38] H. Hou, L. Qiao, Y. Yuan, C. Chen, and G. Wang, "A scalable query pricing framework for incomplete graph data," in *DASFAA*, 2023, pp. 97–113.

[39] C. Chen, Y. Yuan, Z. Wen, Y. Wang, and G. Wang, "Gshop: Towards flexible pricing for graph statistics," in *ICDE*, 2024, pp. 2612–2624.

[40] L. Chen, P. Koutris, and A. Kumar, "Towards model-based pricing for machine learning in a data marketplace," in *SIGMOD*, 2019, pp. 1535–1552.

[41] A. Xu, Z. Zheng, Q. Li, F. Wu, and G. Chen, "VAP: Online data valuation and pricing for machine learning models in mobile health," *IEEE Trans. Mob. Comput.*, vol. 23, no. 5, pp. 5966–5983, 2024.

[42] J. Pei, "A survey on data pricing: From economics to data science," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 10, pp. 4586–4608, 2020.

[43] M. Zhang, F. Beltrán, and J. Liu, "A survey of data pricing for data marketplaces," *IEEE Trans. Big Data*, vol. 9, no. 4, pp. 1038–1056, 2023.

[44] X. Miao, H. Peng, X. Huang, L. Chen, Y. Gao, and J. Yin, "Modern data pricing models: Taxonomy and comprehensive survey," *arXiv preprint arXiv:2306.04945*, 2023.

[45] J. Zhang, Y. Bi, M. Cheng, J. Liu, K. Ren, Q. Sun, Y. Wu, Y. Cao, R. C. Fernandez, H. Xu *et al.*, "A survey on data markets," *arXiv preprint arXiv:2411.07267*, 2024.