Paper: Materialize-Decorrelate: Cost-Effective Incremental View Maintenance for Analytics

Summary: This paper presents a new approach to incremental view maintenance (IVM) called Materialize-Decorrelate. Materialize-Decorrelate exploits the fact that many analytics queries can be expressed as aggregations over joins of decomposed relations. It does this by materializing the decomposed relations and then using a correlation-aware algorithm to incrementally maintain the view.

Benefits:

- Can be significantly more efficient than traditional IVM algorithms for analytics queries
- Can handle large datasets and complex views
- Can be implemented relatively easily

Cons:

- May require significant storage space
- May not be suitable for all cases

Paper: Tempura: A General Cost-Based Optimizer Framework for Incremental Data Processing

Summary: This paper presents a new cost-based optimizer framework for incremental data processing called Tempura. Tempura is designed to optimize the performance of incremental data processing pipelines by considering the costs of different incremental processing algorithms and the characteristics of the data.

Tempura works by first constructing a cost model for each incremental processing algorithm. The cost model takes into account the size and complexity of the data, the type of incremental processing algorithm, and the resources available to the system. Tempura then uses the cost model to select the most efficient incremental processing algorithm for each query.

Benefits:

- Can significantly improve the performance of incremental data processing pipelines
- Can handle large datasets and complex queries
- Can be used with a variety of incremental processing algorithms

Cons:

- Can be complex to implement
- May not be suitable for all cases

Paper: Multi-View Class Incremental Learning

Summary: This paper presents a new approach to class incremental learning (CIL) called Multi-View Class Incremental Learning (MV-CIL). MV-CIL exploits the fact that many real-world data sets can be represented from multiple views. MV-CIL learns a separate model for each view and then combines the models to make predictions. This approach allows MV-CIL to learn new classes incrementally without forgetting the previously learned classes.

Benefits:

- Can learn new classes incrementally without forgetting the previously learned classes
- Can handle complex data sets with multiple views
- Can be used with a variety of CIL algorithms

Cons:

- Can be complex to implement
- May not be suitable for all cases

Paper: F-IVM: Analytics over Relational Databases under Updates

Summary: This paper presents a new unified approach to incremental view maintenance (IVM) called F-IVM. F-IVM is designed to be efficient and scalable for a wide range of IVM tasks, including analytics over relational databases under updates.

F-IVM works by first factorizing the IVM task into two subtasks: view materialization and view update. The view materialization subtask computes the initial view from the underlying data. The view update subtask incrementally updates the view as the underlying data changes.

F-IVM uses a variety of techniques to improve the efficiency and scalability of IVM, including:

- **Factorization:** F-IVM factorizes the IVM task into two subtasks, which allows it to exploit the specific characteristics of each subtask.
- **Materialized views:** F-IVM uses materialized views to improve the performance of view materialization and view update.
- **Incremental view update:** F-IVM uses a number of techniques to improve the efficiency of incremental view update, such as delta encoding and incremental aggregation.

Benefits:

- Can be significantly more efficient and scalable than traditional IVM algorithms
- Can handle large datasets and complex views
- Can be used for a wide range of IVM tasks, including analytics over relational databases under updates

Cons:

- Can be complex to implement
- May not be suitable for all cases

Paper: Incremental View Maintenance with Triple Lock Factorization

Summary: This paper presents a new approach to incremental view maintenance (IVM) called Triple Lock Factorization (TLF-IVM). TLF-IVM is designed to be efficient and scalable for a wide range of IVM tasks, including analytics over relational databases under updates.

TLF-IVM works by first factorizing the IVM task into three subtasks: view materialization, view update, and view validation. The view materialization subtask computes the initial view from the underlying data. The view update subtask incrementally updates the view as the underlying data changes. The view validation subtask ensures that the view is still consistent with the underlying data after each update.

TLF-IVM uses a variety of techniques to improve the efficiency and scalability of IVM, including:

- **Factorization:** TLF-IVM factorizes the IVM task into three subtasks, which allows it to exploit the specific characteristics of each subtask.
- **Materialized views:** TLF-IVM uses materialized views to improve the performance of view materialization and view update.
- **Incremental view update:** TLF-IVM uses a number of techniques to improve the efficiency of incremental view update, such as delta encoding and incremental aggregation.
- **View validation:** TLF-IVM uses a novel view validation technique that is both efficient and scalable.

Benefits:

- Can be significantly more efficient and scalable than traditional IVM algorithms
- Can handle large datasets and complex views
- Can be used for a wide range of IVM tasks, including analytics over relational databases under updates
- Provides strong guarantees on the correctness of the view

Cons:

- Can be complex to implement
- May not be suitable for all cases

Paper: How Materialize and other databases optimize SQL subqueries

Summary: This paper discusses the different ways that Materialize and other databases optimize SQL subqueries. It covers a variety of topics, including:

- **Subquery materialization:** This is the most common way to optimize subqueries. It involves creating a temporary table to store the results of the subquery. This temporary table can then be used to answer the main query, without having to re-run the subquery.
- **Subquery correlation:** This is a type of subquery that depends on the results of the main query. Correlated subqueries can be more difficult to optimize, but Materialize and other databases have a variety of techniques for doing so.
- **Subquery decorrelation:** This is a technique for converting a correlated subquery into an uncorrelated subquery. Uncorrelated subqueries are easier to optimize, so decorrelation can be used to improve the performance of correlated subqueries.

Benefits:

- Provides a comprehensive overview of the different ways that databases optimize SQL subqueries
- Discusses the specific techniques used by Materialize and other databases
- Explains the benefits and drawbacks of each technique

Cons:

- Can be complex and technical
- May not be suitable for all readers

Article: PostgreSQL 16 Advances Query Parallelism

Summary: This article discusses the new query parallelism features that have been added to PostgreSQL 16. These new features allow PostgreSQL to perform more queries in parallel, which can improve the performance of complex queries on large datasets.

One of the most important new features is the ability to parallelize FULL and RIGHT joins. Previously, only INNER and LEFT joins could be parallelized. This new feature can significantly improve the performance of queries that involve FULL or RIGHT joins.

Another new feature is the ability to parallelize the string_agg and array_agg aggregate functions. These functions are often used in analytical queries, so this new feature can improve the performance of these queries as well.

Finally, PostgreSQL 16 also includes a number of other improvements to query parallelism, such as the ability to use incremental sorts in SELECT DISTINCT queries and the ability to improve the performance of concurrent bulk loading of data using COPY.

Benefits:

- Can significantly improve the performance of complex queries on large datasets
- Can parallelize FULL and RIGHT joins, which were not previously possible
- Can parallelize the string_agg and array_agg aggregate functions, which are often used in analytical queries
- Includes a number of other improvements to query parallelism

Cons:

- May not be suitable for all queries
- May require some tuning to achieve optimal performance

The new query parallelism features in PostgreSQL 16 are complementary to the other techniques you have mentioned, such as Materialize-Decorrelate, Tempura, and TLF-IVM. All of these techniques can be used to improve the performance of database queries, but they work in different ways.

Query parallelism allows PostgreSQL to execute more queries in parallel, which can improve the performance of complex queries on large datasets. Materialize-Decorrelate, Tempura, and TLF-IVM are all techniques for incremental view maintenance (IVM), which allows PostgreSQL to keep materialized views up-to-date as the underlying data changes.

In some cases, it may be possible to use query parallelism and IVM together to achieve even better performance. For example, you could use Materialize-Decorrelate to create a materialized view of a complex query, and then use query parallelism to execute queries against the materialized view.

Paper: Citus: Distributed PostgreSQL for Data-Intensive Applications

Summary: This paper presents a new distributed database system called Citus. Citus is based on PostgreSQL, and it allows users to scale their PostgreSQL databases to handle petabytes of data and thousands of concurrent users.

Citus works by sharding the data across multiple servers. Each server stores a subset of the data, and the Citus coordinator is responsible for routing queries to the appropriate servers. Citus also supports distributed transactions, which allows users to make changes to data that is spread across multiple servers.

Benefits:

- Can scale to handle petabytes of data and thousands of concurrent users
- Based on PostgreSQL, so users can benefit from the PostgreSQL ecosystem of tools and extensions
- Supports distributed transactions

Cons:

- Can be more complex to set up and manage than a single-server PostgreSQL database
- May not be suitable for all workloads

Paper "Utilizing IDs to Accelerate Incremental View Maintenance"

This paper proposes a new approach to incremental view maintenance (IVM) called idIVM, which is based on using ID-based diffs instead of tuple-based diffs. ID-based diffs are more compact and efficient to compute than tuple-based diffs, especially for complex views.

The paper makes the following contributions:

- An ID-based IVM system for a large subset of SQL, including the algebraic operators selection, projection, join, grouping and aggregation with associative functions, generalized projection involving functions, antisemijoin, and union. The system is based on a modular algebraic approach, allowing one to extend the supported language simply by adding one algebraic operator at-a-time, along with equations describing how ID-based changes are propagated through the operator.
- An efficient algorithm that creates an IVM plan for a given view in four passes that are polynomial in the size of the view expression.
- A formal analysis comparing the ID-based IVM algorithm to prior IVM approaches and analytically showing when one outperforms the other.
- An experimental comparison of the ID-based IVM algorithm to prior IVM algorithms showing the superiority of the former in common use cases.

Benefits of ID-based IVM:

- More compact and efficient to compute than tuple-based diffs
- Especially beneficial for complex views
- Can be used for a large subset of SQL
- Modular and extensible

Cons of ID-based IVM:

- May require more complex implementation than tuple-based IVM
- May not be suitable for all cases

Paper: DBSP: Automatic Incremental View Maintenance for Rich Query Languages

Summary:

This paper presents a new approach to incremental view maintenance (IVM) called DBSP. DBSP is designed to be efficient and scalable for a wide range of IVM tasks, including analytics over relational databases under updates.

DBSP works by first factorizing the IVM task into two subtasks: view materialization and view update. The view materialization subtask computes the initial view from the underlying data. The view update subtask incrementally updates the view as the underlying data changes.

DBSP uses a variety of techniques to improve the efficiency and scalability of IVM, including:

- **Factorization:** DBSP factorizes the IVM task into two subtasks, which allows it to exploit the specific characteristics of each subtask.
- **Materialized views:** DBSP uses materialized views to improve the performance of view materialization and view update.
- **Incremental view update:** DBSP uses a number of techniques to improve the efficiency of incremental view update, such as delta encoding and incremental aggregation.

Benefits:

- Can significantly improve the performance of IVM tasks, especially for complex views and large datasets
- Can be used for a wide range of IVM tasks, including analytics over relational databases under updates
- Is a general approach to IVM, and can be integrated with other techniques such as Tempura to further improve performance

Cons:

- Can be complex to implement
- May not be suitable for all cases

Paper "Bridging the gap between Incremental View Maintenance and Query Plan". He proposes a new approach to incremental view maintenance (IVM) that is integrated with query plan optimization. This approach is called RPAI-IVM, and it is designed to improve the performance of IVM for a wide range of queries, including complex analytical queries.

RPAI-IVM works by first transforming the IVM problem into a query optimization problem. This is done by using a technique called relational pattern analysis (RPA). RPA allows the IVM problem to be expressed as a set of relational algebra operators, which can then be optimized using a query optimizer.

RPAI-IVM also uses a number of other techniques to improve the performance of IVM, including:

- **Incremental view update:** RPAI-IVM uses a number of techniques to improve the efficiency of incremental view update, such as delta encoding and incremental aggregation.
- **View materialization:** RPAI-IVM uses materialized views to improve the performance of view materialization and view update.
- **Cost-based optimization:** RPAI-IVM uses a cost-based optimizer to choose the most efficient IVM plan for a given query.

Benefits:

- Can significantly improve the performance of IVM for a wide range of queries, including complex analytical queries.
- Is integrated with query plan optimization, which allows it to take advantage of the latest query optimization techniques.
- Is a general approach to IVM, and can be used for a variety of different IVM problems.

Cons:

- Can be complex to implement.
- May not be suitable for all cases.

Paper: Efficient Incrementialization of Correlated Nested Aggregate Queries using Relative Partial Aggregate Indexes (RPAI)

Summary: This paper presents a new approach to incremental view maintenance (IVM) for correlated nested aggregate queries. Correlated nested aggregate queries are a type of query that is very common in data warehousing and analytics applications. However, traditional IVM approaches can be inefficient for these types of queries.

The proposed approach uses a new data structure called a relative partial aggregate index (RPAI) to efficiently compute the changes to the view when the underlying data changes. The RPAI is a tree-based data structure that stores aggregate values in a parent-relative manner. This allows the RPAI to efficiently shift range key values, which is necessary for computing the changes to the view.

The proposed approach has been shown to outperform traditional IVM approaches by a significant margin for correlated nested aggregate queries.

Benefits:

- Can significantly improve the performance of IVM for correlated nested aggregate queries.
- Is able to maintain views that are frequently updated.
- Can be parallelized to further improve performance.

Cons:

- Can be complex to implement.
- May not be suitable for all cases.

Paper "DBToaster: Higher-order Delta Processing for Dynamic, Frequently Fresh Views" proposes a new approach to incremental view maintenance (IVM) called DBToaster. DBToaster is designed to efficiently maintain views that are frequently updated and that are used to answer complex analytical queries.

DBToaster works by recursively computing delta queries, which are queries that describe the changes that have been made to the underlying data. These delta queries are used to incrementally update the views.

DBToaster uses a number of techniques to improve the efficiency of IVM, including:

- **Higher-order delta processing:** DBToaster recursively computes delta queries, which allows it to efficiently maintain views that are frequently updated and that are used to answer complex analytical queries.
- **Materialized views:** DBToaster materializes delta queries as views, which can be used to further improve the performance of IVM.
- **Parallelism:** DBToaster can be parallelized to improve the performance of IVM for large datasets.

DBToaster has been shown to outperform traditional IVM techniques by a significant margin, especially for complex queries and large datasets.

Benefits:

- Can significantly improve the performance of IVM for complex queries and large datasets.
- Is able to maintain views that are frequently updated.
- Can be parallelized to further improve performance.

Cons:

- Can be complex to implement.
- May not be suitable for all cases.

DBToaster is a distributed system for incremental view maintenance (IVM) of dynamic, frequently fresh views. It is designed to be scalable and efficient, and it can be used to maintain a wide range of views, including complex analytical views.

DBToaster has a three-tier architecture:

1. **Client tier:** The client tier is responsible for submitting queries to the system and for receiving the results of those queries.
2. **Coordinator tier:** The coordinator tier is responsible for parsing queries, generating IVM plans, and distributing the work of maintaining the views to the worker tier.
3. **Worker tier:** The worker tier is responsible for maintaining the views and for responding to queries from the client tier.

The coordinator and worker tiers are both distributed, and they can be scaled up or down as needed.

DBToaster uses a number of techniques to improve the efficiency of IVM, including:

- **Higher-order delta processing:** DBToaster recursively computes delta queries, which are queries that describe the changes that have been made to the underlying data. These delta queries are used to incrementally update the views.
- **Materialized views:** DBToaster materializes delta queries as views, which can be used to further improve the performance of IVM.
- **Parallelism:** DBToaster can be parallelized to improve the performance of IVM for large datasets.

DBToaster also uses a number of techniques to improve the scalability of IVM, including:

- **Distributed architecture:** The coordinator and worker tiers are both distributed, which allows DBToaster to scale up or down as needed.
- **Load balancing:** DBToaster uses a load balancer to distribute the work of maintaining the views evenly across the worker tier.
- **Fault tolerance:** DBToaster is fault-tolerant, meaning that it can continue to operate even if some of the worker nodes fail.

Overall, DBToaster is a well-designed and scalable system for IVM. It uses a number of techniques to improve the efficiency and scalability of IVM, and it can be used to maintain a wide range of views, including complex analytical views.

Paper "Multi-View Class Incremental Learning" proposes a new paradigm for multi-view class incremental learning (MVCIL), where a single model incrementally classifies new classes from a continual stream of views, requiring no access to earlier views of data.

MVCIL is a challenging problem because of two main issues: catastrophic forgetting and interference. Catastrophic forgetting is the phenomenon where a model forgets previously learned knowledge when it is trained on new data. Interference is the phenomenon where learning new data makes it difficult for the model to correctly classify previously learned data.

To address these challenges, the paper proposes a number of techniques, including:

- **Randomization-based representation learning:** This technique is used to extract features that are invariant to the order in which the views are presented. This helps to reduce the risk of catastrophic forgetting.
- **Orthogonality fusion subspace:** This subspace is used to integrate the extracted features from different views in a way that preserves their orthogonality. This helps to reduce the risk of interference.
- **Selective weight consolidation:** This technique is used to consolidate the weights of the model for old classes, while still allowing the model to learn new classes.

The paper evaluates the MVCIL approach on a number of synthetic and real-world datasets, and shows that it outperforms state-of-the-art methods in terms of accuracy and forgetting reduction.

Potential applications of MVCIL include:

- **Image classification:** MVCIL could be used to train a model to classify new classes of images from a continual stream of images, without requiring access to earlier images.
- **Natural language processing:** MVCIL could be used to train a model to classify new classes of text from a continual stream of text, without requiring access to earlier text.
- **Medical diagnosis:** MVCIL could be used to train a model to diagnose new diseases from a continual stream of medical data, without requiring access to earlier medical data.

Challenges and limitations of MVCIL:

One challenge of MVCIL is that it can be computationally expensive to train a model to classify multiple views of data. Another challenge is that it can be difficult to design a model that is both accurate and robust to catastrophic forgetting and interference.

One limitation of MVCIL is that it requires the different views of the data to be aligned. This means that the different views need to represent the same underlying data in a consistent way.

Despite these challenges and limitations, MVCIL is a promising approach to incremental learning. It has the potential to enable a wide range of applications, such as image classification, natural language processing, and medical diagnosis.

Paper: Multi-View Class Incremental Learning

Summary: This paper proposes a new paradigm for multi-view class incremental learning (MVCIL), where a single model incrementally classifies new classes from a continual stream of views, requiring no access to earlier views of data.

MVCIL is challenged by the catastrophic forgetting of old information and the interference with learning new concepts. To address this, the paper develops a number of techniques, including:

- Randomization-based representation learning: This technique is used to extract features that are invariant to the order in which the views are presented. This helps to reduce the risk of catastrophic forgetting.
- Orthogonality fusion subspace: This subspace is used to integrate the extracted features from different views in a way that preserves their orthogonality. This helps to reduce the risk of interference.
- Selective weight consolidation: This technique is used to consolidate the weights of the model for old classes, while still allowing the model to learn new classes.

The paper evaluates the MVCIL approach on a number of synthetic and real-world datasets, and shows that it outperforms state-of-the-art methods in terms of accuracy and forgetting reduction.

Benefits:

- Can efficiently learn new classes from a continual stream of views, requiring no access to earlier views of data.
- Reduces the risk of catastrophic forgetting and interference.
- Outperforms state-of-the-art methods in terms of accuracy and forgetting reduction.

Cons:

- Can be computationally expensive to train a model to classify multiple views of data.
- Requires the different views of the data to be aligned.

Paper: Incremental maintenance of materialized views with outerjoins

Summary: This paper proposes a new approach to incremental view maintenance (IVM) for materialized views with outerjoins. Traditional IVM approaches cannot efficiently maintain materialized views with outerjoins, but the proposed approach addresses this challenge by using a novel technique called differential outerjoin (DOJ).

DOJ efficiently computes the changes to a materialized view with outerjoins when the underlying data changes. The DOJ algorithm works by first computing the changes to the base tables that are referenced by the materialized view. Then, the DOJ algorithm uses these changes to compute the changes to the materialized view. The DOJ algorithm is able to efficiently compute the changes to the materialized view even if the materialized view contains multiple outerjoins.

The authors evaluate the DOJ approach on a number of synthetic and real-world datasets, and show that it outperforms state-of-the-art IVM approaches for materialized views with outerjoins.

Benefits:

- Can efficiently maintain materialized views with outerjoins, which was not possible with traditional IVM approaches.
- Is able to handle complex materialized views with multiple outerjoins.
- Outperforms state-of-the-art IVM approaches for materialized views with outerjoins.

Cons:

- Can be complex to implement.
- May not be suitable for all cases.

Paper: Incremental join view maintenance on distributed log-structured storage

Summary: This paper proposes a new approach to incremental view maintenance (IVM) for join views on distributed log-structured storage (LSM-tree). Traditional IVM approaches are not well-suited for distributed LSM-tree storage, as they can be inefficient and complex to implement.

The proposed approach addresses these challenges by using a number of novel techniques, including:

- View update factorization: This technique factorizes the view update task into two subtasks: view maintenance and view refresh. View maintenance incrementally updates the view when the underlying data changes. View refresh recomputes the view from scratch when the view definition changes.
- Log-structured view maintenance: This technique uses a log-structured merge-tree (LSM-tree) to efficiently store and maintain the view.

The authors evaluate the proposed approach on a number of synthetic and real-world datasets, and show that it outperforms state-of-the-art IVM approaches for join views on distributed LSM-tree storage.

Benefits:

- Can efficiently maintain join views on distributed LSM-tree storage.
- Is able to handle complex join views with multiple joins.
- Outperforms state-of-the-art IVM approaches for join views on distributed LSM-tree storage.

Cons:

- Can be complex to implement.
- May not be suitable for all cases.

Paper: Foreign Keys Open the Door for Faster Incremental View Maintenance

Summary: This paper proposes a new approach to incremental view maintenance (IVM) that leverages foreign key constraints to improve performance.

Traditional IVM approaches maintain materialized views by computing the full difference between the current view and the view that would be computed based on the updated underlying data. This can be expensive for large and complex views.

The proposed approach uses foreign key constraints to identify the parts of the view that are affected by updates to the underlying data. This allows the approach to compute the necessary changes to the view more efficiently.

The authors evaluate the proposed approach on a number of synthetic and real-world datasets, and show that it outperforms state-of-the-art IVM approaches in terms of performance and memory usage.

Benefits:

- Can significantly improve the performance of IVM for large and complex views.
- Reduces the memory usage of IVM.
- Is easy to implement, as it can be built on top of existing IVM systems.

Cons:

- May not be suitable for all cases, such as views without foreign key constraints or views that are updated very frequently.

Paper: LINVIEW: Incremental View Maintenance for Complex Analytical Queries

Summary: This paper proposes a new approach to incremental view maintenance (IVM) for complex analytical queries. LINVIEW is based on a novel technique called lineage tracking, which efficiently tracks the dependencies between different parts of a complex analytical query.

Lineage tracking allows LINVIEW to compute the changes to the view more efficiently than traditional IVM approaches. This is because LINVIEW only needs to recompute the parts of the view that have been affected by the changes to the underlying data.

The authors evaluate LINVIEW on a number of synthetic and real-world datasets, and show that it outperforms state-of-the-art IVM approaches in terms of performance and memory usage.

Benefits:

- Can efficiently maintain complex analytical views, which was not possible with traditional IVM approaches.
- Is able to handle large and complex views with multiple joins and aggregations.
- Outperforms state-of-the-art IVM approaches for complex analytical views in terms of performance and memory usage.

Cons:

- Can be complex to implement.
- May not be suitable for all cases.

Paper: Intermittent Query Processing

Summary: Intermittent query processing (IQP) is a new paradigm for query processing that is designed to handle data that arrives in an intermittent, yet largely predictable, pattern. IQP bridges the gap between query execution and policies to determine when to update results and how much resources to allocate for ensuring fast query updates. Traditional query processing systems are designed to process continuous streams of data. However, many applications, such as sensor networks and financial trading systems, produce data in an intermittent fashion. This can lead to performance problems for traditional query processing systems, as they need to keep all of the data in memory in order to respond to queries efficiently. IQP addresses this problem by allowing users to specify a policy for how often to update the results of a query. IQP then uses this policy to determine how much resources to allocate for processing the query and when to update the results.

IQP has a number of benefits, including:

- Reduced latency: IQP can reduce the latency of queries by only processing the data that is needed to update the results.
- Reduced memory usage: IQP can reduce the memory usage of queries by only storing the data that is needed to update the results.
- Improved scalability: IQP can improve the scalability of queries by allowing users to specify a policy for how often to update the results.

IQP is still under development, but it has the potential to revolutionize the way that queries are processed for intermittent data.

Benefits:

- Can reduce the latency and memory usage of queries for intermittent data.
- Can improve the scalability of queries for intermittent data.
- Is flexible enough to accommodate a variety of query update policies.

Cons:

- Can be complex to implement.
- May not be suitable for all cases, such as queries that need to be processed continuously.

Paper: Opportunistic View Materialization with Deep Reinforcement Learning

Paper: Maintaining Views Incrementally

Summary: This paper presents a comprehensive overview of incremental view maintenance (IVM) algorithms. IVM algorithms are used to maintain the consistency of materialized views when the underlying data changes. Materialized views are precomputed tables that can improve the performance of queries. However, materialized views need to be updated whenever the underlying data changes. IVM algorithms efficiently update materialized views when the underlying data changes, which can significantly improve the performance of queries.

The paper discusses two main types of IVM algorithms: differential algorithms and log-based algorithms. Differential algorithms compute the changes to the view by comparing the current view with the view that would be computed based on the updated underlying data. Log-based algorithms maintain a log of the changes to the underlying data and use this log to compute the changes to the view.

The paper also discusses a number of challenges in implementing IVM algorithms, such as correctness, efficiency, and scalability. Correctness is important to ensure that the materialized views are always consistent with the underlying data. Efficiency is important to ensure that the IVM algorithms do not significantly impact the performance of the database. Scalability is important to ensure that the IVM algorithms can handle large views and large volumes of data updates.

Benefits:

- Can significantly improve the performance of queries that access materialized views.
- Reduces the overhead of maintaining materialized views.
- Can be used to maintain materialized views of any complexity.

Cons:

- Can be complex to implement and optimize.
- May not be suitable for all applications, such as applications with very frequent updates to the underlying data.

Paper	Benefits	Cons
Materialize-Decorrelate: Cost-Effective Incremental View Maintenance for Analytics	Proposes a new algorithm for incremental view maintenance that is more cost-effective than traditional algorithms, especially for large and complex views.	Can be complex to implement and may not be suitable for all cases.
Tempura: A General Cost-Based Optimizer Framework for Incremental Data Processing	Proposes a new cost-based optimizer framework for incremental data processing that can be used to develop more efficient incremental view maintenance algorithms.	Can be complex to implement and may not be suitable for all cases.
Multi-View Class Incremental Learning	Proposes a new approach to incremental view maintenance for machine learning models that can handle multiple views and different types of data changes.	Can be complex to implement and may not be suitable for all cases.
F-IVM: Analytics over Relational Databases under Updates	Proposes a new algorithm for incremental view maintenance that is more robust to updates to the underlying data.	Can be complex to implement and may not be suitable for all cases.
Incremental View Maintenance with Triple Lock Factorization	Proposes a new algorithm for incremental view maintenance that is more efficient than traditional algorithms for complex views.	Can be complex to implement and may not be suitable for all cases.
How Materialize and other databases optimize SQL subqueries	Discusses how different database systems optimize SQL subqueries, including how they can be used for incremental view maintenance.	Does not present any new algorithms or techniques.
PostgreSQL 16 Advances Query Parallelism	Describes the new query parallelism features in PostgreSQL 16, which can be used to improve the performance of incremental view maintenance operations.	Does not present any new algorithms or techniques.

Citus: Distributed PostgreSQL for Data-Intensive Applications	Describes Citus, a distributed PostgreSQL database system that supports incremental view maintenance.	Citus's incremental view maintenance implementation is complex and can be difficult to understand.
Utilizing IDs to Accelerate Incremental View Maintenance	Proposes a new algorithm for incremental view maintenance that uses IDs to improve performance.	Can be complex to implement and may not be suitable for all cases.
DBSP: Automatic Incremental View Maintenance for Rich Query Languages	Proposes a new system for incremental view maintenance that can handle a wider range of query languages than traditional systems.	Can be complex to implement and may not be suitable for all cases.
Bridging the gap between Incremental View Maintenance and Query Plan	Proposes a new approach to incremental view maintenance that is more tightly integrated with the query planner.	Can be complex to implement and may not be suitable for all cases.
Efficient Incrementialization of Correlated Nested Aggregate Queries using Relative Partial Aggregate Indexes (RPAI)	Proposes a new algorithm for incremental view maintenance that is more efficient for correlated nested aggregate queries.	Can be complex to implement and may not be suitable for all cases.
DBToaster: Higher-order Delta Processing for Dynamic, Frequently Fresh Views	Proposes a new system for incremental view maintenance that can handle a wider range of view definitions and update patterns than traditional systems.	Can be complex to implement and may not be suitable for all cases.
Multi-View Class Incremental Learning	Proposes a new approach to incremental view maintenance for machine learning models that can handle multiple views and different types of data changes.	Can be complex to implement and may not be suitable for all cases.
Incremental maintenance of materialized views with outerjoins	Proposes a new algorithm for incremental view maintenance that can handle materialized views with outerjoins.	Can be complex to implement and may not be suitable for all cases.

Incremental join view maintenance on distributed log-structured storage	Proposes a new algorithm for incremental view maintenance for join views on distributed log-structured storage (LSM-tree).	Can be complex to implement and may not be suitable for all cases.
Foreign Keys Open the Door for Faster Incremental View Maintenance	Proposes a new algorithm for incremental view maintenance that leverages foreign key constraints to improve performance.	Can be complex to implement and may not be suitable for all cases.
LINVIEW: Incremental View Maintenance for Complex Analytical Queries	Proposes a new algorithm for incremental view maintenance for complex analytical queries.	Can be complex to implement and may not be suitable for all cases.
Intermittent Query Processing	Proposes a new query processing paradigm for intermittent data.	Can be complex to implement and may not be suitable for all cases.
Opportunistic View Materialization with Deep Reinforcement Learning	Proposes a new approach to incremental view maintenance that uses deep reinforcement learning to optimize the view materialization process.	Can be complex to implement and train the deep reinforcement learning agent.
Maintaining Views Incrementally	A comprehensive overview of incremental	

https://link.springer.com/article/10.1007/s00778-023-00785-1
https://github.com/alibaba/cost-based-incremental-optimizer
https://arxiv.org/abs/2306.09675
https://www.scienceopen.com/document?vid=9f9e5daf-de32-4f3c-8964-2bb632c81129
https://www.scienceopen.com/document?vid=a52739f2-ebd4-4620-ac05-858eb42bc029
https://www.cs.ox.ac.uk/dan.olteanu/papers/no-sigmod18.pdf
https://www.scattered-thoughts.net/writing/materialize-decorrelation/
https://www.infoworld.com/article/3697752/postgresql-16-advances-query-parallelism.html
https://dl.acm.org/doi/pdf/10.1145/3448016.3457551
https://odin.cse.buffalo.edu/research/index.html
https://arxiv.org/pdf/2203.16684.pdf
https://cseweb.ucsd.edu//~ikatsis/publications/sigmod15.pdf
https://qiyanghe1998.github.io/static/files/ivm-report.pdf
https://dl.acm.org/doi/abs/10.1145/3514221.3517889
https://www.cs.ox.ac.uk/files/9137/vldbj2014-dbtoaster-extended.pdf
https://dbtoaster.github.io/docs_architecture.html
https://arxiv.org/pdf/2303.08583.pdf
https://arxiv.org/pdf/2306.09675.pdf

https://dl-acm-org.libproxy1.usc.edu/doi/abs/10.14778/3587136.3587137
DBSP: Automatic Incremental View Maintenance for Rich Query Languages

https://dl-acm-org.libproxy1.usc.edu/doi/10.1016/j.is.2011.06.001
Incremental maintenance of materialized views with outerjoins

https://link-springer-com.libproxy1.usc.edu/article/10.1007/s11704-020-9310-y
Incremental join view maintenance on distributed log-structured storage

Foreign Keys Open the Door for Faster Incremental View Maintenance
https://dl-acm-org.libproxy2.usc.edu/doi/abs/10.1145/3588720

Incremental view maintenance (IVM) is a technique for keeping materialized views consistent with the underlying data when the data changes. IVM systems can be classified into the following categories:

Delta-based IVM systems track the changes to the base tables since the last time the materialized view was refreshed and use this information to update the materialized view.

Version-based IVM systems store the history of changes to the base tables and use this information to update the materialized view to the latest state.

Change-propagation-based IVM systems use a mathematical model to track the dependencies between the materialized view and the base tables. This allows the materialized view to be updated efficiently even if the base tables are complex.

Deep reinforcement learning (DRL)-based IVM systems use DRL to learn a policy for when to materialize views.

Hybrid IVM systems combine the advantages of different IVM methods.

—--

Intermittent query processing (IQP) systems are designed to handle data that arrives in an intermittent, yet largely predictable, pattern. IQP bridges the gap between query execution and policies to determine when to update results and how much resources to allocate for ensuring fast query updates.

Opportunistic view materialization (OVM) systems use machine learning to identify which materialized views are likely to be used in the future and pre-materialize them.