NOV 12, 2024

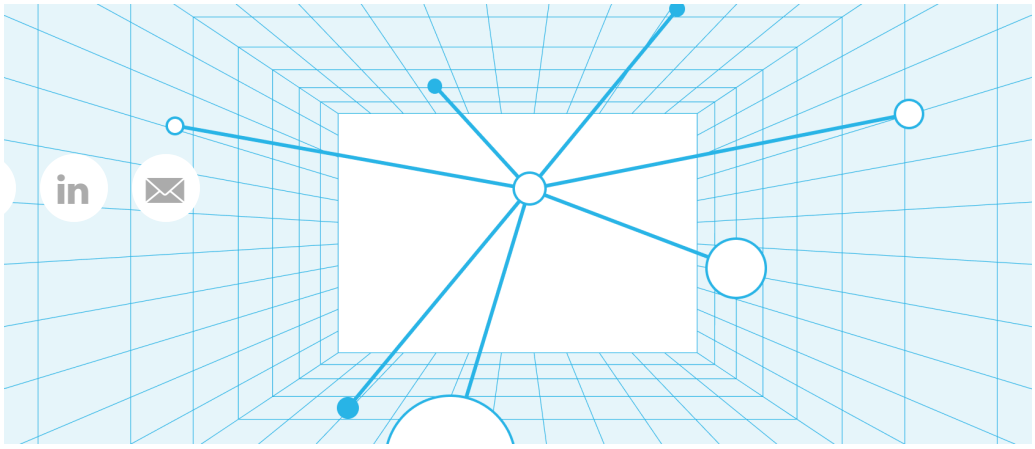
AUTHOR

Yahia Bsat

SHARE

Snowflake Cortex Analyst: Introducing Joins for Star and Snowflake Schemas

Gen AI



Building a reliable text-to-SQL product is a challenging journey, and it becomes even more intricate when introducing the complexity of joins across multiple tables. Joins are foundational to data analysis, allowing us to combine different perspectives on related data, but they also carry inherent challenges. From maintaining data integrity to navigating join paths and mitigating the risks of data loss or double counting, crafting a trustworthy join system is no small feat.

We're thrilled to announce that **Snowflake Cortex Analyst** now supports joins for star and snowflake schemas, and this new functionality is in public preview. This enhancement makes it possible for business users to ask questions that rely on data scattered across multiple tables. Based on our benchmarks, Cortex Analyst achieves an impressive 92% SQL accuracy in generating reliable answers for scenarios involving joins of standard star and snowflake schemas — significantly surpassing other text-to-SQL solutions, which often fall short of this level of precision (with competing products averaging around 73% accuracy, and GPT-4o in single-shot mode at 55%).

AUTHOR

Yahia Bsat

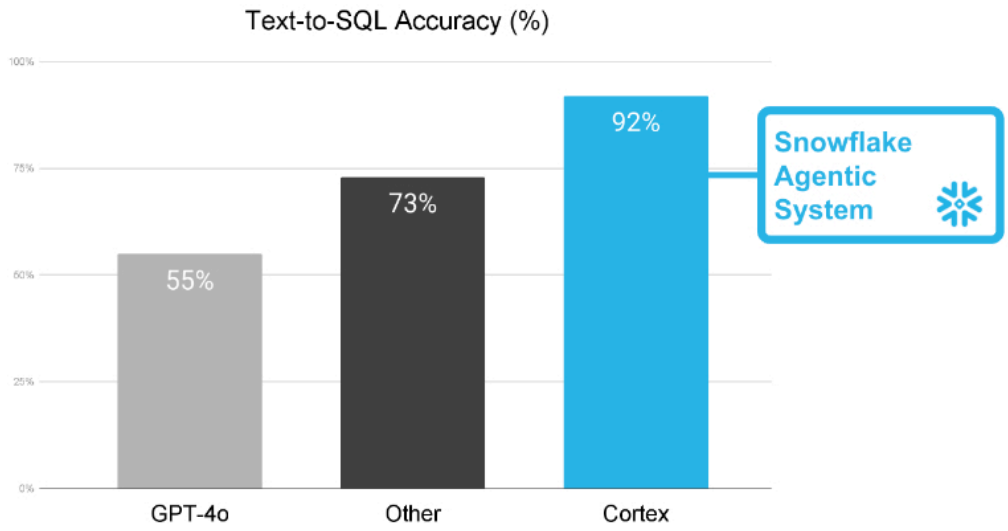


Figure 1. SQL generation accuracy on joins for Snowflake Cortex Analyst vs. alternatives

In this post, we'll explore the intricacies of supporting joins in Cortex Analyst, the challenges introduced by different join types, and how we mit[illegible]e c[illegible]mon pitfalls to deliver more reliable results.

Defining star and snowflake schemas

Before diving into the complexity of joins, it's helpful to clarify what we mean by star and snowflake schemas. These are common database structures designed to support complex analytical queries efficiently.

In a star schema, there is a central "fact table" that contains measures like sales or revenue, surrounded by "dimension tables" that provide context for those facts, such as products, dates or customers.

A snowflake schema is an extension of the star schema, allowing for additional normalization by breaking down dimension tables into more granular tables. This means snowflake schemas can include multiple hops of dimension tables, providing an even-more-normalized representation of data. However, unlike fact tables, dimension tables typically do not contain measures, which keeps the focus on the attributes describing entities.

The following diagram (Fig. 2) illustrates these relationships and helps visualize the core difference between star and snowflake schemas, showing how facts are linked to multiple levels of dimensional data.

AUTHOR

Yahia Bsat

Figure 2. Examples of star and snowflake schemas

Common problems with joins in text-to-SQL

'm er joins isn't just about linking tables together; it requires ensuring that all the nuances of database operations are accounted for. Here, we break down the most common challenges that arise when working with joins and how Cortex Analyst handles them to provide accurate and reliable SQL generation.

For this section, we will use the following schema (Fig. 3) to provide specific SQL examples.

Figure 3. Snowflake schema example

For an overview of our benchmarking methodology, refer to this blog post.

1. Join paths and key integrity

One of the fundamental aspects of joining tables is determining the correct join path. This means joining tables using the appropriate primary/foreign key relationships, which is critical for maintaining data integrity. Mistakes often occur when attempting to join on columns that seemingly align based on their names but don't actually reflect the correct relationship — for instance, joining based on dimension names instead of keys.

AUTHOR

Yahia Bsat

Moreover, SQL allows for joins to be expressed in multiple ways, such as through USING clauses or ON clauses with table order variations. Ensuring consistency here is challenging, as a mix-up could lead to incorrect joins, compromising the results. Cortex Analyst mitigates this by carefully parsing the generated SQL by the LLM, canonicalizing the join path used to a standard format and verifying the correct join keys are used.

Let's look at an example:

Question: What is the total number of orders by customer type?

Incorrect SQL: Joining with customer_name gets rejected by Cortex Analyst.

```
SELECT customers.customer_type,
COUNT(*) as order_count
FROM orders
LEFT JOIN customers ON orders.customer_name =
customers.customer_name
GROUP BY customers.customer_type
```

Correct SQL: Joining with the correct key customer_id.

```
SELECT customers.customer_type,
COUNT(*) as order_count
FROM orders
LEFT JOIN customers ON orders.customer_id =
```

```
customers.customer_id
GROUP BY customers.customer_type
```

2. Challenges in join configuration: Ensuring data completeness and consistency

AUTHOR

Yahia Bsat

Configuring joins correctly is critical to maintaining data completeness and consistency when generating SQL queries. Both the type of join (such as inner or left) and the sequence of joins can significantly impact the accuracy of the data returned, especially in cases where the underlying data relationships are complex or inconsistent.

Using an incorrect join setup can easily lead to undercounting, data loss or overinflated results. For example, an INNER JOIN will exclude rows without a match in both tables, potentially omitting relevant data, while a LEFT JOIN might retain all records from one table even when data is ni in ir other. Additionally, the order of joins matters: OrderItems LEFT JOIN Orders may produce different results than Orders LEFT JOIN OrderItems, especially if there are rows with no corresponding matches.

In environments like Snowflake, where referential integrity isn't enforced, this is even more crucial. Without careful attention to both join type and order, inconsistencies, such as undercounting or data mismatches, can arise. To address these challenges, Cortex Analyst incorporates checks to validate that the generated SQL aligns with the intended table relationships and sequence defined in the semantic model. If a discrepancy is found, Cortex Analyst automatically adjusts the query to ensure consistency and accuracy.

Let's look at an example:

Question: What is the average number of items per completed order across different locations?

Incorrect SQL: The average calculation includes non-completed orders that do not yet have any order items.

```
WITH total_items_per_order AS (
  SELECT
    o.order_id,
    o.location_id,
    COUNT(oi.order_item_id) AS item_count
  FROM orders AS o
  LEFT JOIN order_items AS oi
```

```
    ON o.order_id = oi.order_id
  GROUP BY
    o.order_id,
    o.location_id
)
SELECT
  l.location_name,
  AVG(item_count) AS avg_items_per_order
FROM total_items_per_order AS t
LEFT OUTER JOIN locations AS l
  ON t.location_id = l.location_id
GROUP BY
  l.location_name
```

AUTHOR

Yahia Bsat

Correct SQL: The average calculation only includes completed orders which have at least one order item.


```
WITH total_items_per_order AS (
  SELECT
    o.order_id,
    o.location_id,
    COUNT(oi.order_item_id) AS item_count
  FROM order_items as oi
  LEFT OUTER JOIN orders as o
    ON o.order_id = oi.order_id
  GROUP BY
    o.order_id,
    o.location_id
)
SELECT
  l.location_name,
  AVG(item_count) AS avg_items_per_order
FROM total_items_per_order AS t
LEFT OUTER JOIN locations AS l
  ON t.location_id = l.location_id
GROUP BY
  l.location_name
```

3. Ambiguous columns: Preventing compilation issues

A frequent issue arises from ambiguous column names when multiple tables are joined. Columns with the same names across different tables can lead to compilation errors if not properly qualified. Snowflake's syntax requires fully qualified column names to resolve these ambiguities; otherwise, the query will fail to compile. From our experiments on hundreds of sample queries, simply prompting the model to qualify all columns is not reliable by itself.

AUTHOR

Yahia Bsat

To handle this, Cortex Analyst includes a "join disambiguator" step that carefully parses the SQL query to ensure each column is fully qualified. Such a disambiguation step requires recursive processing of the SQL query, updating the schema used for validation iteratively after each common table expression (CTE). By explicitly indicating which table each column belongs to, we prevent compilation errors and help comply with Snowflake's SQL syntax requirements.

Let's look at an example:

Question: What are the top products by sales amount?

Incorrect SQL: SQL compilation error due to product_id being ambiguous.

```
SELECT product_id,
       product_name,
SUM(net_amount) as total_sales_amount
FROM orders
LEFT JOIN products ON product_id = product_id
GROUP BY product_id
ORDER BY total_sales_amount DESC
```

Correct SQL: Cortex Analyst disambiguates columns generated by the LLM.

```
SELECT products.product_id,
       product_name,
SUM(net_amount) as total_sales_amount
FROM orders
LEFT JOIN products ON orders.product_id =
products.product_id
GROUP BY products.product_id
ORDER BY total_sales_amount DESC
```

4. Avoiding double counting: Aggregation across tables

When aggregating metrics across different granularities, there's always a risk of double counting. This risk is particularly high when combining data in multi-hop fashion from different tables that have a many-to-one relationship. For example, calculating order net amount (from the Order Items table) while also aggregating total tax amount (from the Orders table) can inadvertently inflate results if the granularity differences between Order Items and Orders are not handled correctly.

AUTHOR

Yahia Bsat

To avoid such scenarios, Cortex Analyst adds validation to reject queries that might result in double counting. For now, we take a conservative approach: If measures from dimension tables are being joined to a fact table, these queries are flagged and rejected. This strict rule is designed to prevent inaccurate aggregations. This is why we currently support only standard star and snowflake schemas, and not more degenerate schemas.

However, as we evolve the product, we plan to allow defining metrics across tables and will soon share more on how we'll ensure no double counting occurs in complex join scenarios.

Let's look at an example:

Question: What is the total gross amount by customer?

Incorrect SQL: Double counting of tax amount per order given it joins to order items. Cortex Analyst rejects the query and does error correction.

```
SELECT customers.customer_id,
        customers.customer_name,
SUM(order_items.net_amount) +
SUM(orders.tax_amount) as total_gross_amount
FROM order_items
LEFT JOIN orders ON order_items.order_id =
orders.order_id
LEFT JOIN customers ON orders.customer_id =
customers.customer_id
GROUP BY customers.customer_id
```

Correct SQL: Tax amount is summed only once per order.

```
WITH net_amount_by_customer as (
SELECT customers.customer_id,
```

AUTHOR

Yahia Bsat


```
SUM(order_items.net_amount) as total_net_amount
FROM order_items
LEFT JOIN orders ON order_items.order_id =
orders.order_id
LEFT JOIN customers ON orders.customer_id =
customers.customer_id
GROUP BY customers.customer_id
), tax_amount_by_customer as (
SELECT customers.customer_id,
SUM(orders.tax_amount) as total_tax_amount
FROM orders
LEFT JOIN customers ON orders.customer_id =
customers.customer_id
GROUP BY customers.customer_id
)
SELECT customers.customer_id,
       customers.customer_name,
total_net_amount + total_tax_amount as
total_gross_amount
FROM net_amount_by_customer
JOIN tax_amount_by_customer USING (customer_id)
LEFT JOIN customers USING (customer_id)
GROUP BY customers.customer_id
```

5. Derived entities: Handling distinctness in joins

Working with derived entities — such as calculating metrics on unique customer locations — requires careful consideration when using joins. For instance, if a customer appears multiple times across different orders, and the analysis requires unique customer-level metrics, the system needs to extract distinct records before aggregating.

Out-of-the-box language models often struggle with these scenarios, returning inaccurate results. Cortex Analyst tackles this by following clear instructions for when to extract unique records before performing calculations, ensuring reliable outputs that accurately capture distinct entities in the data.

Let's look at an example:

Question: Compare the % of female customers in different locations.

Incorrect SQL: Customers with multiple orders from the same location are counted multiple times.

```
SELECT
  l.location_id,
  l.location_name,
  (
    COUNT(CASE WHEN c.customer_gender = 'Female'
THEN 1 END) / COUNT(c.customer_id) *   100.0
  ) AS female_customer_percentage
FROM orders AS o
LEFT JOIN customers AS c
  ON o.customer_id = c.customer_id
LEFT JOIN locations AS l
  ON o.location_id = l.location_id
GROUP BY
  l.location,
  l.location_name
```

AUTHOR

Yahia Bsat

Correct SQL: Customers are counted once per location.

```
WITH unique_customers_locations as (
  SELECT DISTINCT
     orders.customer_id,
     orders.location_id
  FROM orders
)
SELECT
  l.location_id,
  l.location_name,
  (
    COUNT(CASE WHEN c.customer_gender = 'Female'
THEN 1 END) / COUNT(c.customer_id) *   100.0
  ) AS female_customer_percentage
FROM unique_customers_locations AS ucl
LEFT JOIN customers AS c
  ON ucl.customer_id = c.customer_id
LEFT JOIN locations AS l
  ON ucl.location_id = l.location_id
GROUP BY
  l.location,
  l.location_name
```

6. Unsupported subqueries with CTEs

Subqueries that reference common table expressions can simplify complex SQL queries, but they are not always supported by Snowflake when derived from CTEs. Specifically, Snowflake requires that subqueries use base tables as their source, rather than CTEs, which can lead to compilation errors if not adhered to.

AUTHOR

Yahia Bsat

For instance, attempting the following subquery will fail in Snowflake because it uses a CTE as its source rather than a base table:

```
WITH battery_products AS (
  SELECT product_id
  FROM products
  WHERE DESCRIPTION ILIKE '%BATTERY%'
)
SELECT order_id, net_amount
FROM orders
LEFT JOIN (SELECT product_id FROM battery_products)
USING (id);
```

LLMs often generate such subqueries, as some other syntaxes do not enforce such constraints, resulting in invalid SQL that cannot compile. Cortex Analyst addresses this by automatically transforming such subquery suggestions into valid SQL that aligns with Snowflake's specific syntax requirements. This includes helping ensure that subqueries are rewritten as CTEs, thereby avoiding compilation errors and enabling successful query execution.

Conclusion

Supporting joins in a text-to-SQL system is far more involved than simply joining two tables together; without careful handling, inaccurate results can easily arise. It requires thoughtful handling of join paths, join types, disambiguation, aggregation and distinctness, along with careful parsing to generate SQL that accurately reflects the user's question.

With Snowflake Cortex Analyst, joins for star and snowflake schemas, now available in public preview, achieve greater than 90% SQL accuracy. While this milestone already delivers substantial capabilities for business users, we are not stopping here. Soon, we will expand to support more complex schemas, including metrics across tables, and we'll share, in an

upcoming blog post, how we're working to prevent double counting when dealing with even-more-advanced join scenarios.

We invite you to try all the new Cortex Analyst features, including JOIN capabilities, and share your feedback. Here are some additional valuable resources to help you get started:

AUTHOR

Yahia Bsat

BUILD Cortex Analyst Announcements: Check out our roundup blog for the latest feature updates.

Quickstart Guide: Use our semantic model generator tooling to create semantic models for Cortex Analyst.

GitHub Samples Repository: Discover inspiring examples on how to put Cortex Analyst to use.

Third-Party Semantic Layers: Learn how to translate existing third-party semantic layers for use with Cortex Analyst.

Stay tuned as we continue to enhance support for enterprise-grade, AI-driven self-serve analytics!

RELATED CONTENT

AUG 14, 2024

Snowflake Cortex Analyst: Behind the Scenes

AUG 29, 2024

Snowflake Cortex Analyst: Evaluating Text-to-SQL

AUG 08, 2024

Snowflake Cortex Search: High-Quality, Performant

Building a conversational self-service analytics product for business users is a complex and challenging endeavor. Trust and accuracy have to be at the core of such a product, as business...

Delve into the details

Accuracy for Real-World Business Intelligence Scenarios

Being able to build a system that allows business users to ask intricate, complex and...

Explore

Search and Retrieval for Enterprise AI

Search and retrieval systems have always been a critical backbone for knowledge management in enterprises....

Here's How

Yahia Bsat

SHARE

Sign up for Snowflake Communications

diana.shaw@snow United States

By submitting this form, I understand Snowflake will process my personal information in accordance with their Privacy Notice. Additionally, I consent to my information being shared with Event Partners in accordance with Snowflake's Event Privacy Notice. I understand I may withdraw my consent or update my preferences here at any time.

AUTHOR

SUBSCRIBE NOW

Yahia Bsat

SHARE