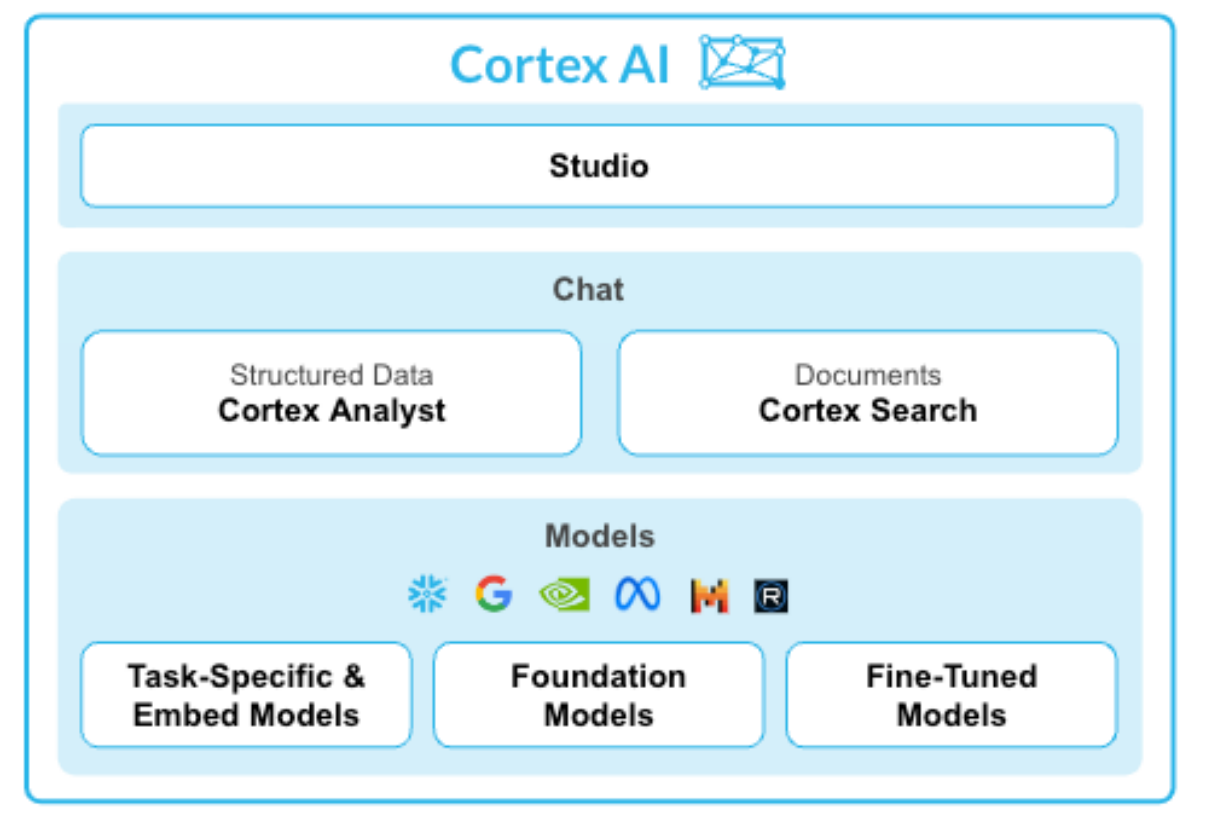
CORTEX AI

Cómo usar la Inteligencia Artificial de Snowflake para Analytics



1. INTRODUCCIÓN

Cortex Analyst es una herramienta de Snowflake construida a partir de dos grandes modelos de lengua (LLMs): LLaMa de Meta y Mistral. A través de este servicio se puede interactuar con los datos ya introducidos en Snowflake. Cortex Analyst crea una interfaz amigable donde el lenguaje natural se procesa y traduce a con una precisión a texto SQL. De esta forma se facilita el desarrollo de aplicaciones conversacionales donde los usuarios de negocio puedes interactuar con el asistente de IA haciéndole preguntas sobre sus datos y recibiendo una respuesta en tiempo real.

Para conseguir una precisión elevada de texto a SQL, Cortex Analyst está basado en agentes que se encargan de diferentes tareas dentro del modelo. A su vez, este se presenta como una REST API que hace de puente entre el usuario y el modelo y puede ser integrada en diferentes aplicaciones como Teams, Streamlit u otras interfaces de chat.

Principios clave

Cortex Analyst sigue unos principios esenciales con los que tratan de lograr una precisión elevada y fiable:

1. **Recopilación de la semántica:** el modelo necesita contexto para comprender mejor la pregunta que se le está haciendo y ser capaz de devolver una respuesta de calidad. Es esencial que el modelo conozca el vocabulario y jerga específica del negocio para ello.

2. **Rechazar preguntas sin respuesta y sugerir alternativas:** el asistente de IA es capaz de reconocer preguntar que no pueden ser contestadas o ambiguas dados los datos disponibles. En dicho caso, produce nuevas preguntas que pueden ser útiles para el usuario en vez de generar una respuesta con resultados erróneos.

3. **Mejora del modelo:** desde Snowflake se hacen cargo de la evolución de su LLM y se comprometen a seguir explorando los avances tecnológicos y añadir nuevos modelos emergentes en su *mix* para impulsar en un futuro el rendimiento y precisión

Referencias

Cortex Analyst: Paving the Way to Self-Service Analytics with AI

Cortex Analyst | Snowflake Documentation

Cortex Analyst semantic model specification | Snowflake Documentation

Cortex Analyst Verified Query Repository | Snowflake Documentation

CREATE VIEW | Snowflake Documentation

Overview of Views | Snowflake Documentation

Understanding compute cost | Snowflake Documentation

2. ARQUITECTURA DE CORTEX ANALYST

El flujo de trabajo que se sigue es relativamente sencillo e incluye interacciones con varios agentes de LLM y puntos de control en cada paso para evitar que el modelo sufra *alucinaciones* y pueda devolver respuestas de calidad.

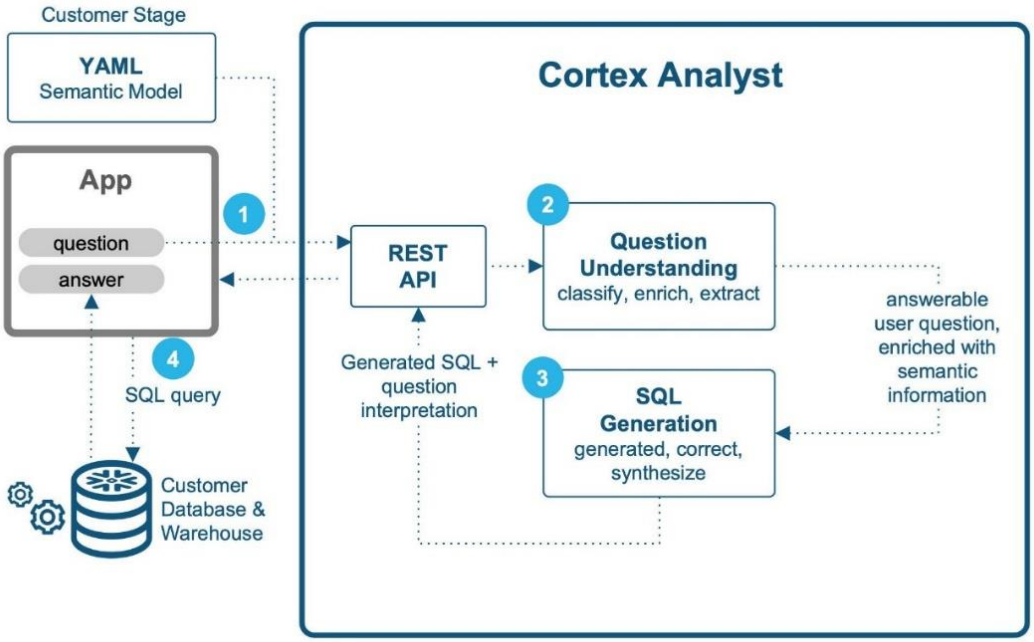
A grandes rasgos los pasos que sigue para transformar texto en SQL son los siguientes:

1. Dentro de la interfaz de chat (App) el cliente manda la petición a la REST API de Cortex Analyst que incluye su pregunta y el modelo semántico con los metadatos relevantes.

2. Dentro de Cortex Analyst, se analiza y clasifica la pregunta según pueda o no ser respondida. En caso de no serlo, se proporcionarán preguntar alternativas que serán devueltas al cliente. En caso contrario, la pregunta se enriquecerá (se le dará contexto) mediante el uso del modelo semántico proporcionado previamente.

3. La pregunta enriquecida es entonces enviada a los diferentes agentes de IA que utilizan diferentes LLMs para generar posibles consultas SQL. Un agente adicional corrige los errores y las posibles *alucinaciones* cometidas en ellas.

4. Un último agente sintetizador es el encargado de generar la consulta SQL final, la cual responde a la pregunta inicial con mayor precisión. Esta es devuelta al usuario a través de la REST API y ejecutada en segundo plano en la App, mostrando así los resultados finales al cliente.



3. CASO DE USO

A continuación, se mostrarán lo pasos que se recomiendan seguir ilustrándolos con un caso de uso en el que se utilizaron los datos de producción de una empresa dedicada a la construcción.

Crear objetos en Snowflake

Antes de poder hacer uso de Cortex Analyst es necesario tener todas las tablas que se van a utilizar en Snowflake, además de cargar sus datos para poder ejecutar las consultas SQL posteriormente.

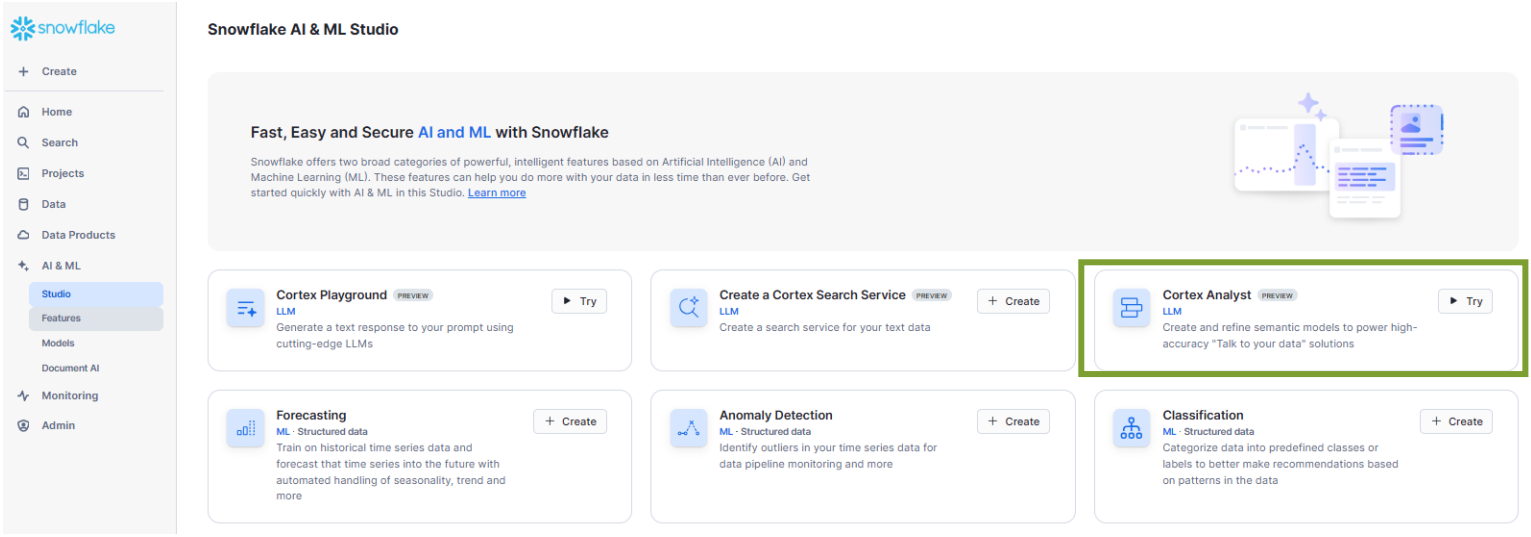
Estas tablas han de estar en un **Schema**, que a su vez se encuentra en la base de datos del usuario. Además, se debe crear un **Stage** que es donde se guardará posteriormente el modelo semántico que se cree.

En el apartado de **Projects** se pueden utilizar las **Worksheets** para crear estos objetivos y tener el entorno preparado.

Modelo Semántico

Como ya se ha mencionado anteriormente, se debe de crear un modelo semántico con metadatos del datamart que se va a utilizar. Esto es un archivo en formato YAML que contiene información sobre las tablas, sus columnas, las relaciones entre las dimensiones y la tabla de hechos y algunas consultas verificadas que se verán más adelante.

El modelo semántico se puede generar a través de herramienta de Cortex Analyst que se encuentra en **Snowflake AI & ML Studio**. Esta herramienta ofrece una interfaz amigable con la que se puede construir el modelo semántico de forma sencilla.

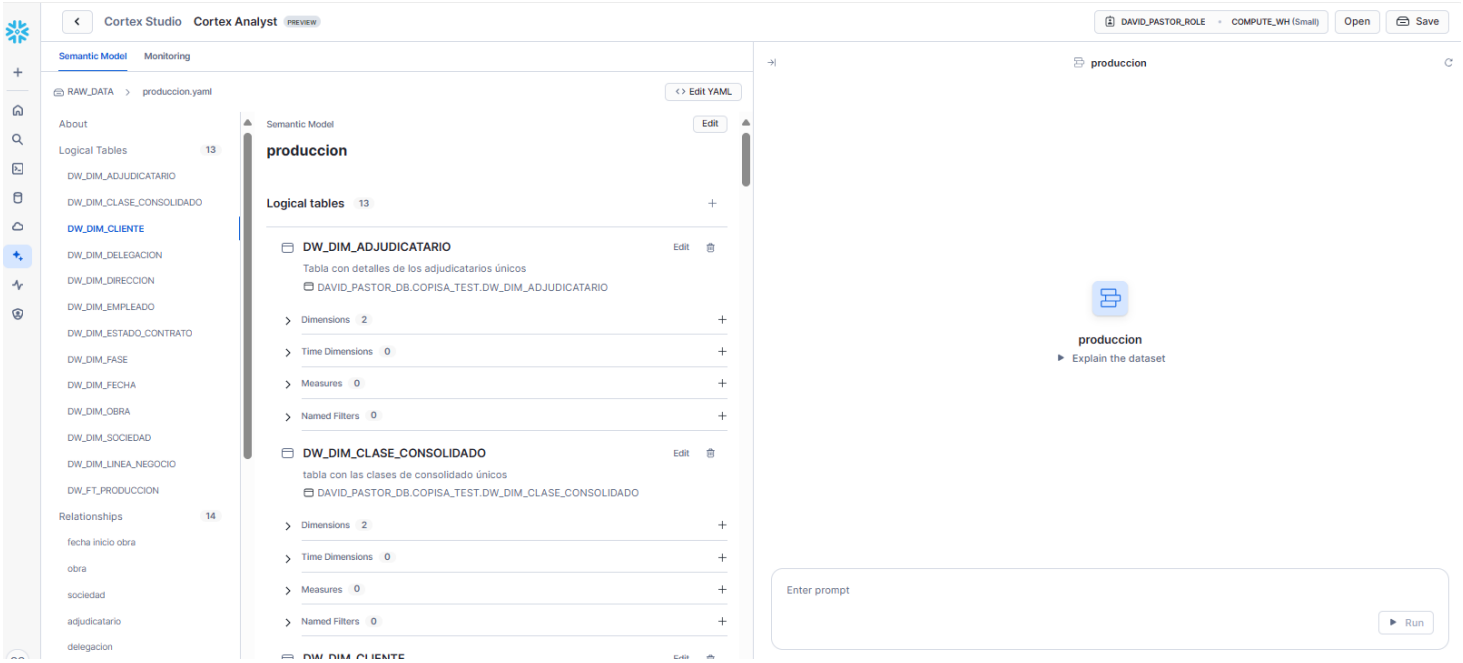


A continuación, se muestra la interfaz de la herramienta, en este caso de uso se incluyeron 13 tablas y 60 columnas en total (superando el límite recomendado). Estas tablas se ubican en el apartado **Logical Tables**.

Cada una de las tablas de dimensiones (DW_DIM_nombre) se relaciona con la tabla de hechos (DW_FT_nombre) al menos una vez. Estas relaciones se pueden especificar en el apartado **Relationships**.

El último apartado es el de las consultas verificadas (**Verified Quieries**). Aquí se almacena un repositorio de preguntas-respuestas que le modelo utilizará como entrenamiento y responder a nuevas preguntas similares correctamente.

En el panel del chat de la derecha se puede interactuar con el asistente de IA y hacerle preguntas. Su respuesta se puede editar y guarda como consulta verificada una vez sea correcta.



El panel central contiene un desglose de todas las columnas que seleccionamos de cada una de las tablas. En ellas hay que escribir detalladamente una descripción y sinónimos para cada columna.

Por defecto, Cortex Analyst añade las columnas como **Dimensions** si se trata de datos de tipo varchar, sin embargo, si los detecta como numéricos, se introducen como **Métrics**. En caso de ser incorrecto, como podría pasar con las columnas que son identificadore (ID_nombre_SK), estas deben ser redirigidas a **Dimensions**. También hay otro tipo que Snowflake categoriza como **Time Dimensions**, aquí se deben de añadir aquellas columnas que indiquen tiempo, ya sean de tipo date/timestamp/time o no.

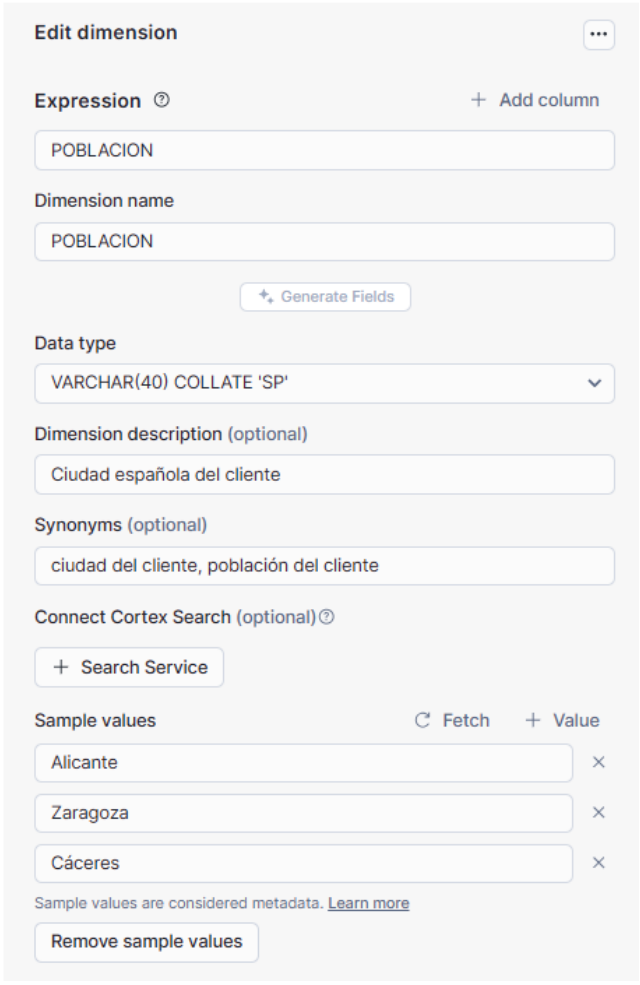
Para profundizar e ilustrar lo anterior, se hará un desglose de diferentes tablas del caso de uso.

Se utiliza la tabla **CLIENTES** y la columna **POBLACION**, para ejemplificar las **Dimensions**.

Al tratarse de una columna física de la tabla, **Expression** y **Dimension name** coinciden (más adelante veremos un caso en el que no). En **Data type** se especifica el tipo de dato que es. También se incluyen ejemplos de valores (**Sample Values**) que son útiles para que el modelo entienda el tipo de dato que incluye la columna. Estos campos se generan automáticamente, por lo que no habría que tocarlos al menos de quererse modificar.

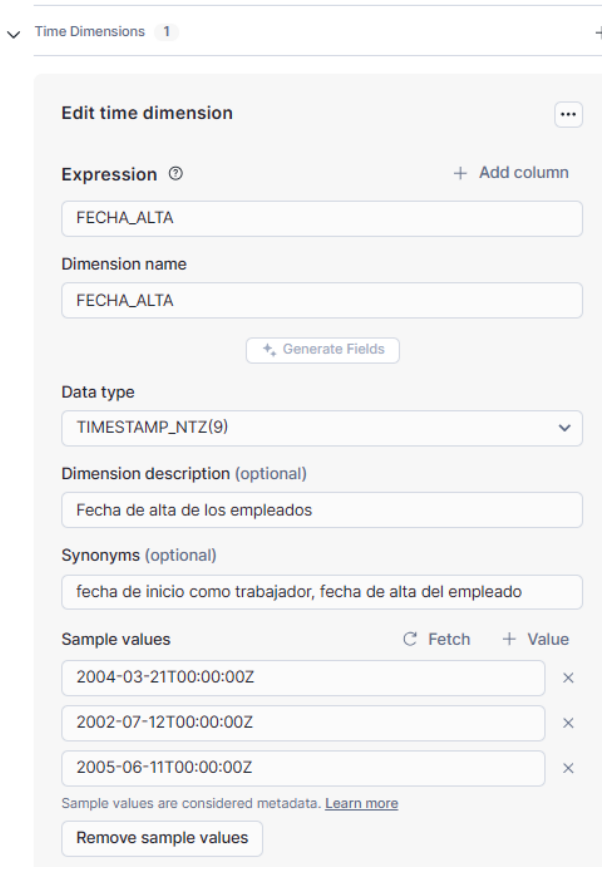
Los siguientes campos se rellenan a mano, y aunque ponga que es *opcional* son necesarios para un buen funcionamiento del modelo.

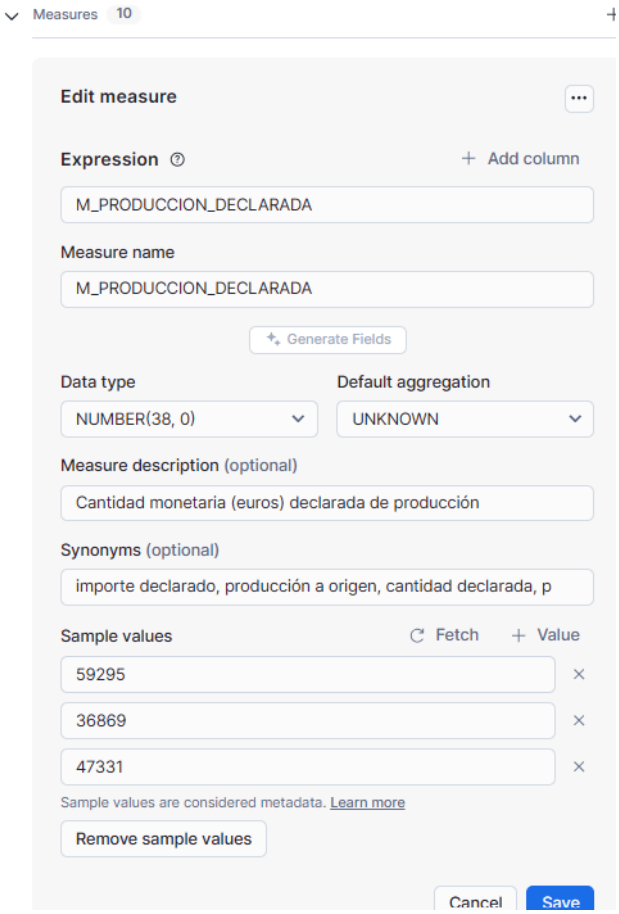
Es importante que la descripción de la columna (**Dimension Description**) se ajuste lo máximo posible a lo que representan los datos, así mismo, se han de poner tantos sinónimos (**Synonyms**) como sean posibles (pensando en las diferentes formas de hacer referencia a dicha columna). En este caso ambos campos son bastantes sencillos ya que esta columna almacena de qué ciudad son los clientes, en este caso como sinónimos se incluye **ciudad** y **población del cliente** ya que son dos formas distintas de hacer referencia a esta columna.


Edit dimension
Expression
Add column
POBLACION
Dimension name
POBLACION
Generate Fields
Data type
VARCHAR(40) COLLATE 'SP'
Dimension description (optional)
Ciudad española del cliente
Synonyms (optional)
ciudad del cliente, población del cliente
Connect Cortex Search (optional)
Search Service
Sample values
Fetch
Value
Alicante
Zaragoza
Cáceres
Sample values are considered metadata. Learn more
Remove sample values

Para ejemplificar las **Time Dimensions** se utiliza la tabla **EMPLEADO** y la columna **FECHA_ALTA**.

Esta columna indica la fecha en la que se dio de alta a los empleados, al tratarse de tiempo, se incluye como dimensión de tiempo. De esta forma se le da al modelo un contexto temporal con el que poder analizar hechos en determinados periodos de tiempo.





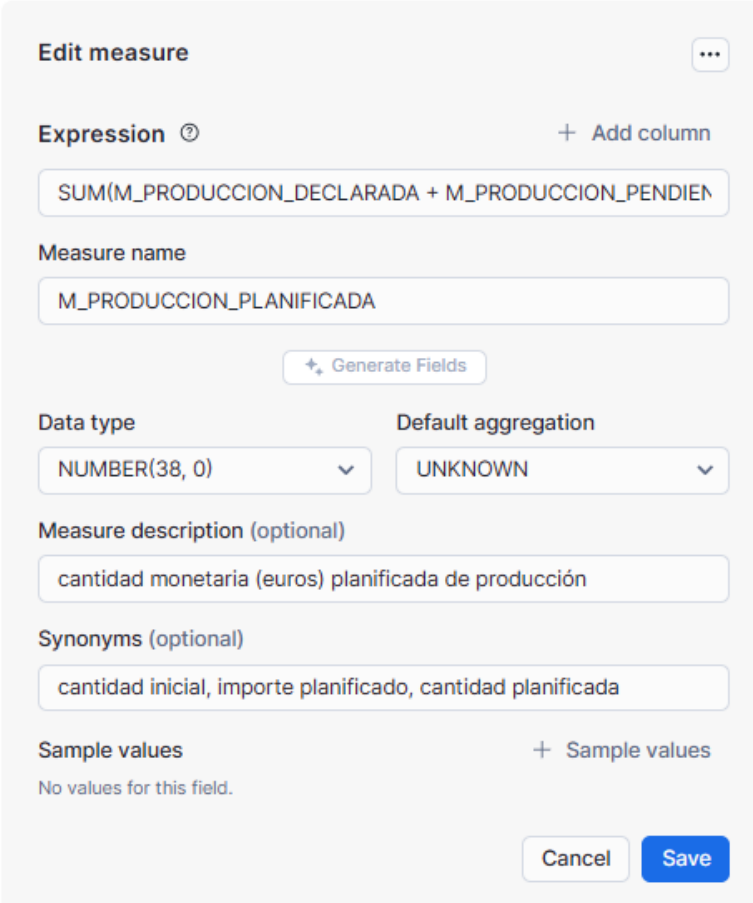
Finalmente, para ilustrar las **Metrics** se mostrarán dos columnas de la tabla de hechos **PRODUCCION**, **PRODUCCION_DECLARADA** y **PRODUCCION_PLANIFICADA**.

PRODUCCION_DECLARADA se trata de una métrica que indica la cantidad (en euros) de producción que se ha declarado. También se puede referir a ella como importe/cantidad/producción declarada y producción a origen.

Esta es una métrica sencilla que su expresión coincide con el nombre ya que es una columna física de esta tabla de hechos.

Por otro lado, **PRODUCCION_PLANIFICADA** es algo más compleja porque es la suma de otras dos columnas: **PRODUCCION_DECLARADA** y **PRODUCCION_PENDIENTE**.

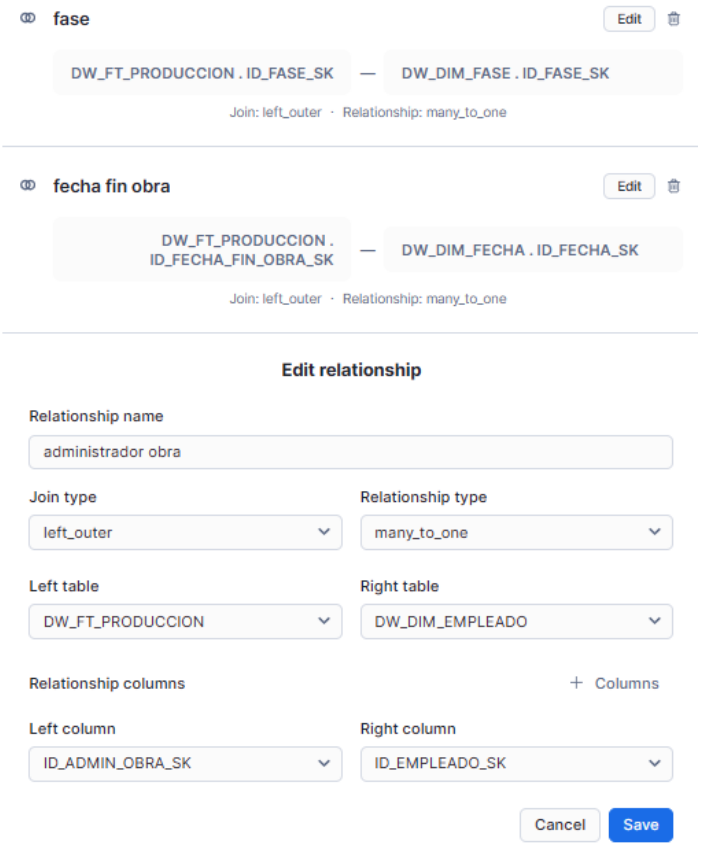
En este caso, la expresión ha de ser esta operación, ya que la columna en sí no es una física de la tabla de hechos.

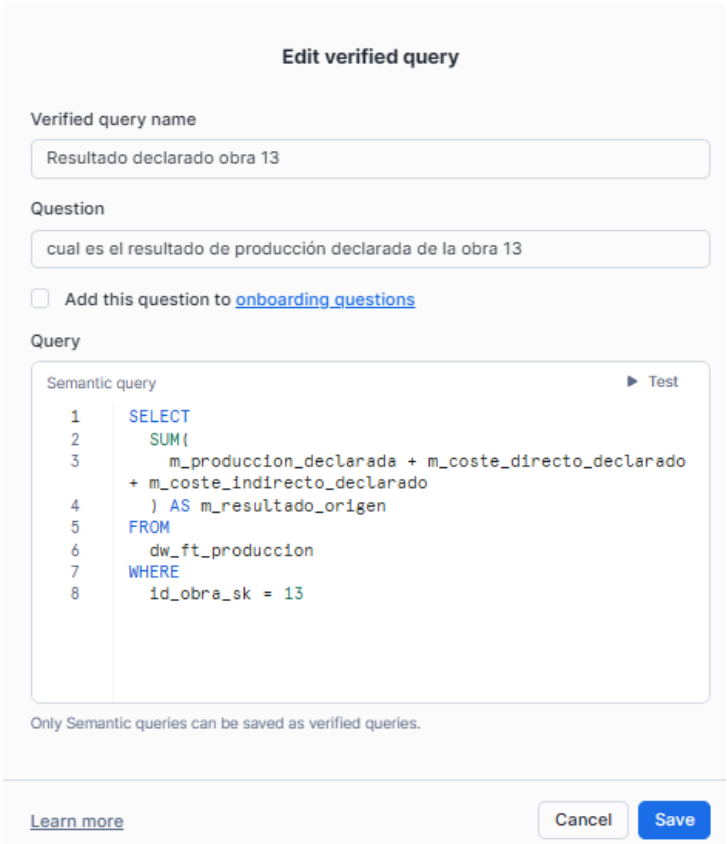


Para la especificación de las relaciones entre las tablas se puede hacer rellenando los campos como se ve a continuación. Como se mencionó previamente, las tablas se pueden relacionar más de una vez, como sucede en el caso de la dimensión **EMPLEADO** y los hechos **PRODUCCION**.

En esta imagen se puede observar relaciones ya establecidas, entre la dimensión **FASE** y **FECHA**, y la tabla de hechos; y los campos que se rellenan para establecer estar relaciones.

En este caso de uso, se utilizó la tabla de **EMPLEADO** para identificar tanto al jefe de la obra como al administrador. Como se menciona previamente, las dimensiones pueden relacionarse con la tabla de hecho más de una vez, sin embargo, a pesar de que la interfaz te permite añadir más de una relación en el mismo paso, se recomienda hacerlo por separado. En este caso de uso, se ha comprobado que cuando se añaden de forma conjunta el modelo comete más errores a la hora de realizar los JOINs en las consultas SQL.





Finalmente, se muestra un ejemplo de consulta verificada. En esta ventana emergente se pueden hacer modificaciones para corregir o simplificar la consulta SQL realizada por el modelo.

En los campos **Question** y **Query** se recoge la pregunta realizada por el usuario en el chat y la respuesta generada, respectivamente. Además, se puede testear la consulta ejecutándola (**Test**).

Cortex Analyst ofrece una interfaz visualmente atractiva y fácil de usar por casi cualquier usuario, sin embargo, cuando se trata con un elevado número de tablas y columnas estos desplegables pueden no ser la forma óptima de rellenar todos los campos relevantes.

Por ello, se recomienda utilizar la vista de YAML (**edit YAML**) e ir añadiendo los campos en la línea de código correspondiente. De esta forma se ahorra tiempo especificando todas las descripciones y sinónimos, así como estableciendo las relaciones. Por otro lado, las consultas verificadas si se recomienda hacerla con la vista del modelo semántico (**Semantic model viewer**) ya que hay que interactuar con el chat e ir haciendo modificaciones en paralelo con pruebas para corregir la sentencia SQL.

La vista de YAML sería la siguiente, mostrando la estructura que se sigue para detallar la tabla de dimensión **ADJUDICATARIO**.

Se puede ver cómo se ha detallado la **primary_key**, aspecto que en la interfaz anterior no se preguntaba y es esencial para que le modelo entienda cómo se realizan las relaciones.

```yaml
name: produccion
tables:
  - name: DW_DIM_ADJUDICATARIO
    description: Tabla con detalles de los adjudicatarios únicos
    base_table:
      database: DAVID_PASTOR_DB
      schema: COPISA_TEST
      table: DW_DIM_ADJUDICATARIO
    primary_key:
      columns:
        - ID_ADJUDICATARIO_SK
    dimensions:
      - name: NOMBRE
        expr: NOMBRE
        data_type: VARCHAR(16777216) COLLATE 'sp'
        sample_values:
          - Hermanos Ramírez S.L.N.E
          - Morena Delgado Salas S.L.
          - Banco Aramburu S.L.
        description: Nombre del adjudicatario
        synonyms:
          - adjudicatario
      - name: ID_ADJUDICATARIO_SK
        expr: ID_ADJUDICATARIO_SK
        data_type: NUMBER(38,0)
        sample_values:
          - '1'
          - '2'
          - '3'
        description: Identificador único para cada adjudicatario
        synonyms:
          - ID del adjudicatario
          - código del adjudicatario
          - número del adjudicatario
    synonyms:
      - dimensión adjudicatario
```

Un ejemplo de métricas sería el que se ve a continuación. Al igual que con las dimensiones, se describe la columna y se añaden posibles sinónimos.

```yaml
  measures:
    - name: M_PRODUCCION_DECLARADA
      expr: M_PRODUCCION_DECLARADA
      data_type: NUMBER(38, 0)
      sample_values:
        - '59295'
        - '36869'
        - '47331'
      description: Cantidad monetaria (euros) declarada de producción
      synonyms:
        - importe declarado
        - producción a origen
        - cantidad declarada
        - producción declarada
    - name: M_PRODUCCION_PENDIENTE
      expr: M_PRODUCCION_PENDIENTE
      data_type: NUMBER(38, 0)
      sample_values:
        - '85795'
        - '29338'
        - '56354'
      description: cantidad monetaria (euros) pendientes por producir
      synonyms:
        - importe pendiente
        - producción pendiente
        - cantidad pendiente
```

Por otro lado, las relaciones vendrían detalladas al final de la descripción de las tablas de la siguiente forma.

```yaml
relationships:
  - name: fecha inicio obra
    join_type: left_outer
    relationship_type: many_to_one
    left_table: DW_FT_PRODUCCION
    relationship_columns:
      - left_column: ID_FECHA_INICIO_OBRA_SK
        right_column: ID_FECHA_SK
    right_table: DW_DIM_FECHA
  - name: obra
    join_type: left_outer
    relationship_type: many_to_one
    left_table: DW_FT_PRODUCCION
    relationship_columns:
      - left_column: ID_OBRA_SK
        right_column: ID_OBRA_SK
    right_table: DW_DIM_OBRA
  - name: sociedad
    join_type: left_outer
    relationship_type: many_to_one
    left_table: DW_FT_PRODUCCION
    relationship_columns:
      - left_column: ID_SOCIEDAD_SK
        right_column: ID_SOCIEDAD_SK
    right_table: DW_DIM_SOCIEDAD
  - name: adjudicatario
    join_type: left_outer
    relationship_type: many_to_one
    left_table: DW_FT_PRODUCCION
    relationship_columns:
      - left_column: ID_ADJUDICATARIO_SK
        right_column: ID_ADJUDICATARIO_SK
    right_table: DW_DIM_ADJUDICATARIO
```

Por último, se incluyen las consultas verificadas, que como se recomendó previamente, estas es mejor añadirlas mediante la interfaz. Aunque se verían de la siguiente forma en el YAML.

```yaml
verified_queries:
  - name: Producción fase 12
    question: cual el la cantidad pendiente, por declarar y planificada de la fase 12
    use_as_onboarding_question: false
    sql: WITH phase_data AS (SELECT f.id_fase_sk, SUM(p.m_produccion_pendiente) AS pending_amount, SUM(p.m_produccion_declarada) AS declared_amount, SUM(p.m_produccion_declarada + p.m_produccion_pendiente) AS planned_amount FROM dw_ft_produccion AS p LEFT OUTER JOIN dw_dim_fase AS f ON p.id_fase_sk = f.id_fase_sk WHERE f.id_fase_sk = 12 GROUP BY f.id_fase_sk) SELECT pending_amount, declared_amount, planned_amount FROM phase_data
    verified_by: Claudia Carratalá
    verified_at: 1739788157
```

Vista Lógica del modelo completo

Como alternativa, se puede crear una vista lógica del modelo completo. Es decir, se crearía una tabla con todas las columnas de las tablas de dimensión y la de hechos. De este modo, no habría que preocuparse por las relaciones entre las tablas ya que se incluyen dentro de la vista.

Es importante crear una vista no materializada (**View**) ya que las materializadas conllevan un coste adicional.

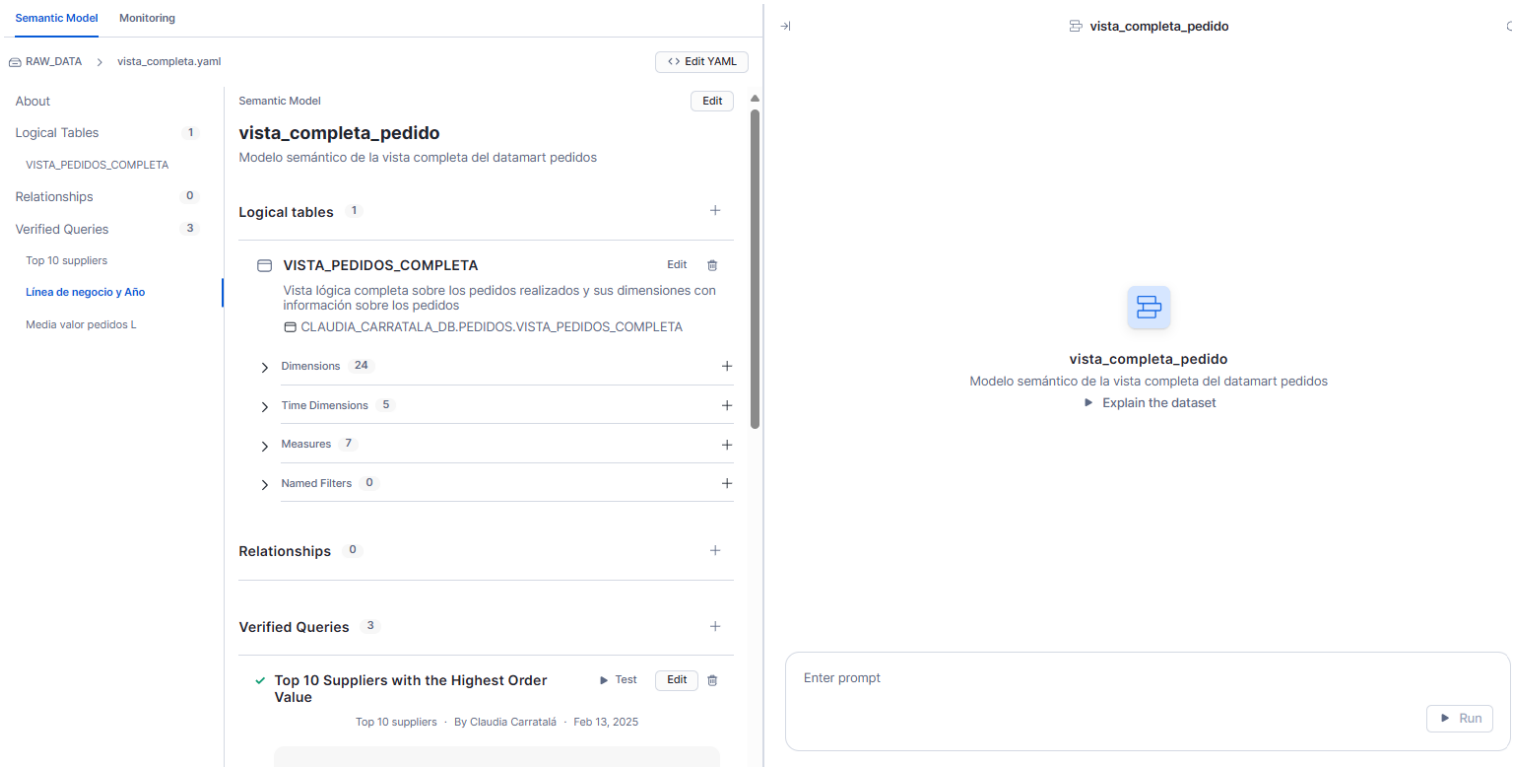
Para crear una vista se puede recurrir a las **Worksheets** y generar una secuencia SQL que cree la vista a partir de un SELECT de todas las tablas que son de interés, así como especificar los JOINS para recuperar ciertas columnas.

Para el caso de uso se creó una vista completa en base al modelo semántico creado previamente, se seleccionaron aquellas tablas y columnas introducidas y descritas en el archivo YAML.

A partir de esta vista se puede crear un modelo semántico siguiendo los mismos pasos que anteriormente. Este modelo será más sencillo ya que solo se selecciona una tabla (la vista completa). Al no tener que especificar las uniones entre tablas, hay columnas que no son necesarias, como las SK (con las que se realizan los JOINS), por lo que, a la hora de crear el modelo semántico, estas se pueden omitir si se cree necesario.

Al no tener que especificar las relaciones, la cantidad de tareas para realizar este archivo YAML es menor que utilizando el método anterior, aunque, sí sería importante introducir consultas verificadas para entrenar al modelo.

Así es cómo se vería la interfaz de Cortex para generar el modelo semántico a partir de una vista lógica del modelo completo.



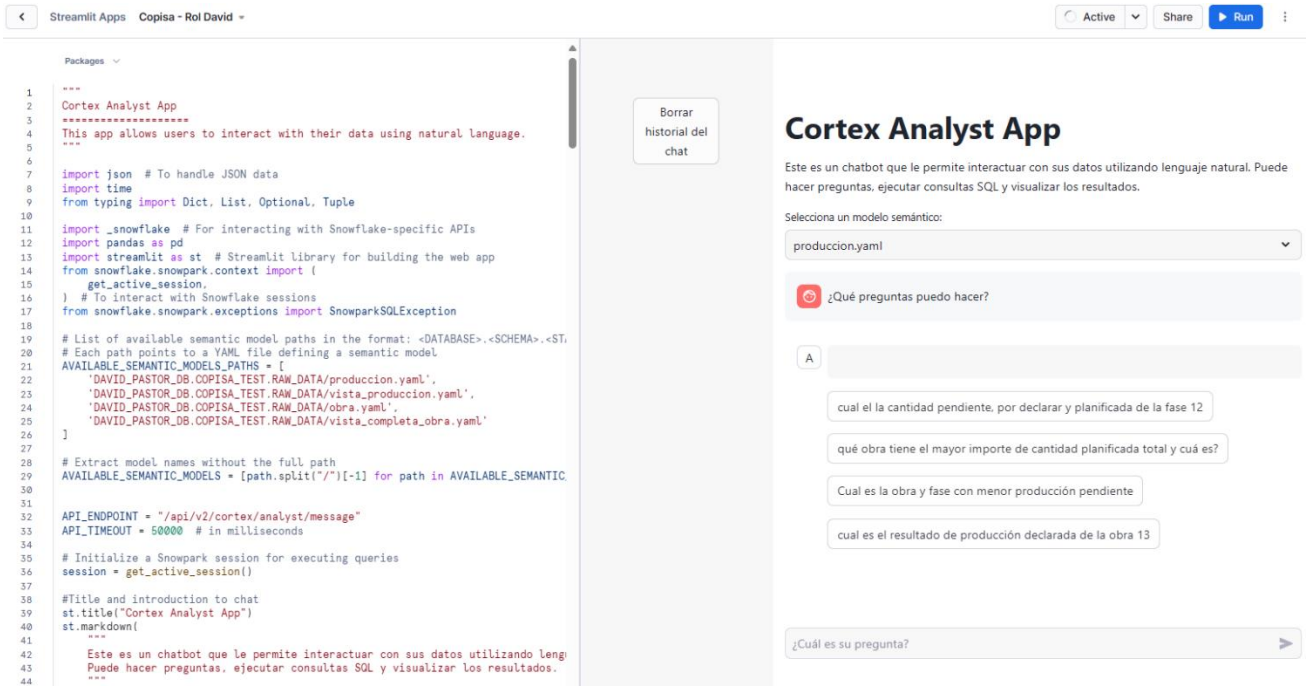
Streamlit App

En el apartado de **Projects** se puede crear una Streamlit App con la que el usuario puede interactuar de manera dinámica con Cortex Analyst.

Se ha de definir el título de la App, así como dónde se guardará y el warehouse que utilizará (con un tamaño *Small* es suficiente).

Una vez creado se puede introducir un script de Python personalizado para crear el chatbot. En este script se debe especificar los *paths* de los modelos semánticos (archivos YAML), estos aparecerán como un desplegable que el usuario puede elegir según quiera hacer preguntas sobre un datamart u otro.

En el panel de la derecha se visualiza el chat generado, donde se pueden lanzar preguntas y obtener repuestas del asistente de IA.



En este caso de uso solo se utilizarán los modelos semánticos relacionados con el datamart de producción: **produccion.yaml** y **vista_produccion.yaml**.

Si se van a usar varios modelos semánticos, se recomienda borrar el historial del chat cada vez que se cambie de archivo ya que el modelo puede confundirse y no responder correctamente.

4. RESULTADOS

El objetivo de este caso de uso es comparar las respuestas obtenidas por el LLM utilizando dos modelos semánticos construidos de forma diferente, pero a partir de los mismos datos. De esta forma se pretende evaluar la capacidad de precisión y rendimiento del modelo ante estas variantes.

Para ello se le preguntó al modelo un conjunto de preguntas, de las cuales a se tenía la consulta SQL correcta, y se compararon con las respuestas obtenidas a través de cada modelo semántico. Para que el modelo entendiera mejor la intención de la consulta, se adaptó la pregunta dándole algo más de detalle sobre lo que se quería obtener cómo resultado.

A continuación, se muestra una tabla con comentarios sobre cómo de precisa ha sido la consulta obtenida con el LLM utilizando los dos modelos semánticos, el construido a partir del modelo en estrella y a partir de la vista lógica.

Consulta	Pregunta realizada	SQL LLM – produccion.yaml Modelo en Estrella	SQL LLM – vista_produccion.yaml Vista Lógica
Diferencia producciones por año y delegaciones **Esta consulta compara la producción declarada con la planificada por año**	¿Cuál es la producción declarada y planificada por año de obra y delegaciones?	El modelo generar correctamente la consulta.	En este caso la consulta se genera mucho más rápido que con el modelo en estrella. Sin embargo, selecciona la fecha de inicio de la fase, y no la de la obra. Así como el ID de la obra. Para evitar esto, se reformula la pregunta para obtener los resultados similares a los esperados.
Resultado declarado vs. Planificado por Delegación. **Esta consulta analiza cuánto del resultado planificado se ha**	¿Cuánto del resultado planificado se ha declarado por delegación?	No comete errores aparte de que no ordena el resultado final por PERCENTAGE_DECLARED en orden descendente.	La consulta que se genera es más simple y similar a la esperada en comparación con el modelo estrella.

declarado en cada delegación.			
Análisis de producción por año y fase. **Selecciona las métricas de producción y las agrupa por año y fase.**	1) Hazme un análisis de la producción por año y fase 2) Hazme un resumen de todas las métricas de producción por año y fase 3) Dime la producción planificada, a origen y pendiente por año y fase 4) Dime la producción planificada, declarada y pendiente por año y fase	Con la primera pregunta no pudo generar ninguna respuesta, se prueba con alternativas. La pregunta 3) no se ejecuta correctamente ya que selecciona las métricas de RESULTADO y no las de PRODUCCION. Se plantea una nueva pregunta y se obtiene el mismo resultado, por lo que se decide borrar el historial y repetir la pregunta 4). Tras esto se obtiene una respuesta mucho más similar a lo esperado, pero con problemas en la fecha, además que selecciona la fecha relativa a las obras y no a la fase como se espera.	En este caso si se le pasa la primera pregunta al modelo, este es capaz de generar la consulta correctamente, sin necesidad de especificarle más campos.
Análisis de costes y resultado en base a adjudicatarios y fases. **Se espera obtener los costes directos e indirectos declarados, así como resultados declarados, pendientes y planificados.**	¿Cuáles son los costes directos e indirectos declarados y resultados de producción declarada, pendiente y planificada por adjudicatario y fase?	Esta pregunta no es capaz de comprenderla, no selecciona las métricas de RESULTADOS, si no que hace una suma de las métricas relacionadas con PRODUCCION. Selecciona correctamente el adjudicatario y las fases, así como los costes directos e indirectos declarados. Tras varios intentos y modificación de la pregunta, se logra que el modelo seleccione las columnas correctas.	Con el modelo semántico de la vista lógica del asistente de IA sí es capaz de generar una consulta similar a la esperada. Aunque, añade las fechas de inicio y fin de las fases, aunque no se pida.
Análisis de producción declarada, pendiente y	¿Cuál es la producción declarada, pendiente	Genera una consulta SQL correcta, devuelve un resultado como el esperado.	La consulta generada es más sencilla y parece realizarla correctamente, sin embrago los datos parecen diferir de

planificada por cliente y fase.	y planificada por cliente y fase?		los obtenidos con el modelo en estrella. Además, de nuevo, añade las fechas de inicio y fin de la fase a pesar de no especificarlas.
Análisis financiero en base a sociedades, obras y fases. **Se espera obtener los valores de producción declarada y planificada, así como los costes directos e indirectos planificados y los resultados planificados y pendientes**	¿Cuál es la producción declarada, planificada, los costes directos e indirectos planificados, los resultados planificados y pendientes en base a la sociedad, la obra y las fases?	Realiza la consulta de forma correcta conforme a la estructura de los datos. RESULTADOS_PENDIENTES equivale a la suma de PRODUCCION_PENDIENTE, por lo que en este caso de uso la consulta SQL generada es similar a la esperada.	AL igual que con el modelo en estrella la consulta se genera correctamente, sin embargo, se vuelve a incluir las fechas de inicio y fin de las fases.

5. CONCLUSIONES

Tras este caso de uso se llegan a diferentes conclusiones y advertencias que el usuario debería de tener en cuenta a la hora de utilizar Cortex Analyst.

El datamart utilizado en este caso de uso era complejo, con numerosas tablas y columnas, y aunque fue moderadamente simplificado, el modelo semántico construido a partir del modelo en estrella superaba los límites recomendados por el sistema. Esto podría verse como limitación ya que no se estarían siguiendo las advertencias de Snowflake, aun así, los resultados obtenidos no cumplen con las expectativas que la plataforma presentaba.

Tras observar que Cortex Analyst tardaba bastante en generar las consultas con el modelo semántico construido a partir del modelo en estrella, se decidió probar el rendimiento con la vista. Como se ha podido comprobar con los resultados, este modelo proporciona respuestas óptimas y precisas en comparación. Esto puede deberse a que no tiene que realizar JOINs a la hora de seleccionar las columnas ya que todas están incluidas en una única vista.

Sin embargo, si las preguntas realizadas son ambiguas o se utiliza un vocabulario diferente al incluido en el modelo semántico, el LLM no es capaz de entender la pregunta o devuelve una consulta vacía, sin resultados.

Los modelos semánticos son base para el buen funcionamiento del LLM, por ello hay que dedicarle tiempo a su construcción. Cuando se trata de múltiples tablas con cientos de columnas en total, esto puede suponer un esfuerzo mayor pudiendo dejar de ser óptimo la construcción de este tipo de archivos. Cortex Analyst no es capaz, por el momento, en asistir al usuario de rellenando los campos forma autómata. Esto es una mejora que podría llegar pronto ya que dese Snowflake se comprometen a mejorar el modelo de forma continua.

Un aspecto importante de Cortex Analyst es que no se entrena con datos de los clientes y estos nunca abandonan los límites de gobernanza de Snowflake, proporcionando un entorno seguro para los datos. Además, a la hora de utilizar la interfaz de chat, el modelo utiliza metadatos que han sido previamente proporcionados en el modelo semántico (YAML). Las consultas SQL son ejecutadas en el warehouse virtual propio del cliente y así generar el resultado final.

Otra parte importante es que todos estos procesos mencionados en el caso de uso consumen unos créditos, los cuáles se traducen a un costo monetario. La monitorización de los gastos es algo fundamental a la hora de utilizar estos servicios para así valorar el coste económico que supone. En este caso de uso, la utilización de la aplicación de Streamlit durante 1h y 15m, en un warehouse de tamaño *S*, supone un gasto de 2.472671463 créditos. Lo que supone un gasto de $9.643 ya que el precio por crédito es de $3.90 a fecha de febrero 2025.

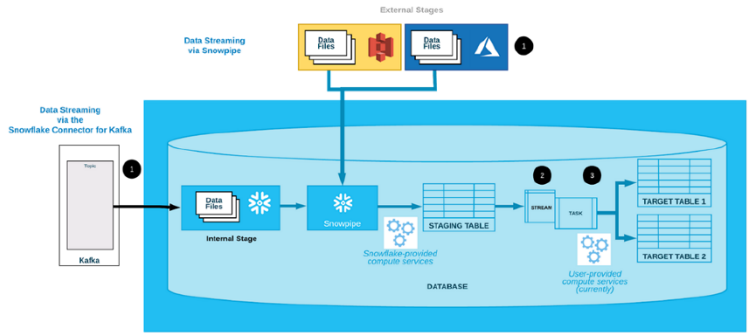
El almacenamiento de todas las tablas que fueron utilizadas para este caso de uso fue de unos 700kB, sin incluir los archivos YAML. El creado a partir del modelo semántico pesa 27.1kB, mientras que el de la vista pesa un poco menos de la mita, 12.4kB.

Para concluir, a pesar de la atractiva interfaz que ofrece Snowflake para interactuar con Cortex Analyst y su sencillez, actualmente el modelo no logra cumplir las expectativas. Cuando se trata con modelos complejos como el presentado aquí, este no es capaz de comprender ciertas preguntas y relaciones entre tablas. Para mejorar el rendimiento, es esencial que le modelo semántico generado sea muy específico y detallado, característica que requiere de tiempo y esfuerzo cuando se manejan grandes tablas. Este proceso debería de optimizarse con la ayuda de Cortex Analyst cuando se avance con este LLM.

6. CURSO DE SNOWFLAKE

Curso de Snowflake

snowflake



Capítulo 1

Introducción a Snowflake y su Arquitectura

Google Cloud · Azure · aws

1. ¿Qué es Snowflake? Historia y evolución del Data Cloud
2. Arquitectura multiclúster en la nube: separación de almacenamiento y cómputo
3. Diferencias entre Snowflake en **AWS, Azure y Google Cloud**
4. Conceptos clave: Virtual Warehouses, Databases y Schemas
5. Seguridad y gobernanza en Snowflake: roles, permisos y cifrado de datos
6. **Ejercicio:** Crear una cuenta en Snowflake, configurar un **Virtual Warehouse** y cargar datos en una tabla

Capítulo 2

Almacenamiento y Gestión de Datos en Snowflake

snowflake · snowpipe

1. Tipos de almacenamiento en **Snowflake:** Databases, Tables, Stages y External Tables
2. **Snowflake File Formats:** Ingesta de datos desde CSV, Parquet y JSON
3. **Ingesta de datos en streaming** con Snowpipe y Auto-Ingest
4. **Time Travel y Fail-safe:** Recuperación de datos históricos
5. Uso de **Zero-Copy Cloning** y **Data Sharing** para colaboración entre cuentas
6. **Ejercicio:** Crear una tabla, cargar datos desde un Stage, restaurar datos con Time Travel

Capítulo 3

Transformación y Procesamiento de Datos en Snowflake

snowflake · python

1. Uso de **Snowflake SQL** para transformación de datos
2. **Streams & Tasks:** Automatización de pipelines de datos
3. **Materialized Views** y **Clustering** para optimización de consultas
4. **Stored Procedures** y **User-Defined Functions (UDFs)** en SQL y Python
5. Uso de **Dynamic Tables** para mantener datos actualizados sin ETL complejo
6. Ejercicio: Crear un **Stream** y un **Task** para capturar cambios en datos en tiempo real

Capítulo 4

Integración de Snowflake con Herramientas Externas

databricks · Spark · PowerBI · tableau · Looker

1. **Conectores y API:** Snowflake Connector for Python, JDBC y ODBC
2. **Integración con Apache Spark y Databricks**
3. **Uso de Snowflake con herramientas de BI:** Power BI, Tableau y Looker
4. **Snowpark:** Desarrollo de código en Python y Java dentro de Snowflake
5. **Uso de Snowflake con Data Lakes:** External Tables y Iceberg Tables
6. **Ejercicio:** Conectar Snowflake con Python y ejecutar consultas con el Snowflake Connector

Capítulo 5

Machine Learning e Inteligencia Artificial con Snowflake

TensorFlow

1. Introducción a **Snowflake Cortex AI:** Modelos preentrenados y funciones de IA
2. **Uso de Snowpark ML** para entrenar modelos dentro de Snowflake
3. Integración con **Hugging Face y TensorFlow**
4. **Feature Engineering** en Snowflake con funciones avanzadas de SQL y Python
5. **MLOps en Snowflake:** Entrenamiento, versionado y despliegue de modelos
6. **Ejercicio:** Crear y ejecutar un **modelo de Machine Learning** dentro de Snowflake con Snowpark ML

Capítulo 6

Integración de Snowflake con Herramientas Externas

snowflake

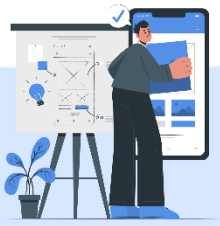
1. **Data Sharing:** Comparticion de datos en tiempo real sin duplicaciones
2. **Snowflake Marketplace:** Acceso y publicacion de datasets
3. **Data Governance:** Uso de **Object Tagging y Data Classification**
4. **Masking Policies y Row Access Policies:** Proteccion de datos sensibles
5. **Auditoría y Monitoreo** con Query History y Access History
6. **Ejercicio:** Configurar un **Data Share** entre cuentas de Snowflake y aplicar una **Masking Policy**

Capítulo 7

Optimización del Rendimiento y Costos en Snowflake

snowflake

1. Estrategias de optimización de consultas: **Micro-partitioning y Clustering**
2. **Query Performance Tuning:** Uso de Query Profile para análisis de ejecución
3. **Resource Monitors:** Control de costos y límites de cómputo
4. **Auto-suspend y Auto-scale:** Ajuste automático de Virtual Warehouses
5. **Optimización de almacenamiento** con Time Travel y Retention Policies
6. **Ejercicio:** Analizar y optimizar una consulta con **Query Profile**



PARTNER · snowflake | stratebi Analytics and Big Data

stratebi
Analytics and Big Data | snowflake

SERVICES PARTNER

SELECT