UNIVERSIDADE DA CORUÑA

Facultade de Informática

Máster Universitario en Enxeñaría en Informática

TRABALLO FIN DE MÁSTER

Análise de Rendemento en Plataformas Data Lakehouse: Snowflake e Databricks

Estudante: Manuel Corujo Muíña

Dirección: Guillermo López Taboada

A Coruña, xuño de 2024.

Aos meus pais

Agradecementos

Considero necesario comezar agradacendo ao meu director de proxecto, Guillermo, pola súa implicación, por todo o tempo empregado e en definitiva porque sen a súa orientación este proxecto non sería posible.

Tamén me gustaría dar as grazas a Ángel Regueiro, compañeiro do CITIC, polos seus consellos e porque traballar con el supuxo unha experiencia moi enriquecedora.

Por suposto, non sería sen a miña familia que tería cumprido logros coma este. Grazas por serdes fonte incondicional e inesgotable de comprensión e apoio. Esfórzome incesablemente para ser quen de vos devolver todo o que me levades dado e me continuades a dar. E, tamén, grazas a María, por ser con quen seguir medrando da man cada día.

Resumo

Este traballo compara o rendemento de dúas das plataformas Data Lakehouse máis populares na actualidade: Snowflake e Databricks, mediante a execución de benchmarks de referencia no seu ámbito, como é o caso de TPC-DS. A análise dos resultados permitiu coñecer en profundidade as distintas configuracións e optimizacións que permiten ambas plataformas. As principais recomendacións e conclusións subliñan a importancia de non sobredimensionar os recursos de cómputo, senón adaptalos ao tamaño dos datos cos que traballar para lograr unha eficiencia óptima. Ademais, tamén se descubriu que o uso de AWS como provedor de *cloud* para Snowflake pode mellorar o seu rendemento respecto a Azure e GCP. Por último, para aqueles escenarios que implican a carga frecuente de grandes volumes de datos, Databricks demostrou ser máis eficaz.

Abstract

This work compares the performance of two of the most popular Data Lakehouse platforms at the moment: Snowflake and Databricks, by running reference benchmarks in their field, such as TPC-DS. The analysis of the results provided in-depth knowledge of the different configurations and optimizations available on both platforms. Key recommendations and conclusions include the importance of not oversizing computational resources but rather tailoring them to match the data workload for optimal efficiency. Additionally, it was found that using AWS as the cloud provider for Snowflake can enhance performance. Lastly, for scenarios involving frequent loading of large datasets, Databricks proved to be more effective.

Palabras chave/Keywords:

- Benchmarking
- Data Lake
- Data Lakehouse
- Data Warehouse
- Databricks
- Snowflake

Índice Xeral

Índice de Figuras

Índice de Táboas

Capítulo 1

Introdución

Na década dos 90 comezaron a popularizarse os Data Warehouses como ferramentas para centralizar a información provinte dos sistemas operacionais e facilitar os procesos de análise de información, substituíndo progresivamente ás bases de datos que se utilizaban con estes fins. Ademais, a partir de 2010 comezaron a empregarse os Data Lakes, convivindo as dúas ferramentas durante a década dos 2010. Neste contexto, a finais desa mesma década xorde a figura do Data Lakehouse, que reune nunha única ferramenta as vantaxes das dúas anteriores. Xunto con esta nova arquitectura aparece a necesidade de avaliar o rendemento das súas características. Sabendo isto, o obxectivo deste proxecto consiste na execución de benchmarks que proben as características de dúas das ferramentas presentes no estado do arte dos Data Lakehouses: Snowflake e Databricks.

Numerosos artigos publicados nos últimos anos sosteñen que a arquitectura Data Warehouse tal e como se coñece hoxe en día desaparecerá, sendo substituída por un novo patrón, coñecido como Data Lakehouse, caracterizado por (i) formatos de datos open source de acceso directo, como Apache Parquet[1] e ORC[2], (ii) soporte out-of-the-box para cargas de traballo de aprendizaxe automática e ciencia de datos, e (iii) rendemento estado do arte [1] [2].

Este TFM aborda a comparación de rendemento entre dúas das plataformas Data Lakehouse máis populares na actualidade: Snowflake[3] e Databricks[4]. Isto resulta de interese para as organizacións e profesionais no campo da xestión e análise de datos, que deben seleccionar a plataforma de datos máis adecuada para as súas necesidades; unha elección que a miúdo se basea en información proporcionada polos propios provedores, o que pode levar a decisións tendenciosas.

[1] https://parquet.apache.org/
[2] https://orc.apache.org/
[3] https://www.snowflake.com/
[4] https://www.databricks.com/

1.1 Motivación

Tanto Snowflake como Databricks publicaron resultados que apoian o rendemento superior das súas respectivas plataformas en diferentes benchmarks [3] [4], polo que, neste contexto, xorde a necesidade de realizar unha avaliación independente e obxectiva, que achegue resultados imparciais e xenuínos á comunidade.

1.2 Obxectivos

Identificouse como obxectivo principal proporcionar unha comparación exhaustiva do rendemento das plataformas, identificando posibles vantaxes e desvantaxes de cada unha en termos de eficiencia, escalabilidade e rendemento. Espérase que os resultados deste estudo resulten de gran utilidade para profesionais, empresas e organizacións que buscan tomar decisións informadas na selección da súa infraestrutura de datos. Este proxecto está destinado a un público amplo no campo da informática e a xestión de datos, incluíndo enxeñeiros de datos, arquitectos de sistemas, analistas de datos e demais traballadores de empresas que desexan implementar solucións Data Lakehouse. Ademais, os resultados contribuirán á comunidade académica ao ofreceren unha avaliación independente e obxectiva destas dúas plataformas.

A raíz deste obxectivo principal, establecéronse os seguintes obxectivos máis específicos:

1. **Análise do estado da arte en plataformas Data Lakehouse**: Coñecer a evolución e as capacidades das plataformas Data Lakehouse, así como as súas ventaxas fronte ás solucións máis tradicionais, Data Warehouses e Data Lakes; identificar aos principais actores neste ámbito, incluindo as súas arquitecturas e funcionalides; e revisar publicacións sobre rendemento e benchmarks existentes, como son os da familia Transaction Processing Performance Council (TPC), analizando como foron empregados en anteriores investigacións.

2. **Execución dos benchmarks**: Seleccionar os benchmarks que se consideren apropiados e relevantes para avaliar o rendemento de plataformas Data Lakehouse; desenvolver unha ferramenta que permita a execución deses benchmarks tanto en Snowflake coma en Databricks, de forma análoga; e realizar múltiples execucións dos benchmarks, seguindo as pautas e boas prácticas establecidas na súa especificación.

3. **Recompilación de Datos e Métricas**: Definir un conxunto de métricas a recompilar durante as execucións dos benchmarks, como tempos de resposta, tempos de execución das consultas ou utilización de recursos; implementar ferramentas e *scripts* que permitan ou faciliten recompilar eses datos e métricas detalladas sobre o rendemento de

Snowflake e Databricks durante estas execucións; e garantizar a precisión e integridade dos datos realizando múltiples execucións.

4. **Recomendacións**: Interpretar os resultados obtidos para obter información sobre as fortalezas e debilidades de Snowflake e Databricks; proporcionar recomendacións prácticas para profesionais, empresas e organizacións baseadas nos resultados da evaluación, tendo en conta factores coma o coste, o rendemento, e casos de uso específicos; analizar as implicacións que podería ter para as organizacións migrar cara arquitecturas Data Lakehouse, incluindo beneficios e riscos potenciais; e suxerir boas prácticas para optimizar o rendemento e a utilización de recursos.

5. **Contribución Académica**: Proporcionar unha metodoloxía detallada e transparente que poida ser reproducida ou ampliada en futuras investigacións; e publicar os resultados en revistas académicas ou congresos para contribuir ao campo de xestión e análise de datos.

1.3 Estado do arte

1.3.1 Data Warehouse

Un Data Warehouse é unha "colección de datos orientada a un tema, integrada, variable no tempo e non volátil, que se utiliza principalmente na toma de decisións da organización" [5]. Normalmente, o Data Warehouse é un sistema informacional que se mantén separado das bases de datos operativas da organización. Hai moitas razóns para iso, pois o Data Warehouse soporta o procesamento analítico en liña (OLAP), cuxos requisitos funcionais e de rendemento son moi diferentes dos das aplicacións de procesamento de transaccións en liña (OLTP) tradicionalmente soportadas polas bases de datos operativas [6].

Os Data Warehouses están orientados ao apoio á toma de decisións, polo que tamén se coñecen coma Decision Support System (DSS). Nestes sistemas os datos históricos, resumidos e consolidados son máis importantes que os rexistros individuais detallados. Dado que os Data Warehouses conteñen datos consolidados, en moitos casos de distintas bases de datos operativas, durante períodos de tempo potencialmente longos, tenden a albergar datos de ordes de magnitude maiores que as bases de datos operativas [6]. Prevese que os Data Warehouses empresariais teñan un tamaño de centos de gigabytes a terabytes. As cargas de traballo son intensivas en consultas, na súa maioría *ad-hoc* e complexas, que poden acceder a millóns de rexistros e realizar moitos escaneos, unións e agregados. O rendemento das consultas e os tempos de resposta son máis importantes que o rendemento das transaccións.

Para facilitar as visualizacións e análises complexas, os datos dun Data Warehouse adoitan modelarse multidimensionalmente. Por exemplo, nun Data Warehouse de vendas, a hora da

venda, o distrito de vendas, o vendedor e o produto poden ser algunhas das dimensións de interese. A miúdo, estas dimensións son xerárquicas; a hora de venda pode organizarse como unha xerarquía día-mes-trimestre-ano, e o produto como unha xerarquía produto-categoría-industria [6]. As operacións OLAP típicas inclúen *roll-up* (aumento do nivel de agregación) e *drill-down* (diminución do nivel de agregación ou aumento do detalle) ao longo dunha ou máis xerarquías de dimensións, *slice_and_dice* (selección e proxección) e *pivot* (reorientación da vista multidimensional dos datos) [6].

Arquitectura

Na Figura 1.1 pódese observar a arquitectura típca dun Data Warehouse. Esta inclúe ferramentas para extraer datos de múltiples bases de datos operativas e fontes externas; para limpar, transformar e integrar estes datos (proceso coñecido como Extract, Transform, Load (ETL)); para cargar datos no Data Warehouse; e para actualizar periodicamente o warehouse para reflectir as actualizacións nas fontes e para eliminar datos do warehouse (traspasándoos a un almacenamento máis lento e económico). Ademais do warehouse principal, pode haber varios *data marts* departamentais. Os datos do warehouse e dos *marts* son almacenados e xestionados por un ou varios servidores, de modo que o warehouse pode estar distribuído para equilibrar a carga, a escalabilidade e unha maior dispoñibilidade. Nunha arquitectura distribuída deste tipo, o repositorio de metadatos adoita replicarse con cada fragmento do warehouse, e todo o warehouse adminístrase de forma centralizada [6]. Unha arquitectura alternativa, implementada por conveniencia cando pode ser demasiado custoso construír un único warehouse loxicamente integrado, é unha federación de warehouses ou *data marts*, cada un co seu propio repositorio e administración descentralizada.

Deseñar e poñer en marcha un Data Warehouse é un proceso complexo, que consta das seguintes actividades: [7]

- Definir a arquitectura, planificar a capacidade e seleccionar os servidores de almacenamento, os servidores de bases de datos e OLAP e as ferramentas.
- Integrar os servidores, o almacenamento e as ferramentas cliente.
- Deseñar o esquema do warehouse e as vistas.
- Definir a organización física do warehouse, a ubicación dos datos, a partición e os métodos de acceso.
- Conectar as fontes utilizando *gateways*, controladores ODBC ou outros *wrappers*.
- Deseñar e aplicar scripts para a extracción, limpeza, transformación, carga e actualización de datos.

- Encher o repositorio coas definicións de esquemas e vistas, scripts e outros metadatos.
- Deseñar e implantar aplicacións de usuario final.
- Despregamento do Data Warehouse e as aplicacións.

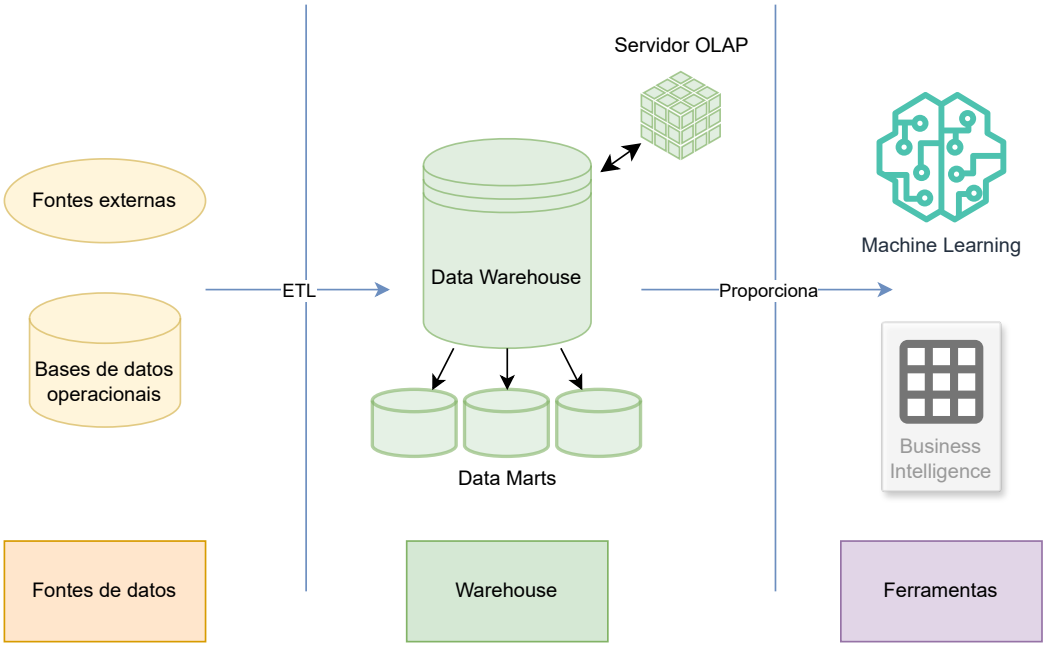


Figura 1.1: Arquitectura típica dun Data Warehouse

1.3.2 Data Lake

Os Data Lakes, plataformas de datos cun enfoque schema on read, propuxéronse para facer fronte aos retos relacionados coas características de variedade e velocidade do Big Data [8]. Nesta arquitectura rica e flexible [9] os datos poden ser explorados inmediatamente, como se mencionou, seguindo un enfoque schema on read, mesmo en diferentes formatos de datos ou con datos cambiantes no tempo, sen custosas transformacións en esquemas predefinidos. Os Data Lakes están orientados ao uso da ciencia de datos [10]. Grazas a estas características, trátase dunha arquitectura moi popular para o almacenamento e procesamento de datos na nube.

Mentras que nos Data Warehouses se manexan datos estruturados, nos Data Lakes faise fincapé nos datos semiestruturados ou non estruturados. Os datos semiestruturados libéranse da estrutura tabular asociada ás bases de datos relacionais: non requiren un esquema, e admiten unha estrutura de datos xerárquica que contén información aniñada. Algúns exemplos destes formatos de datos son JSON, Avro, ORC ou Apache Parquet. Por outra banda, os datos

non estruturados están moi presentes hoxe en día. Exemplos deste tipo de datos son as imaxes e os vídeos, así como todos os ficheiros de texto que unha organización pode almacenar como correos electrónicos ou contidos multimedia en formato de vídeo. Para as imaxes existen os formatos típicos como JPEG, os textos poden gardarse como TXT, PDF, etc. e o vídeo tamén pode estar en varios formatos, como MP4.

Os principais desafíos aos que se enfronta esta arquitecura enuméranse na Sección 1.3.2. Ademais, os Data Lakes deben incluír soporte para aprendizaxe automática *out-of-the-box*, como se detalla na Sección 1.3.2.

Principais desafíos das arquitecturas Data Lake

Os Data Lakes deben facer fronte a unha longa lista de desafíos [11], entre os que destacan os enumerados a continuación:

- Inxesta de datos: necesidade de soportar a inxesta de datos para unha ampla variedade de sistemas; coma rastrexadores web ou rexistros.
- Extracción de datos: mediante a inxesta créanse datasets sen procesar. A extracción de datos é a tarefa que transforma estes datos nun modelo de datos predefinido.
- Limpeza de datos: antes de crear esquemas de metadatos ou calquera outra tarefa, é importante limpar os datos correctamente.
- Descubrimento de datos: debido ás cantidades masivas de datos e aos deficientes ou mesmo inexistentes esquemas, o descubrimento de datos converteuse nun problema importante. Algunhas solucións céntranse en crear e enriquecer catálogos de datos, así como en facilitar a procura neles.
- Xestión de metadatos: sen xestión de metadatos nin esquemas, un Data Lake pode converterse nun *Data Swamp* [11]. Isto pódese evitar creando catálogos con metadatos.
- Integración de datos: o reto no referente a este punto é a integración baixo demanda, a tarefa de integrar datos sen procesar no momento da consulta.
- Xestión de versións de datos: existen ferramentas para o versionado de datos, máis complexas e útiles que o versionado de nomes ou o almacenamento de todas as versións. Os maiores retos son a xestión da evolución dos esquemas e as peculiaridades dos formatos de datos entre versións.

Soporte para Aprendizaxe Automática

A Intelixencia Artifical (IA), e en particular a Aprendizaxe Automática (AA), está cada vez máis estendida. Estes algoritmos adoitan consistir en modelos complexos que teñen que lidar con enormes cantidades de datos. Por exemplo, a primeira versión de ChatGPT[5] utilizaba GPT-3, un modelo lingüístico extremadamente grande, formado por uns 175.000 millóns de parámetros, que se desenvolveu mediante o adestramento cun dataset de 570 GB de texto plano, consumindo varios miles de petaflop/s-día de cálculo [12]. ChatGPT utiliza actualmente modelos máis grandes e complexos, como GPT3.5 e GPT4, e é utilizado por millóns de usuarios en todo o mundo. Nos últimos anos, a potencia de cálculo necesaria para a IA e a AA aumentou enormemente, e converteu o adestramento e despregamento destes modelos na nube nunha situación problemática, xa que pode xerar unha gran pegada de carbono [13].

Por isto, o desenvolvemento de produtos de AA pode supoñer un gran reto cando se trata de automatizalos e operacionalizalos, o que a miúdo dá lugar a expectativas insatisfeitas. Para abordar este problema introduciuse un novo paradigma, Machine Learning Operations (MLOps) [14]. MLOps, que xurdiu en 2009 para reducir os problemas no desenvolvemento de software no ámbito de analítica avanzada, introduce mellores prácticas e conxuntos de conceptos, similares a DevOps. Existen varios marcos de desenvolvemento (framework) de código aberto dispoñibles para IA e AA, como son MLFlow [15], Airflow e Kubeflow [16], así como opcións comerciais como as que ofrecen Amazon Web Services (Sagemaker)[6] ou Microsoft Azure[7]. Ademais, hai outros aspectos cruciais a ter en conta, como o soporte para notebooks, especialmente importante para que os científicos de datos axilicen e compartan o seu traballo [17].

1.3.3 Data Lakehouse

A arquitectura Data Lakehouse [18] [19], unha recente proposta de combinación de Data Lake e Data Warehouse e analítica avanzada de negocio [1], da resposta á necesidade dunha plataforma de xestión de Big Data orientada ao negocio, clave para acelerar o ritmo da transformación dixital das organizacións dando o salto á analítica avanzada baseada en Big Data. Esta recente proposta presenta unha capa subxacente de Data Lake para o almacenamento eficiente de datos estruturados, semiestruturados e non estruturados, estreitamente axustada tanto con SQL (Business Intelligence, informes e visualización) como co procesamento de dataframes baseado en Spark para AA. Isto é posible grazas a un *middleware* que indexa os metadatos e automatiza as cargas de traballo, tanto por lotes como en tempo real. A plataforma Data Lakehouse responde principalmente a unha necesidade empresarial. De feito,

[5] https://chat.openai.com/
[6] https://aws.amazon.com/es/sagemaker/mlops/
[7] https://azure.microsoft.com/en-us/products/machine-learning/mlops/

existe un mercado multimillonario en crecemento con dous principais protagonistas, Databricks [20] e Snowflake [21]. Con todo, os retos de investigación son moi importantes, xa que esta plataforma: (i) baséase en formatos de datos de acceso directo, como Parquet, o que expón a necesidade dunha xestión eficaz dos datos para extraer automaticamente a información cun enfoque de lectura de esquemas; (ii) proporciona soporte para AA out-of-the-box nos datos do Data Lake; e (iii) ofrece un rendemento excepcional, mesmo baseado en múltiples microservizos. O Data Lakehouse baséase en servizos na nube, como o almacenamento de obxectos, o procesamento de datos e os servizos de xestión de API, entre outros, a diferenza dos Data Warehouses e Data Lakes tradicionais que adoitan basearse, por motivos de rendemento, en software despregado en servidores físicos. A Táboa 1.1 resume as características de Data Lakehouse, en contraste coas de Data Warehouse e Data Lake. Aquelas características que si inclúen cada unha das plataformas aparece reflexada cunha celda de cor rosa, mentres que as que non posúen atópanse en branco, segundo corresponda.

		Data Warehouse	Data Lakehouse	Data Lake
Datos	Estruturados & procesados	■	■	
	Non estruturados e en crú		■	■
	Asistencia para a xestión de datos		■	
Procesado	Schema-on-write	■		
	Schema-on-read		■	■
Arquitectura	Escalabilidade do almacenamento		■	■
	Compoñentes flexibles		■	■
	Nativo na nube		■	
	AA out-of-the-box		■	
	Alto Rendemento	■		
Aplicacións	Intelixencia de Negocio	■	■	
	Exploración de datos		■	■
	AA/Ciencia de Datos		■	■

Táboa 1.1: Características dun Lakehouse, contrastadas coas de Warehouses e Data Lakes

Solucións como Delta Lake[8] achegan unha solución a o reto de lograr un equilibrio entre o baixo custo, as capacidades masivas de almacenamento e o soporte para datos semiestruturados e non estruturados dos Data Lakes; e o desprazamento no tempo, as transaccións ACID e o alto rendemento que requiren os Data Warehouses [22]. Isto consíguese ofrecendo unha capa de almacenamento de táboas ACID de código aberto sobre obxectos de almacenamento na nube. Delta utiliza un rexistro de transaccións que se compacta en Apache Parquet para proporcionar as características anteriormente mencionadas dos Data Warehouses [20].

[8] https://delta.io/

Converxencia entre os sistemas Data Warehouse e Lakehouse

Existe unha notable tendencia cara á converxencia entre os sistemas Data Warehouse e Data Lakehouse, como demostra Snowflake, debido á súa dependencia mutua de solucións de almacenamento de obxectos economicamente viables. Cabe sinalar, con todo, que os Data Warehouses difiren dos formatos de almacenamento habituais nas arquitecturas Data Lakehouse en que non admiten formatos de datos dispoñibles abertamente, como é o caso do anteriormente citado Delta [22].

1.4 Estrutura da memoria

Nesta Sección realízase unha breve explicación do contido de cada capítulo.

Capítulo 1 Capítulo introdutorio, onde se mencionan a motivación e os obxectivos do proxecto, ademais de explicar conceptos relacionados co estado do arte, como son Data Warehouse, Data Lake e Data Lakehouse. Ademais, tamén se presentan os principais obxectivos do proxecto.

Capítulo 2 Breve explicación de Kanban, a metodoloxía utilizada para a xestion do proxecto, así como xustificación de por que foi escollida. Tamén se realizan unha estimación dos custos do proxecto, menciónase a planificación inicial apoiándose nun diagrama de Gantt e indícanse os riscos identificados, así como a súa importancia e probabilidade e como xestionalos no caso de apareceren.

Capítulo 3 Detalle das principais tecnoloxías utilizadas. Menciónanse linguaxes de programación e ferramentas de soporte, como ferramentas de xestión de paquetes ou de control de versións; pero sobre todo detállanse tanto Snowflake como Databricks, as plataformas Data Lakehouse a avaliar, e TPC-DS, o benchmark utilizado para dita avaliación.

Capítulo 4 Detállase a ferramenta desenvolvida para executar TPC-DS: Menciónanse funcionalidades, detalles da implementación como patróns de deseño utilizados, tests, etc. Tamén se menciona unha proba que se realizou para conectar Power BI tanto con Snowflake coma con Databricks.

Capítulo 5 Detalle dos experimentos realizados, así coma os resultados obtidos en cada un deles. Por unha banda indícanse cales foron os experimentos realizados con TPC-DS e a implementación que se desenvolveu, mentras que por outra se fai mención ao experimento realizado conectando as plataformas Data Lakehouse con Power BI.

Capítulo 6 Conclusións obtidas tras a execución dos experimentos realizados no Capítulo anterior, así como posibles liñas de traballo futuro, onde se mencionan outros benchmarks suscpetibles de ser utilizados, así como outras plataformas Data Lakehouse para comparar, entre outras cousas.

Capítulo 2

Metodoloxía, Planificación e Ferramentas

Neste Capítulo preséntanse a metodoloxía de desenvolvemento escollida, a estimación de custos e duración, a planificación temporal, a xestión de riscos e as ferramentas de desenvolvemento e de soporte utilizadas.

2.1 Metodoloxía de desenvolvemento

A metodoloxía escollida para a realización do proxecto é Kanban [23]. No anteproxecto indicouse que se utilizaría Scrum, pero tras as primeiras semanas de desenvolvemento e vistas as facilidades que promove GitHub Project, que permite pechar *issues* do repositorio con tan só mover as tarxetas coas tarefas polo taboleiro (entre outras integracións que ofrece entre o taboleiro, que se pode ver na Figura 2.2 e o repositorio), decidiuse que o proxecto se desenvolvese utilizando a metodoloxía Kanban. Esta metodoloxía utilizouse por primeira vez a gran escala na década de 1940, cando o fabricante de automóbiles Toyota empezou a utilizala para seguir e xestionar o seu fluxo de traballo. Ao asignar as tarefas e os produtos en curso a tarxetas e permitir unicamente aos membros do equipo realizar o traballo vinculado ás tarxetas que tiñan diante, o sistema Kanban axudoulles a facer máis eficiente o seu fluxo de traballo e a garantir que se desperdiciase menos tempo e materiais ao longo de todo o proceso de fabricación.

Non foi ata principios da década dos 2000, co auxe da programación áxil[1], cando o método Kanban empezou a utilizarse no mundo do desenvolvemento de software (concretamente en 2004, cando David Anderson colaboraba nun pequeno equipo de Microsoft [24]). Dado que a metodoloxía Kanban se centra no fluxo de traballo a través dun sistema, os mesmos principios que a fixeron útil na fabricación poden transferirse tamén a un proceso de desenvolvemen-

[1] https://martinfowler.com/agile.html

to de software [25]. Algunhas das prácticas básicas de Kanban, como visualizar o fluxo de traballo, limitar a cantidade de traballo en curso e implementar bucles de retroalimentación; entre outras, son especialmente útiles no desenvolvemento de software. Por iso, equipos de software de todo o mundo empezaron a adoptar Kanban como metodoloxía para dirixir os seus proxectos, e hoxe en día segue sendo moi popular.

As principais características ou elementos que definen a esta metodoloxía son os seguintes [23]:

- Visualizar o traballo: Debe existir un taboleiro no que organizar as diferentes tarefas por columnas, segundo a fase de realización na que se atopen. Como mínimo, este taboleiro debe conter tres columnas: a primeira delas, coas tarefas pendentes de realizar; a segunda coas tarefas en curso; e a última delas coas tarefas xa rematadas. Aínda así, columnas adicionais como unha adicada a tarefas en revisión poden resultar útiles. Pódese ver un exemplo baleiro de como é un taboleiro kanban, coas súas tarefas organizadas en columnas, na Figura 2.1.

- Limitar o traballo en curso: Esto conséguese ao permitir visualizar o fluxo de traballo e establecer límites máximos de tarefas para cada etapa, o que garante a concentración, reduce a multitarefa e mellora a eficacia.

- Mellora continua: A mellora continua en Kanban foméntase mediante a revisión e adaptación periódicas do proceso. Isto implica analizar métricas e realizar axustes graduais para optimizar o fluxo de traballo e aumentar a produtividade co tempo.

- Liderazgo en todos os niveis: O correcto seguemento da metodoloxía obriga a implicarse a todos os membros do equipo na definición das tarefas.

Figura 2.1: Taboleiro kanban de exeplo

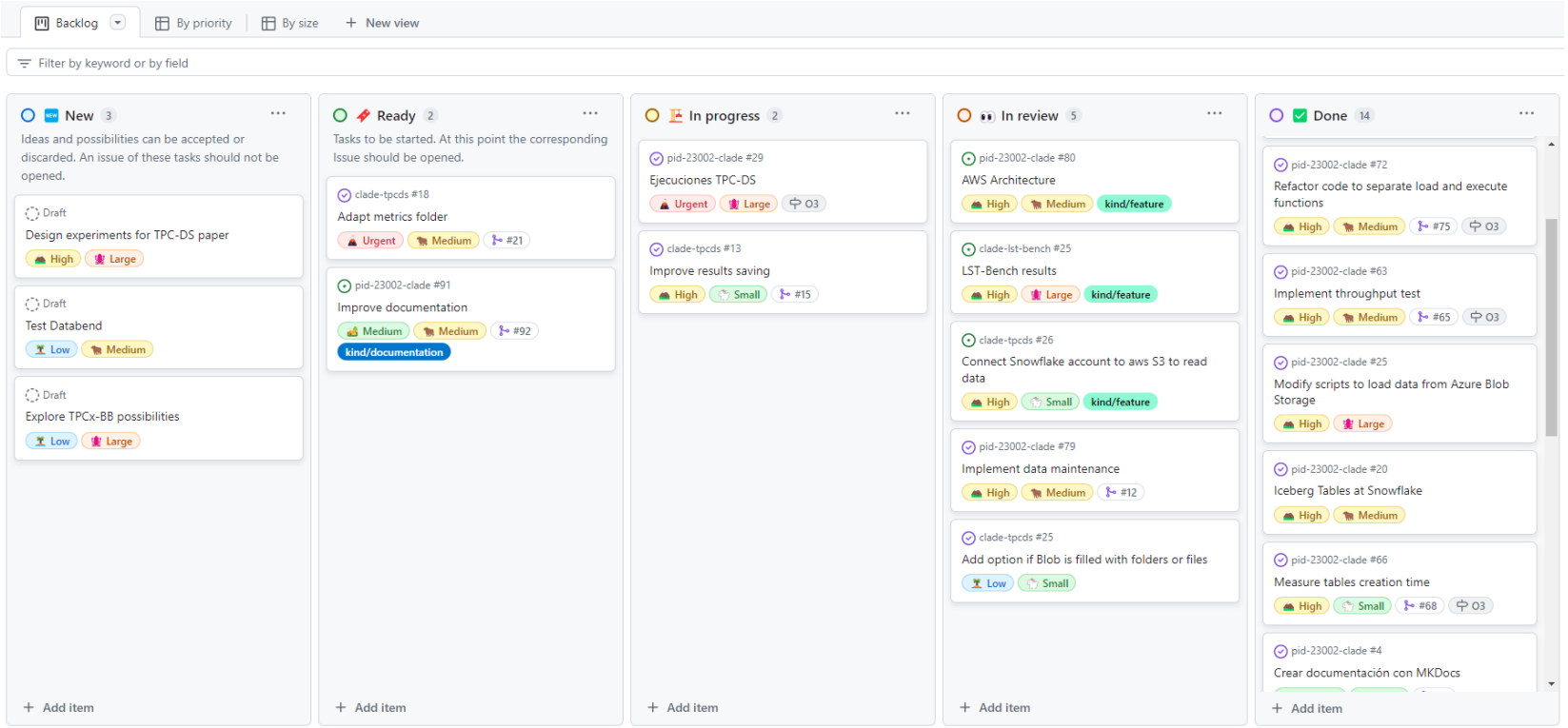


Figura 2.2: Taboleiro kanban de GitHub Project utilizado

2.2 Estimación de custos

Realizouse un cálculo dos custos do proxecto. Para iso, calculáronse de forma separada os custos dos recursos técnicos e os dos recursos humanos, detallados nas Táboas 2.1 e 2.2.

2.2.1 Recursos técnicos

Para a realización deste proxecto utilizáronse dous ordenadores, un de sobremesa e o outro portátil, cuxas especificacións e custos estimados, baseándose no custo de amortización no período estimado (cada equipo foi un 5% da súa vida útil) se detallan na Táboa 2.1.

CPU	GPU	Memoria principal	Coste estimado	Custo amortización
Intel(R) Core(TM) i5-1235U CPU @ 1.30 GHz	Intel(R) UHD Graphics	8 GB	1.600 €	80 €
Intel(R) Core(TM) i5-8250U CPU @ 1.60 GHz	NVIDIA GeForce MX150	8 GB	900 €	45 €

Táboa 2.1: Custo estimado dos recursos informáticos empregados no proxecto

2.2.2 Recursos humanos

Neste proxecto soamente se conta cun contribuínte, cuxos custos se detallan na Táboa 2.2. Hai que ter en conta que é unha simulación, xa que esta parte se desenvolve entre o estudante e o director do proxecto.

Recurso	Custo por hora	Número de horas	Custo total
Desenvolvedor	20 €	450	9.000 €
Director	50 €	20	1.000 €

Táboa 2.2: Custo estimado dos recursos humanos partícipes no proxecto

Deste modo, sumando os costes dos recursos técnicos e os recursos humanos, o coste total do proxecto ascende a 12.500 €.

2.3 Planificación

Nesta Sección detállanse as tarefas que se identificaron ao comezo do proxecto, así como a súa organización coa axuda dun diagrama de Gantt, ilustrado na Figura 2.3. Así, o proxecto pódese dividir nas seguintes cinco fases, coas súas correspondentes tarefas:

1. **Estudo das plataformas**.
    - **1.1**: Estudo das plataformas.

2. **Implementación da ferramenta de benchmarking**.
    - **2.1**: Implementación da fase de carga de datos.
    - **2.2**: Implementación da fase de execución de consultas.
    - **2.3**: Implementación de tests unitarios.
3. **Execución das probas nos entornos Snowflake e Databricks**.
    - **3.1**: Xeración dos datos e carga na nube.
    - **3.2**: Execucións variando o *scale factor*.
    - **3.3**: Execucións variando o tamaño do warehouse utilizado.
4. **Extracción de métricas e conclusións**.
    - **4.1**: Implementación do módulo de métricas.
    - **4.2**: Cálculo de métricas.
    - **4.3**: Conclusións das métricas obtidas.
5. **Documentación (da ferramenta e memoria do proxecto)**.
    - **5.1**: Manual de usuario e documentación da ferramenta.
    - **5.2**: Redacción da memoria do proxecto.

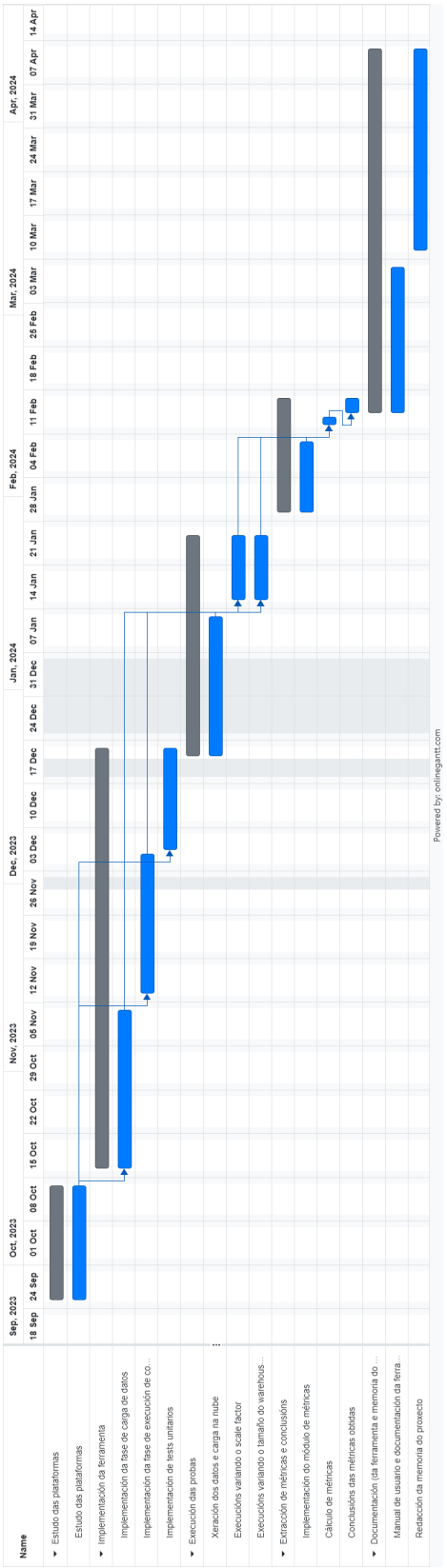


Figura 2.3: Diagrama de Gantt do proxecto

2.4 Xestión de riscos

O principal risco identificado é a inexistencia de benchmarks que se puidesen executar. O deseño Data Lakehouse é recente, e podía ocurrir que non se atopara ningún benchmark apropiado para avaliar as súas características. Ademais, por seren tanto Snowflake coma Databricks dúas plataformas incipientes, atópanse en constante cambio, o que pode dificultar a adquisición de coñecemento sobre características novidosas.

Tamén se identificou como risco a posibilidade de que durante o desenvolvemento do proxecto se publique algún outro proxecto ou resultado que realice unha comparativa similar. Esto podería ser un problema porque ese hipotético proxecto podería recibir a atención e recoñecemento que doutro modo se tería obtido; pero sobre todo porque pode repercutir na orixinalidade do proxecto, xa que se xa existiran publicacións similares tanto a orixinalidade como a novidade do proxecto se verían afectadas. Ademais, se presentase resultados dificilmente replicables podería xurdir controversia entre os seus resultados e os presentados neste traballo.

Debido á crecente popularidade das plataformas Data Lakehouse é probable que isto ocorra, polo que resulta necesario manterse en constante revisión do estado do arte para estar informado das publicacións relacionadas. No caso de se publicase algún resultado similar, este proxeco debería diferenciarse del dalgún modo, como realizar unha análise en maior profundidade (comparando tamén os distintos formatos de datos cos que permiten traballar as plataformas, por exemplo) ou incluír outras ferramentas Data Lakehouse. Por isto, identificamos este risco cunha posibilidade e un impacto medio.

2.5 Ferramentas

2.5.1 Linguaxes de programación

Bash

Bash, acrónimo de "Bourne Again Shell", é un shell de liña de comandos e linguaxe de creación de scripts moi utilizado en sistemas operativos tipo Unix, incluídos Linux e macOS. Proporciona aos usuarios unha interface para interactuar co sistema operativo mediante a execución de comandos e secuencias de comandos.

Neste proxecto utilizouse para a creación de scripts para procesar ficheiros, principalmente SQL.

Python

Python é unha linguaxe de programación interpretada de alto nivel que ofrece un rico conxunto de características; como módulos, excepcións, tipado dinámico e clases. Admite múltiples paradigmas de programación, como a programación orientada a obxectos, a funcional e a imperativa. Ademais, a súa sintaxe resulta elegante e sinxela. Ademais, a súa compatibilidade multiplataforma permítelle funcionar sen problemas tanto en Windows como en sistemas baseados en Unix, como son Linux e macOS.

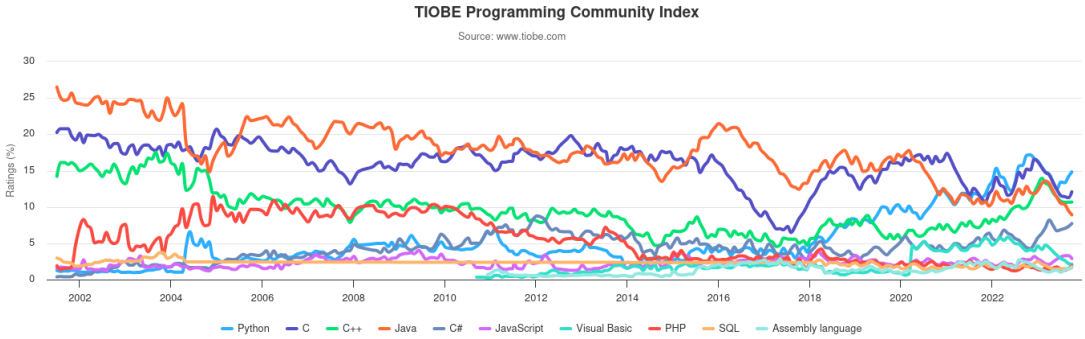


Figura 2.4: Índice TIOBE de popularidade das linguaxes entre 2002 e 2022

Aínda que a natureza interpretada de Python resulta nunha execución de código relativamente máis lenta en comparación con linguaxes compiladas como C [26], segue sendo a linguaxe de programación máis popular, como indica o índice TIOBE [27] (ver Figura 2.4), o que subliña a súa importancia no desenvolvemento de software na actualidade. Por tanto, aínda que Python pode ter inconvenientes en relación ao rendemento, a súa eficiencia na rápida creación de prototipos e o apoio da comunidade convérteno nunha ferramenta indispensable para moitas tarefas computacionais. Ademais, dada a natureza deste proxecto, o código Python realiza tarefas computacionalmente lixeiras e nas que o rendemento non é un requisito, posto que o código que debe desempeñar cun bo rendemento son as consultas SQL que se executan nas plataformas.

SQL

Structured Query Language (SQL) é unha linguaxe de programación que se utiliza para xestionar e manipular bases de datos relacionais. Permite aos usuarios realizar operacións tales como consultar datos, actualizar rexistros, inserir datos novos e eliminar datos existentes nunha base de datos. SQL ofrece unha sintaxe e un conxunto de comandos estandarizados para interactuar coas bases de datos, o que o converte nunha potente ferramenta para a xestión e a análise de datos.

Esta linguaxe é parte fundamental dos sistemas Data Warehouse e Data Lakehouse, e a linguaxe na que se basean benchmarks como TPC-DS, executado neste proxecto. En concreto, TPC-DS ofrece a posibilidade de xerar as consultas en distintos dialectos, e nós decidimos utilizar o estándar SQL92, compatible tanto con Snowflake coma con Databricks.

2.5.2 Ferramentas de desenvolvemento

Editor de texto

- **Visual Studio Code**. Visual Studio Code, desenvolto por Microsoft, é un editor lixeiro de ficheiros fonte dispoñible tanto en Windows, Linux e macOS. Ofrece un amplo conxunto de funcións que inclúen, entre outras, depuración de código, autocompletado, macros e refactorización. Ademais, ten unha interface fácil de usar e admite a instalación de extensións, o que permite altos niveis de personalización en aparencia e comportamento. Ademais, debido á popularidade de VS Code, o *marketplace* de extensións inclúe funcionalidades que axudan nunha ampla e variada multitude de tarefas. Cabe destacar que, aínda que o código fonte está dispoñible baixo licenza MIT[2], as versións de Microsoft seguen sendo propietarias.

2.5.3 Ferramentas de soporte

Xestión de paquetes

- **venv**. O módulo venv permite crear contornas virtuais lixeiras, que poden illarse opcionalmente dos directorios do sistema. Cada contorna virtual ten o seu propio intérprete de Python e pode ter o seu propio conxunto independente de paquetes. Este módulo forma parte da biblioteca estándar de Python.

Control de versións

Git e GitHub

- **Git**. Git é un sistema de control de versións creado por Linus Torvalds. Está deseñado para manexar as complexidades tanto de aplicacións a pequena escala como de proxectos a gran escala, e a súa filosofía de deseño prioriza a facilidade de uso e o rendemento.

  O estrito control de versións de Git garante unha rastrexabilidade precisa dos cambios e a evolución do proxecto, o que o converte nunha ferramenta indispensable para garantir a calidade do desenvolvemento de software. Comparado con outros sistemas de control

[2] https://opensource.org/license/mit

de versións, como SVN, Git ten varias vantaxes que o consolidaron como o estándar de facto para a xestión do código fonte.

Unha destas vantaxes é a súa capacidade *offline*. A arquitectura de Git permite realizar operacións de control de versións tanto para repositorios locais como remotos. Esta característica garante que os programadores poidan manter a funcionalidade completa e realizar tarefas de control de versións localmente, independentemente da conectividade de rede.

A sólida compatibilidade de Git coa creación de ramas promove un enfoque non lineal do desenvolvemento, o que permite aos equipos traballar simultaneamente en varias funcións ou correccións de erros. Git maximiza a produtividade e axiliza a colaboración entre os membros do equipo ao facilitar fluxos de desenvolvemento paralelos.

- **GitHub**. GitHub é unha plataforma para aloxar repositorios Git e mellorar as prácticas de desenvolvemento colaborativo. Ofrece funcións para mellorar a colaboración en equipo, a revisión de código e a xestión de proxectos aproveitando as capacidades de Git. Os programadores poden utilizar a sinxela interface de GitHub para crear, bifurcar e fusionar repositorios. Tamén poden aproveitar ferramentas como o seguimento de incidencias, *pull requests* e wikis de proxectos. Ademais, GitHub ofrece servizos de integración e despregamento continuos (CI/CD), o que facilita as probas automatizadas e as canalizacións de despregamento e axiliza o ciclo de vida do desenvolvemento de software. Ademais, dada a súa popularidade, GitHub permite aos programadores mostrar o seu traballo, contribuír a proxectos de código aberto e participar en iniciativas de codificación colaborativa, enriquecendo o panorama mundial do desenvolvemento de software coa súa ampla comunidade de usuarios e o seu vasto ecosistema de repositorios.

Xestión do proxecto

- **Github Projects**. Como se explica na súa documentación[3], os GitHub Projects son ferramentas versátiles que se integran perfectamente coas incidencias e *pull requests* de GitHub, facilitando a planificación e o seguimento eficaz do traballo. Esto inclúe unha folla de cálculo adaptable, un taboleiro de tarefas e unha folla de ruta. Os usuarios poden xerar e personalizar múltiples vistas aplicando filtros, opcións de clasificación e mecanismos de agrupación ás súas incidencias e *pull requests*. Ademais, GitHub Projects ofrece a posibilidade de visualizar o traballo mediante gráficos personalizables e campos personalizados para realizar un seguimento dos metadatos relevantes para

[3] https://docs.github.com/es/issues/planning-and-tracking-with-projects/learning-about-projects/about-projects

as operacións do equipo. Esta flexibilidade permite aos equipos adaptar as funcións ás súas necesidades e fluxos de traballo específicos, en lugar de verse limitados por metodoloxías ríxidas.

En concreto, durante o desenvolvemento deste proxecto utilizouse principalmente a vista de taboleiro Kanban (ver Sección 2.1).

Capítulo 3

Análise do Plan de Benchmarking

Neste Capítulo preséntanse as tecnoloxías e ferramentas que se utilizaron para a realización deste proxecto.

3.1 Data Lakehouses

3.1.1 Snowflake

Snowflake é un Data Lakehouse baseado na nube que se presentou por primeira vez en 2015 con soporte para Amazon Web Services (AWS), aínda que desde entón se ampliou para incluír Microsoft Azure e Google Cloud Platform (GCP). A arquitectura de Snowflake, incluído o seu motor de procesamento, desenvolveuse desde cero para proporcionar escalabilidade e flexibilidade masivas para xestionar e analizar datos. As características clave da plataforma inclúen soporte completo para transaccións ACID, formatos de datos estruturados e semiestruturados como JSON ou Avro, elasticidade mediante o uso de recursos independentes de almacenamento e computación, e alta dispoñibilidade.

Existen tres edicións diferentes que se poden contratar: Standard, Enterprise e Business Critical; ordeadas de menor a maior custo por crédito. A primeira delas é o nivel introductorio, pero que ofrece acceso e ilimitado a todas as funcionalidades. A versión Enterprise ofrece as mesmas funcións e servizos que a Standard, engadindo funcionalidades a maiores pensadas especificamente para empresas que manexen grandes volumes de datos. Por último, Business Critical ofrece niveles maiores de seguridade protección de datos, pensada para organizacións que manexen datos sensibles. Pódense consultar con maior detalle as diferenzas entre as diferentes versións na propia documentación de Snowflake[1].

[1] https://docs.snowflake.com/en/user-guide/intro-editions

Arquitectura

A principal característica da arquitectura de Snowflake é a separación entre almacenamento e computación. A computación proporciónase a través do motor propietario de Snowflake, mentras que o almacenamento se pode implementar a través de Amazon S3, Azure Blob Storage ou Google Cloud Storage. Para reducir o tráfico de rede entre os nodos de computación e os nodos de almacenamento, cada nodo de computación almacena en caché algunhas táboas.

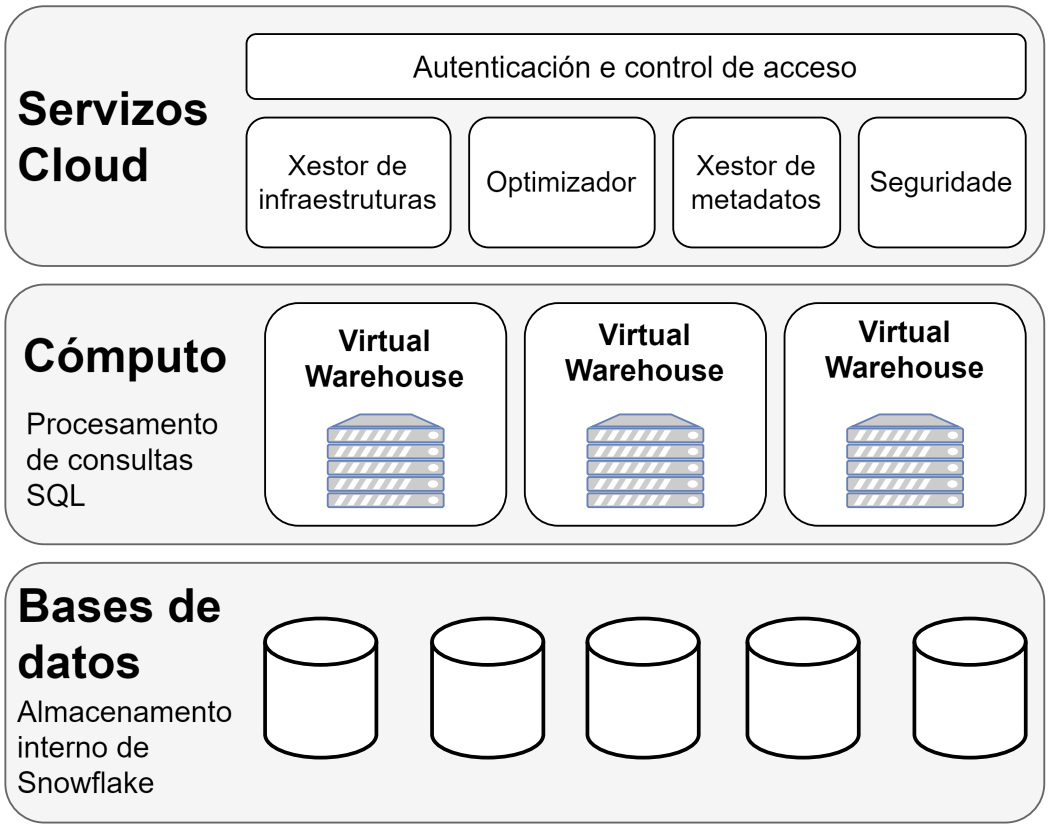


Figura 3.1: Arquitectura de Snowflake

Na Figura 3.1 móstrase un esquema da arquitectura Snowflake, onde se aprecia claramente a separación das tres compoñentes: Servizos na nube, procesamento de consultas e almacenamento [21].

- O almacenamento baséase nos nodos de almacenamento aloxados no provedor de nube, como pode ser Amazon S3. Para almacenar os datos Snowflake utiliza o seu formato interno de datos por columnas, mediante o que consegue compresión e optimización de consultas.
- No procesamento de consultas interveñen os nodos de computación, que forman a capa de procesamento. Estes nodos son Virtual Warehouse (VW), *clústers* elásticos de

máquinas virtuais independentes. Un VW proporciona os recursos necesarios, como CPU, memoria e almacenamento temporal, para realizar operacións nunha sesión de Snowflake, como sentenzas SQL SELECT ou operacións DML. Snowflake ofrece unha gama de VW de distintos tamaños e prezos, que van desde *X-Small*, o de menores capacidades, ata *4X-Large*. Con todo, os recursos de cada VW non se dan a coñecer, polo que se deben probar para determinar o tamaño adecuado para cada situación. Ademais, Snowflake ofrece actualmente dous tipos de VW:

  - Standard: recomendado para SQL e consultas DML, aínda que tamén é capaz de executar cargas de traballo de snowpark.
  - Snowpark-optimized: recomendado para executar cargas de traballo con grandes requisitos de memoria, como é o caso do adestramento de modelos de AA.

- A capa de servizos na nube xestiona varios servizos, como a autenticación, a xestión de infraestruturas, a xestión de metadatos, a optimización de consultas e o control de acceso. Esta capa é fortemente *multi-tenant*, o que significa que cada servizo é de longa duración e compartido por moitos usuarios. Para garantir unha alta dispoñibilidade e escalabilidade, cada servizo está replicado.

Sistema de almacenamento

Os datos en Snowflake almacénanse en táboas de bases de datos, estruturadas loxicamente como coleccións de columnas e filas. Como se mencionou anteriormente, a plataforma inicialmente elixida por Snowflake para almacenar estes datos foi AWS, xa que nese momento era a oferta máis madura na plataforma na nube e ofrecía o maior conxunto de usuarios potenciais [21].

A diferenza dun Data Warehouse normal, Snowflake implementa a súa propia forma de particionado denominada *micro-partitioning* [28]. Trátase de unidades contiguas de almacenamento, que conteñen entre 50 MB e 500 MB de datos sen comprimir (a compresión realízase cando se almacenan en Snowflake). As filas das táboas asígnanse a micro-particións individuais organizadas en columnas. Snowflake almacena metadatos sobre todas as filas almacenadas nunha micropartición, incluíndo:

- O rango de valores para cada unha das columnas da micro-partición.
- O número de valores distintos.
- Propiedades adicionais utilizadas tanto para a optimización como para o procesamento eficaz das consultas.

Esta característica achega varias vantaxes, dúas das cales son as máis destacadas por Snowflake. En primeiro lugar, as operacións DML poden aproveitar os metadatos subxacentes das micro-particións para facilitar e simplificar o mantemento das táboas. Doutra banda, a "poda de consultas" (query pruning), permite podar con precisión as columnas das micro-particións en tempo de execución da consulta, incluídas as columnas que conteñen datos semiestruturados [29].

Interfaces de consumo de datos

Os datos son consumidos polos VW mediante o motor de procesamento de Snowflake, columnar e vectorizado. Orixinalmente, podíase acceder aos datos de Snowflake en liña mediante consultas SQL, ofrecendo tamén extensións SQL con funcións incorporadas. Para escribir e executar consultas SQL, pódense utilizar worksheets da UI (Snowsight[2]) ou diversas ferramentas para conectarse a Snowflake, como o seu propio conector JDBC.

Para traballar con datos semiestruturados e sen esquema, Snowflake amplía o SQL estándar con tres tipos de datos: VARIANT, ARRAY e OBJECT, onde os dous últimos non son máis que restricións do tipo VARIANT. Mediante estes tipos de datos dáse soporte aos formatos semiestruturados JSON, Avro, ORC, Parquet e XML.

Na actualidade, pódese acceder aos datos de Snowflake, así como xestionalos, desde Python, Java ou Scala utilizando snowpark, un marco de desenvolvemento software que amplía o rendemento, a gobernanza e a elasticidade de Snowflake máis aló de SQL para soportar de forma nativa estas tres plataformas. A Figura 3.2 mostra dúas formas de utilizar Snowpark, no lado do cliente ou no lado do servidor:

- Lado do cliente: pódese programar e executar localmente na nosa contorna para que se conecte a Snowflake e se execute nos seus VW.
  - API DataFrame: escribe consultas e transformacións de datos utilizando os seus propios dataframes, similares aos de Pandas ou Spark. O procesamento realízase no motor de procesamento elástico de Snowflake, que tamén permite recuperar os datos localmente.
  - Snowpark ML API: a parte de modelado permite adestrar modelos de AA directamente en Snowflake, unha funcionalidade que actualmente se atopa en vista previa pública. En conxunto coa parte de operacións pódese obter un bo ecosistema de MLOps, pero esta última polo momento non está aberta ao público.
- Lado do servidor: almacenado en Snowflake, usando notebooks dipoñibles en Snowsight.

[2] https://docs.snowflake.com/en/user-guide/ui-snowsight

- User Defined Function (UDF) ou procedementos almacenados escritos en calquera das tres linguaxes que se executan nos VW cando é necesario. Poden programarse, utilizarse en consultas SQL, etc.
- Snowpark Container Services: permite aos programadores despregar, xestionar e escalar sen esforzo cargas de traballo en contedores (jobs, servizos, funcións de servizo) utilizando unha infraestrutura segura xestionada por Snowflake con opcións de hardware configurables, como as GPU. Actualmente en vista previa pública.

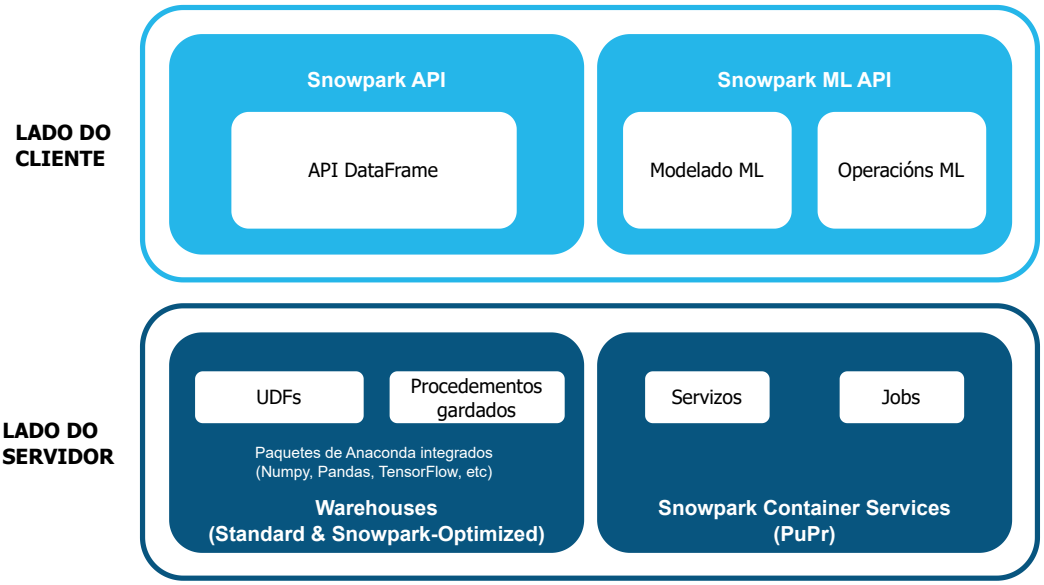


Figura 3.2: Arquitectura de Snowpark

Prezos

Dado que a arquitectura Snowflake se divide en tres capas distintas, o prezo baséase no uso real de cada capa e as funcións sen servidor.

- Almacenamento: cobro mensual en función do uso. A compresión que realiza Snowflake reduce os custos.
- Virtual Warehouse: pagados con créditos de Snowflake por segundos de uso.
- Servizos na nube: Idéntico aos Virtual Warehouse.

Os créditos Snowflake son unha unidade de medida utilizada para pagar o consumo de recursos en Snowflake. O valor dos créditos depende da edición de Snowflake (Standard, Enterprise ou Business Critical), do provedor da nube (AWS, Azure ou GCP) e da rexión.

Por exemplo, se se utiliza AWS en Canadá (Central) coa edición On Demand Standard, o custo por crédito é de 2,25 $. Doutra banda, se se utiliza Azure en Europa Occidental (Países Baixos) coa edición Estándar, o custo por crédito é de 2,60 $[3]. En xeral, cambiando entre provedores de cloud apenas se producen cambios; as maiores diferenzas prodúcense cambiando a rexión (sendo por norma xeral as situadas en Estados Unidos as máis económicas, e contando as de fóra de Estados Unidos cun prezo similar entre elas). Ademais, tamén se debe ter en conta o número de rexións por provedor: 23 por AWS, 15 por Azure e 4 por GCP, dúas en Estados Unidos e dúas en Europa.

Databricks ofrece a súa calculadora oficial[4] para calcular costes en base a tipo de recurso de cómputo, tipo de instancia do provedor de cloud, número de instacias, horas/día e días/mes; e a partir desa información realizar conversións entre DBUs e dólares. Porén, Snowflake non ofrece nada similar. Debido a isto, se se queren utilizar ferramentas con esas funcionalidades, deben ser ferramentas non oficiais, como a desenvolta por *cleartelligence*, que ofrece funcionalidades similares á calculadora de Databricks, e a maiores ofrece visualizacións e gráficas sobre proxeccións de gasto por uso e almacenamento.

3.1.2 Databricks

Databricks é unha plataforma de analítica unificada e aberta para construír, despregar, compartir e manter solucións de datos, analítica e IA. Intégrase co almacenamento na nube nas tres plataformas cloud máis populares: AWS, Azure e GCP; e xestiona e desprega infraestrutura na nube no seu nome. Como outras plataformas de Data Lakehouse, Databricks proporciona ferramentas que axudan a conectar fontes de datos a unha plataforma para procesar, almacenar, compartir, analizar, modelar e monetizar datasets con solucións desde Business Intelligence (BI) a IA xerativa.

Arquitectura

Databricks funciona a partir dunha capa de control e unha capa de cálculo.

- A capa de control inclúe os servizos backend que Databricks xestiona na conta de Databricks. Os comandos executados nos notebooks e moitas outras configuracións do espazo de traballo almacénanse na capa de control e cífranse en repouso.

- A capa de cómputo é onde os datos son procesados.

[3] https://www.snowflake.com/pricing/pricing-guide/?utm_cta=website-pricing-page-optimized-storage-pricing-guide

[4] https://www.databricks.com/product/pricing/product-pricing/instance-types

- Para a maioría dos cálculos de Databricks, os recursos de cómputo atópanse na conta do provedor da nube no que se denomina a capa de cómputo clásico. Databricks utiliza a capa de cómputo clásico para notebooks, jobs e Databricks SQL warehouses, tanto na versión *pro* como na *classic*.
- Para Databricks SQL warehouses na versión *serverless*, os recursos de cómputo *serverless* execútanse nunha capa de cómputo *serverless* na conta de Databricks.

Como se pode ver na Figura 3.3, Databricks ofrece tres tipos distintos de SQL Warehouses: *serverless*, *pro* e *classic*. O primeiro deles é o máis distintos dos demais, posto que se executan na capa de cómputo da conta de Databricks. Polo contrario, tanto os Data Warehouses *pro* coma os *classic* se executan en recursos da subscrición do provedor cloud. Entre *pro* e *classic* a principal diferenza é que os Data Warehouses *pro* poden utilizar *Predictive IO*, un conxunto de funcións para acelerar aas operacións de exploración selectiva nas consultas SQL. *Predictive IO* pode proporcionar unha amplia gama de melloras de velocidade.

Os conectores Databricks permiten conectar *clústers* a fontes de datos externas fóra da conta do provedor de cloud para inxerir datos ou para o seu almacenamento. Tamén é posible inxerir datos de fontes de datos de streaming externas, como datos de eventos, streaming, IoT, etc.

O Data Lake almacénase en repouso na conta do provedor cloud e nas súas fontes de datos para que os usuarios manteñan o control e a propiedade dos seus datos.

Os resultados dos jobs almacénanse na conta do provedor cloud. Para os resultados dos notebooks, o almacenamento realízase nunha combinación do plano de control (*control plane*) (resultados parciais para a súa presentación na interface de usuario e o almacenamento do provedor cloud). Se os usuarios desexan que os resultados dos notebooks se almacenen unicamente na súa conta do provedor cloud, poden configurar a localización de almacenamento para os resultados dos notebooks.

Aínda que as arquitecturas poden variar en función das configuracións personalizadas, o diagrama da Figura 3.3 representa a estrutura e o fluxo de datos máis habituais para as contornas de Databricks.

Almacenamento

Databricks utiliza tanto volumes de almacenamento en bloques como localizacións de almacenamento de obxectos nun servizo na nube [30]. Na computación en nube, o almacenamento de obxectos (almacenamento *blob*) refírese a contedores de almacenamento que manteñen os datos como obxectos, nos que cada obxecto consta de datos, metadatos e un identificador de recurso único global (URI). Doutra banda, o almacenamento en bloques ou almacenamento en disco refírese a volumes de almacenamento que corresponden a unidades

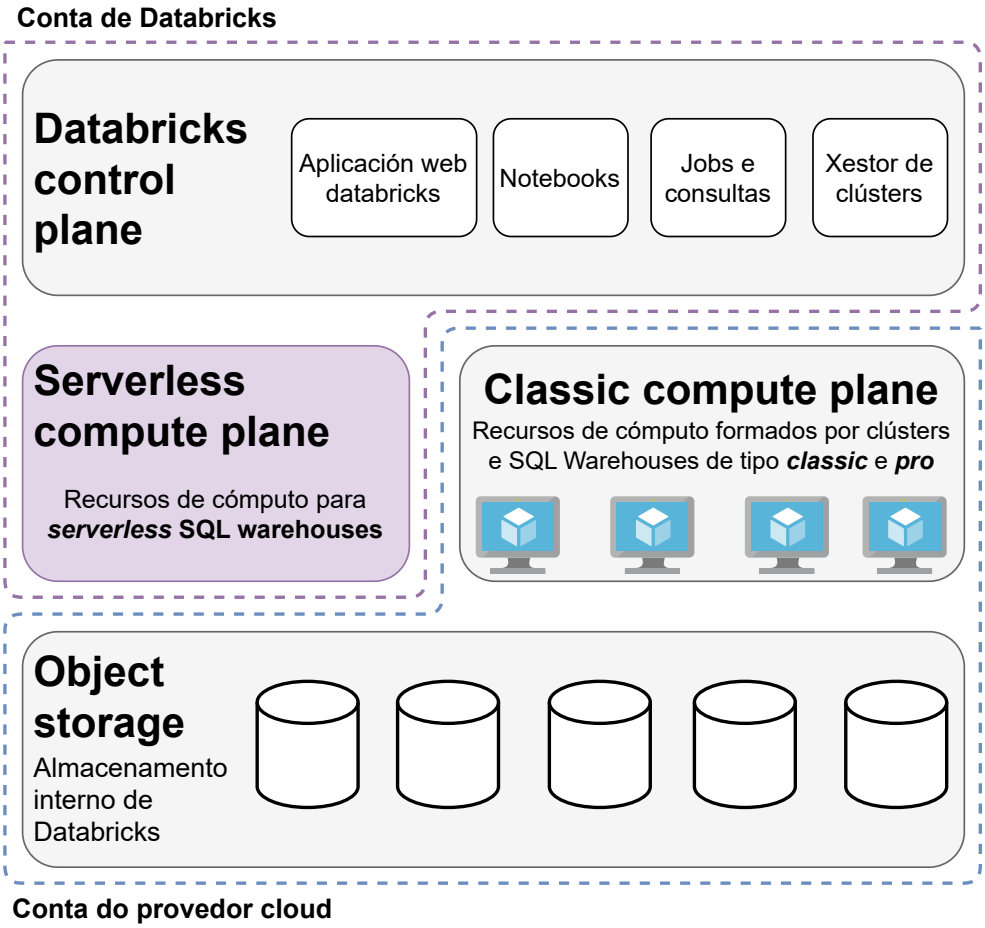


Figura 3.3: Arquitectura de Databricks

de disco duro tradicionais (HDD) ou unidades de estado sólido (SSD).

- Object storage: por defecto utiliza Databricks File System (DBFS), aínda que se pode configurar para utilizar un sistema externo. Pódese acceder mediante Spark.

- Block storage: As máquinas virtuais na nube precisan dun almacenamento en bloques adxunto, polo que Databricks configura e desprega as máquinas virtuais con este tipo de almacenamento, e utilízao para ficheiros de datos efímeros durante a vida útil do cómputo. Neste almacenamento é onde se atopan o sistema operativo e as distintas bibliotecas instaladas. Ademais, os programadores poden utilizalo para gardar e cargar ficheiros.

Interfaces de consumo de datos

Databricks ofrece os seus propios notebooks, que soportan linguaxes como Scala, Python e R [31]; pero tamén proporciona soporte para scripts SQL. Ademais, tamén soporta bibliotecas populares para procesado de datos, como Pandas.

Ademais disto, tamén se pode escribir código fóra da contorna Databricks e conectarse á plataforma usando conectores oficiais como o *Databricks SQL connector for Python*[5].

Prezos

Os prezos de Databricks baséanse en unidades Databricks (DBU), unidades de capacidade de procesamento por hora baseadas no tipo de instancia. Así, Databricks ofrece un enfoque de pago por uso, exactamente o mesmo concepto que Snowflake cos créditos (aínda que non existen correspondencias exactas entre os prezos das dúas plataformas).

Existen tres plans de conta: Estándar, *Premium* e *Enterprise*. Cada servizo ofrecido por Databricks ten un prezo en unidades de $/DBU, e pode variar en función do tipo de conta, o provedor de nube utilizado e a súa rexión [32]. Como se mencionou na Sección 3.1.1, Databricks ofrece a súa calculadora oficial[6] para calcular costes en base a tipo de recurso de cómputo, tipo de instancia do provedor de cloud, número de instacias, horas/día e días/mes; e a partir desa información realizar conversións entre DBUs e dólares.

3.2 Benchmarking de Data Lakehouse

Avaliar o rendemento dun sistema informático resulta un reto importante. As únicas métricas fiables son o custo (relación prezo/rendemento) e a capacidade e velocidade de procesamento de datos. Os intentos de proporcionar estas métricas preséntanse en forma de benchmarks, que agrupan múltiples *microprobas* e simulacións de cargas de traballo [33]. Neste punto entra en xogo Transaction Processing Performance Council (TPC), unha organización sen ánimo de lucro constituída por axentes da industria, creada para definir os benchmarks de referencia en distintos ámbitos. Así, é posible distribuír información xusta e verificable entre interesados en distintas tecnoloxías. Os benchmarks desenvolvidos por TPC permiten comparar o rendemento dos sistemas. Estes benchmarks ofrecen aos provedores a oportunidade de mellorar os seus produtos e publicar os seus resultados en liña para comparalos cos da competencia, normalmente previo pago.

En concreto, entre todos os benchmarks da familia TPC, este proxecto centra a atención en TPC Decision Support (TPC-DS), detallado na Sección 3.2.1.

[5] https://docs.databricks.com/en/dev-tools/python-sql-connector.html
[6] https://www.databricks.com/product/pricing/product-pricing/instance-types

3.2.1 TPC-DS

TPC-DS modela un Data Warehouse, centrándose nas tarefas de Online Analytical Processing (OLAP). Modela as funcións de soporte á toma de decisións dun provedor retallista de produtos, o que facilita a familiaridade cos compoñentes do benchmark [34]. O retallista, aínda que ficticio, vende os seus produtos a través de tres canles de distribución: tenda, catálogo e Internet.

Dataset

TPC-DS implementa un esquema de floco de neve múltiple, que é un "enfoque híbrido entre un esquema 3NF e un esquema de estrela puro" [34].

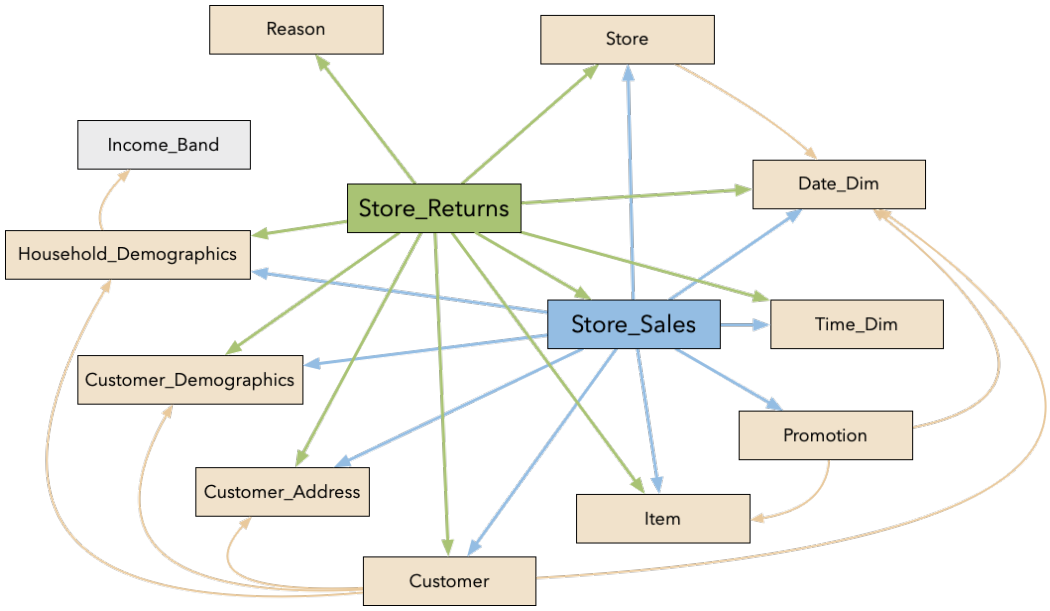


Figura 3.4: Extracto do diagrama entidade-relación de TPC-DS

O diagrama da Figura 3.4 mostra unha parte do esquema de TPC-DS a través dun formato entidade-relación. Nel pódense ver dúas das seis táboas de feitos, *Store_Sales* e *Store_Returns*, xunto coas táboas de dimensión asociadas como *Customer* e *Store*.

As relacións verdes e azuis ilustran o enfoque convencional do esquema en estrela, mentres que as relacións laranxas mostran a normalización das táboas de dimensión, introducida polo enfoque do esquema en floco de neve. As táboas normalizadas permiten as relacións entre si, e tamén poden separarse das táboas de feitos, como demostra a táboa *Income_Band*. Deste xeito, TPC-DS emprega un complexo esquema de floco de neve composto por 24 táboas en total.

Despois de deseñar o esquema da base de datos de proba, é esencial encher as táboas con

datos. Para determinar o tamaño da base de datos, TPC-DS introduce o concepto de *scale factors*, que consiste en valores discretos que miden o volume de datos en *gigabytes*. Os *scale factors* van de 1 (1 GB, só apto para cualificación) a 100.000 (100 TB). Neste contexto, é esencial ter en conta que os resultados do benchmark limítanse a *scale factors* específicos e non son comparables aos resultados obtidos con outros *scale factors*.

No tocante ao escalado de datos, é crucial diferenciar o escalado de dominios e o de tuplas. Este último refírese ao número de tuplas presentes nas táboas de feitos, que se escalan linealmente co *scale factor* en TPC-DS. O escalado de dominios refírese ás táboas de dimensións, que xeralmente non escalan linealmente. TPC-DS escala os seus dominios de forma sublineal e evita as relacións de táboas pouco realistas presentes en TPC-H (un antigo benchmark de apoio á toma de decisións) [34]. Ademais, a proporción de clientes, artigos, tendas e outras dimensións segue sendo realista, mesmo con *scale factors* de gran tamaño.

A xeración de valores en TPC-DS utiliza modelos matemáticos e distribucións ben estudadas (como as distribucións Normal ou Poisson). Os datasets sintéticos resultantes destas distribucións posúen varias vantaxes para os benchmarks, pero tamén presentan un notable inconveniente: non son compatibles cunha técnica de variables vinculantes empregada en TPC-DS para reducir a predictibilidade [34]. Por iso, TPC-DS sintetiza datos do mundo real para varias distribucións críticas para eludir este problema. Para máis detalles sobre este proceso, pódese consultar o artigo "The making of TPC-DS" [34].

Consultas de TPC-DS

TPC-DS define un conxunto de 99 consultas SQL diferentes (incluídas as extensións OLAP) deseñadas para cubrir todo o dataset [34]. O deseño do esquema permite dous tipos de consulta distintos: unha sección de informes (canle de venda por catálogo, que representa o 40% do dataset) e un segmento ad-hoc (canles de venda por Internet e en tenda). A carga de traballo de consulta contén catro clases de consulta que inclúen consultas de informes puras, consultas ad-hoc puras, consultas iterativas e consultas de extracción ou minaría de datos.

As consultas de informes son coñecidas de antemán, o que permite utilizar métodos de optimización como a colocación de datos e estruturas de datos auxiliares (por exemplo, vistas materializadas e índices), tal e como establece a especificación de TPC-DS.[7].

As consultas ad-hoc, pola súa banda, están deseñadas para simular consultas de usuario que non están predeterminadas. As optimizacións explícitas mediante estruturas de datos auxiliares (como DDL, opcións de sesión e parámetros de configuración global) non están permitidas para a sección ad-hoc do esquema.

As consultas nas que interveñen as seccións ad-hoc e de información do esquema, como

[7] https://www.tpc.org/TPC_Documents_Current_Versions/pdf/TPC-DS_v3.2.0.pdf)

as consultas OLAP iterativas e de minaría de datos, clasifícanse como consultas híbridas e entran dentro de ambos os tipos [34].

Curiosamente, ningunha das consultas é completamente estática, nin sequera as utilizadas para a elaboración de informes. No seu lugar, utilízase un modelo con substitucións pseudo-aleatorias para crear todas as consultas co fin de modelar o uso dinámico do DSS. Con todo, o modelo está estreitamente vinculado ao xerador de datos para garantir a comparabilidade das consultas entre substitucións [34].

Métricas de TPC-DS

Os benchmarks TPC ofrecen métricas sinxelas para comparar o rendemento dos sistemas. No caso de TPC-DS, son importantes catro intervalos de tempo específicos, que se poden consultar na Figura 3.5.

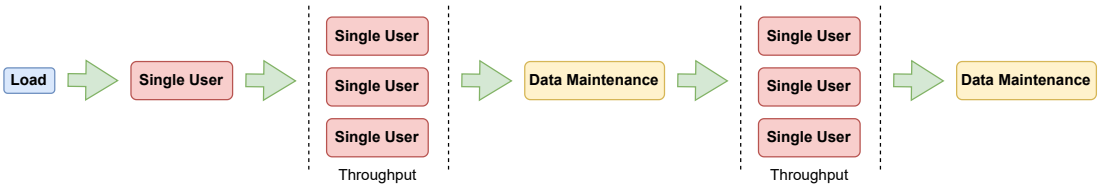


Figura 3.5: Orde de execución das consultas de TPC-DS

Dita figura mostra de forma efectiva os períodos nos que TPC-DS avalía os tempos de execución. Nela faise referencia ás seguintes etapas:

- *Data Load.*
- *Power Test (Single User).*
- *Throughput Test.*
- *Data Maintenance.*

Unha vez rexistrados os tempos necesarios, TPC-DS ofrece diferentes métricas a obter a partir deles. A métrica principal é a métrica de rendemento, que se pode ver na Ecuación 3.1.

$$QphDS@SF = \left\lfloor \frac{SF \cdot Q}{\sqrt[4]{T_{PT} \cdot T_{TT} \cdot T_{DM} \cdot T_{LD}}} \right\rfloor \tag{3.1}$$

Na Fórmula 3.1, *SF* representa o *scale factor*, mentres que Q representa o número de consultas: $Q = S_q \cdot 99$, sendo S_q o número de fluxos executados no *Throughput Test.* TPC-DS emprega estes fluxos de consultas para simular múltiples usuarios operando en paralelo. Como resultado, cada fluxo de consulta executa as 99 consultas de TPC-DS secuencialmente,

pero cada un nunha orde diferente. O número mínimo de fluxos de consulta necesarios depende do *scale factor* elixido e está documentado na especificación oficial. En xeral, canto maior é o *scale factor*, maior é o número de fluxos de consulta que deben procesarse. Como se explica na especificación, o resto de siglas significan o seguinte:

- $T_{PT} = T_{POWER} \cdot S_q$, sendo T_{POWER} o tempo total empregado en completar o *Power Test*.
- $T_{TT} = T_{TT1} + T_{TT2}$, sendo T_{TT1} o tempo total empregado en completar o *Throughput Test* 1, e T_{TT2} o tempo empregado en completar o *Throughput Test* 2.
- $T_{DM} = T_{DM1} + T_{DM2}$, sendo T_{DM1} o tempo total empregado en completar o *Data Maintenance Test* 1, e T_{DM2} o tempo empregado en completar o *Data Maintenance Test* 2.
- $T_{LD} = 0.01 \cdot S_q \cdot T_{LOAD}$, sendo T_{LOAD} o tempo no que se executou o *Load Test*.
- T_{PT}, T_{TT}, T_{DM} e T_{LD} están en unidades de horas decimais.

A pesar de que a métrica de rendemento é a máis popular, tamén se utiliza a métrica de prezo-rendemento (Ecuación 3.2, sendo P o prezo ($) do sistema). A maiores, tamén se poden empregar como métricas secundarias o tempo total da fase *Load Data* e o das fases *Power Test* (tanto total coma separado por cada consulta) e *Data Maintenance*.

$$\$/kQphDS@SF = \frac{1000 \cdot P}{QphDS@SF} \tag{3.2}$$

Retos e limitacións

O principal inconveniente de TPC-DS de cara a utilizalo neste proxecto é a súa limitación aos datos estruturados. TPC-DS emprega exclusivamente datos estruturados, mentres que os Data Lakehouses poden xestionar datos semiestruturados e non estruturados. Existen outros benchmarks, como TPCx-BB[8], que tamén proban as capacidades para xestionar datos semiestruturados semiestruturados (como *weblogs*) e non estruturados (como *reviews*); pero que non está adaptado a Data Lakehouses na nube, polo que non foi posible utilizalo neste proxecto.

[8] https://www.tpc.org/tpcx-bb/

Capítulo 4

Desenvolvemento do Plan de Benchmarking

Neste Capítulo detállase a proposta do proxecto. Deste modo, na Sección 4.1 explícase a implementación que se realiza para executar o benchmark TPC-DS, e na Sección 4.2 detállase a proba realizada de conectividade entre Snowflake e Databricks e Power BI.

4.1 Deseño e Implementación da Ferramenta de Benchmarking

Nesta Sección detállanse diferentes aspectos de CLADE-TPC-DS, a ferramenta desenvolvida para dar resposta aos requisitos vistos na análise do plan de *benchmarking* para implementar TPC-DS. Na páxina oficial do benchmark[1] pódense descargar tanto un documento coa súa especificación coma un *kit* con ferramentas útiles para a súa implementación. De entre as utilidades proporcionadas nese *kit*, as principais son as seguintes:

- **tpcds.sql**: esquema SQL que define as 24 táboas que conforman a base de datos, especificando as súas columnas e tipos de datos.
- **dsdgen**: ficheiro binario (non se atopa directamente no *kit*. Para obtelo é preciso compilar previamente os ficheiros fonte, que si se atopan entre os provistos) que permite xerar en ficheiros CSV os datos que posteriormente van encher as táboas.
- **dsqgen**: ficheiro binario que, como ocorre con **dsdgen**, para obtelo é necesario compilar previamente os ficheiros fonte. Permite xerar, a partir duns modelos proporcionados no *kit*, as 99 consultas SQL que conforman o benchmark.

Deste xeito, a forma máis sinxela de implementar o benchmark é mediante unha aplicación que de forma local execute as consultas xeradas por dsqgen nas plataformas a probar. Porén,

[1] https://www.tpc.org/tpc_documents_current_versions/current_specifications5.asp

dado que tamén é necesario medir o tempo de carga dos datos nas plataformas, subir os datos desde o ordenador local non é unha opción, pois a medición veríase totalmente alterada pola velocidade da conexión a Internet que utilice o ordenador. Por isto, finalmente decidiuse implementar unha arquitectura como a que se pode consultar na Figura 4.1. Como se pode ver, a decisión final foi almacenar estes datos na nube, nun *Azure Blob Storage*[2], de forma que é este último o que realiza a conexión coas plataformas a probar. Así, a latencia entre a fonte de datos e a plataforma a probar non depende do usuario, senón do Blob de Azure e a plataforma, que se asume constante (permitindo unhas lixeiros marxes).

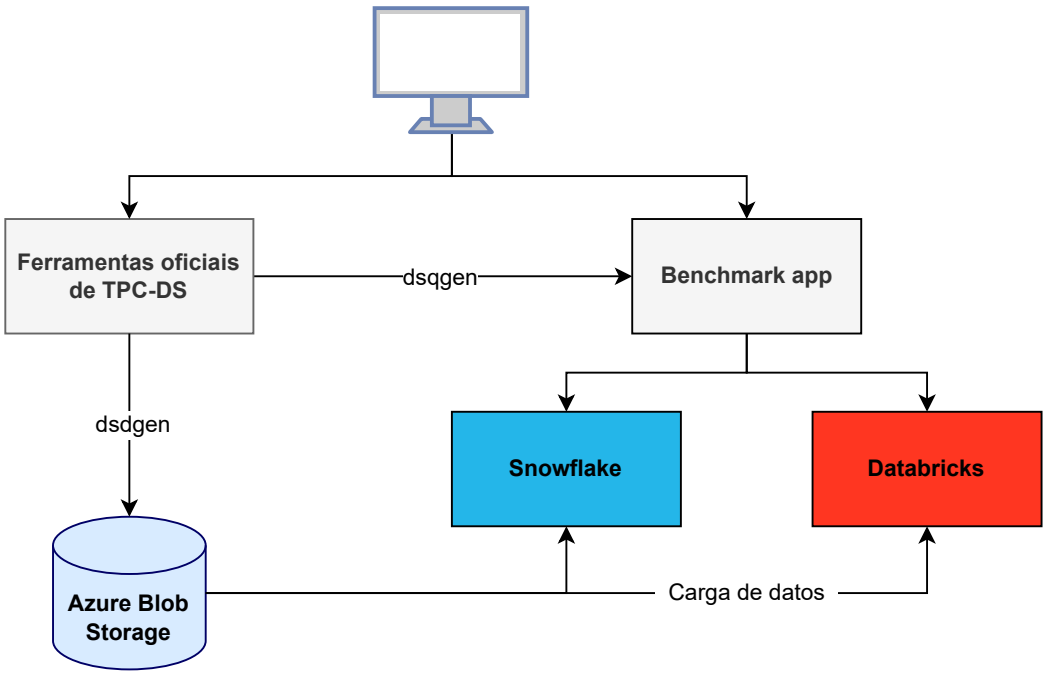


Figura 4.1: Arquitectura da implementación realizada de TPC-DS

Así, o que mostra o esquema da Figura 4.1 é o seguinte:

- **Ferramentas oficiais de TPC-DS**: ferramentas desenvolvidas por TPC. En concreto as dúas seguintes: **dsdgen**, utilizada para xerar os datos do benchmark en formato CSV; e **dsqgen**, empregada para xerar as consultas SQL coas que se avalían os sistemas.

- **Benchmark app**: aplicación CLI desenvolvida en Python para este proxecto. Permite subir os datos ás plataformas para avaliar, lanzar as consultas cun número dado de repeticións e fíos e obter as métricas para a avaliación. Nas seguintes Seccións (4.1.1, 4.1.2 e 4.1.3) danse máis detalles acerca da implementación e funcionamento.

[2] https://azure.microsoft.com/es-es/products/storage/blobs

- **Azure Blob Storage**: sistema de almacenamento en liña empregado para gardar os datos xerados por dsdgen. Deste xeito, disponse o dataset de TPC-DS cos *scale factors* cos que se traballou: 1, 10, 25, 50 e 100.

- **Snowflake e Databricks**: sistemas a evaluar utilizando a aplicación. Estes sistemas terán que ler os datos desde o *Blob Storage* e cargalos nunha base de datos do seu almacenamento interno, para despois executar as consultas SQL. Como se mencionou na Sección 2.5.1, estas consultas están no dialecto SQL92, e son as mesmas as que se utilizan para Snowflake e para Databricks, que se executan en SQL Warehouses *serverless* (ofrécese maior detalle sobre estas configuracións no Capítulo 5).

4.1.1 Deseño Software

A aplicación presenta tres comandos a través dos cales interactuar con ela: **load_data**, **execute_queries** e **data_maintenance**, cuxa función é cargar o dataset desde o *Blob* na plataforma a probar, lanzar as consultas na plataforma e executar a fase de *Data Maintenance*, respectivamente. Ademais, existe un módulo para obter tanto as diferentes métricas do benchmark coma gráficas de interese. Porén, este módulo tan só está formado por funcións Python ás que se chama de forma consecutiva (en forma tanto de script coma de notebook), pola que esta Sección centrarase na propia implementación da aplicación, da que destacan os seguintes aspectos:

- **BenchBase**: Clase utilizada a modo de interface, de modo que as plataformas nas que se implemente o benchmark (por agora Snowflake e Databricks) deben implementar os métodos indicados nesta clase. Formado por set_up_connection, load_data, execute_queries e data_maintenance. Tamén se utiliza para pasar os parámetros lidos a través dun ficheiro de configuración ou CLI a atributos de clase, que posteriormente se poden utilizar.

- **Patrón Método Factoría**. Nas Figuras 4.1 e 4.2 pódese ver como se implementa a instanciación das clases que executan o benchmark utilizando o patrón de deseño Método Factoría. O uso deste patrón de deseño permite encapsular o proceso de creación de obxectos no método *create_bench()*. Así, o seu uso permite engadir sen problemas novas clases de benchmarking específicas da plataforma sen necesidade de modificar o código existente, o que favorece o mantemento e a extensibilidade. Isto, ademais de axilizar a instanciación de obxectos, tamén mellora a claridade xeral e o mantemento do código. Grazas a esto, cúmplense os seguintes principios SOLID:

- **Single Responsibility**: A creación de obxectos das distintas clases de benchmarking realízase nun único punto, facendo o código máis sinxelo de manter.
- **Open/Closed**: Pódense introducir novas clases de benchmarking (implementacións para distintas plataformas) sen editar o código dos scripts que lanzan as execucións.

Para maior aclaración, na Figura 4.2 ilústrase cal é o diagrama de clases do patrón de deseño Método Factoría, mentras que na Figura 4.3 se ilustra a implementación realizada.

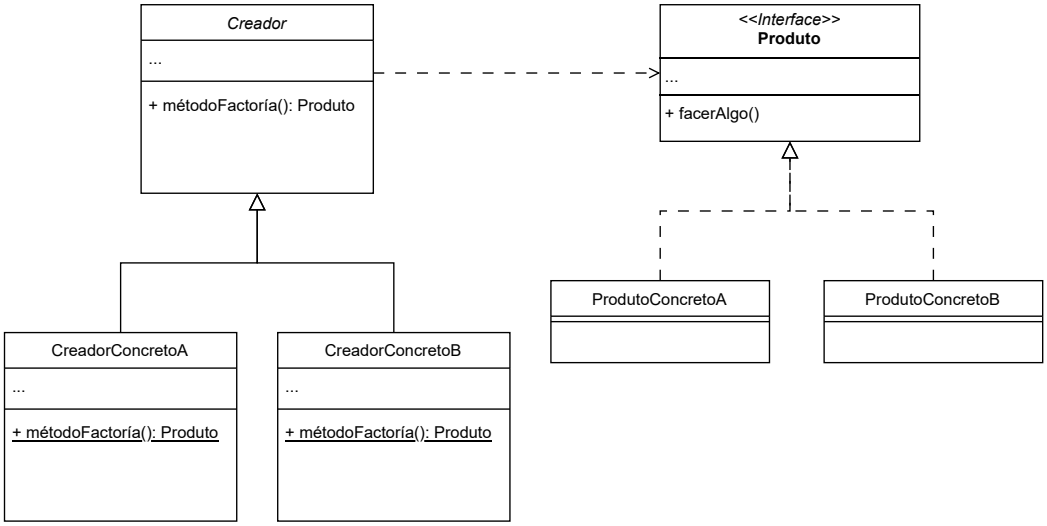


Figura 4.2: Diagrama de clases do patrón de deseño Método Factoría

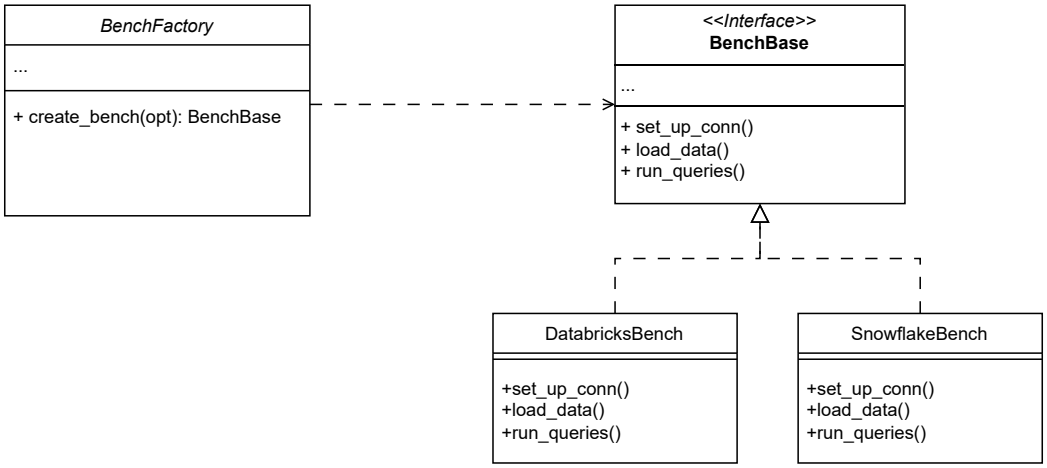


Figura 4.3: Diagrama de clases da implementación do patrón de deseño Método Factoría

```python
if __name__ == "__main__":
    opt = LoadOptions()
    bench = BenchFactory.create_bench(opt)
    cs = bench.set_up_conn()
    bench.load_data(cs, opt.get_load_options())
```

Listing 4.1: Código de load_data.py

```python
class BenchFactory:

    @staticmethod
    def create_bench(opt):
        """
        Create a benchmarking object based on the platform
    specified in the options.

        Args:
            opt: Options object.
        """
        if opt.get_platform() == "databricks":
            return DatabricksBench(opt)
        elif opt.get_platform() == "snowflake":
            return SnowflakeBench(opt)
        else:
            raise ValueError(f"Invalid platform:
    {opt.get_platform()}")
```

Listing 4.2: Código de bench_factory.py

4.1.2 Funcionalidades requeridas

A aplicación que se desenvolveu cobre as seguinte funcionalidades[3]:

- Subida dos datos a Snowflake e Databricks. A ferramenta proporciona a posibilidade de subir os datos á plataforma a probar, da que tan só é necesario incluir a dirección e as credenciais para escribir os ficheiros. Estes indícanse nun ficheiro de configuración, que se pode consultar no Anexo A.

- Execución das consultas en Snowflake e Databricks. Permítese axustar o número de fíos concurrentes, o que permitiu implementar tanto o *Power Test* coma o *Throughput Test* de TPC-DS.

[3] existe un terceiro *endpoint*, *data_maintenance*, que executa a fase Data Maintenance de TPC-DS, pero non se inclúe nestes puntos porque esa fase tan só é unha modificación das dúas anteriores.

- Obtención de métricas e gráficas. Implementáronse, tanto en scripts coma en notebooks, métodos para a obtención tanto de métricas do rendemento das plataformas na execución do benchmark coma gráficas de interese. As gráficas mostradas no Capítulo 5 obtivéronse utilizando esta implementación.

4.1.3 Testing

Na actualidade resulta dabondo coñecida a importancia de realizar un *testing* completo e exhaustivo de calquera proxecto de desenvolvemento de *software* [35]. Por isto, utilizouse a biblioteca Pytest[4] para a implementación de probas unitarias. Estas permitiron comprobar durante o desenvolvemento do proxecto que o código escrito era eficaz. Ademais, tamén permitiron durante o desenvolvemento a execución completa do código sen necesidade de realizar as consultas nas plataformas a probar, o que permitiu aforrar os custos destas, así como unha non despreciable cantidade de diñeiro e tempo. Para isto foi necesaria a implementación dunha clase *maqueta*: en lugar de traballar con conexións reais ás plataformas, creouse unha clase que imita o comportamento dos conectores reais. Esta maqueta implementa os mesmos métodos que os verdadeiros conectores, pero en lugar de realizar operacións na nube simula as respostas esperadas. Isto permite executar o *workload* completo do benchmark sen executar realmente as consultas na plataforma.

4.2 Inxesta a Power BI

Power BI é unha ferramenta de BI desenvolvida por Microsoft que permite aos usuarios visualizar datos, compartir información e tomar decisións baseadas en datos. Proporciona cadros de mando e informes interactivos, intégrase con varias fontes de datos e ofrece funcións para preparar, descubrir e compartir datos. Power BI está deseñado para ser fácil de usar, permitindo tanto a usuarios técnicos como non técnicos crear e colaborar en potentes visualizacións e análises. Unha das múltiples formas de empregalo é utilizando unha plataforma Data Lakehouse para almacenar datos, e conectándoa a Power BI. Porén, non existe ningún benchmark da familia de TPC ou similar que avalíe o rendemento destes conectores. Por isto, e debido tamén á súa crecente popularidade [36], decidiuse explorar as posibilidades que ofrecen tanto Snowflake coma Databricks para utilizar as súas plataformas como fontes de datos para inxestar a Power BI.

Para isto implementouse, seguindo a documentación oficial, a "conexión manual de Power BI Desktop a Azure Databricks" [5] e o "conector ao almacén de datos de Snowflake desde o

[4] https://docs.pytest.org/en/8.0.x/
[5] https://learn.microsoft.com/es-es/azure/databricks/partners/bi/power-bi

espazo de traballo de Power Query" [6]. Deste modo trátase avaliar cualitativamente as funcionalidades que ofrece cada un destes conectores, así como avaliar o rendemento que ofrecen á hora de transferir os datos desde as plataformas onde están almacenados ata Power BI. Estes experimentos están detallados na Sección 5.3.1.

[6] https://learn.microsoft.com/es-es/power-query/connectors/snowflake

Capítulo 5

Avaliación do Rendemento

Neste Capítulo explícanse os experimentos realizados a partir da proposta do Capítulo 4, así coma os resultados obtidos. Así, existen dúas seccións: na primeira, a 5.2, detállanse os experimentos realizados a través da implementación de TPC-DS que se realizou. Estes divídense en dous: un primeiro grupo de experimentos nos que tan só se executaron dúas fases concretas de TPC-DS, Load Data e Power Test; e un segundo grupo no que se executou o benchmark completo: unha fase de Load Data, unha de Power Test, dúas de Throughput Test e dúas de Data Maintenance. Na segunda sección, a 5.3, precísanse as probas que se realizaron utilizando os conectores que existen entre Power BI e as plataformas probadas.

5.1 Configuración experimental

Os recursos de cómputo empregados nos experimentos pódense consultar na Táboa 5.1. Nela compáranse Data Warehouses de Snowflake con SQL Warehouses de Databricks. Os prezos de Snowflake fan referencia á configuración utilizada, provedor AWS e rexión *West Europe Paris*. O mesmo ocorre con Databricks, onde se fai referencia a SQL Warehouses de tipo *serverless*, con provedor Azure e rexión *West Europe*. Ademais, ambas plataformas, a non ser que se indique o contrario (como ocorre na Sección 5.2.2), len os datos dun Azure Blob Storage que se atopa na rexión *West Europe*.

Snowflake			Databricks		
Tamaño	Créditos/h	$/h	Tamaño	DBU/h	$/h
Small	2	5.20$	X-Small	6	5.46$
Medium	4	10.40$	Small	12	10.92$
Large	8	20.80$	Medium	24	21.84$

Táboa 5.1: Warehouses utilizados en Snowflake e Databricks

5.2 Avaliación Empregando CLADE-TPC-DS

Utilizando CLADE-TPC-DS, a ferramenta implementada, realizáronse diferentes experimentos co benchmark TPC-DS. Dado que é un benchmark complexo, formado por distintas fases (ver Figura 3.5), decidiuse realizar a maioría de experimentos acurtando o *workflow* do benchmark: En lugar de executar todas as fases que se ven na Figura 3.5 decidiuse realizar unha serie de experimentos nas que tan só se cargan os datos na plataforma (fase Load Data) e se executan todas as consultas en modo monousuario (fase Power Test). Deste modo consíguese unha execución máis áxil do benchmark aínda que, obviamente, non se obtén información das fases Throughput Test nin Data Maintenance, xa que non se chegan a executar. Así, os experimentos executados utilizando esta "versión reducida" do benchmark pódense consultar na Sección 5.2.1, mentras que aqueles executados utilizando o benchmark completo pódense consultar na Sección 5.2.2. Os resultados máis relevantes deste apartado tamén se atopan recollidos na publicación [37].

5.2.1 Resultados de fases Load Data e Power Test

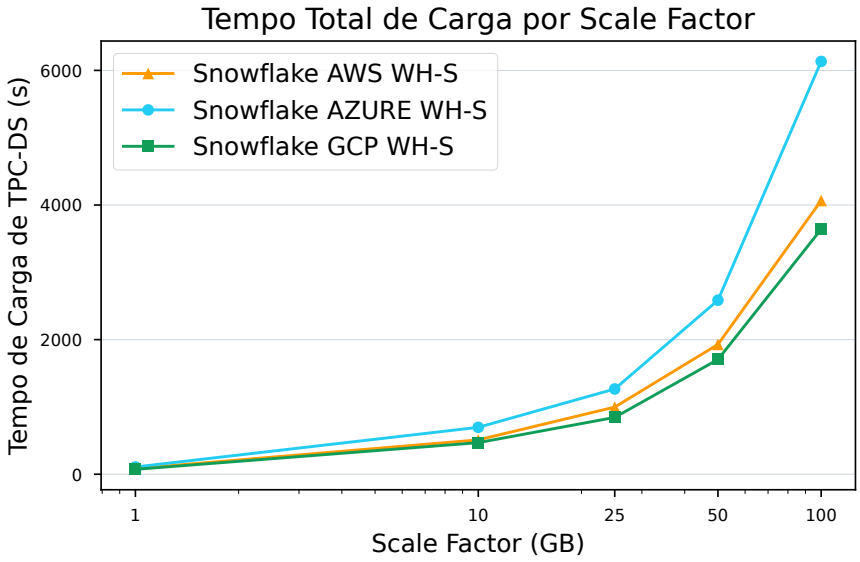
Esta serie de experimentos pódese dividir en tres grandes bloques, segundo a plataforma que evalúan. O primeiro bloque, tratado na Sección 5.2.1, inclúe probas realizadas en Snowflake. O segundo deles, Sección 26, experimentos realizados en Databricks. Finalmente, a Sección 27 trata o bloque de experimentos realizados comparando capacidades entre as dúas plataformas probadas nas seccións anteriores.

Snowflake

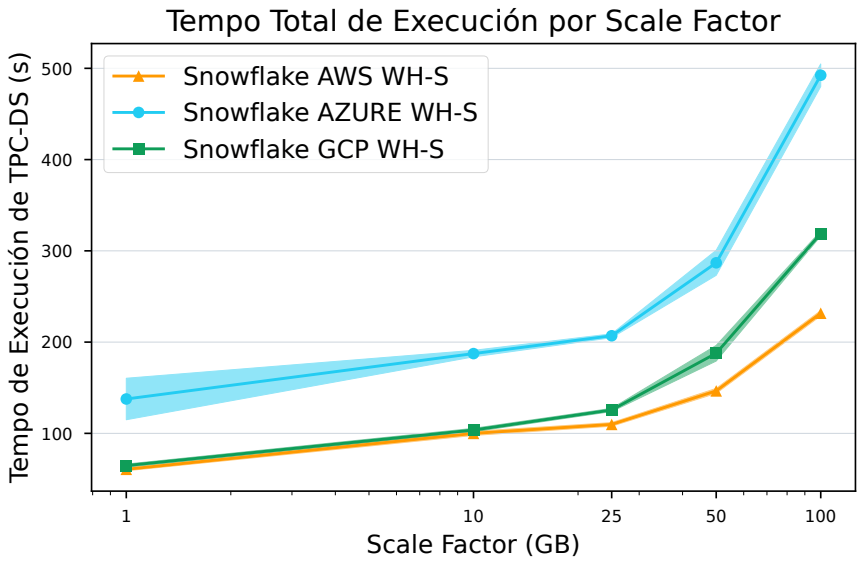
Para avaliar o rendemento de Snowflake decidiuse realizar probas en dous contextos diferentes. No primeiro deles compárase o rendemento de Snowflake con diferentes provedores de cloud: Amazon Web Services (AWS), Microsoft Azure e Google Cloud Platform (GCP). No segundo, seleccionouse AWS como provedor e comparouse o rendemento de tres tamaños diferentes de Virtual Warehouse: Small, Medium e Large.

Na Figura 5.1a compáranse os tempos de carga de datos, fase Load Data, en Snowflake implementado nos tres provedores de cloud para distintos *Scale Factors*. Destaca o feito de que en Azure se obtén un maior tempo de carga, aínda estando a ler dun Blob do mesmo provedor, onde se podería esperar algunha optimización específica. Entre AWS e GCP a plataforma de Google obtén uns resultados lixeiramente mellores, pero a diferenza é menor que respecto ao primeiro.

Doutra banda, na Figura 5.1b compáranse os resultados de executar a fase Power Test do benchmark nestas plataformas 10 veces, onde se mostran a súa media e desviación típica. Neste caso é AWS o que mostra un mellor rendemento, algo que se vai acentuando a cada



(a) Tempos de carga

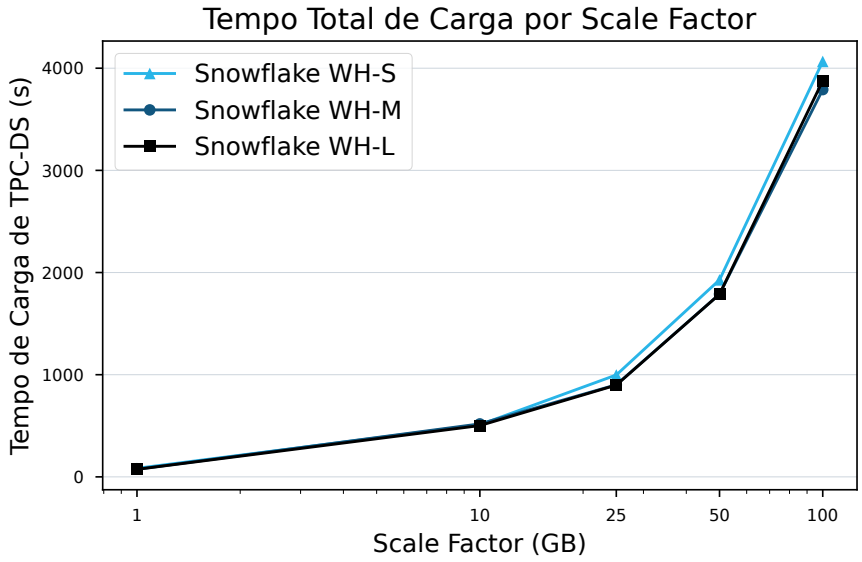


(b) Tempos de execución

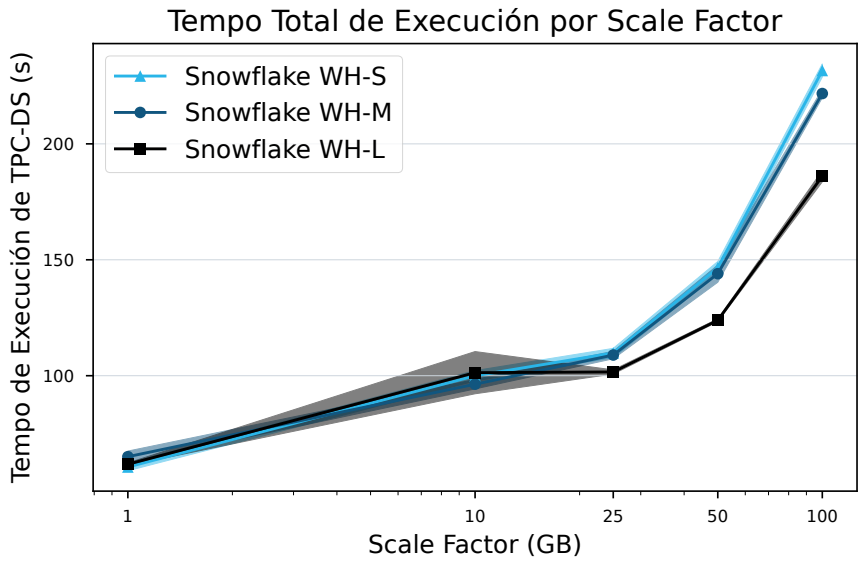
Figura 5.1: Resultados da execución de TPC-DS en Snowflake utilizando AWS, GCP e Azure

aumento do *Scale Factor*. Un feito a ter en conta é que Snowflake foi desenvolto inicialmente en AWS, e por tanto é de esperar que nesta plataforma se obteña un mellor rendemento. Por isto, para o resto de experimentos do proxecto utilizarase a implementación de Snowflake sobre AWS.

Na Figura 5.2a repítese a mesma proba en Snowflake (fases Load Test e Power Test de TPC-DS), esta vez variando o tamaño do warehouse e non o provedor cloud. O resultado é



(a) Tempos de carga



(b) Tempos de execución

Figura 5.2: Resultados da execución de TPC-DS en Snowflake utilizando tres tamaños diferentes de warehouse

que se aprecia pouca diferenza en canto a tempo empregado. Porén, a diferenza de prezo (Créditos/hora) entre estes tamaños (Táboa 5.1) duplícase por cada aumento. A diferenza que se pode apreciar entre o warehouse Small e Medium (Figura 5.2b) é case imperceptible, mentras que si se produce un aumento de rendemento utilizando o tamaño Large, aínda que non moi pronunciado. As cifras coas que se constrúen estas gráficas pódense consultar na

Táboa 5.2, onde se pode apreciar que os tempos dunha execución da fase Power Test con *scale factor* 100 se reducen nun 4.29% (232 segundos a 222 segundos) e un 16.10% (222 segundos a 186 segundos). Con todo, en contraste con estes lixeiros aumentos, o custo de cada execución aumenta con cada incremento de tamaño do warehouse nun 93.93% (0.33 $ a 0.64 $) e un 68.75% (0.64 $ a 1.08 $).

Así, comparando as Figuras 5.1b e 5.2b apréciase que se producen maiores diferenzas no tempo total de execución variando o provedor de cloud que variando o tamaño de warehouse. Isto é interesante porque aumuentar o tamaño de warehouse implica maior custo por crédito, mentres que os distintos provedores de cloud (polo menos na rexión "West Europe") teñen todos o mesmo custo por crédito. Deste modo, pódese afirmar que aínda que se utilice un tamaño maior, cun custo dúas veces superior, o benchmark non ten por que executarse na metade de tempo. Cos tamaños de *Scale Factor* cos que se traballaron, ata 100 GB, un tamaño operable para pequenas e medianas empresas, non se atoparon diferenzas significativas que animen a usar tamaños grandes, polo que non resulta escalable. Cun warehouse de tamaño Small (2 nodos, 16 cores/thread e 32 GB de RAM[1]) pódese traballar de forma eficiente con bases de datos de 100 GB, sen necesidade dunha maior capacidade de cómputo. Por tanto, é necesario estimar de forma correcta o tamaño de warehouse adecuado á tarefa para realizar, para así evitar sobrecustos innecesarios.

En conclusión, vistos os resultados dos experimentos, pódese concluir que Snowflake AWS é capaz de executar as consultas de TPC-DS na metade de tempo que Snowflake Azure ao utilizar un warehouse do mesmo tamaño. Pola súa banda, Snowflake GCP obtivo un rendemento intermedio entre os outros dous. As diferenzas á hora de cargar os datos son menores, sendo GCP o provedor de cloud sobre o que Snowflake ofrece un mellor desempeño, aínda que seguido moi de preto por AWS.

Tamaño	$\$/h$	$T(s)$	$\Delta T(s)$	$	$\Delta$$
Small	5.20	231.64	-	0.33	-
Medium	10.40	221.71	$\downarrow$ 4.29%	0.64	$\uparrow$ 93.93%
Large	20.80	186.05	$\downarrow$ 19.68%	1.08	$\uparrow$ 227.50%

Táboa 5.2: Custos dunha execución de Power Test en distintos warehouses de Snowflake

[1] Snowflake non publica as especificacións dos seus warehouses. Porén, estes atópanse en distintas páxinas da web de forma non oficial, como é o caso de https://select.dev/posts/snowflake-warehouse-sizing.

Databricks

En Databricks utilizouse a versión *Azure Databricks*, realizando probas de rendemento para comparar tres tamaños diferentes de *SQL serverless warehouses*, posto que son o recurso de cómputo dos que dispón Databricks máis similar aos warehouses de Snowflake por seren *serverless* e non ter capacidade que hai que levantar e apagar (ou auto-apagar). A alternativa sería utilizar un clúster[2], recurso de cómputo cuxas capacidades difiren das dos warehouses que ofrece Snowflake.

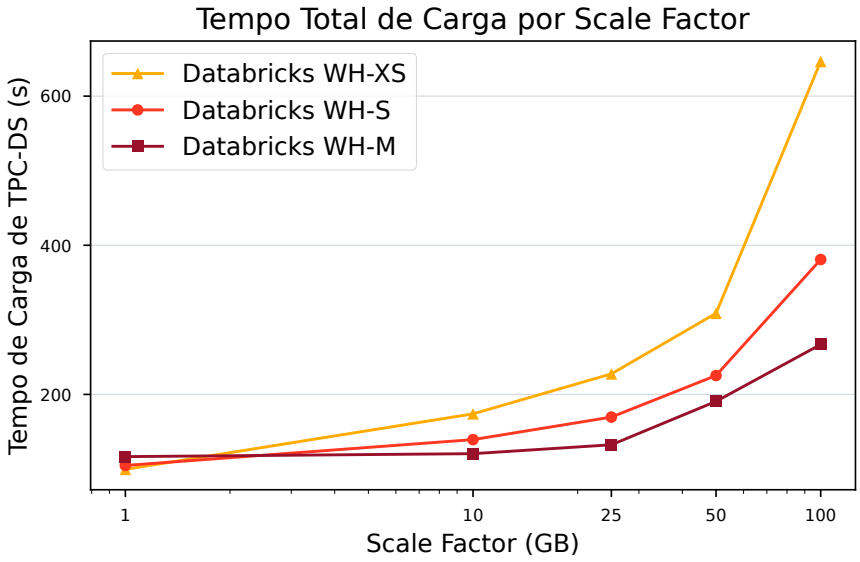
Á hora de cargar os datos obtéñense diferenzas de tempos de carga significativas entre os tres tamaños, como se pode apreciar na Figura 5.3a. Ao pasar de Warehouse XS a S pódese comprobar que o tempo diminúe case ata a metade en SF100, polo que neste caso podería compensar e xustificar o aumento do prezo correspondente, o cal se duplica (ver Táboa 5.1). Porén, esta diminución non resulta tan significativa cando o tamaño se aumenta de S a M, mentres que o prezo séguese duplicando.

Analizando os tempos de execución das consultas da fase Power Test de TPC-DS na Figura 5.3b obsérvase que o comportamento repite o mesmo patrón en SF100. En ambas as gráficas se observa que canto maior é o *Scale Factor*, maior é o efecto de utilizar warehouses de maior tamaño, polo que se recomenda ter especial coidado de non empregar innecesariamente unha potencia de cómputo maior da necesaria. Na Táboa 5.3 pódese comprobar o custo exacto de executar o benchmark SF100 en Databricks, onde se aprecia que aumentar de X-Small a Small supón unha redución do tempo case da metade (40,28%), cun aumento do 17% do custo. Con todo, ao pasar de Small a Medium a duración redúcese soamente un 23% e o custo aumenta un 34,5%, de tal forma que este cambio é moito menos rendible que o anterior. Pódese obter a mesma conclusión que con Snowflake: un aumento de tamaño de warehouse non compensa en todos os casos, senón que é dependente do tamaño da base de datos para poder aproveitar a potencia extra.

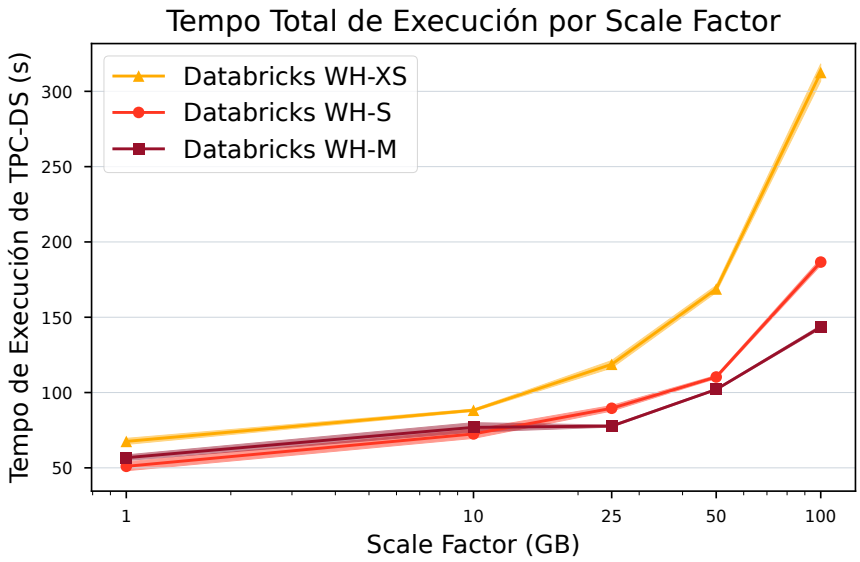
Tamaño	$\$/h$	$T(s)$	$\Delta T(s)$	$\$$	$\Delta\$$
X-Small	5.46	312.51	-	0.47	-
Small	10.92	186.63	$\downarrow 40,28\%$	0.57	$\uparrow 17,54\%$
Medium	21.84	143.48	$\downarrow 54.09\%$	0.87	$\uparrow 45.98\%$

Táboa 5.3: Custos dunha execución de Power Test en distintos warehouses de Snowflake

[2] https://learn.microsoft.com/es-es/azure/databricks/compute/



(a) Tempos de carga



(b) Tempos de execución

Figura 5.3: Rendemento de TPC-DS en Databricks utilizando tres tamaños diferentes de warehouse

Snowflake e Databricks

Co fin de comparar o rendemento entre as dúas plataformas, Snowflake e Databricks, seleccionouse AWS como provedor de cloud para Snowflake, dado que o seu rendemento é superior ao dos outros dous provedores (como se explicou na Sección 5.2.1). Así, dado que teñen un custo similar (Táboa 5.1), decidiuse comparar os warehouses das dúas plataformas

enfrontándoos entre si do seguinte modo:

- Snowflake S – Databricks X-Pequeno
- Snowflake M – Databricks Pequeno[3]
- Snowflake L – Databricks Mediano

Tamaño	$\$/h$	$T(s)$	$\Delta T(s)$	$\$$	$\Delta\$$
Small	5.20	4066.30	-	5.87	-
Medium	10.40	3790.70	$\downarrow 6.78\%$	10.95	$\uparrow 86.54\%$
Large	20.80	3877.20	$\uparrow 2.28\%$	22.40	$\uparrow 104.47\%$

Táboa 5.4: Custos dunha carga de datos de TPC-DS (SF100) en distintos tamaños de warehouses de Snowflake

Tamaño	$\$/h$	$T(s)$	$\Delta T(s)$	$\$$	$\Delta\$$
X-Small	5.46	646.53	-	0.98	-
Small	10.92	380.89	$\downarrow 41.09\%$	1.16	$\uparrow 18.37\%$
Medium	21.84	266.98	$\downarrow 29.91\%$	1.62	$\uparrow 39.66\%$

Táboa 5.5: Custos dunha carga de datos de TPC-DS (SF100) en distintos tamaños de warehouses de Databricks

Existe unha gran disparidade entre as plataformas avaliadas para a métrica do tempo de carga[4] dos datos desde o Blob Storage ata o almacenamento interno da propia plataforma (Figura 5.4a). Tal e como se deseñou a arquitectura para implementar estes experimentos (Sección 4.1) Azure Databricks atópase baixo a mesma subscrición que o Blob Storage, pero non nun mesmo grupo con localización por proximidade[5], nin sequera no mesmo Grupo de Recursos, polo que non é posible que exista unha vantaxe debido a isto. Con todo, a diferenza é moi notable: Para o *Scale Factor* 100 e o maior tamaño de warehouse comprobado en ambas as

[3] Este tamaño non se engadiu ás gráficas por simplicidade, xa que os seus resultados se atopaban nun punto intermedio entre os dos outros dous casos.

[4] Esta medida difire lixeiramente da dictada na especificación de TPC-DS: nesta explcítase que se debe medir tanto o tempo de creación das táboas coma o de cargar os datos nelas, mentres que nestes experimentos tan só se mediu o tempo que se tarda en cargar os datos unha vez as táboas xa están creadas.

[5] https://learn.microsoft.com/en-us/azure/virtual-machines/co-location

plataformas, Snowflake tarda 3877 segundos (1 hora e 4 minutos) en cargar os datos en todas as táboas (consultar Táboa 5.4), mentres que Databricks logra facelo en 266 segundos (4 minutos e 25 segundos) (consultar Táboa 5.5). As diferenzas intraplataforma son moito máis lixeiras, aínda que son maiores no caso de Databricks que no de Snowflake. Por tanto, pódese afirmar que independentemente do provedor de cloud e do tamaño do warehouse, Snowflake resulta máis lento que Databricks cargando os datos desde un Azure Blob Storage. As diferenzas, establecendo as equivalencias mencionadas ao comezo da Subsección 27, chegan a ser da orde de dez veces superiores en canto ao rendemento. Coa intención de intentar paliar estas diferenzas, executouse o benchmark de novo, pero desta vez con Azure Databricks lendo os datos desde o Azure Blob Storage e con Snowflake AWS lendo os datos desde un AWS S3; en ambos casos estando tanto a plataforma coma o almacenamento de datos na mesma rexión xeográfica. Porén, Snowflake non mellorou os resultados na fase de Load Data respecto ás probas anteriores, nas que lía os datos desde o Azure Blob Storage.

En canto aos tempos de execución das consultas, as diferenzas entre as dúas plataformas non son tan notorias como no caso da carga de datos. Na Figura 5.4b pódese comprobar que a configuración que demostra un maior tempo de execución é Databricks cun tamaño de warehouse XS, mentres que a que menor é Databricks co warehouse M, o comprobado de maior tamaño. Pódense apreciar nas Táboas 5.2 e 5.3 os custos exactos das execucións para SF100. Destacar que o warehouse S de Snowflake ten un custo similar (mesmo lixeiramente inferior) ao warehouse XS de Databricks, pero con todo realiza a execución completa das consultas nun tempo inferior.

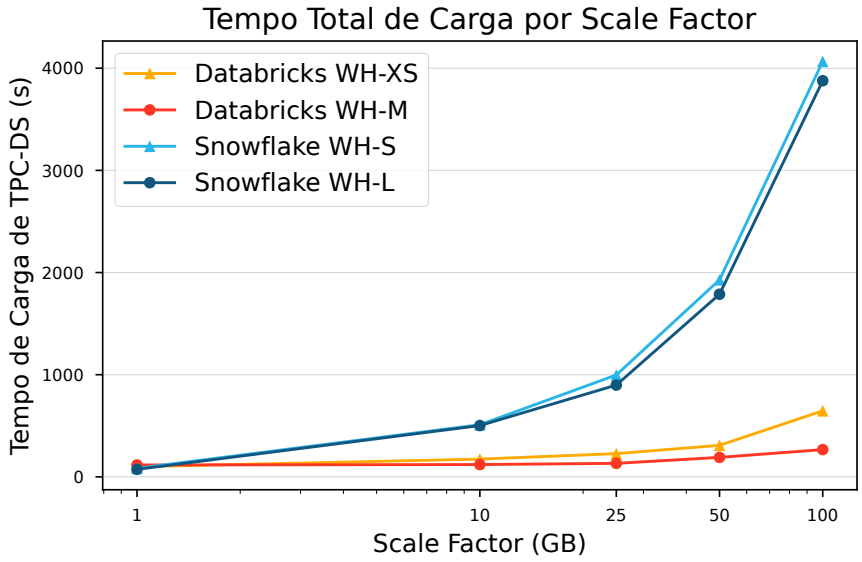
Espazo en disco

Na Figura 5.5 pódese comprobar o espazo que ocupa en disco a base de datos que se utilizou. Na figura apréciase que tanto Snowflake como Databricks realizan unha compresión ao redor dun 30% cando o *Scale Factor* é 100, obtendo as dúas plataformas un resultado moi similar, aínda que situándose Snowflake lixeiramente por encima.

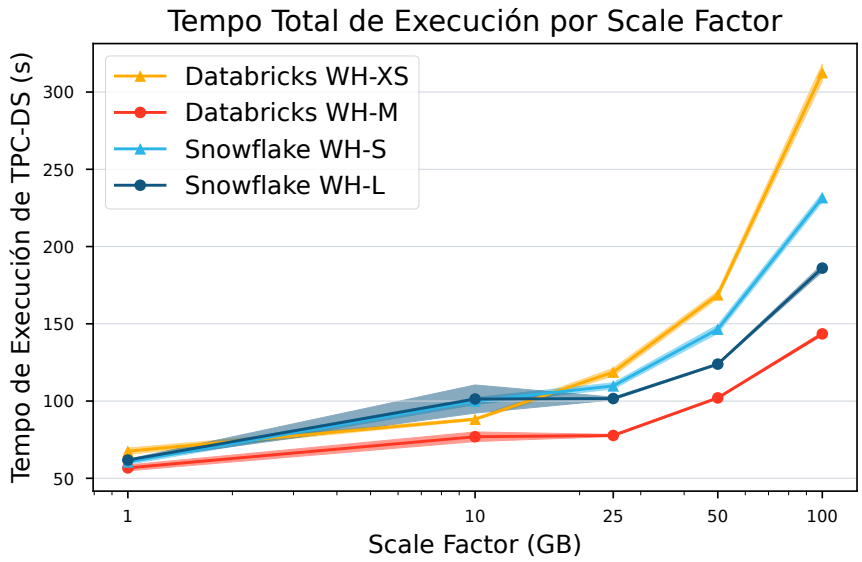
5.2.2 Benchmark completo

Debido a que Snowflake, independentemente da configuración, tarda máis en cargar os datos que Databricks, decidiuse realizar unha proba na que os datos estiven gardados nun Amazon S3[6] e non nun contedor de Azure para comprobar se Snowflake AWS aproveitaba esta ventaxa. Así, realizouse unha execución completa de TPC-DS con *scale factor* 100 e 4 fíos de execución no *Throughput Test*, con Azure Databricks cargando os datos desde contedor de Azure (o mesmo que nos experimentos anteriores) e Snowflake AWS cargando os datos desde

[6] https://aws.amazon.com/es/s3/



(a) Tempos de carga



(b) Tempos de execución

Figura 5.4: Resultados da execución de TPC-DS tanto en Databricks coma Snowflake

un AWS S3 an rexión West Europe (Paris). Os resultados poden verse na Táboa 5.6.

Para a execución en Databricks utilizouse un *warehouse* de tamaño X-Pequeno, mentras que para a execución en Snowflake se utilizou un de tamaño S (que como se explicou na Subsección 27 son prácticamente equivalentes en canto a prezo).

Dados estes resultados, utilizando a fórmula 3.1, pódese calcular a métrica principal de TPC-DS, *QphDS@SF*, que quedaría do seguinte modo:

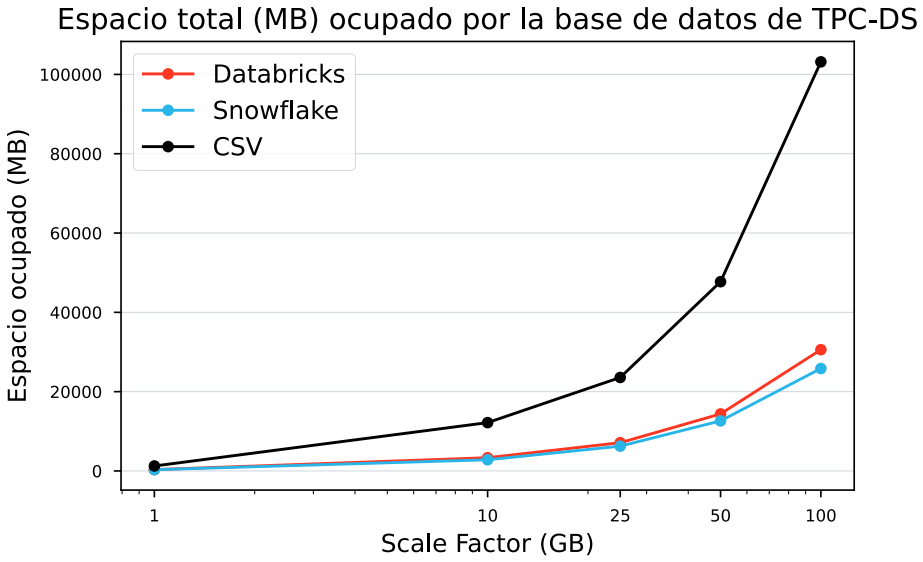


Figura 5.5: Espazo que ocupa a base de datos por *Scale Factor* en cada un dos formatos

	Databricks	**Snowflake**
Load Data	652.91	4088.10
Power Test	95.16	334.23
Throughput Test 1	274.17	833.67
Data Maintenance 1	123.19	75.50
Throughput Test 2	263.99	852.53
Data Maintenance 2	112.05	54.81

Táboa 5.6: Tempos (segundos) das fases de TPC-DS@100 en Databricks e Snowflake

- Databricks:

$$QphDS@100 = \left\lfloor \frac{100 \cdot 396}{\sqrt[4]{0.0264 \cdot 0.1512 \cdot 0.0653 \cdot 0.1814}} \right\rfloor = 477549\ QphDS@100 \tag{5.1}$$

- Snowflake:

$$QphDS@100 = \left\lfloor \frac{100 \cdot 396}{\sqrt[4]{0.0928 \cdot 0.4684 \cdot 0.0362 \cdot 1.1356}} \right\rfloor = 192606\ QphDS@100 \tag{5.2}$$

Vendo as métricas pódese comprobar que Databricks consegue uns valores superiores aos

obtidos por Snowflake, entre dúas e tres veces maiores. Isto, vendo a Táboa 5.6, dedúcese que se debe principalmente á gran diferencia entre as plataformas na fase de Load Data.

Como amosa a Figura 5.6, na web oficial de TPC tan só se atopan resultados publicados para *scale factors* de 10.000 GB e 100.000 GB. Por tanto, os resultados obtidos durante a realización deste proxecto non son comparables, xa que tan só se realizaron experimentos con s*scale factors* comprendidos entre 1 GB e 100 GB. Isto decidiuse por dous motivos principais: en primeiro lugar, aportar á comunidade resultados con conxuntos de datos dun volume operable pola maioría de organizacións. Os resultados publicados na web utilizan volumes de datos que non manexan a maioría de organizacións, polo que non lles resultan de interese eses resultados. Ademais, por outro lado, reducindo o volume dos conxuntos de datos a avaliar tamén se reducen os custos das execucións, e por tanto os custos do proxecto.

10,000 GB Results

Company	System	Performance (QphDS)	Price/kQphDS	Watts/KQphDS	System Availability	Database	Operating System	Date Submitted	Cluster
Alibaba.com	Alibaba Cloud AnalyticDB	18,998,559	59.27 CNY	NR	06/17/20	Alibaba Cloud AnalyticDB 3.0.12	Alibaba Group Enterprise Linux Server 7.2 (Paladin)	06/17/20	Y
Alibaba.com	Alibaba Cloud E-MapReduce	11,569,838	237.03 CNY	NR	04/17/20	Alibaba Cloud E-MapReduce 4.0.1	CentOS Linux Release 7.4	04/16/20	Y
H3C	H3C UniServer R4900 G3	8,944,478	423.13 CNY	NR	12/23/20	GBase 8a V9	Red Hat Enterprise Linux Server 7.8	12/23/20	Y
SUPERMICRO	Supermicro A+ Server 2123BT-HNC0R	4,418,054	110.29 USD	NR	08/31/19	Transwarp ArgoDB V1.2.1	Red Hat Enterprise Linux Server 7.6	08/07/19	Y

100,000 GB Results

Company	System	Performance (QphDS)	Price/kQphDS	Watts/KQphDS	System Availability	Database	Operating System	Date Submitted	Cluster
databricks	Databricks SQL 8.3	32,941,245	157.57 USD	NR	11/02/21	Databricks Photon Engine 8.3	Ubuntu 18.04.5 LTS	11/02/21	Y
Alibaba.com	Alibaba Cloud E-MapReduce	14,861,137	175.23 USD	NR	09/16/19	Alibaba Cloud E-MapReduce 3.21.2	CentOS Linux Release 7.4	09/16/19	Y

Figura 5.6: Resultados publicados na web de TPC-DS

5.3 Benchmarks desde Servizos de Business Intelligence

5.3.1 Inxesta de datos a Power BI

Seguíronse as guías oficiais [7] [8] que Power BI ofrece para conectarse desde Power BI Desktop a ambas plataformas.

O primeiro que se pode apreciar, antes de probar nada, é o maior abano de posibilidades que ofrece o conector de Azure Databricks respecto ao conector de Snowflake. Este último tan só permite conectar Power BI Desktop a Snowflake utilizando Power Query; pensado isto para utilizar a plataforma na nube como almacén de datos, desde onde Power BI se encarga de ler e posteriormente procesar para a súa visualización. Porén, o conector de Azure Databricks,

[7] https://learn.microsoft.com/es-es/azure/databricks/partners/bi/power-bi

[8] https://learn.microsoft.com/es-es/power-query/connectors/snowflake

ademais desta mesma funcionalidade, tamén permite a publicación en Power BI Online e a conexión á versión Desktop a través de *Databricks Partner Connect*[9].

Dado isto, a única proba realizada é unha comparativa de rendemento entre os conectores a través de Power Query. Nos dous casos se debe indicar que táboa ou vista se quere consultar (ou introducir a man a consulta), e Power BI encárgase de descargar os datos necesarios (ben táboas completas ou ben o resultado da consulta) utilizando DirectQuery para posteriormente traballar con eles. É dicir, a lóxica da consulta SQL execútase na plataforma da nube e non en Power BI. Por isto, realizar consultas complexas para esta proba carece de sentido, xa que o que se quere avaliar é o rendemento do conector, e a consulta se executa na plataforma na nube. Deste modo, simplemente se realizaron consultas que recuperaran táboas cos mesmos rexistros nas dúas plataformas, para comparar o tempo que tarda en cada unha delas. Utilizáronse *warehouses* de tamaño S en Snowflake e X-Pequeno en Databricks dada a equivalencia descrita na Sección 27, e as táboas *store_sales*, *catalog_returns* e *promotion* de TPC-DS na versión de *scale factor* 10. Estas táboas ocupan en disco en formato CSV un tamaño de 3.72GB, 211.31MB e 60.7KB, respectivamente; polo que se considera que cobren un rango de tamaño suficiente. Amais, utilizouse o analizador de rendemento[10] que ofrece Power BI. Este mide o tempo de procesamento necesario para actualizar os elementos do informe iniciados como resultado de calquera interacción do usuario que teña como resultado a execución dunha consulta e, dado que neste experimento simplemente se inclúen de forma directa os datos da consulta na táboa (non existe representación gráfica), asúmese que a maior parte do tempo se emprega recibindo ou procesando o resultado da consulta.

En xeral, os resultados obtidos contan cunha variabilidade moi elevada, e resulta imposible extraer conclusións acerca de que táboas se cargan en menor tempo, pois o resultado cambia en cada execución. Porén, si semella que o conector de Snowflake ofrece un rendemento lixeiramente superior ao ofrecido polo de Databricks. Na Táboa 5.7 pódese ver o resultado de refrescar cinco veces a vista, onde se amosa o tempo medio de refresco e a desviación típica desas cinco actualizacións. O ancho de banda non inflúe porque as consultas son limitantes e devolven poucos rexistros. Aínda así, o conector de Databricks ofrece máis capacidades, como conectarse a Power BI Service ou a publicación en Power BI Online desde Databricks.

[9] https://learn.microsoft.com/en-us/azure/databricks/partner-connect/
[10] https://learn.microsoft.com/es-es/power-bi/create-reports/desktop-performance-analyzer

Táboa	Snowflake		Databricks	
	Media	Desviación típica	Media	Desviación típica
Promotion	1390.6	469.28	1307.6	390.00
Catalog Returns	873.8	335.18	1019	357.39
Store Sales	925.8	384.37	1155.8	402.76

Táboa 5.7: Tempo de carga (segundos) das táboas en PowerBI no experimento realizado

Capítulo 6

Conclusións e Traballo Futuro

6.1 Conclusións

A partir dos experimentos e resultados descritos no Capítulo anterior, pódese afirmar que se cumpliron todos os obxectivos definidos no Capítulo 1. Nel pódese comprobar que, previamente á realización do traballo, se definiron os seguintes obxectivos:

1. **Análise do estado da arte en plataformas Data Lakehouse**. A través de artigos como [1] [2] decidiuse que Snowflake e Databricks eran bos representantes da arquitectura Data Lakehouse, e por tanto as plataformas escollidas para avaliar. Ademais, o feito de que ambas plataformas publicaran artigos [3] [4] nos que expoñían o maior rendemento da súa plataforma no benchmark TPC-DS, provocou que este fora o benchmark escollido. Deste modo trataríase de realizar avaliacións imparciais que puideran aclarar o publicado polas propias plataformas (obviamente partes interesadas e non imparciais).

2. **Execución dos benchmarks**. Este obxectivo completouse satisfactoriamente mediante o desevolvemento de CLADE-TPC-DS, a ferramenta que permite executar TPC-DS, o benchmark escollido, tanto en Snowflake coma en Databricks (e que ademais fai uso de patróns e principios de deseño software coa fin de permitir de forma sinxela a extensión da ferramenta para avaliar novas plataformas).

3. **Recompilación de datos e métricas**. Mediante a execución de TPC-DS puidéronse obter métricas parciais, como son T_{LOAD} e T_{POWER}, e métricas da execución completa do benchmark como é o caso de $QphDS@SF$; todas elas detalladas na Sección 3.2.1.

4. **Recomendacións**. Tras a execución dos experimentos, analizando as métricas e os datos obtidos, emitíronse recomendacións de valor á industria, como é o caso de priorizar

o uso de Databricks sobre Snowflake no caso de que o uso que se lle vaia dar á plataforma conteña cargas de grandes volumes de datos de forma reiterada. Isto baséase nos resultados que amosan unha clara diferenza no tempo de carga entre as dúas plataformas, como se pode apreciar na Figura 5.4. Ademais, baseándose naos resultados amosados na Sección 5.2.1, tamén se recomenda o uso de AWS como provedor cloud para Snowflake, xa que mellora o rendemento sobre Azure e GCP tanto na carga de datos (fase *Load Data* de TPC-DS) coma no procesado (fase *Power Test* de TPC-DS).

5. **Contribución académica**. Acadouse este obxectivo chegando a publicar un artigo nas Xornadas Sarteco 2024: "Análisis de Rendimiento de Plataformas Data Lakehouse: Snowflake y Databricks" [37].

6.2 Traballo futuro

Dada a natureza do proxecto, existen múltiples vías a través das cales poder amplialo, posto que o abano de probas ou comparacións que se poden realizar é realmente amplo.

En primeiro lugar, poderíase ampliar a ferramenta desenvolvida. Esta permite realizar unha execución completa de TPC-DS tanto en Snowflake coma en Databricks, pero poderíase ampliar engadindo máis plataformas sobre as que executar o benchmark. Por exemplo, podería resultar de interese que a ferramenta ofreza soporte para executar o benchmark en Databend[1], un cloud Data Lakehouse open source de recente publicación; ou incluso OneLake[2] ou Microsoft Fabric[3].

Ademais de extensións da ferramenta, tamén se pode ampliar o traballo utilizándoa no seu estado actual para realizar novas probas. Estas poderían incluir novos *scale factors*, novos formatos de datos (TPC-DS xera os datos en CSV, pero pódense transformar a formatos coma Parquet, por exemplo), ordenación dos rexistos, etc.

Por outra banda, tamén se poderían explorar outros benchmarks, como TPCx-BB, que proben as capacidades das plataformas manexando outros formatos de datos. TPC-DS tan só utiliza formatos de datos estruturados, que consulta a través de SQL, pero existen benchmarks, como é o caso do mencionado TPCx-BB, que tamén utilizan formatos de datos semiestruturados e mesmo non estruturados, coma *reviews* de produtos. Non se chegou a incluir no proxecto porque está deseñado para sistemas *on premise* e a súa execución en plataformas na nube, como é o caso de Snowflake e Databricks, non é óptima; pero pode ser unha vía interesante de traballo futuro.

Por último, tamén sería válido como traballo futuro calquera proba non vinculada a TPC

[1] https://www.databend.com/
[2] https://learn.microsoft.com/en-us/fabric/onelake/onelake-overview
[3] https://www.microsoft.com/es-es/microsoft-fabric

que compare cualidades das plataformas Data Lakehouse, similares á realizada con Power BI. Estas probas poden servir para comprobar que funcionalidades ofrecen as plataformas, e non tan só para avaliar rendemento.

Apéndices

Apéndice A

Ficheiros de configuración

Neste Capítulo móstranse exemplos dos distintos ficheiros de configuración utilizados na aplicación. En concreto, móstrase un exemplo do ficheiro de configuración da conexión e outro do ficheiro de configuración dos experimentos para cada plataforma.

A.1 Ficheiro JSON para a configuración da conexión en Databricks

```
{
    "host": "databricks.host",
    "token": "token",
    "http_path_xs":
    ↪ "http_path_to_databricks_warehouse_xsmall",
    "http_path_s": "http_path_to_databricks_warehouse_small",
    "http_path_m": "http_path_to_databricks_warehouse_medium"
}
```

A.2 Ficheiro JSON para a configuración da conexión en Snowflake

```
{
    "user": "USER",
    "password": "password",
    "account": "ORGNAME-ACCOUNTNAME"
}
```

A.3 Ficheiro JSON para a configuración dos experimentos en Databricks

```
{
    "warehouse": "S",
    "schema": "tpc_ds",
    "blob_url": "wasbs://container@url",
    "blob_folder": "/sf1/",
    "sql_schema": "/tmp/tpcds/tpcds.sql",
    "raw_queries": "/tmp/tpcds/queries/query_0.sql",
    "processed_queries": "/tmp/tpcds/queries/queries.sql",
    "scale_factor": 1,
    "results_path": "../results/databricks/",
    "permutation_orders_folder":
    ↪ "./config/permutation_orders",
    "streams": 1
}
```

A.4 Ficheiro JSON para a configuración dos experimentos en Snowflake

```
{
    "warehouse": "wh_small",
    "db": "DB",
    "schema": "TPC_DS",
    "stage": "tpcds_stage",
    "blob_url": "cloud://url/container/",
    "blob_folder": "sf1/",
    "sql_schema": "/tmp/tpcds/tpcds.sql",
    "raw_queries": "/tmp/tpcds/queries/query_0.sql",
    "processed_queries": "/tmp/tpcds/queries/queries.sql",
    "scale_factor": 1,
    "results_path": "../results/snowflake/",
    "permutation_orders_folder":
    ↪ "./config/permutation_orders",
    "streams": 1
}
```

Apéndice B

Manual de uso

Neste Capítulo preséntanse as instruccións para realizar un uso básico da ferramenta CLI desenvolta para executar TPC-DS. Na Sección B.1 explícase como se cargan os datos desde o contedor na plataforma a executar o benchmark. Na Sección B.2 especifícase como se executan as consultas unha vez cargados os datos na plataforma.

B.1 Carga dos datos

```
load_data.py [-h] [--scale_factor SF] [--warehouse WH] --platform
    {databricks,snowflake} --experiment_config EXP_FILE
    --connection_config CONN_FILE

options:
  -h, --help            show this help message and exit

  --scale_factor SCALE_FACTOR, -s SCALE_FACTOR
                        Scale Factor for the dataset (default: None)

  --warehouse WAREHOUSE, -w WAREHOUSE
                        Size of warehouse to use (default: None)

  --platform {databricks,snowflake}, -p {databricks,snowflake}
                        Platform to use (databricks or snowflake)
    (default: None)

  --experiment_config EXPERIMENT_CONFIG, -e EXPERIMENT_CONFIG
                        Path to the experiment configuration file

  --connection_config CONNECTION_CONFIG, -c CONNECTION_CONFIG
                        Path to the connection configuration file
```

Listing B.1: Parámetros para executar load_data.py

B.2 Execución das consultas

```
execute_queries.py [-h] [--scale_factor SF] [--warehouse WH]
    --platform {databricks,snowflake} --experiment_config
    EXPERIMENT_CONFIG --connection_config CONNECTION_CONFIG
    [--queries QUERIES] [--times TIMES] [--streams STREAMS]

options:
  -h, --help            show this help message and exit

  --scale_factor SCALE_FACTOR, -sf SCALE_FACTOR
                        Scale Factor for the dataset (default: None)

  --warehouse WAREHOUSE, -w WAREHOUSE
                        Size of warehouse to use (default: None)

  --platform {databricks,snowflake}, -p {databricks,snowflake}
                        Platform to use (databricks or snowflake)
    (default: None)

  --experiment_config EXPERIMENT_CONFIG, -e EXPERIMENT_CONFIG
                        Path to the experiment configuration file
    (default: None)

  --connection_config CONNECTION_CONFIG, -c CONNECTION_CONFIG
                        Path to the connection configuration file
    (default: None)

  --queries QUERIES, -q QUERIES
                        Specify "all" or a specific query number
    (default: all)

  --times TIMES, -t TIMES
                        Number of times to execute the queries
    (default: 5)

  --streams STREAMS, -s STREAMS
                        Number of streams to use to execute the
    queries (default: 1)
```

Listing B.2: Parámetros para executar execute_queries.py

Relación de Acrónimos

AA Aprendizaxe Automática. 7, 8, 25, 26

ACID Atomicity, Consistency, Isolation, and Durability. 8, 23

API Application Programming Interfaces. 8

AWS Amazon Web Services. 23, 25, 27, 28, 46, 49

BI Business Intelligence. 7, 28, 42

CI/CD Continuous Integration & Continuous Delivery. 21

CLI Command Line Interface. 38, 67

CPU Central Processing Unit. 25

CSV Comma Separated Values. 37, 38, 57, 60

DBFS Databricks File System. 30

DBU Databricks Unit. 31

DDL Data Definition Language. 33

DML Data Manipulation Language. 25

DSS Decision Support System. 3, 34

ETL Extract, Transform, Load. 4

GCP Google Cloud Platform. 23, 27, 28, 46, 49

HDD Hard Disk Drive. 30

IA Intelixencia Artifical. 7, 28

JDBC Java Database Connectivity. 26

MLOps Machine Learning Operations. 7, 26

ODBC Open Database Connectivity. 4

OLAP Online Analytical Processing. 3, 4, 32–34

OLTP Online Transaction Processing. 3

SQL Structured Query Language. 18, 19, 25–27, 29, 31, 33, 37–39, 57, 60

SSD Solid State Drive. 30

TPC Transaction Processing Performance Council. 2, 31, 34, 38, 42, 56, 60

TPC-DS TPC Decision Support. 1, 9, 20, 31–35, 37, 39, 41, 45–47, 49, 50, 53, 54, 57, 59, 60, 67, III

TPCx-BB TPC Express Benchmark BB. 35, 60

UDF User Defined Function. 27

UI User Interface. 26

VW Virtual Warehouse. 24–27, 46

Glosario

backend Partes dunha aplicación informática ou do código dun programa que permiten o seu funcionamento e ás que non pode acceder un usuario. 28

benchmark Programa informático, conxunto de programas ou outras operacións, cuxo fin é avaliar o rendemento relativo dun obxecto, normalmente mediante a execución dunha serie de probas e ensaios estándar con el. 1, 2, 9, 10, 18, 31–34, 37–39, 42, 45, 46, 49, 53, 59, 60, 67

biblioteca Colección de programas e paquetes de software postos ao dispor xeral, a miúdo cargados e almacenados en disco para o seu uso inmediato. 30, 31, 42

Big Data Datos tan grandes ou complexos que é difícil ou mesmo imposible procesalos mediante métodos tradicionais. 5, 7

binario Ficheiro informático que contén información de calquera tipo codificada en binario para o propósito de almacenamento e procesamento en computadores. 37

clúster Conxunto de ordenadores que funcionan xuntos de forma que poden considerarse un único sistema. 24, 29

Data Lake Ferramenta formada por un repositorio de datos almacenados no seu formato natural, normalmente *blobs* de obxectos ou ficheiros. Este pode incluír datos estruturados de bases de datos relacionais (filas e columnas), datos semiestruturados (CSV, XML, JSON, *logs*), datos non estruturados (correos electrónicos, documentos) e mesmo datos binarios (imaxes, audio, vídeo). 1, 2, 5–9, 29

Data Lakehouse Ferramenta encargada de implementar estruturas de datos e funcións de xestión de datos similares ás dun Data Warehouse, directamente no tipo de almacenamento de baixo custo utilizado polos Data Lakes. 1–3, 7–10, 18, 20, 23, 28, 35, 42, 59–61

Data Warehouse Ferramenta utilizada como repositorio central de datos estruturados, que poden proceder de fontes dispares. Almacena datos actuais e históricos nun único lugar, e estes utilízanse para crear informes analíticos, que á súa vez permiten tomar decisións informadas en base aos datos. 1–5, 7–9, 20, 25, 29, 32, 45

dataframe Estrutura de datos construída con filas e columnas, similar a unha base de datos ou a unha folla de cálculo. 7, 26

dataset Colección estruturada de información que se utiliza para analizar patróns, realizar investigacións, adestrar modelos de aprendizaxe automática e apoiar decisións baseadas en datos. 6, 7, 28, 33, 39

DevOps Conxunto de prácticas e ferramentas que automatizan e integran os procesos entre os equipos de desenvolvemento de software e de TI. 7

diagrama de Gantt ferramenta gráfica cuxo obxectivo é expoñer o tempo de dedicación previsto para diferentes tarefas ou actividades ao longo dun tempo total determinado. 15

estado do arte Fase máis recente de desenvolvemento dun produto, que incorpora a tecnoloxía, as ideas e as características máis novas. 1, 9, 18

job Procedemento mediante o que executar aplicacións de procesamento e análise de datos nun espazo de traballo Databricks. 29

máquina virtual Equipo virtual definido por software dentro de servidores físicos, onde só existe como código. 30

notebook Contorna virtual utilizada para *literate programming*, unha metodoloxía de porgramación. Os notebooks comparten algúns obxectivos e características coas follas de cálculo e os procesadores de texto, pero van máis aló dos seus limitados modelos de datos. 7, 26, 28, 29, 31, 39, 42

open source Software cuxo código fonte e outros dereitos que normalmente son exclusivos para quen posúe os dereitos de autor, son publicados baixo unha licenza de código aberto ou forman parte do dominio público. 1, 60

out-of-the-box Usabilidade ou funcionalidade inmediata dun produto, normalmente un dispositivo electrónico ou un programa informático. 1, 6, 8

patrón de deseño Técnica utilizada para resolver problemas comúns no desenvlvemento de software e outros ámbitos referentes ao diseño de interacción ou interfaces. 39, 40, III

Power BI Servizo de análise de datos de Microsoft orientado a proporcionar visualizacións interactivas e capacidades de intelixencia empresarial cunha interface o suficientemente simple como para que os usuarios finais poidan crear por si mesmos os seus propios informes e paneis. 37, 42, 43, 45, 56

probas unitarias Práctica de desenvolvemento software que consiste en escribir e executar probas automatizadas para verificar o correcto funcionamento de unidades individuais de código, como funcións, métodos ou clases. 42

programación áxil Metodoloxías de desenvolvemento de software centradas na idea do desenvolvemento iterativo, no que os requisitos e as solucións evolucionan mediante a colaboración entre equipos interfuncionais autoorganizados. 11

schema on read Estratexia de análise de datos, mediante a cal os datos se aplican a un plan ou esquema a medida que se extraen dunha localización almacenada, en lugar de á súa chegada. 5

script Programa ou secuencia de instrucións que é interpretado ou executado por outro programa en lugar de polo procesador do computador (como ocorre nun programa compilado). 4, 5, 18, 31, 39, 40

shell Programa informático que expón os servizos dun sistema operativo a un usuario humano ou a outros programas. 18

sistema operativo Conxunto de programas que permite manexar a memoria, disco, medios de almacenamento de información e os diferentes periféricos ou recursos dun ordenador. 30

snowpark Conxunto de bibliotecas que despregan e procesan de forma segura Python e outras linguaxes de programación en Snowflake para desenvolver canalizacións de datos, modelos de aprendizaxe automática, aplicacións, etc. 25, 26

testing Método utilizado para verificar se ao deseñar un produto dixital este cumpre cos requisitos esperados e se atopa libre de erros e brechas. 42

worksheet Interface para crear e enviar consultas SQL e ver os resultados a medida que se completan as sentenzas. 26

Bibliografía

[1] M. Armbrust, A. Ghodsi, R. Xin, and M. Zaharia, "Lakehouse: a new generation of open platforms that unify data warehousing and advanced analytics," in *Proceedings of CIDR*, vol. 8, 2021.

[2] D. Mazumdar, J. Hughes, and J. Onofre, "The data lakehouse: Data warehousing and more," *arXiv preprint arXiv:2310.08697*, 2023.

[3] B. Dageville and T. Cruanes. (2021) Industry benchmarks and competing with integrity. [En liña]. Dispoñible en: https://www.snowflake.com/blog/industry-benchmarks-and-competing-with-integrity/

[4] Databricks. (2023) Databricks vs. snowflake. [En liña]. Dispoñible en: https://www.databricks.com/databricks-vs-snowflake

[5] W. H. Inmon, *Building the data warehouse*. John wiley & sons, 2005.

[6] S. Chaudhuri and U. Dayal, "An overview of data warehousing and olap technology," *ACM Sigmod record*, vol. 26, no. 1, pp. 65–74, 1997.

[7] R. Kimball and M. Ross, *The data warehouse toolkit: the complete guide to dimensional modeling*. John Wiley & Sons, 2011.

[8] H. Fang, "Managing data lakes in big data era: What's a data lake and why has it became popular in data management ecosystem," in *2015 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)*. IEEE, 2015, pp. 820–824.

[9] C. Madera and A. Laurent, "The next information architecture evolution: the data lake wave," in *Proceedings of the 8th international conference on management of digital ecosystems*, 2016, pp. 174–180.

[10] P. P. Khine and Z. S. Wang, "Data lake: a new ideology in big data era," in *ITM web of conferences*, vol. 17. EDP Sciences, 2018.

[11] F. Nargesian, E. Zhu, R. J. Miller, K. Q. Pu, and P. C. Arocena, "Data lake management: challenges and opportunities," *Proceedings of the VLDB Endowment*, vol. 12, no. 12, pp. 1986–1989, 2019.

[12] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[13] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, "Green ai," *Communications of the ACM*, vol. 63, no. 12, pp. 54–63, 2020.

[14] D. Kreuzberger, N. Kühl, and S. Hirschl, "Machine learning operations (mlops): Overview, definition, and architecture," *IEEE Access*, 2023.

[15] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. A. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe *et al.*, "Accelerating the machine learning lifecycle with mlflow." *IEEE Data Eng. Bull.*, vol. 41, no. 4, pp. 39–45, 2018.

[16] E. Bisong and E. Bisong, "Kubeflow and kubeflow pipelines," *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*, pp. 671–685, 2019.

[17] Y. Zhang and Z. G. Ives, "Juneau: data lake management for jupyter," *Proceedings of the VLDB Endowment*, vol. 12, no. 12, 2019.

[18] D. Oreščanin and T. Hlupić, "Data lakehouse-a novel step in analytics architecture," in *2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO)*. IEEE, 2021, pp. 1242–1246.

[19] B. Shiyal, "Modern data warehouses and data lakehouses," in *Beginning Azure Synapse Analytics: Transition from Data Warehouse to Data Lakehouse*. Springer, 2021, pp. 21–48.

[20] M. Armbrust, T. Das, L. Sun, B. Yavuz, S. Zhu, M. Murthy, J. Torres, H. van Hovell, A. Ionescu, A. Łuszczak *et al.*, "Delta lake: high-performance acid table storage over cloud object stores," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 3411–3424, 2020.

[21] B. Dageville, T. Cruanes, M. Zukowski, V. Antonov, A. Avanes, J. Bock, J. Claybaugh, D. Engovatov, M. Hentschel, J. Huang *et al.*, "The snowflake elastic data warehouse," in

Proceedings of the 2016 International Conference on Management of Data, 2016, pp. 215–226.

[22] P. Jain, P. Kraft, C. Power, T. Das, I. Stoica, and M. Zaharia, "Analyzing and comparing lakehouse storage systems," in *CIDR*. CIDR, 2023.

[23] R. B. Wakode, L. P. Raut, and P. Talmale, "Overview on kanban methodology and its implementation," *IJSRD-International Journal for Scientific Research & Development*, vol. 3, no. 02, pp. 2321–0613, 2015.

[24] M. O. Ahmad, J. Markkula, and M. Oivo, "Kanban in software development: A systematic literature review," in *2013 39th Euromicro conference on software engineering and advanced applications*. IEEE, 2013, pp. 9–16.

[25] K. Koppenhaver. (2022, Jul) Using kanban in software development. [En liña]. Dispoñible en: https://steady.space/blog/using-kanban-in-software-development/

[26] L. Jun and L. Ling, "Comparative research on python speed optimization strategies," in *2010 International Conference on Intelligent Computing and Integrated Systems*, 2010, pp. 57–59.

[27] (2022, Jun) Tiobe index. [En liña]. Dispoñible en: https://www.tiobe.com/tiobe-index/

[28] R. Soni, "Clustering and micro-partitions," in *Snowflake SnowPro™ Advanced Architect Certification Companion: Hands-on Preparation and Practice*. Springer, 2023, pp. 91–104.

[29] (2024) Understanding snowflake table structures. [En liña]. Dispoñible en: https://docs.snowflake.com/en/user-guide/tables-micro-partitions

[30] (2024, Mar) Connect to data sources. [En liña]. Dispoñible en: https://docs.databricks.com/en/connect/index.html

[31] (2023, Oct) Language-specific introductions to databricks. [En liña]. Dispoñible en: https://docs.databricks.com/en/languages/index.html

[32] Databricks pricing. [En liña]. Dispoñible en: https://www.databricks.com/product/pricing

[33] B. Gregg, *Systems performance: enterprise and the cloud*. Pearson Education, 2014.

[34] R. O. Nambiar and M. Poess, "The making of tpc-ds," in *Proceedings of the 32nd International Conference on Very Large Data Bases*, ser. VLDB '06. VLDB Endowment, 2006, p. 1049–1058.

[35] A. Anand and A. Uddin, "Importance of software testing in the process of software development," *International Journal for Scientfic Research and Development*, vol. 12, no. 6, 2019.

[36] A. Kumar and A. Shawkat Ali, "Big data visualization tools, challenges and web search popularity-an update till today," in *International Conference on Big Data Intelligence and Computing*. Springer, 2022, pp. 305–315.

[37] M. Corujo, A. Regueiro, A. Xuiz, I. Garcia, G. Lopez, R. Rey, and J. Touriño, "Evaluación de rendimiento de plataformas data lakehouse: Snowflake y databricks," in *Jornadas Sarteco*, 2024.