2 – Delta Lake vs Iceberg

Delta Lake to rozwiazanie stworzone przez Databricks i mocno wspierane przez Microsoft. Jest ono naturalnie zintegrowane z platforma Azure i srodowiskiem Spark. Zapewnia pelne wsparcie transakcji ACID, latwe wersjonowanie danych oraz automatyczna optymalizacje danych – bez potrzeby dodatkowej konfiguracji. Schema evolution dziala bezproblemowo, co oznacza ze zmiany w strukturze danych nie wymagaja duzej ingerencji w pipeline.

Apache Iceberg natomiast to projekt rozwijany w ramach Apache Software Foundation – powstal w Netflixie i stawia na otwartosc i elastycznosc. Iceberg lepiej sprawdza sie w srodowiskach z wieloma silnikami zapytan np. Trino, Presto, Hive czy Flink. Ma bardziej zaawansowane mechanizmy wersjonowania i zarzadzania metadanymi, ale ich wdrozenie wymaga wiecej pracy i znajomosci konfiguracji. Nie jest tak dobrze zintegrowany z Azure jak Delta.

Jesli chodzi o optymalizacje – Delta oferuje mechanizmy typu Z-Ordering i Auto Optimize, co znacznie upraszcza zarzadzanie danymi. Iceberg natomiast bazuje na filtracji po metadanych, co daje wieksza kontrole, ale wymaga zrozumienia wiecej niz tylko samego kodu SQL.

Delta sprawdzi sie najlepiej gdy:

- korzystamy z Databricks lub ekosystemu Azure
- zalezy nam na szybkim wdrozeniu z automatyczna optymalizacja
- chcemy miec wsparcie dla machine learningu, streamingu
- potrzebujemy prostych upsertow i transformacji typu batch/stream

To dobre rozwiazanie do budowy lakehouse'ow w srodowisku silnie opartym o Spark.

Iceberg to lepszy wybor gdy:

- pracujemy w srodowisku z roznymi silnikami (Presto, Hive, Flink itd.)
- dzialamy on-prem lub hybrid
- zalezy nam na pelnej kontroli wersjonowania i pracy na snapshotach danych
- mamy czeste zmiany schematu i potrzebujesz zaawansowanej ewolucji danych

Iceberg swietnie radzi sobie w duzych organizacjach z rozproszona architektura danych.

3 – Krytyka medalion

Architektura medalionowa czyli podzial danych na warstwy bronze silver i gold ma swoje zalety, ale tez sporo ograniczen. W wielu przypadkach jej stosowanie jest przesadne lub wrecz przeszkadza w efektywnej pracy z danymi.

Po pierwsze jest po prostu zbyt zlozona – trzy warstwy oznaczaja wiecej kodu, wiecej testow i wiecej punktow awarii. Dane sa kopiowane miedzy warstwami co podnosi koszty i utrudnia sledzenie ich drogi. Dla czystych zrodel danych warstwa bronze jest czesto niepotrzebna, a mimo to ja sie tworzy na sile.

Kolejna sprawa to opoznienia – dane musza przejsc przez wszystkie warstwy, zanim dotra do uzytkownika, co jest problematyczne np. w analizach bliskich rzeczywistego czasu. Architektura nie wspiera tez dobrze real-time streamingu. Dodatkowo logika transformacji potrafi sie dublowac w warstwach silver i gold, przez co trudniej to utrzymac.

Dla analitykow biznesowych system bywa nieprzejrzysty – trzeba wiedziec z ktorej warstwy korzystac, nie ma jednej wersji prawdy. Jesli chodzi o uprawnienia, to kazda warstwa moze wymagac osobnych konfiguracji, a bez Unity Catalog robi sie to szybko problematyczne. Nie ma tez dobrego mechanizmu do sledzenia historii zmian – dokumentacja czesto nie nadaza za transformacjami.

Schematy danych trzeba aktualizowac oddzielnie dla kazdej warstwy, co bywa upierdliwe. Kazda dodatkowa warstwa zuzywa wiecej zasobow: CPU, IO, storage. Debugowanie problemow w takich pipeline'ach jest jak szukanie igly w stogu siana – trudne, wolne i kosztowne.

Medalion wymusza tez tworzenie duzej liczby tabel, monitoringu, pipeline'ow – dla kazdej warstwy osobno. Nowoczesne narzedzia jak DuckDB albo dbt pozwalaja pominac czesc warstw, zachowujac przejrzystosc. W efekcie architektura medalionowa potrafi doprowadzic do izolacji danych – dane zamkniete w jednej warstwie moga byc niewidoczne w innej.

Na koniec trzeba wspomniec, ze ta architektura nie jest latwa do wdrozenia bez doswiadczonego zespolu data engineeringowego. I ze jest mocno promowana przez Databricks – ciezko ja przeniesc do innych platform 1:1 bez utraty funkcjonalnosci.