Zadanie 2

**Delta Lake** to technologia stworzona przez firmę Databricks. Jest głęboko zintegrowana z platformą Azure Databricks i działa natywnie z Apache Spark. Pozwala na operacje typu *upsert* (merge), oferuje wersjonowanie danych, time-travel (czyli cofanie się do wcześniejszego stanu danych), obsługuje ACID i jest zoptymalizowana pod kątem przetwarzania strumieniowego i batchowego. Jest idealnym wyborem, gdy system opiera się głównie na Databricks i Spark, a celem są analizy biznesowe, transformacje danych i machine learning.

**Apache Iceberg** to format otwarty, wspierany przez społeczność i firmy takie jak Netflix czy Apple. Jest bardziej neutralny technologicznie – współpracuje z wieloma silnikami, takimi jak Spark, Flink, Trino, Presto czy Snowflake. Działa bardzo dobrze w dużych środowiskach hurtowni danych i w projektach, gdzie wymagana jest wysoka skalowalność, elastyczność i niezależność od konkretnego dostawcy platformy (np. Databricks). Iceberg również wspiera wersjonowanie, time-travel i transakcje ACID.

**Rekomendacja dla klienta:**

* Jeśli klient korzysta z **Azure Databricks** i zależy mu na prostym, wydajnym i dobrze wspieranym rozwiązaniu – **Delta Lake** będzie najlepszym wyborem.
* Jeśli klient chce budować rozwiązanie bardziej otwarte, wieloplatformowe, z integracją z różnymi silnikami lub zamierza przetwarzać olbrzymie ilości danych w skalowalny sposób – **Apache Iceberg** będzie lepszy.

Zadanie 3

Architektura medalionowa dzieli dane na trzy warstwy:  
**Bronze** (surowe dane), **Silver** (oczyszczone dane) i **Gold** (dane do analizy biznesowej). Choć jest to popularny model, ma też swoje wady i nie zawsze jest najlepszym wyborem.

Poniżej znajduje się 20 punktów krytyki wraz z krótkim wyjaśnieniem każdego z nich:

1. **Złożoność utrzymania** – trzeba tworzyć i zarządzać wieloma pipeline’ami między warstwami.
2. **Duplikacja danych** – te same dane są przetwarzane i przechowywane w trzech miejscach, co zwiększa koszty.
3. **Większe zużycie zasobów** – każda warstwa to osobny proces, co zwiększa zużycie CPU, RAM i czasu obliczeń.
4. **Opóźnienie dostępu do danych** – pełna analiza możliwa dopiero po przetworzeniu danych aż do warstwy Gold.
5. **Zbędność w prostych przypadkach** – dla małych projektów wystarczyłaby tylko jedna lub dwie warstwy.
6. **Trudności w debugowaniu błędów** – problem może się pojawić na różnych etapach, co utrudnia identyfikację źródła.
7. **Wysoki próg wejścia** – dla małych zespołów lub firm może być to zbyt skomplikowane rozwiązanie.
8. **Brak automatyzacji** – często wymaga ręcznej konfiguracji i utrzymywania przepływów między warstwami.
9. **Utrudniona elastyczność** – sztywna struktura powoduje, że trudno szybko zmienić logikę przetwarzania.
10. **Koszt storage** – każda warstwa zwiększa potrzebę na przestrzeń dyskową.
11. **Przesadne skupienie na strukturze** – może ograniczyć kreatywność analityków w eksploracji danych.
12. **Potrzeba dodatkowego monitorowania** – każda warstwa wymaga osobnych logów i alertów.
13. **Ryzyko niespójności danych między warstwami** – jeśli coś pójdzie nie tak w jednej z warstw, dane mogą się rozjechać.
14. **Zbyt duży narzut dla danych jednorazowych** – dla danych ad-hoc lub tymczasowych tworzenie całej struktury nie ma sensu.
15. **Nieefektywne dla danych strumieniowych** – czasami nie ma potrzeby trzech etapów w czasie rzeczywistym.
16. **Nadmierna liczba plików pośrednich** – generuje wiele plików, które trzeba przechowywać i zarządzać nimi.
17. **Utrudnione testowanie** – testy jednostkowe dla takiej architektury są bardziej złożone.
18. **Zależność od konkretnej platformy (np. Databricks)** – trudniej przenieść logikę gdzie indziej.
19. **Brak standaryzacji między firmami** – każdy może interpretować warstwy nieco inaczej.
20. **Zwiększone ryzyko technicznego długu** – każda nowa warstwa to dodatkowy kod, który trzeba utrzymywać i aktualizować.