Lernzettel

Datenbank-Management vs. Data-Stream-Management: Unterschiede, Schnittstellen, Herausforderungen

Universität: Technische Universität Berlin
Kurs/Modul: Informationssysteme und Datenanalyse

Erstellungsdatum: September 19, 2025

Zielorientierte Lerninhalte, kostenlos!
Entdecke zugeschnittene Materialien für deine Kurse:

https://study.AllWeCanLearn.com

Informationssysteme und Datenanalyse

Lernzettel: Datenbank-Management vs. Data-Stream-Management

(1) Grundbegriffe.
Datenbank-Management-System (DBMS): systematisch verwaltet persistente, strukturierte Daten in einer Datenbank. Typische Merkmale:

- relationale oder NoSQL-Datenmodelle,
- SQL- bzw. deklarative Abfragen,
- Transaktionen mit ACID-Eigenschaften (Atomarität, Konsistenz, Isolation, Dauerhaftigkeit),
- persistente Speicherung, Indizes, Stabilität bei Fehlertoleranz.

Data-Stream-Management-System (DSMS): verarbeitet kontinuierlich entstehende Datenströme in Echtzeit oder nahezu Echtzeit. Typische Merkmale:

- Stream-Modelle mit Tupel- oder Ereignisorientierung,
- kontinuierliche, window-basierte Abfragen (z. B. für aggregierte Kennzahlen),
- geringe Latenz, oft Handel zwischen Konsistenz und Verfügbarkeit,
- Zustandsverwaltung über Streams, Fault Tolerance, Skalierbarkeit.

Data Warehousing kann als zentrale, persistente Historie gesehen werden, die aus verschiedensten Quellen zusammengeführt wird; Streams können in DW-Architekturen eingeflossen werden, um Echtzeit- oder Near-Real-Time-Analytics zu unterstützen.

(2) Architektur und Schnittstellen.
DBMS-Architektur: Schema, Speicherengine, Transaktionslog, Indizes, SQL-Operatoren, reliable Update/Query-Operatoren, Persistenz und Snapshot-Isolation in Transaktionen.

DSMS-Architektur: Streaming-Operatoren (Filter, Join, Aggregation, Projection), Fenster-Definitionen (Tumbling, Hopping, Sliding), kontinuierliche Abfragen, Zustandsverwaltung und Verlauf (Checkpointing, Fault Tolerance).

Interoperabilität und Schnittstellen:

- Change Data Capture (CDC) zur Integration von DBMS-Daten in Streaming-Pipelines,
- Messaging-Backbones (z. B. Kafka) als Puffer/Brücke zwischen DBMS, DSMS und Consume-Tools,
- ETL/ELT-Prozesse, Echtzeit-Adapter, Data-Lake- bzw. Data-Warehouse-Schnittstellen,
- Abfragesprachen: SQL im DBMS; Continuous-Query- oder Streaming-SQL-Ansätze im DSMS.

(3) Unterschiede in Eigenschaften.

- *Latenz und Durchsatz*: DBMS preceding zahlreiche Transaktionen in Upload-/Persistenz-Schritten; DSMS zielt auf geringe Latenz pro Tupel ab, oft mit hohem Durchsatz.

- *Konsistenz vs. Verfügbarkeit*: DBMS typischerweise stark konsistent (ACID); DSMS priorisiert oft zeitnahe Ergebnisse, Konsistenzmodelle variieren (z. B. eventual oder exactly-once bei Backends).
- *Datenmodell*: DBMS speichert deterministische, persistente Zustände; DSMS verarbeitet kontinuierliche Ereignisse in Flussformen (Events, Streams).
- *Verarbeitungskonzept*: DBMS verwendet Batching/Persistenz; DSMS arbeitet mit kontinuierlicher Abfrage bzw. window-basierten Berechnungen.
- *Zustand und Windowing*: DSMS benötigt oft definierte Fenster (z. B. 5-Minute-Windows) und Zustand, DBMS eher transaktionsbasiert.

(4) Schnittstellen und Interoperabilität (Anwendungsfälle).

- CDC aus einem relationalen DBMS in eine Streaming-Pipeline, um Echtzeit-Kennzahlen zu berechnen.
- Streaming-ETL, um Rohdaten aus Streams in ein Data Warehouse zu laden (ELT-Ansatz möglich).
- Konsolidierung von Historie (DW) und Echtzeit-Ansichten (DSMS) für Dashboarding und Alarmierung.
- CEP (Complex Event Processing) über Streams zur Erkennung komplexer Muster in Echtzeit.

(5) Typische Herausforderungen.

- *Event-Latenz und Ordering*: verspätete oder out-of-order eingehende Ereignisse erfordern robuste Zeitstempel-Strategien.
- *Genau-Eins-Verarbeitung/Idempotenz*: sicherstellen, dass wiederholte Tupel keine doppelten Effekte erzeugen.
- *Backpressure und Skalierbarkeit*: Geräte/Anwendungen produzieren mehr Daten als verarbeitet werden können.
- *Schema-Evolution*: Änderungen am Datenmodell müssen rückwärtskompatibel oder migrierbar sein.
- *Konsistenzüberbrückung*: Brücken zwischen transaktionalen Datenbanken und Streaming-Analysen müssen Konsistenz- und Latenzanforderungen berücksichtigen.
- *Zustandsmanagement*: DSMS benötigt angemessene Zustandsverwaltung, Checkpoints, Recovery-Strategien.

(6) Architekturmuster und Entscheidungsrichtlinien.

- *Lambda-Architektur*: Batch- (DW) + Speed- (DSMS) Layer; einfache Wartung zu trennen, aber potenziell duplizierte Logik.

- *Kappa-Architektur*: alles als Stream, geringere Komplexität, ideal bei starken Streaming-Anforderungen.
- *Hybrid-Ansätze*: DSMS erzeugt Echtzeit-Features, DBMS speichert persistente, bereinigte Datenversionen, DW dient langfristiger Analyse.
- *Datenintegrationsmuster*: CDC, Change-Feeds, Event-Sourcing, Messaging-Integrationen.

(7) Checkliste zur Systemauswahl im Praxisfall.

- Benötigen Sie Echtzeit- oder Near-Echtzeit-Analysen? Dann DSMS oder Streaming-Komponenten sinnvoll.
- Muss Konsistenz strikt garantiert werden? Dann stärker DBMS-zentriert arbeiten oder starke Exactly-Once-Guarantie sicherstellen.
- Welche Datenmengen ankommen? Bei hohen Volumen eventuell DSMS mit window-basierten Aggregationen.
- Welche Analysen? Ad-hoc-Abfragen bevorzugen DBMS; fortlaufende Kennzahlen in DSMS.
- Wie sieht die Integrationslandschaft aus? Vorhandene Systeme, Kafka/Kubil, CDC-Quellen, ETL/ELT-Strategien.

(8) Fazit und Merksätze.

- DBMS und DSMS adressieren unterschiedliche Anforderungen: Persistenz und transaktionale Konsistenz versus geringe Latenz und kontinuierliche Verarbeitung.
- Eine integrierte Architektur nutzt beides dort sinnvoll, wo Echtzeit-Responsen und stabile Langzeitanalysen zusammenkommen.
- Die Wahl der Architektur hängt stark vom Anwendungsprofil, den Datenströmen und den Anforderungen an Konsistenz, Latenz und Kosten ab.

(9) Beispiel-Illustration (du kannst diese als Mini-Fallstudie verwenden).
Ein E-Commerce-System produziert Bestell-Events in Streams. Ein DSMS berechnet in Echtzeit Kennzahlen wie durchschnittlicher Bestellwert pro Region in 5-Minuten-Windows. Parallel speichert das DBMS transaktionale Orders in einer relationalen Datenbank. CDC-Feeds liefern Änderungen in den Streaming-Pfad, damit Dashboards sowohl Echtzeit-Metriken als auch historische Trends aus DW/DBMS zeigen können.