

M 6

6.1 · DB Design for a Startup: Keep It Simple

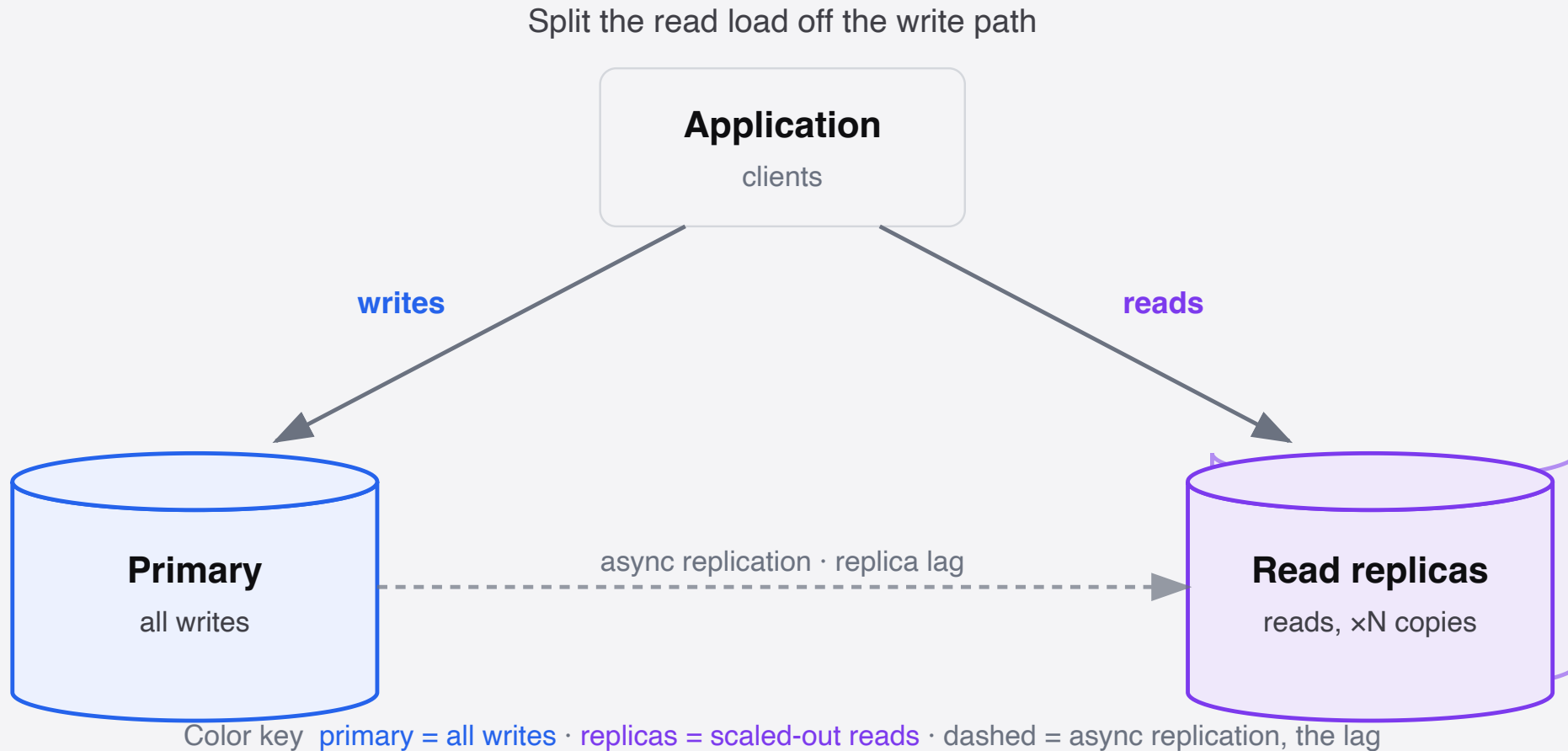
Most apps stay within one machine.

PostgreSQL on AWS RDS/Cloud SQL/Azure will handle your first 10 million users.

Examples

Company	Exit Value	Database Strategy
Instagram	\$1B	PostgreSQL until acquisition
WhatsApp	\$19B	PostgreSQL handled 900M users
Notion	\$10B valuation	Still on PostgreSQL + replicas
Stack Overflow	Still thriving	4 SQL Servers, 1.3B pageviews/month
OpenAI (ChatGPT)	800M weekly users	One primary Postgres + ~50 read replicas (2026)

Read replicas



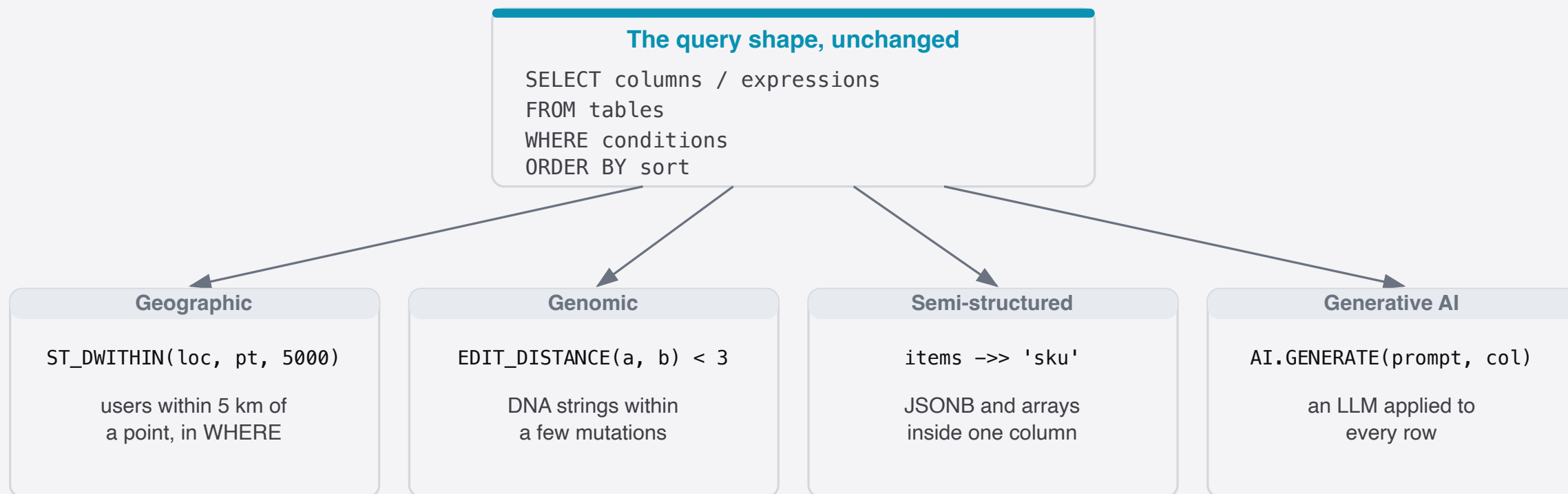
M6

SQL's Complex Types

SQL extends beyond INT and TEXT.

Same SQL shape, new data types

SELECT / FROM / WHERE / ORDER BY stays; only the function in the clause changes



cyan = the query shape that never changes · **grey** = the domain function you drop into a clause

JSONB: a document column, still with ACID and joins

```
-- One Postgres node with JSONB: ACID and joins from the relational side,  
-- a flexible document schema from the JSONB column.
```

```
CREATE TABLE orders (  
  id          SERIAL PRIMARY KEY,  
  user_id     INTEGER REFERENCES users(id),  
  items       JSONB,           -- Flexible per-order document  
  created_at  TIMESTAMP  
);
```

[illegible]

Generative AI: an LLM call as a column

```
-- Tag, summarize, and translate songs with an LLM (BigQuery)
SELECT
  title,
  artist,
  AI.CLASSIFY(lyrics, ["Happy", "Melancholic", "Energetic", "Chill"]) AS vibe,
  AI.GENERATE("Summarize the emotional arc IN one sentence", lyrics) AS ai_summary,
  AI.TRANSLATE(title, "Spanish") AS title_es
FROM Songs
WHERE AI.IF("The song IS about a summer romance", lyrics)
      AND play_count > 100000;
```

Custom logic: define a function once

```
-- SQL UDF: simple, fast, inlined
CREATE TEMP FUNCTION viralScore(plays INT64, shares INT64)
RETURNS FLOAT64
AS (plays * 0.3 + shares * 10);

-- JavaScript UDF: when SQL alone falls short
CREATE TEMP FUNCTION normalizeTitle(t STRING)
RETURNS STRING
LANGUAGE js AS """
    return t.toLowerCase().trim();
    """;
```

Custom logic: call it like a built-in

```
SELECT
  normalizeTitle(s.title) AS title,
  s.artist,
  viralScore(s.play_count, s.share_count) AS viral_potential
FROM Songs s
WHERE s.release_year = 2024
      AND s.play_count > 1000
ORDER BY viral_potential DESC
LIMIT 100;
```

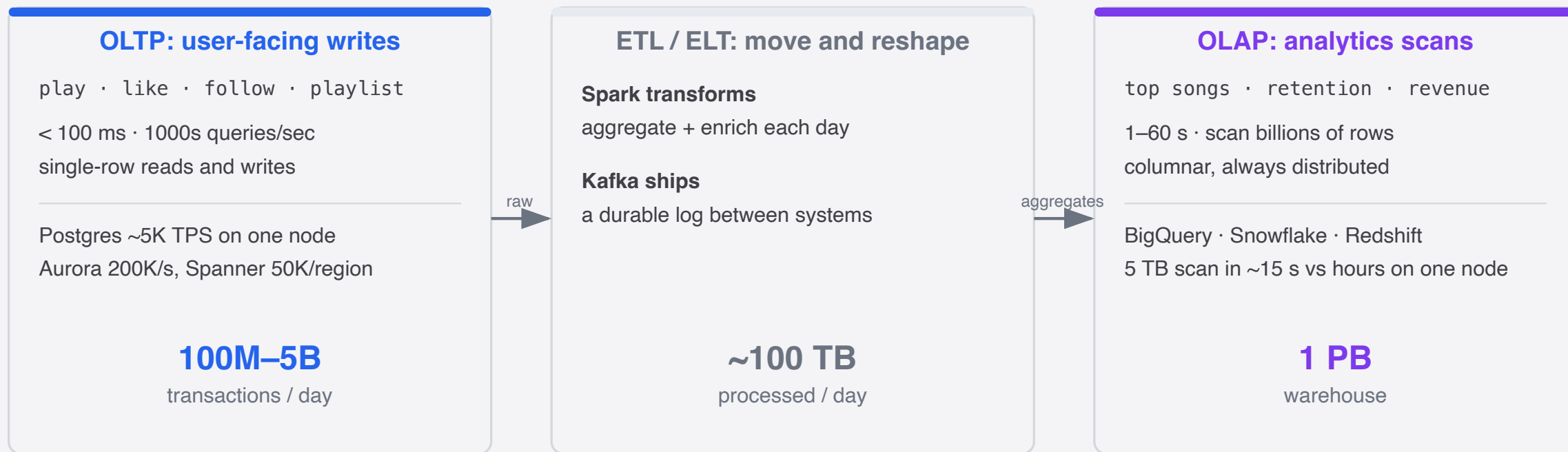
M 6

Database Design at Big-Tech Scale

At big-tech scale, the data stack splits into three systems: OLTP for user-facing writes, OLAP for analytics over billions of rows, and an ETL/ELT pipeline that moves data from one to the other.

Three systems, one data stack

OLTP serves users, OLAP answers analytics, ETL or ELT moves data between them



blue = transactional OLTP · **violet** = analytical OLAP warehouse · **grey** = the ETL/ELT bridge

A Spark ETL job, roughly as it looks in real code

```
# Simplified Apache Spark job: aggregate yesterday's plays
from pyspark.sql.functions import count, countDistinct, avg

def spotify_daily_job(spark, songs_dim, users_dim):
    # 1. EXTRACT from OLTP (a PostgreSQL read replica)
    raw_listens = spark.read.format("jdbc") \
        .option("url", "jdbc:postgresql://replica.spotify.com/spotify") \
        .option("query", "SELECT * FROM listens WHERE date = CURRENT_DATE - 1") \
        .load()

    # 2. TRANSFORM: aggregate and enrich
    daily_stats = raw_listens \
        .join(songs_dim, "song_id") \
        .join(users_dim, "user_id") \
        .groupBy("date", "genre", "country", "age_group") \
        .agg(
            count("*").alias("play_count"),
            countDistinct("user_id").alias("unique_users"),
            avg("listen_duration").alias("avg_duration")
        )

    # 3. LOAD to OLAP (BigQuery)
    daily_stats.write.format("bigquery") \
        .mode("append") \
        .option("table", "spotify_warehouse.daily_stats") \
        .option("partitionField", "date") \
        .save()
```

Your CS145 Data Systems License



UNIV. OF DATA MAGIC

CS145 DATA SYSTEMS LICENSE

Certified dangerous to operate data systems





Platforms:
BigQuery • Postgres • Spark • Snowflake



Skills:
Window functions • CTEs • Subqueries



Status:
Safe for GBs



Restriction:
< 10 GB scale



TB SCALE: NOT AUTHORIZED
One bad query = one very bad invoice

Query responsibly 