Workshop 2

Data Architecture for a Music Streaming Platform based on Spotify

Brayan Stiven Yate Prada – Student ID: 20192020151
Holman Andres Alvarado Diaz – Student ID: 20201020032
Universidad Distrital Francisco José de Caldas

Faculty of Engineering
Course: Databases II
Professor: Carlos Andrés Sierra Virgüez

May 29, 2025

Contents

1 Data System Architecture

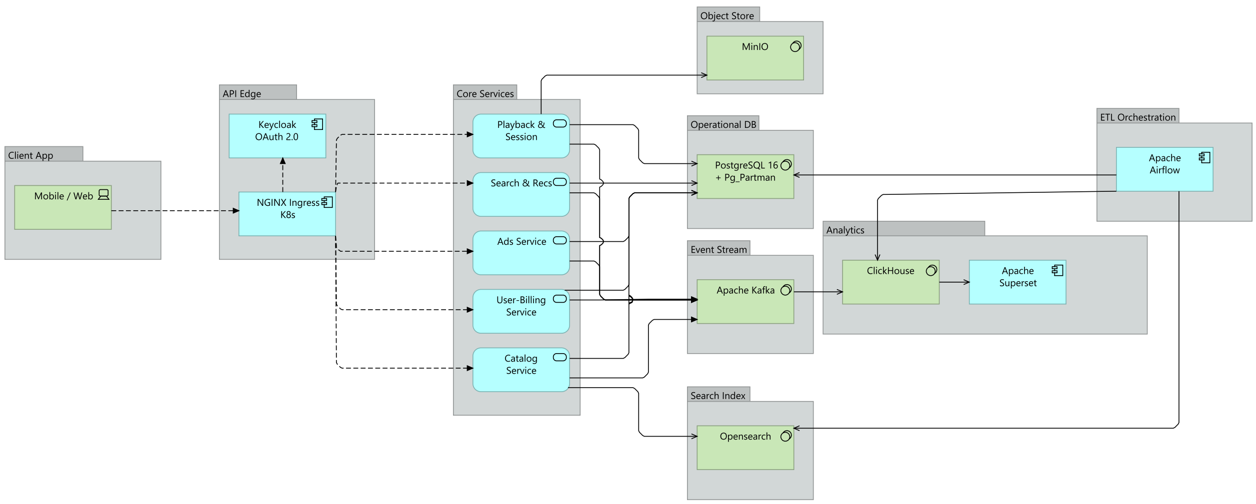

Figure 1: High-level Architecture Diagram

1.1 Justification for Each Technology

Component	Technology	Reason	Primary Data Stored
API Edge	**NGINX Ingress Controller**	Ships with every Kubernetes distribution; trivial TLS termination; rich ecosystem.	Proxies all HTTPS traffic.
Authentication	**Keycloak**	OAuth2/OIDC server; social log-ins, multi-factor authentication.	User credentials and tokens (hashed / JWT).
Operational SQL	**PostgreSQL 16 + pg_partman**	Logical/physical replication; partitioning and native JSONB; easy to query.	Users, subscriptions, catalog (artists, albums, tracks), playlists, payments, ads, social graph.
Search / Autocomplete	**OpenSearch**	Apache-licensed Elasticsearch fork; scales from single node to multi-node; SQL plug-in for analysts.	Denormalised documents for tracks, artists, playlists.
Object Storage	**MinIO**	Lightweight S3 clone; single binary; erasure coding; deployable on-prem or on any cloud.	Audio files, cover art, long-term raw logs, GDPR exports.

Event Streaming	**Apache Kafka**	Industry-standard durable, ordered, replayable event log; hundreds of client SDKs.	Play/skip events, ad impressions, change-data capture (CDC) from PostgreSQL via Debezium.
Real-Time / Batch Analytics	**ClickHouse**	Blazing fast, column-oriented, SQL-native; licence-free; handles $> 100\,000$ inserts/s and sub-second dashboards.	Aggregated play counts, creator dashboards, advertising metrics, Wrapped-style reports.
Orchestration / ETL	**Apache Airflow**	Familiar DAG UI; Python-first; vast provider library (Debezium → ClickHouse, backups, ML).	Schedules nightly royalty jobs, table vacuums, data-quality checks.
Business Intelligence	**Apache Superset**	Modern OSS BI; drop-in ClickHouse driver; RBAC and SSO via Keycloak.	Front-end only – reads ClickHouse for charts.

1.2 End-to-End Data Flow (grounded in the schema)

Creator Upload

1. The *Catalog Service* receives the audio file and its metadata.
2. Binary data are stored in a MinIO bucket `/audio/`.
3. Rows for artist, album and track are inserted into PostgreSQL (ER diagram tables).
4. Debezium streams these row-level changes to Kafka; an OpenSearch sink connector indexes the documents so the track becomes searchable within seconds.

Listener Playback

1. The *Playback Service* retrieves `track_id`, rights and bitrate information from PostgreSQL, signs a presigned MinIO URL and returns it to the client.
2. The client begins streaming; every 5 s it emits a `play-tick` JSON message to Kafka.

Real-Time Aggregation ClickHouse materialised views ingest the `play-tick` topic directly—no Spark or Flink cluster is required for the minimum viable product (MVP). Dashboards in Superset refresh automatically for artists and advertising operations.

Nightly Jobs *(Airflow)*

- `royalty_etl.py`: SQL executed in ClickHouse summarises plays by `track_id` and writes the payout CSV to the PostgreSQL `payouts` table.
- `search_reindex.py`: Performs a bulk re-sync of OpenSearch from PostgreSQL to catch edge cases.
- **Back-ups**: pg_dump streams are stored as compressed objects in MinIO.

Social Graph and Recommendations The `user_follow` bridge table lives in PostgreSQL; for the MVP we run simple collaborative-filtering in Python inside Airflow once per day and write top-N recommendations into `user_recs`. When sub-50 ms cold-start latencies become necessary these features can migrate to Redis, but not in version 1.

1.3 Why This Trimmed Stack Meets Our Goals

- **Feasible for a small team:** only **eight** stateful services (PostgreSQL, MinIO, Kafka, OpenSearch, ClickHouse, Keycloak, Cassandra, Trino + Iceberg) – all ship Helm charts and run on three modest VMs or a single Kubernetes cluster.
- **Single source of truth:** PostgreSQL hosts the entire ER diagram with just two bridge tables (`playlist_track`, `user_follow`) and one fact table (`ad_impression`). OLTP rows are partitioned by tenant (`user_id mod N`) enabling future shard-out.
- **Analytics without Hadoop:** ClickHouse reads Kafka directly, avoiding Spark/HDFS.
- **Search within seconds:** Debezium + Kafka keeps OpenSearch in near-real-time sync.
- **Fully open-source:** no licences, no managed-only products.
- **Clear upgrade path:** if traffic explodes we can add Postgres replicas, expand Kafka, deploy more ClickHouse shards or adopt Citus; none require a rewrite.

2 Information Requirements

#	Information Type	Description	Storage	Business Link	Key User Stories
1	**Catalog Metadata**	Track, album, artist info. Enables search & legal playback.	PostgreSQL (`tracks`, `albums`, `artists`); OpenSearch cache.	Global audio library; Key Partners.	Search; Playback; Upload Audio.
2	**Audio References**	Presigned URLs + bitrate list for playback via CDN or MinIO.	Generated by Playback Service from object metadata.	Seamless streaming.	Playback; Device Integration.
3	**User Profile**	Display name, avatar, language, tier, parental controls. UI personalization + policy enforcement.	PostgreSQL (`users`).	Personalisation; Segmentation.	Profile Management; Register; Parental Control.
4	**Subscription & Billing**	Plan, renewal, token, grace state. Controls privileges and ad access.	PostgreSQL (`subscriptions`, `payment_method`).	Recurring revenue.	Subscription Management; Premium Access.

5	**Playback State**	Position, device ID, network, heartbeat. Enables adaptive bitrate + resume.	Kafka → ClickHouse or in-memory cache.	Playback continuity.	Adaptive Streaming; Resume Playback.
6	**Suggestion**	Ranked track IDs + labels (e.g., repeat, trending).	PostgreSQL JSONB (`user_recs`).	Discovery; ML-driven value.	Discover; Playlist Creation.
7	**Search Suggestions**	Tokens, fuzzy matches (tracks, artists, playlists).	OpenSearch.	Search efficiency.	Content Search.
8	**Social Graph**	Follows, playlist rights, friend activity.	PostgreSQL (`user_follow`, `playlist_track`); Kafka.	Community features.	Follow Users; Collaborative Playlists.
9	**Ads & Creatives**	Targeting, CPM, assets; user context.	PostgreSQL (`ad_campaign`, `ad_creative`) + in-memory rules.	Ad revenue.	Launch Campaigns; Real-Time Ads.
10	**Creator Analytics**	Play counts, geo maps, promo-lift.	ClickHouse views via Superset.	Creator tools; Partner support.	View Stats; Track Promotion Impact.
11	**Royalty Reports**	Stream counts, payouts per rightsholder.	Airflow ETL → PostgreSQL (`payouts`); MinIO.	Royalty costs.	(Internal workflows).
12	**Compliance Logs**	Admin actions, GDPR links, token audits.	Append-only Postgres + WORM MinIO.	Trust; Legal compliance.	Data Requests; Audit Trails.

Traceability to Business Model and User Stories

- **Value Proposition** ("global audio library", ad-free Hi-Fi, discovery) requires fast access to items 1, 2, 6 and 5.
- **Revenue Streams** (subscriptions & ads) rely on items 4 and 9.
- **Customer Relationships / Segments** (community, stats) depend on items 3, 8 and 10.
- **Cost Structure / Key Partners** (royalty payouts) require items 1 and 11.
- **Compliance & Trust** are covered by item 12 and secured payment tokens.

3 Query Proposals

3.1 User and Subscription Management (PostgreSQL)

```
-- 1A. Premium users in top-5 high-usage countries
```

```
WITH top_countries AS (
    SELECT country
    FROM   play_events_daily  -- daily roll-up loaded by Airflow
    ORDER  BY total_plays DESC
    LIMIT  5
)
SELECT u.user_id,
       u.display_name,
       s.plan,
       s.country
FROM   users u
JOIN   subscriptions s USING (user_id)
WHERE  s.status   = 'active'
  AND  s.plan     = 'premium'
  AND  s.country  IN (SELECT country FROM top_countries);
```

Purpose: identify premium markets for targeted marketing.

```
-- 2A. Potential account sharing (>=3 countries in 30 days)
SELECT user_id,
       COUNT(DISTINCT ip_country) AS unique_countries_30d
FROM   session_logs PARTITION FOR (CURRENT_DATE - INTERVAL '30 days')
GROUP  BY user_id
HAVING COUNT(DISTINCT ip_country) > 3;
```

Purpose: detect family-plan abuse.

3.2 Search and Discovery (OpenSearch)

```
GET /tracks/_search
{
  "query": {
    "function_score": {
      "query": { "match": { "title": "drake" }},
      "field_value_factor": {
        "field":   "play_count",
        "factor":  0.1,
        "modifier":"sqrt",
        "missing": 1
      },
      "boost_mode": "multiply"
    }
  },
  "sort": [
    { "_score": "desc" },
    { "release_date": "desc" }
```

```
  ]
}
```

Purpose: relevancy-boosted autocomplete.

3.3 Playback Behaviour (ClickHouse)

```
-- 3A. Early-churn indicator for new sign-ups (<=30s skips)
SELECT
    user_id,
    COUNT()            AS total_plays,
    sum(duration < 30) AS short_skips
FROM  play_events
WHERE signup_date >= today() - 7
GROUP BY user_id
HAVING total_plays > 5;

-- 3B. Rapid-skip bot detection (last 10 min)
SELECT *
FROM   play_events
PREWHERE user_id = 'u_999'
    AND event_time > now() - INTERVAL 10 MINUTE
WHERE  duration < 15;
```

3.4 Campaign and Cohort Analytics (ClickHouse)

```
-- 4A. Retention by sign-up cohort
WITH cohort AS (
  SELECT user_id,
         toDate(min(event_time)) AS cohort_date
  FROM   play_events
  GROUP  BY user_id
)
SELECT cohort_date,
       activity_date,
       uniqExact(user_id) AS active_users
FROM (
  SELECT user_id,
         toDate(event_time) AS activity_date
  FROM   play_events
) e
JOIN cohort USING user_id
GROUP BY cohort_date, activity_date
ORDER BY cohort_date, activity_date;
```

```sql
-- 4B. Streams attributable to promotional campaigns
SELECT
    t.artist_id,
    t.track_id,
    c.campaign_id,
    count() AS plays_during_campaign
FROM  play_events      AS pe
JOIN  tracks           AS t  ON pe.track_id  = t.track_id
JOIN  ad_campaign      AS c  ON pe.event_time BETWEEN c.start_date AND c.end_date
WHERE c.campaign_type  = 'promotional'
GROUP BY
    t.artist_id, t.track_id, c.campaign_id;
```

3.5 Royalties and Finance (PostgreSQL)

```sql
-- 5A. Royalty calculation weighted by local rates
SELECT
    t.artist_id,
    t.track_id,
    pe.country,
    COUNT(*) * r.rate AS royalty_amount
FROM   play_events_monthly pe  -- ETL roll-up table
JOIN   tracks            t  ON pe.track_id = t.track_id
JOIN   royalty_rates     r  ON r.country   = pe.country
                            AND r.artist_id = t.artist_id
WHERE  pe.month = '2025-05'
GROUP  BY t.artist_id, t.track_id, pe.country, r.rate;

-- 5B. Cross-check: plays vs payments divergence >1 %
WITH plays AS (
    SELECT artist_id,
           COUNT(*) AS total_plays
    FROM   play_events_monthly
    WHERE  month = '2025-05'
    GROUP  BY artist_id
)
SELECT p.artist_id,
       SUM(ap.royalty_amount) AS total_paid,
       total_plays
FROM   artist_payouts ap
JOIN   plays          p USING (artist_id)
GROUP  BY p.artist_id, total_plays
HAVING ABS(SUM(ap.royalty_amount) / NULLIF(total_plays,0) - expected_rate) > 0.01;
```

Technology Mapping

Use-Case	Query Engine	Rationale
Operational joins, financial exactness	PostgreSQL	ACID semantics and referential integrity.
Event-stream ad-hoc, cohort, fraud analysis	ClickHouse	100 000 RPS ingestion and sub-second scans.
Search / autocomplete	OpenSearch	Token scoring and fuzzy match.
Event ingestion	Kafka	Decouples writes and enables real-time pipelines.

4 Improvements Over Workshop 1

This section refines all sections based on the feedback received after Workshop 1.

4.1 Business Model Canvas

Melody UD is an end-to-end platform that empowers independent musicians and spoken-word creators to publish, *monetize*, and analyze their audio content while giving listeners a friction-free, ad-supported or subscription experience. The canvas below summarizes the operating context.[1]

Key Partners	Key Activities	Value Proposition
midrule Labels, distributors	Ingest audio and artwork	One-stop publishing hub with global reach for independent artists
Cloud providers	Run scalable micro-services	Freemium listening with personalization and offline playback
Advertisers, agencies	Sell campaigns	Analytics dashboard with near real-time insights
Music-rights societies	Set royalty rules	Transparent royalty calculation and timely payouts

4.2 Requirements Documentation

4.2.1 Functional Requirements

Domain	ID	Requirement

[1] All references to *Spotify* in Workshop 1 were updated to *Melody UD*.

midrule Account & Identity	F01	Provide registration and authentication via e-mail + password, social OAuth, and enterprise SSO; store only hashed credentials and MFA tokens.
	F02	CRUD profile data (display name, avatar, language, payment country, parental controls).
	F03	Manage subscription life-cycle events (upgrade, downgrade, cancel, pause, reactivate) with prorated billing, grace periods, and an immutable event log.
midrule Content Catalog	F04	Ingest audio assets and validate metadata before persisting to catalog DB and object storage.
	F05	Extract audio feature vectors (tempo, key, loudness, mood) via an ML micro-service and persist them to a feature store.
midrule Discovery & Personalization	F06	Full-text search with autocomplete and typo tolerance, meeting NFR-P1 targets.
	F07	Recommendation engine returns ranked lists (Daily Mix, Discover Weekly, radio) with end-to-end latency $\leq$ 300 ms at p95.
	F08	Editors can create, version, and experiment with editorial playlists.
midrule Playback & Delivery	F09	Negotiate streaming sessions (codec, bit-rate, CDN edge) and return signed chunk URLs.
	F10	Dynamically adjust audio quality based on network telemetry.
midrule Social & Community	F11	Enable real-time, optimistic-locking edits on shared playlists and publish change events to an activity feed.
midrule Advertising	F12	CRUD campaigns, creatives, and targeting rules; log impressions to an append-only column store.
	F13	Ad-decision engine responds in $\leq$ 50 ms for 99 % of requests.
	F14	Finance service aggregates stream counts, applies contractual overrides, and calculates royalties.
midrule Creator Tools	F15	Creator dashboard shows play counts, geo heat-maps, and playlist adds with < 30 min lag.
	F16	Promotion workflow lets creators schedule and pay for promotions, persisting billing events to a ledger.
midrule Governance & Compliance	F17	Write immutable audit logs of admin actions, data exports, and rights changes.

F18	Fulfill data-subject requests—export or delete all personal information—within 30 days.

4.2.2 Non-Functional Requirements

ID	Category	Requirement	Priority
midrule NFR-P1	Performance	Search results return in $\leq$ 150 ms for 95 % of queries.	High
NFR-P2	Performance	Playback begins in $\leq$ 300 ms for 95 % of sessions.	High
NFR-P3	Performance	Ad-decision engine responds in $\leq$ 50 ms for 99 % of requests.	High
NFR-S1	Scalability	Handle $\geq$ 1 billion MAUs and $\geq$ 20 million concurrent streams via horizontal sharding and auto-scaling.	High
NFR-A1	Availability	Playback services maintain 99.95 % monthly uptime ($<$ 22 min downtime).	High
NFR-C1	Integrity	Subscription & billing data are strongly consistent.	High
NFR-C2	Integrity	Social interactions may be eventually consistent within 5 s.	Medium
NFR-SEC1	Security	Enforce TLS 1.3 in flight, AES-256 at rest, and tokenize payment data.	High
NFR-OBS1	Observability	Capture end-to-end traces for every playback session.	Medium
NFR-MNT1	Maintainability	Public interfaces documented in OpenAPI; enable zero-downtime blue/green deployments.	Medium

4.3 Architecture and Data Strategy

4.3.1 Demand Characterization

- **Catalog size:** 120 million tracks, 500 TB raw media, 5 TB new uploads per day.
- **Peak usage:** 20 million concurrent streams; average session 40 min.
- **Global footprint:** four latency zones—Americas, EMEA, APAC, LATAM.

4.3.2 Design Goals

1. Ultra-low-latency playback with strong consistency for billing.
2. Linearly scalable ingest and event streaming for analytics and ML.
3. Cost-efficient archival for long-tail media.

4.3.3 Design Overview (Initial Database Architecture)

Designing a large-scale music-streaming database requires balancing ultra-low latency, GDPR compliance, global scalability, and financial integrity. Melody UD distributes data into specialized layers—*only* open-source technologies—to ensure transparency and long-term sustainability.

- **Partitioned PostgreSQL 16**: single source of truth for catalog, subscriptions, payments and social graph; Citus or native sharding handles tenant fan-out.
- **Cassandra:** write-heavy session telemetry with tunable consistency.
- **MinIO:** cost-optimized object storage with lifecycle rules.
- **Kafka:** unified event ingestion; ClickHouse materialised views read Kafka topics directly for analytics.
- **OpenSearch:** full-text discovery; daily sync from PostgreSQL via Kafka Connect.

4.3.4 Operating Model and Systemic Context

Melody UD adopts a *freemium* model: free users are monetized through advertising, while Premium users pay recurring subscriptions (70 % of revenue flows to royalties).

- End users (mobile / web clients)
- Backend micro-services and distributed storage
- Content creators
- Advertising subsystem
- Analytics and reporting platform
- Contractual royalty engine

4.3.5 Information Flow: Inputs, Outputs, Interactions

Inputs

Multimedia files (audio, artwork, metadata); user events (play, like, follow); financial data.

Outputs

Optimized playback; personalized recommendations; analytical reports; real-time targeted ads.

Interactions

Client → API Gateway → Micro-services → Distributed stores; lateral calls to Advertising and Recommendations.

4.3.6 Critical Bottlenecks and Stress Points

Technical Bottlenecks

Subsystem	Problem	Risk	Mitigation
midrule Social graph	p99 latency > 200 ms	Poor UX	Partition graph; add RedisGraph cache.
Advertising	Cache saturation	Ad-decision > 50 ms	Hierarchical LRU cache; pre-computed segments.

Streaming	CDN/DRM negotiation	Start-time > 300 ms	Edge nodes with pre-signed URLs; regional hints.
Royalties	Massive event joins	Legal risk	Incremental aggregation; ClickHouse MVs.
GDPR	Data proliferation	Fines	Unified subject ID and automated exports.

bottomrule

System-Level Stress Points

Potential Crisis	Cause / Impact
midrule Explosive concurrent-user growth	Recommendation, playback, or auto-scaling failures.
Advertising-decision delays	Revenue loss and degraded free-tier UX.
Inaccurate royalty calculations	Contractual breaches and reputational damage.
Content-ingestion saturation	Validation, ML, or storage bottlenecks.
Social-graph inconsistency bottomrule	Likes/follows visible after > 5 s.

4.3.7 Solution Roadmap

1. Strengthen distributed caching for personalization and ads.
2. Apply edge computing to start playback near the user.
3. Decouple royalty calculations into asynchronous stages.
4. Deploy full OpenTelemetry tracing.
5. Optimize graph partitioning and ad-cluster sizing algorithms.

4.4 Entity–Relationship Model: Method and Diagram

4.4.1 Step-wise Method

Step 1 — Define Components Melody UD comprises client apps, service APIs, data lanes, and analytics stores (Table 9).

Table 9: Components per Layer

Layer	Building Blocks
midrule Edge (Client)	Mobile / Web apps, Smart-speaker SDK, Edge CDN
Gateway	API Gateway (rate-limit, authentication)
Core Services	Playback, User, Subscription & Billing, Search, Recommendation, Ad Platform

Messaging	Kafka streaming bus
Operational Data	PostgreSQL 16 clusters (partitioned, Citus-ready)
Telemetry	Cassandra (time-series)
Object Storage	MinIO (audio & artwork)
Real-Time Analytics	ClickHouse
Warehouse / BI bottomrule	Iceberg tables queried by Trino

Steps 2–5 — Entities, Attributes, Relationships

Step 2 — Define Entities

User, **Plan**, **Subscription**, **Invoice**, **Payment_Method**, **Payment_Transaction**, **Playlist**, **Playlist_Track**, **Play_Event**, **Session_Log**, **Follow**, **Artist**, **Album**, **Track**, **Advertiser**, **Ad_Campaign**, **Ad_Impression**, **Royalty_Rate**, **Artist_Payout**

Step 3 — Define Attributes per Entity

User

`user_id, email, password_hash, display_name, created_at, account_type, country`

Subscription

`subscription_id, user_id, plan_id, status, start_date, end_date, payment_method_tok`

Plan

`plan_id, name, price_cents, currency, concurrent_streams_limit, tier`

Invoice

`invoice_id, subscription_id, billing_period_start, billing_period_end, amount_cents`
`status`

Payment_Method

`payment_method_id, user_id, brand, last4, expiry_month, expiry_year, token`

Playlist

`playlist_id, user_id, name, description, is_public, created_at, updated_at`

Playlist_Track

`playlist_id, track_id, position, added_at`

Artist

`artist_id, name, country, biography`

Album

`album_id, artist_id, title, release_date, cover_art_url`

Track

`track_id, album_id, title, duration_sec, bpm, musical_key, loudness, audio_url`

Play_Event

`play_event_id, user_id, track_id, timestamp, device, location_id`

Session_Log

`session_id, user_id, ip_country, device_id, started_at, ended_at`

Payment_Transaction
`payment_id, user_id, amount, currency, provider, status, created_at`
Follow
`follower_user_id, followed_id, followed_type, created_at`
Advertiser
`advertiser_id, name, contact_email, billing_country`
Ad_Campaign
`campaign_id, advertiser_id, name, start_date, end_date, budget, targeting_json`
Ad_Impression
`impression_id, campaign_id, play_event_id, timestamp`
Royalty_Rate
`artist_id, country, rate`
Artist_Payout
`payout_id, artist_id, month, amount_cents, processed_at`

Steps 4–5 — Relationships and Cardinalities

- User → Subscription (**1 : N**)
- Subscription → Invoice (**1 : N**)
- User → Payment_Method (**1 : N**)
- Artist → Album → Track (**1 : N**)
- Playlist ↔ Track via Playlist_Track (**N : N**)
- User ↔ Track via Play_Event (**N : N**)
- User ↔ Artist via Follow (**N : N**)
- Advertiser → Ad_Campaign → Ad_Impression → Play_Event (**1 : N**)

Step 6 — First ER Draw See Figure 2.

Steps 7–10 — Normalization, Keys and Constraints

Step 7: **Resolve N:M bridges** — playlist_track and user_follow tables already satisfy 3NF by carrying only foreign keys and the bridging payload (`position`, `created_at`).

Step 8: **Introduce surrogate keys** — every base entity gains an immutable UUID PK; natural keys (e.g. `email`, `campaign_id`) receive `UNIQUE` constraints.

Step 9: **Third-normal-form check** — all non-key columns depend solely on the whole key; partial or transitive dependencies were eliminated by moving monetary fields to `invoice` and `artist_payout`.

Step 10: **Define integrity rules** — `CHECK (amount_cents > 0)` on invoices and payouts, `CHECK (expiry_year >= extract(year from current_date))` on payment methods, cascading deletes only on test data; production deletes are soft (boolean `is_deleted`).

(Figure 3).

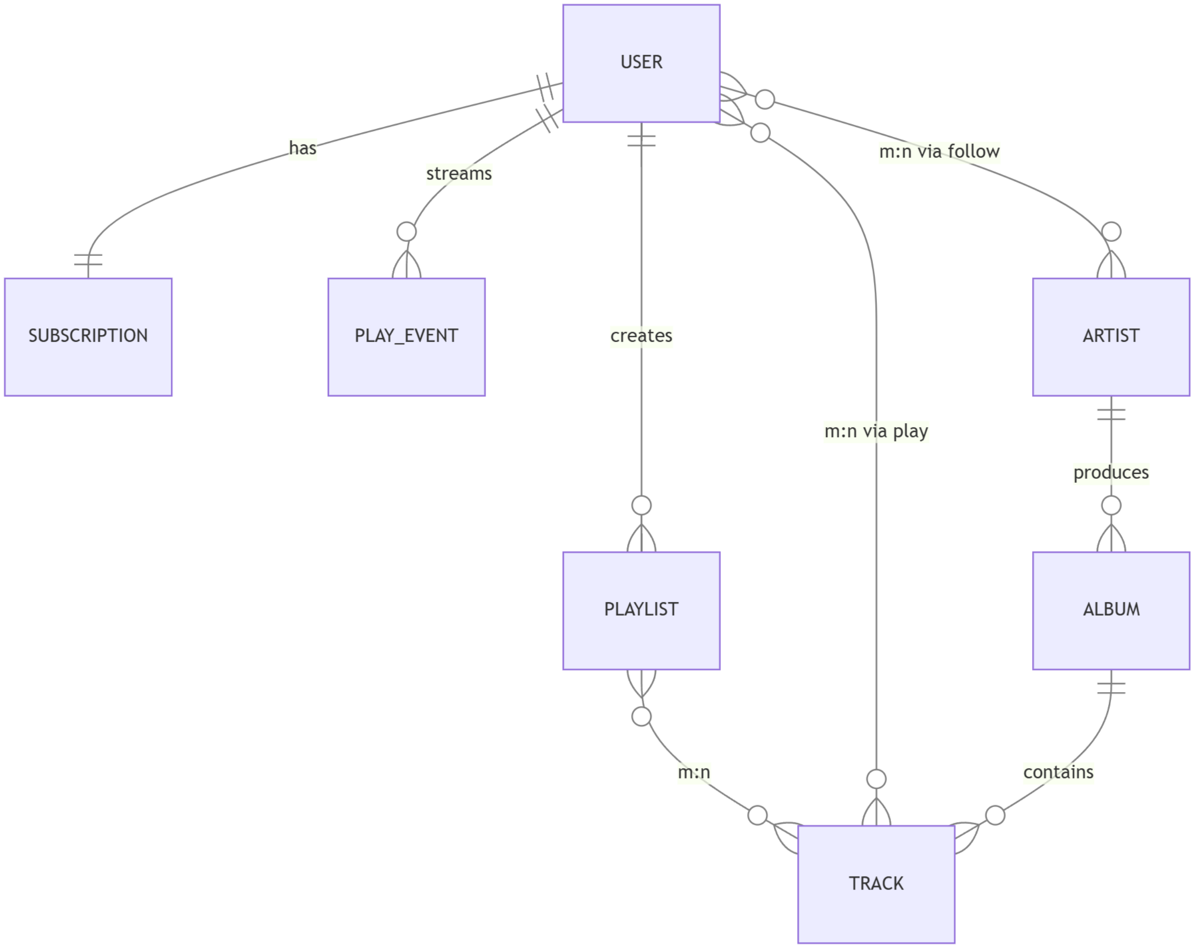

Figure 2: First Entity–Relationship Model.

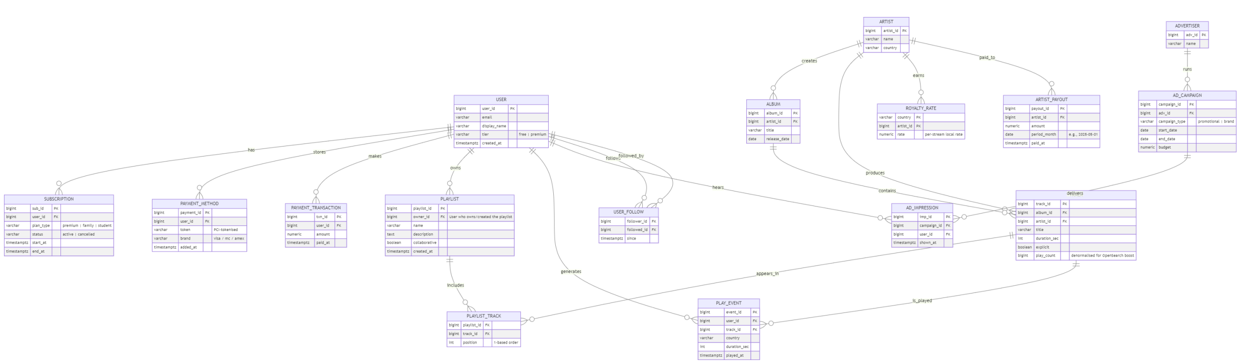

Figure 3: Melody UD Entity–Relationship Model (refined).

5 References

References

[1] Corporate Finance Institute. *Business Model Canvas Examples*. Available at: `https://corporatefinanceinstitute.com/resources/management/business-model-canvas-examples/`

[2] Business Model Analyst. *Spotify Business Model*. Available at: `https://businessmodelanalyst.com/spotify-business-model/`

[3] Music Business Research. (19 August 2024). *The Music Streaming Economy Part 10: Spotify's Business Model*. Available at: `https://musicbusinessresearch.wordpress.com/2024/08/19/the-music-streaming-economy-part-10-spotifys-business-model/`

[4] Investopedia. *How Spotify Makes Money*. Available at: `https://www.investopedia.com/articles/investing/120314/spotify-makes-internet-music-make-money.asp`

[5] IIDE. *Business Model of Spotify*. Available at: `https://iide.co/case-studies/business-model-of-spotify/`

[6] GrowthX Club. *Spotify Business Model*. Available at: `https://growthx.club/blog/spotify-business-model#spotify-revenue-model`

[7] Software Developer Diaries. *How Spotify's Playback Works Under the Hood*. YouTube video. Available at: `https://www.youtube.com/watch?v=K26bGXVR-mE`

[8] Lopez, M. A. *Deep Dive into the Tech Stack Used by Spotify*. LinkedIn article. Available at: `https://www.linkedin.com/pulse/deep-dive-tech-stack-used-spotify-marny-a-lopez`

[9] Intuji. *How Does Spotify Work? – Tech Stack Explored*. Available at: `https://intuji.com/how-does-spotify-work-tech-stack-explored/`