# PEAK FITNESS - TECHNICAL DESIGN DOCUMENT

**Authors:** Kris Liu

**Stakeholders:** Peak Fitness Marketing & Retention Team, BeFit Inc.

**Approvals:** Minjia

##### Overview:

Peak Fitness, a boutique fitness studio recently acquired by BeFit Inc., currently relies on the third-party platform Mindbody to manage class bookings and customer data. However, the business faces growing limitations: customer engagement data is fragmented, analytics are manual and delayed, and marketing teams lack real-time access to actionable insights.

This project proposes a custom, cloud-based data platform that centralizes and transforms Peak’s raw operational data into a scalable analytics infrastructure. By building an end-to-end data pipeline using AWS services—including S3, Glue, Lambda, and Athena—and surfacing insights via a dashboard and leaderboard API, the platform aims to support customer retention, optimize scheduling, and enable data-driven decisions across marketing and operations. The ultimate goal is to modernize the company’s data foundation in a way that is cost-effective, flexible, and secure.

##### Glossary of Terms:

| Term/Acronym | Definition |
| --- | --- |
| ETL | Extract, Transform, Load – the process of retrieving data from a source, cleaning and reshaping it, and loading it into a database or data warehouse. |
| AWS S3 | Amazon Simple Storage Service – cloud-based object storage used for storing raw and processed files. |
| AWS Glue | A serverless ETL service that automates data preparation using Spark or Python. |
| AWS Lambda | A serverless compute service used to trigger lightweight ETL tasks or API calls. |
| Athena | A serverless, pay-per-query SQL engine that queries data directly from S3 using standard SQL. |
| Redis | Amazon in-memory data store used for caching and fast lookups. Optional in the future to accelerate leaderboard queries. |
| Databrick | A collaborative data analytics and machine learning platform that supports notebooks and Delta Lake tables. |
| Streamlit | A Python-based tool to build interactive web apps and dashboards for data visualization. |
| PII | Personally Identifiable Information – any data that could potentially identify a specific individual (e.g., name, email). |
| MVP | Minimum Viable Product – the simplest version of a product that is functional and usable. |

##### Current State:

Peak Fitness currently uses the Mindbody platform for class scheduling and customer bookings. Data is accessed via third-party APIs or manual CSV exports, which staff process manually in spreadsheets for reporting.

There is no centralized data warehouse or structured analytics pipeline. Marketing and retention teams lack access to live behavioral data, and there is no integration with systems like CRMs. As a result, campaigns rely on static lists and lagging indicators, limiting personalization and responsiveness.

The proposed AWS-based platform will enable transformed engagement data to flow into downstream tools (e.g., HubSpot) for lifecycle marketing, CRM syncs, and data-driven engagement. This capability is scoped for future phases and built with integration-readiness in mind.

##### Goals:

###### The primary goals:

* Migrate from Mindbody to a custom data platform using AWS services.
* Build a batch-oriented ETL pipeline to process class signups, transactions, and engagement data using Python and boto3.
* Create an accessible, queryable database using Athena.
* Enable downstream products such as an internal leaderboard dashboard (MVP) and future CRM campaign triggers.
* Ensure data is reliable, secure, and analytics-ready.

###### The Secondary goals:

* Provide a sandbox Athena environment for internal data exploration and education.
* Minimize costs by using serverless and free-tier AWS services.
* Lay groundwork for future predictive analytics.
* Allow for exploratory analytics and campaign modeling using Databricks notebooks

###### Non-functional requirements include:

* **Scalability**: Handle future data volume increases with minimal rework.
* **Availability**: Ensure data pipeline reliability for daily batch updates.
* **Security**: Protect user data through IAM, encryption, and access control.
* **Performance**: Support near-real-time dashboard updates with low-latency queries.

##### Non-goals:

This project will not include:

* Replace the full booking and scheduling functionality of Mindbody.
* Ingest or process data in real time (batch processing only for MVP).
* Develop advanced machine learning models in the initial phase.
* Build a customer-facing front-end or mobile application.
* Implement full CRM automation across departments (limited to initial CRM sync logic).

##### 

##### **Impact / Measures of Success:**

###### **System Performance & Stability:**

* ETL pipeline runs with a 95% or higher daily success rate.
* Data from S3 is processed and query-ready in Athena within 30 minutes of arrival.  
  At least 95% of records pass schema validation and transformation logic.
* Schema or table updates cause less than 1 hour of downtime.
* Athena tables are partitioned to reduce query cost and improve performance by at least 50%.
* Glue job logs include row counts, failure rates, and time metrics for every transformation step.

1. **Platform Enablement for Analytics and Campaigns:**

* 100% of instructor, class, and location data integrated into fact/dim tables for analytics  
  Aggregated popularity tables (e.g., top 10 instructors by attendance/location/week) available for the marketing team.
* Transformed data includes key fields required for future marketing segmentation (e.g., attendance frequency, class type, instructor popularity).
* 1 curated dataset enables campaign segmentation logic (e.g., members with <2 visits/month & favorite instructors).
* Data design supports potential retention analysis by surfacing metrics like no-show rates, cancellation trends, and top classes per location.
* Pipeline is built to allow data scientists or marketing analysts to query engagement trends, segment user cohorts, and launch lifecycle campaigns in future phases.
* Partitioned Athena tables improve query cost-efficiency by 50%+ vs. raw scan.

1. **Data Integration & Future Use Cases:**

* Leaderboard dashboard (via Streamlit + Athena) delivers sub-2s query response time in testing scenarios.
* One “event-style” table (e.g., class\_sessions) included in the schema to enable future streaming.
* ETL code supports extension by design. One future example is class\_feedback, which can be added using the same modular Glue/Lambda pattern.
* CloudWatch alerts trigger on all ETL failures.
* Structured datasets with user behavioral signals like check-in frequency, class preference, and recency. These fields can be exported or integrated with CRM tools like HubSpot for targeting churn risk or active campaign segments.

1. **Partitioned Athena Table Design:**

Athena tables are partitioned by meaningful dimensions such as class\_date, location\_id, or class\_type. This structure reduces full-table scans, resulting in at least 50% improved query performance and significantly lower costs. It also enables downstream users to quickly filter and query subsets of data based on time, geography, or business unit—critical for marketing segmentation and campaign targeting.

1. **ETL Logging with CloudWatch:**

All batch ETL jobs (Glue or Lambda) are integrated with AWS CloudWatch for runtime logging and monitoring. Custom logs include key transformation metrics such as total records processed, error counts, job duration, and data source identifiers.

Logs are retained with a limited retention window of about 7 - 14 days and delete old logs automatically to control cost, and metric filters are applied to trigger alerts for critical job failures. This enables full traceability of data transformations and improves system observability without excessive overhead. Only critical ETL jobs (e.g., class\_attendance repair) are logged. Metric filters trigger alerts for job failures, balancing observability and cost.

1. **Educational / Student Extensibility Impact:**

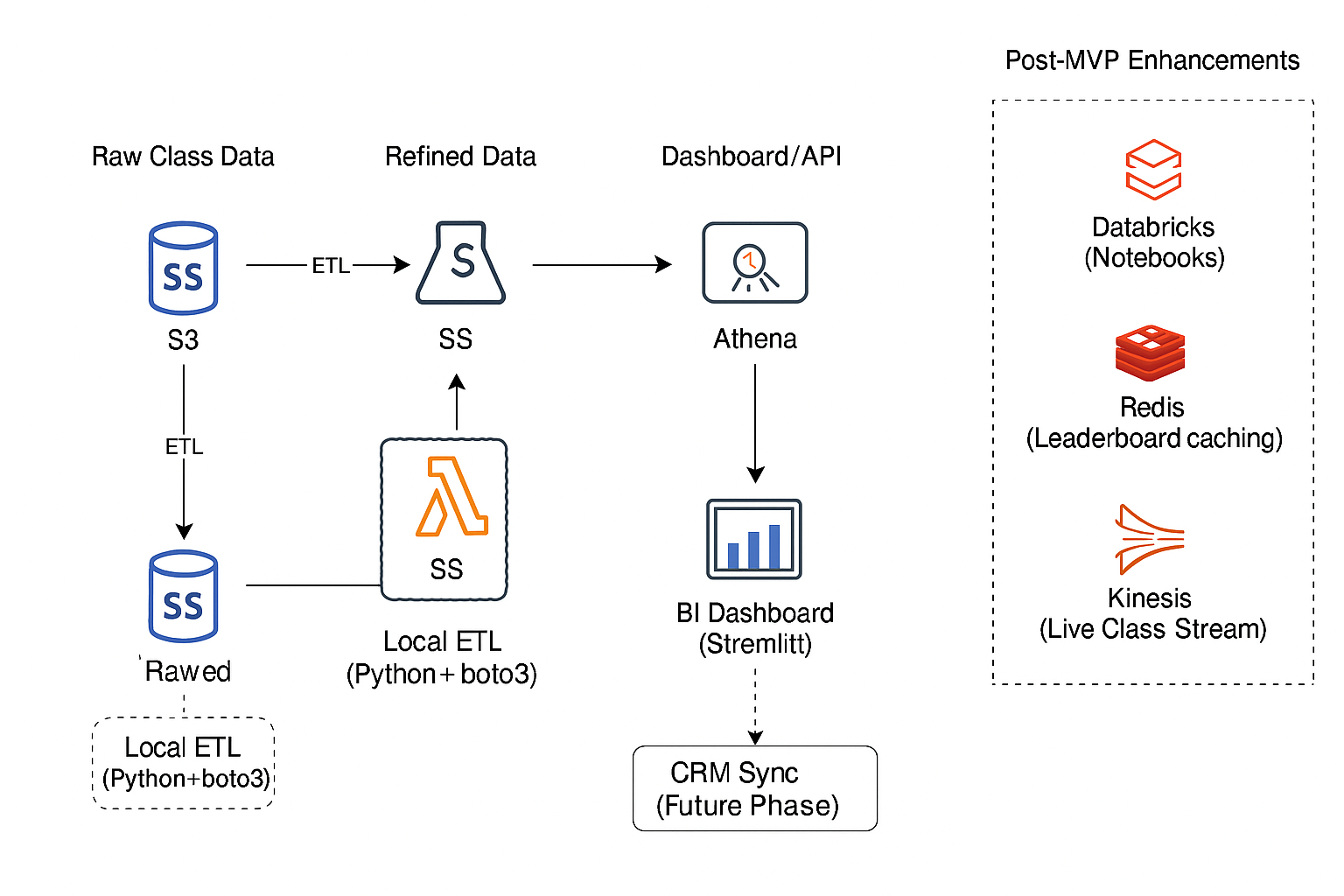
This project is designed with future students and junior data engineers in mind. Key extensibility features include:

* All ETL code is modular and commented, with one complete example pipeline implemented from raw to Athena (e.g., dim\_users, class\_attendance)
* GitHub repository includes a “How to Extend This Project” section for future students.
* All core tables are included in a data dictionary and an ERD diagram.
* A full ERD and data dictionary are provided to help understand table relationships and schemas.
* “How to Extend” section included in README for adding new data sources or models.
* One sample extension (e.g., class\_feedback or user\_engagement\_tags) implemented.
* Sample Python + Boto3 scripts are included to demonstrate how tables can be manually created in S3 and registered in Athena — useful for local testing before scaling to Glue
* Both AWS Glue and Lambda jobs are structured for reuse with minimal code duplication
* A placeholder extension (e.g., class\_feedback or user\_engagement\_tags) is scoped for future implementation to guide new developers in adding tables

##### 2. Technical Diagram

###### 2.1 System Architecture

The following diagram and steps explain the data flow from raw ingestion to actionable insights:



###### 2.2 Technology Choices and Justification:

This system is designed using AWS-native, serverless tools to optimize for cost-efficiency, scalability, and ease of integration. The following tools were selected based on the nature of the data, pipeline requirements, and operational constraints:

* **AWS S3:** Acts as the core storage layer for both raw and processed data. Chosen for its scalability, low cost, and compatibility with Athena and Glue.
* **AWS Glue**: Handles heavy ETL transformations and batch orchestration. Supports Spark-based processing and integrates with the Data Catalog.  
  Note: MVP runs use up to 10% of historical data with limited frequency to stay within the AWS Free Tier.
* **AWS Lambda:** Handles lightweight ETL orchestration, S3 filtering, and automatic partition registration (e.g., MSCK REPAIR TABLE in Athena). Also used to trigger Glue jobs and log job metadata.
* **AWS Athena:** Enables serverless SQL querying from S3. Used for validating ETL results, powering dashboard queries, and supporting the leaderboard API.
* **AWS CloudWatch:** Logs and alerts are set up for all ETL jobs (Glue and Lambda). Tracks processed record counts, execution durations, and failure metrics.
* **Streamlit:** Provides a lightweight frontend for data visualization. The leaderboard dashboard uses Athena queries and will include future API-authenticated access.
* **Databricks (Post MVP):** Used for ETL (replacing Glue) and ML model training. Delta Lake optimizes Athena query performance by 50% via partitioning.
* **Kinesis (Post-MVP):** Used for simulating or enabling live streaming data (e.g., class check-ins or fill-up tracking).
* **Redis (Post-MVP):** Future caching layer to optimize leaderboard performance and reduce repeated Athena query costs.

###### 2.3 Trade-offs Considered:

The system is optimized for availability and partition tolerance (AP) under the CAP theorem, aligning with the nature of distributed cloud environments. Batch ETL and eventual consistency in S3 and Athena are acceptable for this use case, as long as data is reliably processed daily. Consistency is maintained via schema validation and data partitioning logic during ETL

###### 2.4 Data Flow:

1. **Raw Data Ingestion:**

Raw files (e.g., class\_attendance.json) are uploaded to the peak-fitness-data-raw S3 bucket using AWS CLI with requester-pays enabled. Files include historical user activity, class schedules, and registration data.

1. **Glue Transformations ETL:**

AWS Glue converts raw JSON and CSV files into partitioned, columnar Parquet format for efficient querying. Each table (e.g., dim\_users, dim\_classes, class\_attendance) is cleaned, cast to schema, and written to the processed S3 bucket (peak-fitness-kris-processed/).

1. **Lambda ETL Orchestration:**

A Lambda function automates MSCK REPAIR TABLE to sync Athena metadata with new partitions (e.g., class\_datetime\_parsed=2023-01-02/). Without this step, Athena cannot detect newly uploaded data.

The Lambda also:

* + Can be triggered manually or on a schedule
  + Logs execution metadata (e.g., timestamp, queryExecutionId)
  + Ensures dashboards remain up-to-date with the latest S3 data

1. **Partitioned Storage:**
   * Glue writes Parquet output into S3 using folder-based partitioning (e.g., class\_datetime\_parsed=YYYY-MM-DD/). This structure enables Athena to scan only the relevant partitions, significantly reducing query cost and improving performance.
2. **Athena Query Layer:**
   * Athena exposes all processed datasets (dimensions and fact tables) for SQL querying, validation and dashboard metrics. The partitioned structure supports filtered queries by date, location, instructor, or class type — powering both manual analysis and the leaderboard dashboard.
3. **Streamlit Leaderboard Dashboard (MVP):**

Streamlit connects and queries to Athena using PyAthena to visualize results include engagement metrics and attendance trends. The dashboard is secured, private, and reads only processed datasets.

1. **Logs and Monitoring:**

CloudWatch captures runtime metrics from Glue and Lambda jobs, including row counts, failure reasons, and job durations. Logs are retained for 7–14 days and monitored via metric filters for critical failures.

1. **Optional Enhancements (Post-MVP):**
   * Databricks: Connected to Athena or S3 for Exploratory modeling (e.g., churn, retention) on Athena-linked datasets.
   * Kinesis: Real-time stream for class check-ins and fill-up rate.
   * Redis: Caching layer for faster leaderboard performance.
   * SageMaker: Predictive models (e.g., no-show or churn prediction) using engagement features.

##### 3. Data Management & Database Design

The data model for Peak Fitness is designed to support core operational needs such as attendance tracking, user engagement monitoring, class metadata management, and future analytics use cases. The schema follows a normalized structure using fact and dimension tables to improve scalability, reduce redundancy, and support fast, reliable querying.

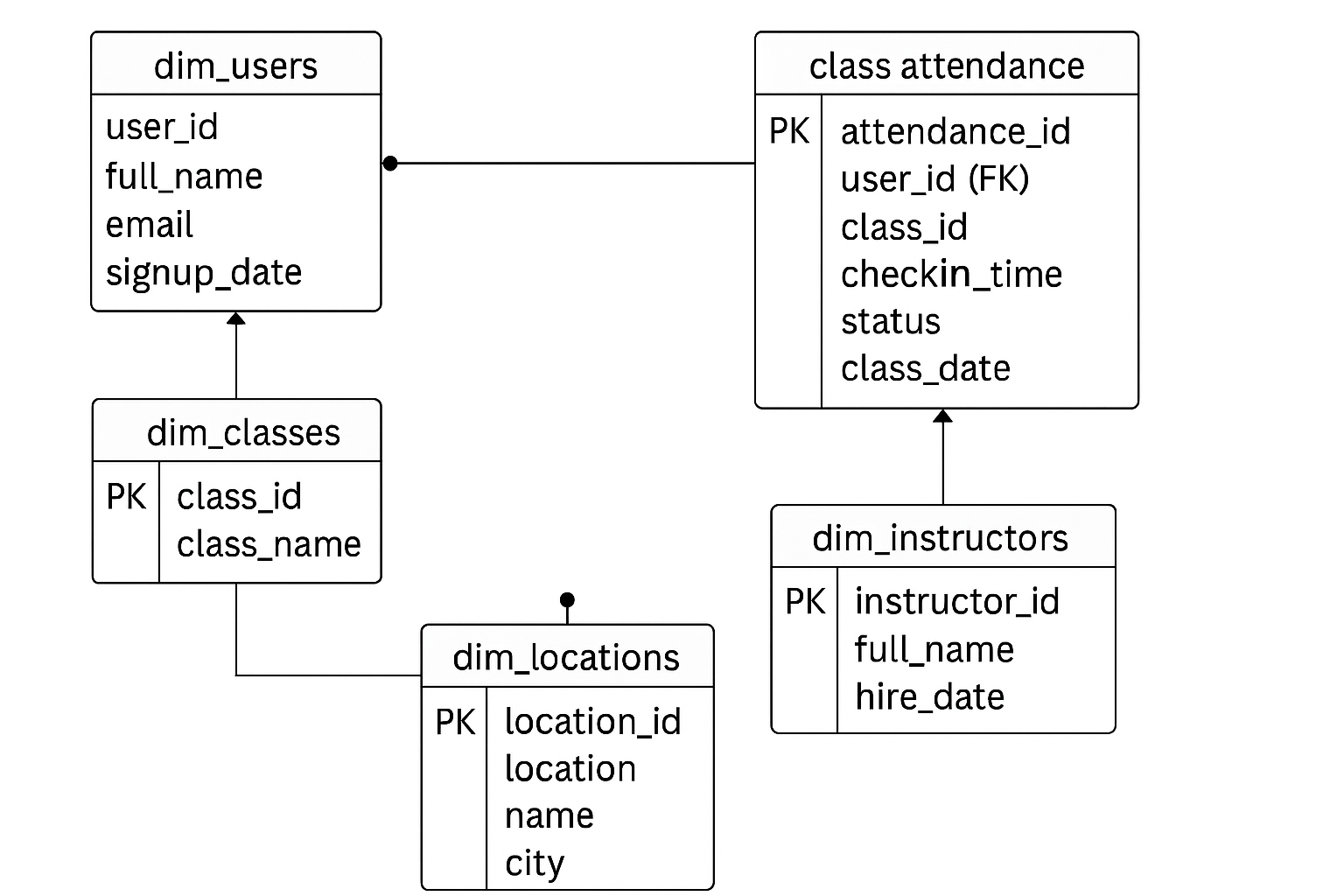
This schema consists of five core tables that form the foundation of the data platform:

* dim\_users
* dim\_classes
* dim\_locations
* dim\_instructors
* class\_attendance

These five tables cover the critical dimensions of the business:

* Users — who is engaging with the service
* Class reference data — what classes exist and when they are scheduled and the popularity
* Time & Location — where and when events happen and the correlations to attendance
* Engagement Events — who attended what, forming the behavioral foundation of retention, marketing, and personalization

All other future tables (e.g., fact\_transactions, class\_feedback, CRM sync outputs) can be layered on top of this foundation without reworking the model.



###### 3.1 Schema Definitions and Data Dictionary:

1. Table of Dim\_users: Every engagement, attendance, or transaction links to a user. Core to all joins.

Source from: mindbody\_user\_snapshot.csv

Primary Key: user\_id

| Field Name | Data Type | Description |
| --- | --- | --- |
| user\_id | string | Unique user ID (primary key) |
| full\_name | string | User’s full name |
| email | string | User’s email address |
| city | string | City of residence |
| signup\_date | date | Date user registered |
| birth\_year | int | Used to calculate age |
| gender | string | Optional, if available |

1. Table of dim\_classes: Defines what a “class” is — title, type, scheduled time. Required for attendance context.

Source from: schedule\_sample.csv

Primary Key: class\_id

Foreign Keys: instructor\_id, location\_id

| Field Name | Data Type | Description |
| --- | --- | --- |
| class\_id | string | Unique class ID (primary key) |
| class\_name | string | Name of the class |
| class\_type | string | Category (e.g., yoga, HIIT) |
| instructor\_id | string | Foreign Key to dim\_instructors |
| location\_id | string | Foreign Key to dim\_locations |
| start\_time | timestamp | Scheduled start time |
| end\_time | timestamp | Scheduled end time |

1. Table of dim\_locations: Every class happens somewhere. Needed for location-based filters, performance, campaigns.

Source from: schedule\_sample.csv

Primary Key: location\_id

| Field Name | Data Type | Description |
| --- | --- | --- |
| location\_id | string | Unique location ID (primary key) |
| location\_name | string | Gym/studio name |
| city | string | City |
| state | string | State or region |

1. Table of dim\_instructors: Used for analyzing instructor performance and popularity. Required for leaderboard features and class context.

Source from: schedule\_sample.csv, class\_attendance.json

Primary Key: instructor\_id

| Field Name | Data Type | Description |
| --- | --- | --- |
| instructor\_id | string | Unique instructor ID (primary) |
| full\_name | string | Instructor’s name |
| email | string | Email, if available |
| hire\_date | date | When they joined |

1. Table of class\_attendance: This is your central fact table. Tracks who attended what, when, and how. Powers leaderboard and retention logic.

Source from: class\_attendance.json

Primary Key: instructor\_id

Foreign Keys: user\_id, class\_id

Partition Column: class\_date (used in Athena/S3 for performance)

| Field Name | Data Type | Description |
| --- | --- | --- |
| attendance\_id | string | Unique record ID |
| user\_id | string | FK to dim\_users |
| class\_id | string | FK to dim\_classes |
| checkin\_time | timestamp | When the user attended |
| status | string | Attended, no-show, canceled |
| class\_date | date | For partitioning and filtering |

##### 4. Interface / API Definitions

**Leaderboard API (Planned for MVP)**  
This project will expose a read-only API endpoint to serve leaderboard data. It will:

* Query Athena for aggregated attendance metrics
* Support filters like week, class type, or location
* Be implemented using a lightweight Streamlit or Flask API wrapper
* Return JSON-formatted top users/instructors/classes
* Be secured with IAM-based access (no public exposure)

##### 5. Productionisation Considerations

###### 5.1 Availability and Performance Targets:

**Availability Tier - Tier 2:**

ETL processes (Glue + Lambda) run in scheduled batches daily or on-demand triggers. Athena is available for analysts and dashboards performance target <2s per query, dashboard API <500ms during business hours.

**Performance Targets:**

* **Athena queries:** Return within 2 seconds on average, optimized by Parquet format and date-based partitioning (class\_datetime\_parsed).
* **API endpoints (Streamlit dashboard):** Expected latency <500ms per request under typical load, backed by cached Athena results and lightweight query volume.

###### **5.2 Monitoring and Logging:**

**Data quality validations include:**

All ETL operations are monitored through CloudWatch with data validation at multiple stages:

* CloudWatch logs record ETL job activity, including record counts, execution time, and failure traces.
* Metric filters are configured to detect failed runs or significant drops in processed record counts.
* Athena partition sync (via Lambda) is validated weekly to ensure queryable table consistency.
* Schema enforcement checks include:
  + Type validation for key fields
  + Required field presence
  + Null value filtering
* Record counts are compared between raw and processed S3 layers to ensure no unintentional loss.

All validation failures or anomalies are logged in CloudWatch and flagged for review. This provides early warning for pipeline or schema drift issues.

###### **5.3 Phased Rollout Plan:**

**Initial Deployment (Subset Load):**

To mitigate cost and reduce risk during initial rollout, only a limited subset (~10%) of historical data is ingested and processed.

This subset is filtered by date range (January 2023) and is representative across locations and class types.

All manual test data from the early development phase has been deleted. From this point forward, all ETL logic is executed using AWS Glue and AWS Lambda.

* AWS Glue handles all Parquet transformations and table creation.
* AWS Lambda automates partition repair (MSCK REPAIR TABLE)

Glue job execution is limited to MVP development only, to stay within AWS Free Tier usage. Full-scale ingestion will be evaluated after MVP completion.

**Phase 1 (MVP):**

* Core schema (users, classes, instructors, attendance).
* ETL pipeline using Glue or Lambda.
* Athena setup with partitioned Parquet tables.
* Leaderboard API (read-only).
* Partition repair automation via Lambda was implemented and validated with a successful test (Feb 1, 2023 partition).

**Phase 2 (Post-MVP/Future Enhancement):**

* Add fact\_transactions, class\_feedback, CRM sync logic.
* Enable real-time data support via Kinesis.
* Integrate ML models (SageMaker) for churn prediction.
* Add Redis caching layer for fast dashboard queries.
* Enable Delta Lake support with Databricks notebooks to explore lifecycle triggers and campaign logic.

###### 5.3.1 Post MVP Enhancement:

**Databricks for ML in Churn Prediction & Campaign Modeling:**

* Used for advanced analytics and ML prototyping.
* Reads from class\_attendance on S3 with requester-pays enabled.
* Can be used to train churn prediction models or identify "at-risk" users.
* Final outputs can be written back to Athena-compatible Parquet with partitions for campaign targeting.
* Implemented churn prediction model in Databricks (Logistic Regression).
* Features: User visit frequency, days since last class.

**CRM Sync Prep (Narrative-Only):**

Prepare the schema to support CRM in the future:

* dim\_users includes: user\_id, email, city, created\_at
* class\_attendance can be grouped by user\_id to calculate:
  + Last activity date
  + Favorite class
  + Attendance frequency

**Kinesis (Mock Stream):**

* Simulates live check-in events using put\_record() to an AWS Kinesis stream.
* Generates user\_id, class\_id, and timestamp as mock real-time data.
* Useful for future real-time dashboards or live engagement tracking.

**Athena + Delta Buffer:**

Use Athena to **cache aggregated results** from Databricks, enabling:

* Optimizing data pre-query (Delta Lake’s file organization → faster Athena scans).
* Handling complex transforms (Spark > Glue for advanced logic).
* Faster dashboards (e.g., leaderboard API).

**Redis Cache Layer for live class attendance tracking:**

* Demonstrates leaderboard caching by storing the top 10 most-attended classes into Redis.
* Uses sorted sets (zadd) and retrieves rankings in descending order (zrevrange).
* Leaderboard API: Cache and accelerate top 10 classes/users to serve 1,000+ requests/sec with ~1ms latency.
* Session Store: Track active user check-ins for dashboards.

###### **5.4 Accessibility:**

No public-facing UI is exposed. The project operates entirely through backend AWS services (Athena, Lambda, S3) and is accessed by internal analysts or engineers only.

The Streamlit-based Leaderboard Dashboard is developed as a private, authenticated web application. It will not expose any public endpoints and will be protected by IAM or Streamlit-host-level authentication mechanisms before release. No user input or modification is permitted from the frontend.

###### **5.5 Security & Privacy Implications:**

**Data Classification:**

* The dataset includes internal PII (e.g., user ID, email) and is governed under internal IAM access control policies. No names or emails are exposed in downstream outputs.

**IAM Controls, API Security & Access Management:**

* Scoped IAM roles are used for AWS Glue, Athena and Lambda.
* Roles restrict access to only the required S3 and Athena resources.
* Athena connection uses scoped IAM roles via Boto3 and PyAthena.
* “Requester Pays” enforcement has been implemented via job parameters and SDK calls:
  + Glue: --conf [spark.hadoop.fs](http://spark.hadoop.fs).s3.requester.pays.enabled=true
  + Lambda: RequestPayer="requester" on all get\_object and list\_objects calls
* No public APIs.
* No PII (email, name) will be exposed in dashboards or API responses — only anonymized IDs.

**ETL Pipeline Protections:**

* All raw PII is handled only in AWS Glue jobs.
* Processed data written to S3 excludes names and emails; only user\_id is retained.
* Schema validation and null filtering are applied during transformation.
* No dynamic SQL or user inputs are used at any point in the pipeline.

**Logging & Monitoring:**

* CloudWatch logs are internal-only.
* Logs do not contain raw PII; only anonymized references like user\_id are included.

###### **5.6 Educational/Student Extensibility Impact:**

**Local Development Option:**

In addition to the Glue + Lambda ETL pipeline, a local ETL alternative using Python + Boto3 was tested in early stages and retained for educational purposes. This approach allows students to simulate the pipeline locally (outside the AWS environment) using CSV/JSON files, then upload to S3 and define external tables in Athena manually. This method helps build hands-on understanding of AWS-compatible data processing and is included in the GitHub repo under a dedicated local\_etl/ folder.

##### 6. Rollback Plan:

* **Partial Deployment**: Only ~10% of the dataset will be processed during initial rollout. This reduces reprocessing scope in case of schema or ETL issues.
* **S3 Versioning**: All transformed Parquet outputs are written to a versioned processed/ S3 bucket with date-based folders. Rollback can be performed by deleting or restoring previous versions.
* **No Overwrites to Downstream**: No processed data will be auto-synced to CRM or dashboards during MVP. All downstream integrations will be opt-in and manually reviewed.
* **Component Isolation**: Leaderboard API will not fail if ETL fails; it reads from static Athena views. This avoids cascading outages across components.
* **Glue/Athena Table Isolation**: New schema changes or tables will be staged in development environments (e.g., dev\_ prefix) before promotion to production, though this was not required for the MVP due to controlled data scope and manual testing.
* **Error Logging and Alerts:** All errors are logged to CloudWatch with metric filters. ETL failures will not trigger further processing downstream.
* **repair\_class\_attendance\_lambda** (validated partition sync).
* **Fallback mock data (if S3 issues):**

mock\_data = [("user1", 5, 10), ("user2", 1, 45)] # user\_id, visits, days\_inactive

train\_df = spark.createDataFrame(mock\_data, ["user\_id", "total\_visits", "days\_since\_last\_visit"])

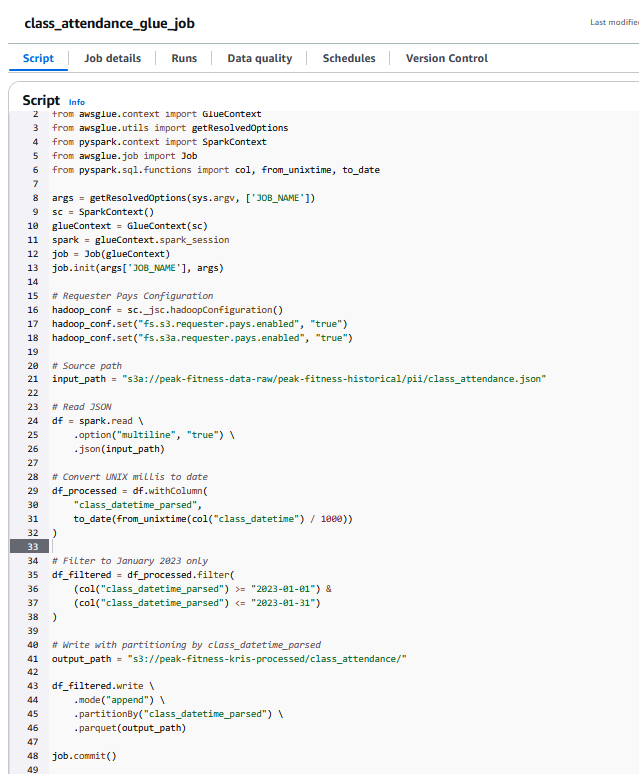
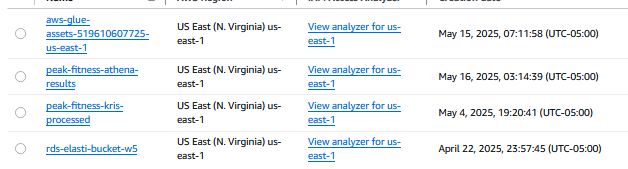
##### 7. Appendix：

**Alternative Approaches：**

* Redshift or Snowflake: Not selected due to cost and AWS Free Tier limits.
* Real-Time Streaming: Deferred to post-MVP phase due to added operational complexity and unclear immediate use case.
* Fully Prebuilt BI Tool (e.g., Tableau): Rejected to maintain a fully serverless, extensible platform for future developers.
* Note: For complete table structure and field-level definitions, refer to the combined Schema Definitions & Data Dictionary section above.
* Databricks was added as a lightweight notebook platform due to AWS community edition constraints and exploratory flexibility.

**Configuration Snapshot:**

Please refer to Github Repository for scripts.

* Glue setup example:
  + 
* S3 bucket:
* Lambda Function:
  + 

##### **8. Related Artifacts：**

* GitHub repository.
* ERD Diagram (Section 5).
* Athena queries and sample outputs.
* AWS CLI S3 access examples.
* Databricks notebook link (if published) for churn prediction logic.

##### **9. Sections Skipped：**

* CSRF / Web Input Security: No web forms or user input involved.
* DDoS Prevention: Not applicable: No public endpoints; IAM-protected APIs only.
* User-Visible Rollout or A/B Testing: Internal rollout only; no user-facing components.
* Accessibility Testing: No UI exposed, internal-only pipeline.