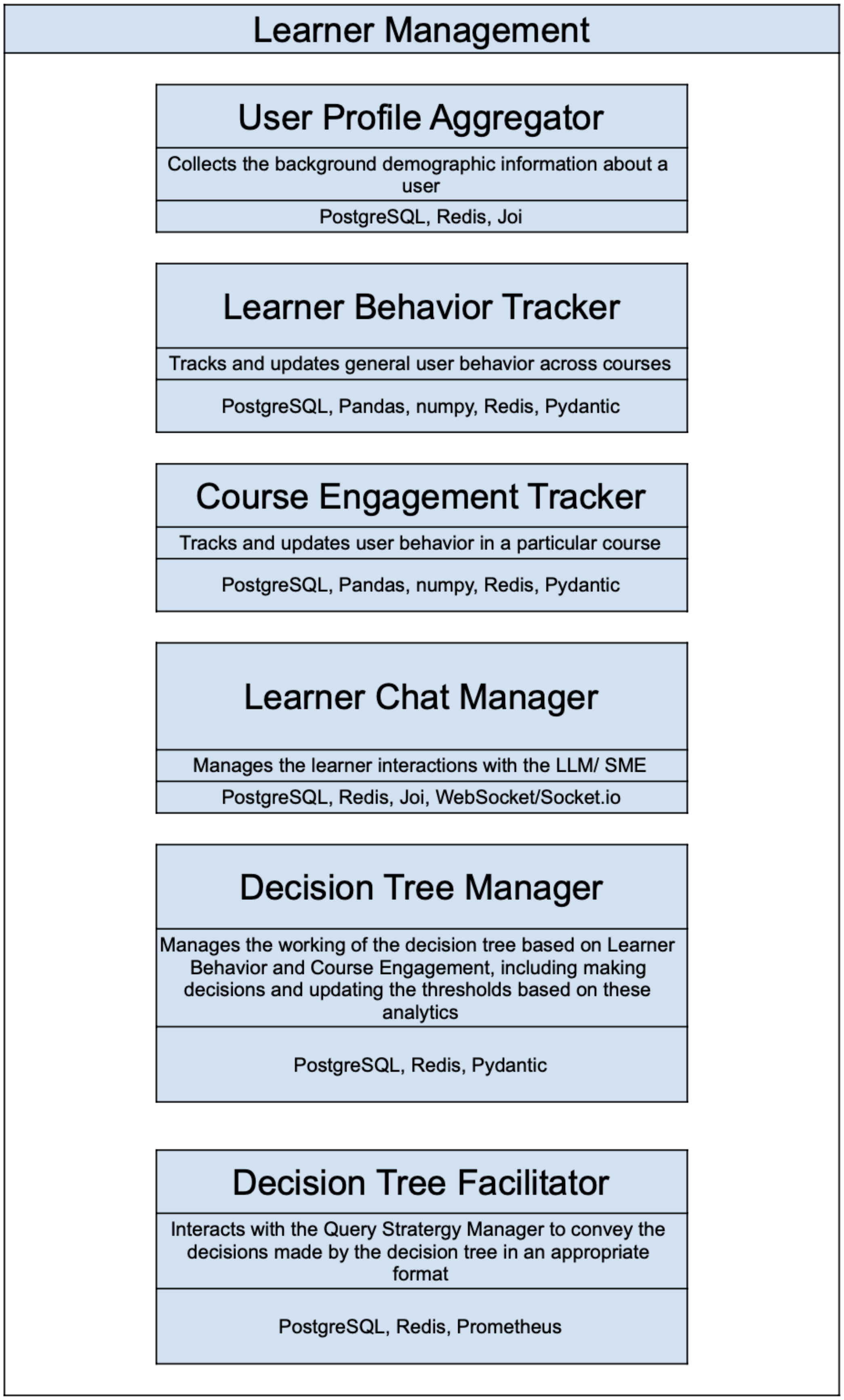

Learner Management

User Profile Aggregator

Collects the background demographic information about a user

PostgreSQL, Redis, Joi

Learner Behavior Tracker

Tracks and updates general user behavior across courses

PostgreSQL, Pandas, numpy, Redis, Pydantic

Course Engagement Tracker

Tracks and updates user behavior in a particular course

PostgreSQL, Pandas, numpy, Redis, Pydantic

Learner Chat Manager

Manages the learner interactions with the LLM/ SME

PostgreSQL, Redis, Joi, WebSocket/Socket.io

Decision Tree Manager

Manages the working of the decision tree based on Learner Behavior and Course Engagement, including making decisions and updating the thresholds based on these analytics

PostgreSQL, Redis, Pydantic

Decision Tree Facilitator

Interacts with the Query Stratergy Manager to convey the decisions made by the decision tree in an appropriate format

PostgreSQL, Redis, Prometheus

1. User Profile Aggregator
   Purpose: Collect background demographic info (static or infrequently updated)

   - PostgreSQL
     - As KV sir mentioned, this is best for storing user profiles because it is well-structured + supports indexing and constraints
   - MongoDB
     - This would be necessary if our profile structures become highly dynamic— not expected for our demographics

   Proposed DB: PostgreSQL

2. Learner Behavior Tracker
   Purpose: Track and update user behavior across courses— lots of events, semi-structured data.
   - Overall learning patterns
   - Session habits across subjects
   - Preference trends (e.g., format, time-of-day, interaction types)
   - Skill progression across topics
   - Global engagement indicators

   - MongoDB
     - Aggregated user learning profiles evolve over time
     - Per-user modeling to feed the decision tree
   - ClickHouse/TimescaleDB
     - Aggregating time-series behavior such as how frequently a user interacts with learning materials across courses and cross-course skill improvement over time

- PostgreSQL
    - Useful for storing analytics-read summaries
    - Can power decision tree training datasets and external analytics

```
user_id | total_courses | avg_score | dominant_format | last_activity_date
--------+---------------+-----------+-----------------+-------------------
abc123  |      4        |   0.68    |      visual     |   2025-07-12
```

Proposed DB: MongoDB + ClickHouse/PostgreSQL

3. Course Engagement Tracker
   Purpose: Per-course engagement monitoring, micro-logs of interaction within a course.

- MongoDB
    - Flexible schema to store per-user learning journeys as not all user journeys will need to store the same info
    - Nested data support to track quiz attempts, hint requests, confidence score trends, etc.
    - Storage and querying for time-stamped data (session durations, module completions, time spent on quizzes, etc.)
- ClickHouse/TimescaleDB
    - Fast time series behavioral logs
- ElasticSearch
    - searching/filtering logs ("Users who watched video x and then did y")
    - For querying interaction patterns
- PostgreSQL
    - Would not work well as structures will be more dynamic

Proposed DB: MongoDB + ClickHouse/TimescaleDB (for querying behavior patterns efficiently)

4. Learner Chat Manager

   Purpose: Real-time learner interaction management with LLM/SME; persistent storage of chat history, session tracking.

   - Redis Streams
     - Fast data structure in Redis for streaming data
     - Low latency so good for real time communication
     - Acts as a queue between frontend users and the LLM/SME processing engine
     - Temporary storage— does not persist full conversation history
   - ElasticSearch
     - Full-text search across past conversations, useful for LLM context recall and feedback mining
     - Store structured chat logs for search, filtering, and analytics
     - Can also power dashboards (e.g. with Kibana)
   - MongoDB
     - Lets us store full conversations as a single document
   - PostgreSQL
     - Not useful for real time, unstructured messaging but can store session summaries

   Proposed DB: Redis Streams(real-time) + ElasticSearch (search/analytics). PostgreSQL for chat summaries if needed.

5. Decision Tree Manager

Purpose: Run decision trees using learner behavior and engagement; manage and update thresholds; storing past decisions and logs, supporting retraining and threshold tuning through analytics.

- PostgreSQL
  - Store structured configuration threshold data + decision logs
  - Track which thresholds were in place when a decision was made (as it is ACID compliant)
- Redis
  - Fast and in-memory access to threshold values
- ClickHouse
  - Useful for model evaluation, retraining, threshold tuning (once we get to a place where we need to frequently retrain)
- DuckDB
  - Local/in-process analytics for on-the-fly training or decision testing

Proposed DB: PostgreSQL + Redis as baseline. DuckDB as we need to retrain frequently but on a small number of additional data points (as data comes in through more and more usage).

6. <u>Decision Tree Facilitator</u>

   Purpose: Deliver tree outputs to the query strategy manager; likely needs speed and traceability.

   - Redis
     - In-memory data store with instantaneous read/write performance
     - low-latency delivery of decisions to other services

- PostgreSQL
    - To do the same things that it does in the decision tree manager
- Kafka/NATS
    - Allow reliable, asynchronous event driven delivery at scale
- Prometheus
    - Monitors system health through decision execution latency, error rates, decisions made per hour, ache hit/miss ratios

Proposed DB: Redis, PostgreSQL, Kafka, Prometheus

1. User Profile Aggregator (PostgreSQL)

Purpose: Store static/infrequently updated learner background data
Attributes:

- `user_id` (UUID / Primary Key)
- `age` (Integer)
- `class_year` (String / Enum)
- `courses_completed` (Integer)
- `gpa` (Float)
- `learner_purpose` (Enum: 'scratch', 'revising', 'exploring')
- `format_preference` (Enum: 'text', 'image', 'video')
- `format_comfort_rating_text` (Integer 1–5)
- `format_comfort_rating_image` (Integer 1–5)
- `format_comfort_rating_video` (Integer 1–5)
- `format_comprehension_score_text` (Float 0–1)
- `format_comprehension_score_image` (Float 0–1)
- `format_comprehension_score_video` (Float 0–1)

- `session_preference` (Enum: 'short_chunks', 'long_sessions')
- `ui_theme_preference` (Enum: 'dark', 'light')
- `layout_preference` (String or Enum)
- `study_mode` (Enum: 'solo', 'group')
- `learner_level` (Enum: 'basic', 'intermediate', 'advanced')
- `writing_style_category` (Enum: 'formal-analytical', 'conversational', etc.)
- `writing_metrics`: (JSON)
  - `avg_sentence_length` (Float)
  - `type_token_ratio` (Float)
  - `pronoun_usage` (Float)
  - `contractions_count` (Integer)

---

2. Learner Behavior Tracker (MongoDB + ClickHouse/PostgreSQL)

Purpose: Track behavior and time-series engagement
Attributes:

- `user_id`
- `course_id`
- `total_time_spent` (Float)
- `content_type_engagement`: (JSON)
  - `quiz_time` (Float)
  - `video_time` (Float)
  - `reading_time` (Float)
  - `hint_usage_count` (Integer)
- `module_order_log`: List of `{module_id, timestamp}` (for path deviation)
- `retention_span` (Float avg session duration)
- `active_time_distribution`: JSON of time-of-day vs. activity
- `exploratory_behavior_score` (Boolean / Float)
- `content_flags`: List of `{topic_id, flagged_type}`

---

3. Course Engagement Tracker (MongoDB + ClickHouse)

Purpose: Track per-course micro-interactions
Attributes:

- `user_id`
- `course_id`
- `module_id`
- `quiz_attempts`: List of `{question_id, score, skill_tag, objective_tag, retries, hint_used}`
- `confidence_score_by_topic`: Dict `{topic_id: score}`
- `confidence_score_trends`: List of `{topic_id, trend_value}`
- `redo_topics`: List of topic IDs flagged for review
- `topic_scores`: `{topic_id: percentage}`
- `PMI`: Float
- `skill_scores`: Dict `{skill_name: score (0–10)}`
- `flagged_topics`: List of topic IDs from surveys
- `session_logs`: JSON list of `{start_time, end_time, duration, activity_type}`

- `format_performance_mismatch`: JSON `{format: {"comfort": 5, "performance": 2}}`

---

4. Learner Chat Manager (Redis Streams + ElasticSearch + MongoDB)

Purpose: Manage LLM chat history and feedback
Attributes:

- `user_id`
- `session_id`
- `timestamp`
- `message_direction`: 'user' / 'system'
- `message_text`
- `contextual_topic_id`
- `related_skill` (if mapped)
- `related_objective`
- `hint_used` (Boolean)
- `feedback_flags` (e.g., “needs revision”)

- `searchable_tags`: keywords/topics
- (Redis Stream Message Queue: ephemeral chat log)
- (ElasticSearch structured storage: for full-text retrieval)
- (MongoDB fallback: full JSON transcript)

---

5. Decision Tree Manager (PostgreSQL + Redis + DuckDB)

Purpose: Store thresholds and decision outputs
Attributes:

- `user_id`
- `timestamp`
- `tree_id` (Enum: `primary_classification`, `intervention_strategy`)
- `input_attributes`: JSON of inputs used (e.g., confidence_score, objective_score, etc.)
- `decision_label` (e.g., `Struggling Novice`, `Fast Tracker`)
- `llm_prompt_output` (string)
- `thresholds_at_time`: JSON snapshot of rules in effect

- `flagged_rules_hit`: List of rule IDs or conditions met
- `retraining_logs`: JSON of `{date, changes_made, data_points_used}`
- `decision_version` (Integer or UUID)

⚡ 6. Decision Tree Facilitator (Redis + PostgreSQL + Kafka/NATS)

Purpose: Deliver decisions to orchestrator/query layer
Attributes:

- `user_id`
- `decision_tree_id`
- `strategy_label`
- `llm_prompt_summary`
- `delivery_timestamp`
- `response_status` (e.g., success, fail, timeout)
- `latency_ms`
- `cache_hit` (Boolean)

- `trace_id` (for linking decisions across services)
- `event_source` (e.g., `onboarding`, `quiz_submit`, `topic_change`)
- Kafka/NATS topic log: for distributed async delivery
- Prometheus metrics: decision rates, failures, latencies