# Database Schema Detailed Design Documentation

Tameem Hossain (GitHub Username: THossain03)

A logo with arrows pointing up

AI-generated content may be incorrect.

Table of Contents

[Database Schema Detailed Design Documentation 1](#_Toc201352490)

[Introduction 3](#_Toc201352491)

[Table Schemas 3](#_Toc201352492)

[1. Users 3](#_Toc201352493)

[2. UserProfiles 4](#_Toc201352494)

[3. UserSessions 5](#_Toc201352495)

[4. UserDevices 7](#_Toc201352496)

[5. Habits 8](#_Toc201352497)

[6. HabitLogs 9](#_Toc201352498)

[7. HabitStats 11](#_Toc201352499)

[8. UserAIModels 12](#_Toc201352500)

[9. HabitPredictions 14](#_Toc201352501)

[10. AIModelPerformance 15](#_Toc201352502)

[11. ExternalFeedback 16](#_Toc201352503)

[12. FeedbackActions 18](#_Toc201352504)

[13. HabitAdviceLinks 19](#_Toc201352505)

[14. UserSettings 20](#_Toc201352506)

[15. NotificationPrefs 20](#_Toc201352507)

[16. HabitTemplates 22](#_Toc201352508)

[17. Categories 23](#_Toc201352509)

[18. HabitTags 24](#_Toc201352510)

[19. UserActivityLog 25](#_Toc201352511)

[20. ErrorLogs 26](#_Toc201352512)

[21. UserAchievements 27](#_Toc201352513)

[22. APIKeys 28](#_Toc201352514)

[23. WebhookLogs 29](#_Toc201352515)

[Summary of Tables (22 total) 31](#_Toc201352516)

[Additional Notes on Schema Decisions 32](#_Toc201352517)

[How to Use This Schema 33](#_Toc201352518)

## Introduction

We have gathered here a more detailed, expanded schema design document for ~22 DynamoDB tables for NeuroHabits, designed with recommendation and formatting from OpenAI ChatGPT o4-mini. For each table, it lists:

* **Purpose/description**
* **Attributes** with types and roles (PK, SK, other indexed attributes)
* **Primary Key schema** (Partition Key and Sort Key)
* **GSIs / LSIs** (when relevant) with their key schemas and projected attributes
* Notes on access patterns that motivate each index.

Note on DynamoDB normalization vs. access-pattern modeling:  
We still think in terms of normalized “entities” to clarify domains and dependencies (1NF/2NF/3NF), but in DynamoDB you often co-locate items or even collapse multiple logical tables into one physical table. Here we present them as separate logical tables; in implementation you may choose single-table designs or keep separate tables depending on query patterns, scale, and team preference.

## Table Schemas

### 1. Users

**Purpose**: Master user account record (authentication, basic info).

* **Attributes**:
  + UserID (String) — PK
  + Email (String) — unique login email
  + HashedPassword (String)
  + CreatedAt (String, ISO8601 timestamp)
  + Status (String) — e.g. ACTIVE, SUSPENDED
  + LastLoginAt (String, ISO8601)
  + PreferredLocale (String)
  + EmailVerified (Boolean)
  + MFAEnabled (Boolean)
  + PasswordResetToken (String, nullable)
  + PasswordResetExpiry (String, ISO8601, nullable)
  + *(other audit fields as needed)*
* **Primary Key**:
  + PK: UserID
* **Sort Key**: none
* **LSIs**: none (no sort key on base)
* **GSIs**:
  + **GSI\_EmailIndex**
    - PK: Email
    - SK: none
    - Projection: ALL or at least UserID, HashedPassword, Status, EmailVerified.
    - Purpose: lookup UserID and credential info by email during login.
  + **GSI\_StatusIndex** (optional)
    - PK: Status
    - SK: CreatedAt
    - Projection: UserID, Email, CreatedAt
    - Purpose: admin queries of users by status (e.g. list suspended accounts).
* **Access patterns**:
  + Get user by UserID.
  + Authenticate: query by Email → get UserID + hashed password.
  + Admin listing by Status.

### 2. UserProfiles

**Purpose**: Extended user profile details separate from core auth.

* **Attributes**:
  + UserID (String) — PK
  + ProfileID (String) — same as UserID or versioned; here we assume 1:1 so ProfileID = UserID
  + FullName (String)
  + DisplayName (String)
  + Bio (String)
  + AvatarURL (String)
  + Timezone (String)
  + Preferences (Map) — e.g. theme, default reminders
  + CreatedAt (String)
  + UpdatedAt (String)
  + Demographics (Map) — e.g. birthYear, gender (if collected)
  + Locale (String)
  + *(other profile fields)*
* **Primary Key**:
  + PK: UserID
* **Sort Key**: none
* **GSIs**: rarely needed, since profile looked up by UserID. If you want search by DisplayName:
  + **GSI\_DisplayNameIndex**
    - PK: DisplayName
    - SK: none
    - Projection: UserID, AvatarURL
    - Use-case: search users by display name (if public social features).
* **Access patterns**:
  + Get / update profile by UserID.
  + Possibly search by DisplayName for social.

### 3. UserSessions

**Purpose**: Track login sessions/tokens for users (for logout, revoke tokens).

* **Attributes**:
  + UserID (String) — PK
  + SessionID (String) — SK (e.g. UUID)
  + CreatedAt (String)
  + ExpiresAt (String)
  + DeviceInfo (Map) — e.g. OS, browser/app version
  + IP (String)
  + UserAgent (String)
  + IsActive (Boolean)
  + LastSeenAt (String) — for heartbeat
  + *(other session metadata)*
* **Primary Key**:
  + PK: UserID
  + SK: SessionID
* **LSIs**: none in practice.
* **GSIs**:
  + **GSI\_ExpiresAtIndex**
    - PK: IsActive (Boolean or String “ACTIVE”/“INACTIVE”)
    - SK: ExpiresAt
    - Projection: UserID, SessionID, DeviceInfo
    - Purpose: find sessions to expire or cleanup.
  + **GSI\_IPIndex** (optional)
    - PK: IP
    - SK: LastSeenAt
    - Projection: UserID, SessionID, DeviceInfo
    - Purpose: detect suspicious sessions from same IP.
* **Access patterns**:
  + List all sessions for a user (PK query).
  + Revoke session: delete by UserID+SessionID.
  + Cleanup expired sessions: scan GSI\_ExpiresAtIndex where IsActive=“ACTIVE” and ExpiresAt < now.
  + (Optional) Security monitoring via IP index.

### 4. UserDevices

**Purpose**: Track user devices for push notifications, device-specific settings.

* **Attributes**:
  + UserID (String) — PK
  + DeviceID (String) — SK (UUID per device install)
  + DeviceType (String) — e.g. Android, iOS, Web
  + PushToken (String) — for notifications
  + AppVersion (String)
  + LastActiveAt (String)
  + RegisteredAt (String)
  + IsActive (Boolean)
  + *(other device metadata)*
* **Primary Key**:
  + PK: UserID
  + SK: DeviceID
* **GSIs**:
  + **GSI\_PushTokenIndex**
    - PK: PushToken
    - SK: UserID
    - Projection: DeviceID, DeviceType, IsActive
    - Purpose: given a push token, find user/device for sending.
  + **GSI\_LastActiveIndex** (optional)
    - PK: UserID
    - SK: LastActiveAt
    - Projection: DeviceID, IsActive
    - Purpose: list devices by recency (though base SK might suffice).
* **Access patterns**:
  + List devices for user.
  + Find device by DeviceID if needed (via PK+SK).
  + Given push token, find UserID to send.
  + Cleanup inactive devices.

### 5. Habits

**Purpose**: Master record of each habit a user defines.

* **Attributes**:
  + UserID (String) — PK
  + HabitID (String) — SK (UUID)
  + Name (String)
  + Description (String)
  + CategoryID (String) — FK to Categories
  + TemplateID (String, nullable) — if based on template
  + CreatedAt (String)
  + UpdatedAt (String)
  + IsActive (Boolean)
  + FrequencyType (String) — e.g. DAILY, WEEKLY, CUSTOM
  + FrequencySpec (Map) — e.g. days of week, interval
  + GoalTarget (Number) — e.g. times per period
  + Timezone (String) — optional override
  + StreakCount (Number) — kept in HabitStats but could be mirrored
  + NextDueAt (String) — next recommended time
  + *(other config fields)*
* **Primary Key**:
  + PK: UserID
  + SK: HabitID
* **LSIs**:
  + Could define LSI on UpdatedAt if wanting to query habits by recency:
    - LSI\_UpdatedAt: PK=UserID, SK=UpdatedAt
    - Projection: include HabitID, Name, IsActive.
* **GSIs**:
  + **GSI\_CategoryIndex**
    - PK: CategoryID
    - SK: UserID#HabitID or UserID
    - Projection: Name, UserID, HabitID, IsActive
    - Purpose: for analytics or sharing templates—e.g. find habits of all users in a category (if public/shared analysis). Use with caution re: multi-tenancy; could restrict to templates only.
  + **GSI\_NameIndex** (optional)
    - PK: UserID
    - SK: Name
    - Projection: HabitID, CategoryID
    - Purpose: detect duplicate names or search user’s habits by name prefix.
  + **GSI\_NextDueIndex**
    - PK: UserID
    - SK: NextDueAt
    - Projection: HabitID, Name, NextDueAt
    - Purpose: quickly fetch upcoming habits to remind.
* **Access patterns**:
  + List all habits for a user (PK query).
  + Get specific habit by UserID+HabitID.
  + Query upcoming due habits via GSI\_NextDueIndex.
  + Query by category if needed.
  + Search by name.

### 6. HabitLogs

**Purpose**: Each time user logs a habit occurrence (or skip), timestamped record.

* **Attributes**:
  + UserID (String) — PK
  + LogID (String) — UUID for each log
  + HabitID (String)
  + LoggedAt (String, ISO8601 timestamp) — time of occurrence
  + Status (String) — e.g. COMPLETED, MISSED, SKIPPED
  + Notes (String)
  + Duration (Number) — optional (minutes)
  + Metadata (Map) — arbitrary key-values, e.g. mood, location
  + CreatedAt (String) — same as LoggedAt or entry time
  + *(other fields like Geo location if used)*
* **Primary Key**:
  + PK: UserID
  + SK: HabitID#LoggedAt#LogID
    - e.g. SK format: "HABIT#<HabitID>#<LoggedAtISO>#<LogID>"
    - This ensures logs for a given habit are clustered by timestamp.
* **LSIs**:
  + **LSI\_StatusIndex**
    - PK: UserID
    - SK: Status#LoggedAt
    - Projection: HabitID, LogID, Notes
    - Purpose: query logs by status (e.g. find all MISSED entries in last period).
* **GSIs**:
  + **GSI\_HabitTimestampIndex**
    - PK: HabitID
    - SK: LoggedAt#UserID#LogID
    - Projection: Status, Notes, UserID
    - Purpose: cross-user aggregate analytics (e.g. count how many users logged this habit template today). *Use with caution for scale*
  + **GSI\_DateIndex**
    - PK: UserID
    - SK: LoggedAt
    - Projection: HabitID, Status, Notes
    - Purpose: query all logs for user in a date range (all habits), ignoring per-habit grouping.
  + **GSI\_MetadataIndex** (optional): if wanting to query logs by certain metadata fields (e.g., mood). For each metadata key used for queries, create a sparse GSI:
    - e.g. GSI\_MoodIndex: PK: UserID, SK: Mood#LoggedAt (where Mood is extracted from Metadata when present).
    - Projection: HabitID, LogID, Notes.
* **Access patterns**:
  + List logs for a habit: Query PK=UserID, SK begins with HABIT#<HabitID>#.
  + List all logs in a timeframe: GSI\_DateIndex with range on LoggedAt.
  + Query by status via LSI.
  + Cross-user analytics for a template habit (via GSI\_HabitTimestampIndex).

### 7. HabitStats

**Purpose**: Aggregated metrics per habit to avoid heavy scans. Computed/updated after each log.

* **Attributes**:
  + UserID (String) — PK
  + HabitID (String) — SK
  + TotalCount (Number)
  + StreakCurrent (Number)
  + StreakBest (Number)
  + LastLoggedAt (String)
  + FirstLoggedAt (String)
  + MissedCount (Number)
  + SuccessRate (Number) — e.g. percentage
  + AverageInterval (Number) — e.g. days
  + RollingWindowCounts (Map) — e.g. counts per week/month
  + NextReviewAt (String) — when next stats review is scheduled
  + UpdatedAt (String)
  + *(other aggregated fields)*
* **Primary Key**:
  + PK: UserID
  + SK: HabitID
* **LSIs**:
  + **LSI\_NextReviewIndex**
    - PK: UserID
    - SK: NextReviewAt
    - Projection: HabitID, StreakCurrent, SuccessRate
    - Purpose: trigger scheduled reviews/notifications.
* **GSIs**:
  + **GSI\_StreakBestIndex** (optional)
    - PK: UserID
    - SK: StreakBest#HabitID
    - Projection: StreakBest, HabitID
    - Purpose: list top streak habits.
  + **GSI\_AverageIntervalIndex** (optional)
    - PK: UserID
    - SK: AverageInterval#HabitID
    - Projection: AverageInterval.
* **Access patterns**:
  + Get stats for a habit: PK+SK.
  + List habits by next review: LSI\_NextReviewIndex.
  + Show top streaks/other metrics.

### 8. UserAIModels

**Purpose**: Track AI models created or fine-tuned per user (local or managed). Versioning and metadata.

* **Attributes**:
  + UserID (String) — PK
  + ModelID (String) — SK (UUID or meaningful version string)
  + ModelType (String) — e.g. “habit-predictor”, “recommendation-engine”
  + CreatedAt (String)
  + Version (String or Number)
  + Parameters (Map) — hyperparameters or config
  + ModelLocation (String) — S3 path or endpoint info
  + Status (String) — TRAINING, ACTIVE, FAILED, ARCHIVED
  + LastTrainedAt (String)
  + Metrics (Map) — e.g. accuracy, loss
  + TrainingDataSummary (Map) — e.g. #records, date range
  + Notes (String)
  + *(other metadata)*
* **Primary Key**:
  + PK: UserID
  + SK: ModelID
* **LSIs**:
  + **LSI\_StatusIndex**
    - PK: UserID
    - SK: Status#CreatedAt
    - Projection: ModelID, ModelType
    - Purpose: list active vs. archived models.
* **GSIs**:
  + **GSI\_ModelTypeIndex**
    - PK: ModelType
    - SK: UserID#ModelID
    - Projection: CreatedAt, Status
    - Purpose: system-wide view of models of a given type (for multi-tenant monitoring). Use with caution.
  + **GSI\_LastTrainedIndex**
    - PK: UserID
    - SK: LastTrainedAt
    - Projection: ModelID, Status
    - Purpose: find models needing retraining or review.
* **Access patterns**:
  + List user’s models.
  + Get a model’s metadata.
  + Query active models or by type.

### 9. HabitPredictions

**Purpose**: Store AI model outputs/predictions or recommendations for each habit instance or future suggestion.

* **Attributes**:
  + UserID (String) — PK
  + PredictionID (String) — SK (UUID or composite HabitID#Timestamp)
  + HabitID (String)
  + GeneratedAt (String)
  + PredictionType (String) — e.g. “next\_best\_time”, “likelihood\_of\_completion”
  + PredictionValue (Map) — details depending on type
  + ModelID (String)
  + Confidence (Number)
  + ExpiresAt (String) — if prediction valid only until certain date
  + Context (Map) — e.g. recent logs summary
  + *(other fields)*
* **Primary Key**:
  + PK: UserID
  + SK: HabitID#GeneratedAt#PredictionID
* **LSIs**:
  + **LSI\_TypeIndex**
    - PK: UserID
    - SK: PredictionType#GeneratedAt
    - Projection: HabitID, PredictionValue
    - Purpose: fetch latest predictions of a given type.
* **GSIs**:
  + **GSI\_ModelPredictionIndex**
    - PK: ModelID
    - SK: GeneratedAt#UserID#PredictionID
    - Projection: HabitID, PredictionType, Confidence
    - Purpose: analyze predictions generated by a model across users.
  + **GSI\_ExpiryIndex**
    - PK: UserID
    - SK: ExpiresAt
    - Projection: PredictionID, HabitID
    - Purpose: cleanup expired predictions.
* **Access patterns**:
  + List predictions for a habit: Query PK with SK begins HabitID#.
  + Fetch latest of a certain type: LSI\_TypeIndex with descending sort on GeneratedAt.
  + Cleanup expired via GSI\_ExpiryIndex.

### 10. AIModelPerformance

**Purpose**: Time-series of model performance metrics (for tracking drift, retraining triggers).

* **Attributes**:
  + UserID (String) — PK
  + ModelID (String)
  + Date (String, e.g. YYYY-MM-DD) — part of SK
  + MetricType (String) — e.g. “accuracy”, “loss”
  + MetricValue (Number)
  + RecordedAt (String) — timestamp
  + Notes (String)
* **Primary Key**:
  + PK: UserID
  + SK: ModelID#Date#MetricType
* **LSIs**:
  + **LSI\_DateIndex**
    - PK: UserID
    - SK: Date#ModelID#MetricType
    - Projection: MetricValue
    - Purpose: query all metrics for a date or date range.
* **GSIs**:
  + **GSI\_ModelDateIndex**
    - PK: ModelID
    - SK: Date#UserID#MetricType
    - Projection: MetricValue, RecordedAt
    - Purpose: compare performance across users/models.
* **Access patterns**:
  + Query performance metrics for a model over time: query PK=UserID with SK begins ModelID#.
  + Date-based queries via LSI.
  + Cross-user via GSI.

### 11. ExternalFeedback

**Purpose**: Store advice/feedback fetched from third-party APIs for habits or user in general.

* **Attributes**:
  + UserID (String) — PK
  + FeedbackID (String) — SK (UUID)
  + Source (String) — e.g. API provider name
  + FetchedAt (String)
  + Content (String) — advice text or JSON blob
  + FeedbackType (String) — e.g. “habit-improvement”, “nutrition-suggestion”
  + RelatedHabitID (String, nullable) — if advice tied to a habit
  + ProviderMetadata (Map) — raw response fields
  + Processed (Boolean) — whether processed into UI
  + ExpiresAt (String) — when advice is stale
  + *(other fields)*
* **Primary Key**:
  + PK: UserID
  + SK: FeedbackID
* **LSIs**:
  + **LSI\_TypeIndex**
    - PK: UserID
    - SK: FeedbackType#FetchedAt
    - Projection: Source, RelatedHabitID, Processed
    - Purpose: list advice by type for user.
* **GSIs**:
  + **GSI\_SourceIndex**
    - PK: Source
    - SK: FetchedAt#UserID#FeedbackID
    - Projection: FeedbackType
    - Purpose: monitor usage of particular API provider across users / quotas.
  + **GSI\_RelatedHabitIndex**
    - PK: RelatedHabitID
    - SK: UserID#FetchedAt#FeedbackID
    - Projection: Content, Source
    - Purpose: for a habit, fetch all feedback across users (e.g., template-based analytics). Use carefully.
* **Access patterns**:
  + List all feedback for user: PK query.
  + Fetch by type for UI grouping.
  + Cleanup stale advice via ExpiresAt (via scan or TTL).
  + Analytics on source usage.

### 12. FeedbackActions

**Purpose**: User’s interactions with a piece of feedback/advice (accept, reject, comment).

* **Attributes**:
  + UserID (String) — PK
  + ActionID (String) — SK (UUID or composite FeedbackID#action timestamp)
  + FeedbackID (String) — FK to ExternalFeedback
  + ActionType (String) — e.g. “ACCEPTED”, “REJECTED”, “COMMENTED”
  + ActionAt (String)
  + Comment (String, nullable)
  + Metadata (Map) — e.g. reason codes
* **Primary Key**:
  + PK: UserID
  + SK: FeedbackID#ActionAt#ActionID
* **LSIs**:
  + **LSI\_ActionTypeIndex**
    - PK: UserID
    - SK: ActionType#ActionAt
    - Projection: FeedbackID, Comment
    - Purpose: query recent accepts/rejects.
* **GSIs**:
  + **GSI\_FeedbackIndex**
    - PK: FeedbackID
    - SK: ActionAt#UserID#ActionID
    - Projection: ActionType, Comment
    - Purpose: see how users responded to a particular advice (for analytics).
* **Access patterns**:
  + List actions for a feedback: PK query with SK begins FeedbackID.
  + Recent actions by type: LSI\_ActionTypeIndex.
  + Analytics cross-user via GSI\_FeedbackIndex.

### 13. HabitAdviceLinks

**Purpose**: Link table between Habits and ExternalFeedback (many-to-many). In Dynamo terms, separate items.

* **Attributes**:
  + UserID (String) — PK
  + LinkID (String) — SK, e.g. HabitID#FeedbackID
  + HabitID (String)
  + FeedbackID (String)
  + LinkedAt (String)
  + Notes (String, nullable)
* **Primary Key**:
  + PK: UserID
  + SK: HabitID#FeedbackID
* **GSIs**:
  + **GSI\_HabitIndex**
    - PK: HabitID
    - SK: UserID#FeedbackID
    - Projection: LinkedAt, FeedbackID
    - Purpose: for a habit, list all feedback links (cross-user analytics).
  + **GSI\_FeedbackIndex**
    - PK: FeedbackID
    - SK: UserID#HabitID
    - Projection: LinkedAt, HabitID
    - Purpose: for a feedback, list all habits linked by user(s).
* **Access patterns**:
  + Given user and habit, check if linked advice exists.
  + List advice linked to a habit: query PK=UserID with SK begins HabitID.
  + Analytics across users via GSI\_HabitIndex.

### 14. UserSettings

**Purpose**: Store per-user app-wide settings.

* **Attributes**:
  + UserID (String) — PK
  + Settings (Map) — e.g. theme, default reminder times, privacy toggles
  + UpdatedAt (String)
* **Primary Key**:
  + PK: UserID
* **Sort Key**: none
* **Indexes**: none typically (lookup by UserID).
* **Access patterns**:
  + Get/update settings by UserID.

### 15. NotificationPrefs

**Purpose**: Reminder / notification configuration per user (multiple entries if user sets reminders per habit or general times).

* **Attributes**:
  + UserID (String) — PK
  + NotificationID (String) — SK (UUID)
  + Type (String) — e.g. “HABIT\_REMINDER”, “DAILY\_SUMMARY”
  + Channel (String) — EMAIL, PUSH, SMS
  + ScheduleSpec (Map) — e.g. cron-like or interval, time of day
  + Enabled (Boolean)
  + CreatedAt (String)
  + UpdatedAt (String)
  + LastSentAt (String)
  + NextScheduledAt (String)
  + RelatedHabitID (String, nullable) — if reminder tied to a habit
* **Primary Key**:
  + PK: UserID
  + SK: NotificationID
* **LSIs**:
  + **LSI\_TypeIndex**
    - PK: UserID
    - SK: Type#NotificationID
    - Projection: Channel, Enabled, NextScheduledAt
    - Purpose: list certain kinds of notifications for user.
* **GSIs**:
  + **GSI\_NextScheduledIndex**
    - PK: NextScheduledAt (String)
    - SK: UserID#NotificationID
    - Projection: Channel, Type, Enabled
    - Purpose: scheduler service: query notifications due at or before now.
  + **GSI\_RelatedHabitIndex**
    - PK: RelatedHabitID
    - SK: UserID#NotificationID
    - Projection: NextScheduledAt, Enabled
    - Purpose: if habit deleted/modified, find associated reminders.
* **Access patterns**:
  + List notifications for user.
  + Scheduler queries due notifications by timestamp.
  + Find reminders linked to a habit.

### 16. HabitTemplates

**Purpose**: Shared/common habit templates users can pick from.

* **Attributes**:
  + TemplateID (String) — PK
  + Name (String)
  + Description (String)
  + CategoryID (String)
  + DefaultFrequencyType (String)
  + DefaultFrequencySpec (Map)
  + DefaultGoalTarget (Number)
  + CreatedBy (String) — maybe admin or community user
  + CreatedAt (String)
  + IsPublic (Boolean)
  + Tags (List of String)
  + Metadata (Map)
  + UsageCount (Number) — how many users have used
  + LastUpdatedAt (String)
* **Primary Key**:
  + PK: TemplateID
* **Sort Key**: none
* **GSIs**:
  + **GSI\_CategoryIndex**
    - PK: CategoryID
    - SK: UsageCount#TemplateID
    - Projection: Name, IsPublic
    - Purpose: list popular templates by category.
  + **GSI\_NameIndex**
    - PK: Name
    - SK: TemplateID
    - Projection: CategoryID, IsPublic
    - Purpose: search templates by name.
  + **GSI\_TagIndex** (sparse)
    - PK: generate items per tag or use GSI with PK=Tag, SK=TemplateID
    - Projection: Name, CategoryID
    - Purpose: search by tag.
* **LSIs**: none
* **Access patterns**:
  + Get template by ID.
  + List/search templates by category, name, tag.
  + Analytics on usage.

### 17. Categories

**Purpose**: Define habit categories (static reference).

* **Attributes**:
  + CategoryID (String) — PK
  + Name (String)
  + Description (String)
  + ParentCategoryID (String, nullable) — hierarchical
  + CreatedAt (String)
  + UpdatedAt (String)
* **Primary Key**:
  + PK: CategoryID
* **GSIs**:
  + **GSI\_NameIndex**
    - PK: Name
    - SK: none
    - Projection: CategoryID, ParentCategoryID
    - Purpose: lookup category by name.
  + **GSI\_ParentIndex**
    - PK: ParentCategoryID
    - SK: CategoryID
    - Projection: Name
    - Purpose: list subcategories.
* **Access patterns**:
  + Fetch category by ID.
  + Build category tree via ParentCategoryID index.

### 18. HabitTags

**Purpose**: User-assigned tags for habits (many-to-many per habit).

* **Attributes**:
  + UserID (String) — PK
  + TagID (String) — SK, but composite key format: Tag#<TagValue>#Habit#<HabitID> or simpler SK=<HabitID>#<TagValue>
  + HabitID (String)
  + TagValue (String)
  + TaggedAt (String)
* **Primary Key**:
  + PK: UserID
  + SK: HabitID#TagValue
* **GSIs**:
  + **GSI\_TagValueIndex**
    - PK: TagValue
    - SK: UserID#HabitID
    - Projection: TaggedAt
    - Purpose: search global trending tags or cross-user analytics.
  + **GSI\_HabitIndex**
    - PK: HabitID
    - SK: TagValue#UserID
    - Projection: TaggedAt
    - Purpose: list tags for a given habit.
* **Access patterns**:
  + List tags for a user’s habit: query PK=UserID SK begins HabitID.
  + Search by tag across users: GSI\_TagValueIndex.

### 19. UserActivityLog

**Purpose**: Audit log of user actions (for analytics, auditing).

* **Attributes**:
  + UserID (String) — PK
  + ActionID (String) — SK: ActionTimestamp#<UUID>
  + ActionType (String) — e.g. “LOGIN”, “CREATE\_HABIT”, “UPDATE\_PROFILE”
  + ActionTimestamp (String)
  + Details (Map) — additional context: e.g. which habitID was created
  + IPAddress (String)
  + UserAgent (String)
  + *(other audit fields)*
* **Primary Key**:
  + PK: UserID
  + SK: ActionTimestamp#ActionID
* **LSIs**:
  + **LSI\_ActionTypeIndex**
    - PK: UserID
    - SK: ActionType#ActionTimestamp
    - Projection: Details
    - Purpose: query specific action types for troubleshooting (e.g. show all “DELETE\_HABIT” events).
* **GSIs**:
  + **GSI\_GlobalActionIndex**
    - PK: ActionType
    - SK: ActionTimestamp#UserID#ActionID
    - Projection: Details
    - Purpose: system-wide audit of certain actions across users.
* **Access patterns**:
  + Fetch user’s recent actions.
  + Fetch by type via LSI.
  + Global audit via GSI.

### 20. ErrorLogs

**Purpose**: Application error tracking, per user or global.

* **Attributes**:
  + ErrorID (String) — PK (UUID)
  + OccurredAt (String)
  + UserID (String, nullable) — if tied to user
  + Service (String) — e.g. “HabitService”, “AuthService”
  + ErrorType (String) — e.g. “ValidationError”, “Timeout”
  + Message (String)
  + StackTrace (String)
  + Metadata (Map)
  + Resolved (Boolean)
  + ResolvedAt (String, nullable)
* **Primary Key**:
  + PK: ErrorID
* **GSIs**:
  + **GSI\_UserErrorIndex**
    - PK: UserID
    - SK: OccurredAt#ErrorID
    - Projection: Service, ErrorType, Resolved
    - Purpose: list errors per user for support.
  + **GSI\_ServiceErrorIndex**
    - PK: Service
    - SK: OccurredAt#ErrorID
    - Projection: ErrorType, Resolved
    - Purpose: monitor errors by service.
  + **GSI\_ResolvedIndex** (optional)
    - PK: Resolved (String “TRUE”/“FALSE”)
    - SK: OccurredAt#ErrorID
    - Projection: Service, UserID
    - Purpose: list unresolved errors older than threshold.
* **Access patterns**:
  + Lookup error by ID.
  + Query errors by user or service.
  + Cleanup/resolution workflows.

### 21. UserAchievements

**Purpose**: Gamification: badges or milestones earned by user.

* **Attributes**:
  + UserID (String) — PK
  + AchievementID (String) — SK (UUID or code)
  + Type (String) — e.g. “STREAK\_7\_DAYS”, “LOGGED\_FIRST\_HABIT”
  + EarnedAt (String)
  + Metadata (Map) — e.g. streak length
  + NotifiedAt (String) — when user was informed
  + Claimed (Boolean)
* **Primary Key**:
  + PK: UserID
  + SK: AchievementID
* **LSIs**:
  + **LSI\_TypeIndex**
    - PK: UserID
    - SK: Type#EarnedAt
    - Projection: Metadata, Claimed
    - Purpose: check if a user already has an achievement type.
* **GSIs**:
  + **GSI\_TypeGlobalIndex**
    - PK: Type
    - SK: UserID#EarnedAt
    - Projection: Metadata
    - Purpose: analytics: how many users earned a badge.
* **Access patterns**:
  + List achievements for user.
  + Check if user has certain achievement: query LSI\_TypeIndex.
  + Cross-user stats via GSI.

### 22. APIKeys

**Purpose**: (Internal) Store external API keys or integration tokens per user.

* **Attributes**:
  + UserID (String) — PK
  + APIKeyID (String) — SK (UUID)
  + Service (String) — e.g. “OpenAI”, “Fitbit”
  + KeyValue (String) — encrypted or token reference
  + CreatedAt (String)
  + ExpiresAt (String, nullable)
  + LastUsedAt (String)
  + Permissions (List) — scopes
  + Status (String) — ACTIVE/REVOKED
  + Metadata (Map)
* **Primary Key**:
  + PK: UserID
  + SK: APIKeyID
* **LSIs**:
  + **LSI\_ServiceIndex**
    - PK: UserID
    - SK: Service#APIKeyID
    - Projection: Status, ExpiresAt
    - Purpose: quickly find key for a service.
* **GSIs**:
  + **GSI\_ServiceGlobalIndex**
    - PK: Service
    - SK: UserID#APIKeyID
    - Projection: LastUsedAt, Status
    - Purpose: monitor usage of a given API across users (e.g. quota management).
* **Access patterns**:
  + List keys for user.
  + Lookup key by service.
  + Global analytics on API usage.

### 23. WebhookLogs

**Purpose**: Track outbound webhook calls (e.g., sending notifications, third-party callbacks).

* **Attributes**:
  + UserID (String) — PK
  + WebhookID (String) — SK (UUID or timestamp-based)
  + Payload (Map or String)
  + Endpoint (String)
  + SentAt (String)
  + Status (String) — PENDING, SENT, FAILED
  + ResponseCode (Number)
  + ResponseBody (String)
  + RetryCount (Number)
  + NextRetryAt (String, nullable)
  + CreatedAt (String)
  + UpdatedAt (String)
* **Primary Key**:
  + PK: UserID
  + SK: SentAt#WebhookID
* **LSIs**:
  + **LSI\_StatusIndex**
    - PK: UserID
    - SK: Status#SentAt
    - Projection: Endpoint, RetryCount
    - Purpose: find failed webhooks for retry.
* **GSIs**:
  + **GSI\_GlobalStatusIndex**
    - PK: Status
    - SK: SentAt#UserID#WebhookID
    - Projection: Endpoint
    - Purpose: monitor all failed or pending webhooks globally.
  + **GSI\_NextRetryIndex**
    - PK: NextRetryAt
    - SK: UserID#WebhookID
    - Projection: RetryCount, Endpoint
    - Purpose: scheduler to retry at correct time.
* **Access patterns**:
  + List user’s webhooks.
  + Retry failed via LSI\_StatusIndex or GSI\_GlobalStatusIndex.
  + Schedule via NextRetryAt index.

## Summary of Tables (22 total)

1. Users
2. UserProfiles
3. UserSessions
4. UserDevices
5. Habits
6. HabitLogs
7. HabitStats
8. UserAIModels
9. HabitPredictions
10. AIModelPerformance
11. ExternalFeedback
12. FeedbackActions
13. HabitAdviceLinks
14. UserSettings
15. NotificationPrefs
16. HabitTemplates
17. Categories
18. HabitTags
19. UserActivityLog
20. ErrorLogs
21. UserAchievements
22. APIKeys
23. WebhookLogs

*(Note: we counted 23 here; you may merge some or add additional tables for e.g. TeamSharing, SocialInteractions, etc.)*

## Additional Notes on Schema Decisions

1. **1NF**: All attributes atomic (no nested lists except in Map/List attributes which we treat as atomic in schema but avoid multi-valued attributes in keys).
2. **2NF**: Composite PKs ensure non-key attributes depend fully on the key (e.g., HabitLogs SK includes HabitID and LoggedAt).
3. **3NF**: No transitive dependencies: aggregated stats moved to HabitStats; profile separate from Users; feedback separate from habits.
4. **Timestamps as Sort Keys**: Use ISO8601 strings (e.g. 2025-06-20T22:00:00Z) for lexicographical ordering.
5. **GSIs for Query Patterns**: Each index is chosen based on likely queries (e.g., upcoming reminders, logs by date, search by tag).
6. **Sparse GSIs**: If an attribute is not present on all items, index is automatically sparse (e.g., GSI\_TagValueIndex only for items with TagValue).
7. **Composite SK formats**: Use consistent delimiters like # to enable begins\_with queries.
8. **TTL / Cleanup**: Use DynamoDB TTL on items like ExternalFeedback (ExpiresAt) or sessions (ExpiresAt) by setting TTL attribute.
9. **Single-Table vs. Multi-Table**: Though we list separate tables, you could combine some (e.g., Habits, HabitLogs, HabitStats in one table distinguished by item type) if access patterns align and to reduce GSIs count. But separate tables simplify clarity and per-table throughput configs.
10. **Attribute Types**: DynamoDB is schemaless; types are enforced by your application. Document expected types clearly for developers.
11. **Provisioned Throughput / On-Demand**: Depending on scale, choose on-demand for development; for production, provision based on traffic patterns (e.g., heavy reads on HabitLogs).
12. **Encryption & Security**: Sensitive attributes (Password, API keys) should be encrypted at rest (via DynamoDB encryption) and in transit; manage IAM so only appropriate services/roles can read.
13. **Consistency**: Most queries (e.g. get habit or logs) can tolerate eventual consistency; critical auth lookups may use strongly consistent reads.
14. **Batch Operations**: For bulk analytics (e.g. generating monthly reports), consider exporting DynamoDB streams to S3/Redshift or use DynamoDB Streams + Lambda to update an OLAP store.

## How to Use This Schema

* **Development**: Create each table with the specified key schema and define GSIs via AWS Console or CloudFormation.
* **Application Code**: Use a data access layer that encapsulates key-building logic, e.g. functions to format SK strings.
* **Index Maintenance**: Whenever you update attributes used in GSIs (e.g. NextDueAt in Habits), ensure you write the new value so index reflects it.
* **Monitoring**: Monitor GSI usage and throttling; adjust capacity or refactor index usage if hot partitions emerge.
* **Testing**: Write unit tests for query functions against a local DynamoDB emulator to verify index behavior.
* **Evolution**: If new features require new query patterns, add GSIs as needed, being mindful of limits. For heavy cross-user analytics, consider exporting data to a data warehouse rather than rely on DynamoDB scans.