OpenHamPrep

Event System Architecture

Data Capture & Storage Infrastructure

Version 1.5 | January 2026

Open Source Documentation

Contents

1. Overview

1.1 Purpose

This document defines the event-sourcing infrastructure for capturing user learning interactions in OpenHamPrep. The system collects granular event data that can power analytics, readiness scoring, and model calibration without prescribing how that data will be computed or analyzed.

OpenHamPrep is an open-source amateur radio exam preparation platform. This architecture document is published openly so that contributors can understand the design decisions and resource constraints that shaped the system.

1.2 Design Principles

- Append-Only: Events are immutable once written; never modified or deleted during normal operation
- Capture Everything: Record all context that might be useful later, even if not immediately needed
- Pool-Aware: Every question event tracks both location (NCVEC ID) and identity (content hash)
- Schema-Aligned: Events reference existing tables via UUID foreign keys
- Compute Separately: This system only captures and stores; all analysis happens in separate systems
- Cost-Conscious: Designed to run within Supabase Pro plan limits (~$25/month)

1.3 Relationship to Existing Schema

The event system integrates with the existing OpenHamPrep database. It does not replace existing tables but runs alongside them, capturing richer data that can eventually power more sophisticated analytics.

Existing Table	Event System Usage
questions	Events store question_id (UUID) as foreign key reference
profiles	Events store user_id (UUID) referencing auth.users
question_attempts	Events capture similar data but with richer context
question_mastery	Downstream system; can be computed from events
practice_test_results	Events capture with full subelement breakdown
user_readiness_cache	Downstream system; computed from events

1.4 Scope

This document covers:

- Event table schema and payload structures
- Question pool versioning and content hashing
- Event types and when they're captured
- Client-side integration patterns
- Data retention and pruning strategy
- Migration approach from existing system

This document does NOT cover:

- Readiness score computation (separate document)
- Analytics dashboards or reporting
- Item statistics aggregation logic
- Model training or calibration

2. Design Rationale

This section explains why we chose this architecture. Understanding these decisions helps contributors make informed changes and helps other projects learn from our approach.

2.1 Why Event Sourcing?

We considered two approaches for capturing user activity:

Option A: Direct Table Updates (Current System)

The existing OpenHamPrep system uses direct table updates. When a user answers a question:

1. Insert a row into question_attempts
2. Update aggregates in question_mastery (increment counters)
3. Update cached metrics in user_readiness_cache

This works but has limitations:

- Lost context: We only store that an attempt happened, not when in a session, how long it took, or what mode the user was in
- Rigid schema: Adding new metrics requires schema migrations and backfilling
- Coupled computation: The application must know how to update all downstream tables
- Hard to evolve: Changing the readiness formula means updating code AND migrating data

Option B: Event Sourcing (This Design)

With event sourcing, we capture raw events with full context:

4. Insert an immutable event record with all relevant data
5. Downstream systems (mastery, readiness) compute from events as needed
6. Event structure can capture new fields without breaking existing consumers

Benefits:

- Complete history: Every interaction preserved with full context
- Flexible analysis: New metrics can be computed from historical events
- Decoupled systems: Event capture is simple; computation logic is separate
- Debuggable: Can trace exactly how a user reached their current state
- Evolvable: Readiness model v2 can reprocess the same events differently

Why We Chose Event Sourcing

OpenHamPrep is building toward a psychometrically-grounded readiness model. This model will evolve as we collect outcome data and validate predictions. Event sourcing lets us:

- Iterate on the model without losing historical data
- A/B test different formulas against the same event history
- Add new signals (time-on-task, session patterns) without migration
- Build item-level statistics (difficulty, discrimination) from attempt history

The overhead of storing events (vs. just aggregates) is modest—about 600 bytes per question attempt—and well within our resource budget.

2.2 Why Direct PostgREST Inserts?

Supabase offers two ways to write data from the client:

Option A: Edge Functions

Edge Functions are serverless functions that run on Supabase's infrastructure. They're useful for complex operations that require server-side logic, secrets, or external API calls.

```
// Edge Function approach
const response = await fetch('/functions/v1/record-attempt', {
  method: 'POST',
  body: JSON.stringify({ questionId, answer, timeSpent })
});

// Server-side: validate, enrich, insert, update caches...
```

Characteristics:

- Latency: 100-200ms typical (cold start can be 500ms+)
- Cost: Charged per invocation ($2/million) and compute time ($0.018/hour)
- Flexibility: Can run arbitrary code, call external APIs, use secrets

Option B: Direct PostgREST

PostgREST is the REST API layer that Supabase provides over your Postgres database. The Supabase client SDK uses it for all standard CRUD operations.

```
// Direct PostgREST approach
const { error } = await supabase
  .from('events')
  .insert({
    event_type: 'question_attempt',
    user_id: user.id,
    payload: { questionId, answer, timeSpent, ... }
  });
```

Characteristics:

- Latency: 30-50ms typical (no cold start)
- Cost: Included in Pro plan, no per-request charges
- Flexibility: Limited to CRUD operations; no custom logic

Cost Comparison at Scale

Let's calculate the cost difference at various scales:

MAU	Questions/User/Month	Total Attempts	Edge Function Cost	PostgREST Cost
1,000	500	500,000	$1.00	$0
5,000	500	2,500,000	$5.00 + compute	$0
10,000	500	5,000,000	$10.00 + compute	$0
25,000	500	12,500,000	$25.00 + compute	$0
50,000	500	25,000,000	$50.00 + compute	$0

At 10,000 MAU, edge functions would add $10-15/month just for question attempt recording. That's 40-60% of the entire Pro plan cost.

Why We Chose Direct PostgREST

For the hot path (recording question attempts), we use direct PostgREST because:

- It's 3-5x faster (better user experience)
- It's free (included in Pro plan)
- Event recording doesn't need server-side logic—it's just an insert
- Row-Level Security handles authorization

We reserve Edge Functions for operations that genuinely need them:

- Readiness computation (complex calculation, caching logic)
- Outcome reporting (needs to snapshot user state)
- Any operation requiring secrets or external APIs

2.3 Why a Rolling Window with Archival?

We keep 300 events per user in the database, with older events archived to Supabase Storage. Here's why:

What the Readiness Model Actually Needs

The OpenHamPrep readiness model uses these inputs:

Metric	Data Needed	Events Required
Recent accuracy	Last 50 attempts	50
Overall accuracy	Last 200-300 attempts	200-300
Coverage	Unique questions seen (computed incrementally)	0 (cached)
Mastery	Questions correct 2+ times (computed incrementally)	0 (cached)
Practice test rate	Last 10-20 tests	10-20 test events
Recency	Last activity timestamp	1 (most recent)

The model doesn't need 6-month-old events for real-time calculations. But that doesn't mean we should throw them away.

Why Archive Instead of Delete?

Supabase Storage is 6x cheaper than database storage:

- Database: $0.125/GB/month
- Storage: $0.021/GB/month (and archives compress ~10:1)

For approximately $1-2/month extra, we can keep complete history forever. This enables:

- Training ML models on long-term learning patterns
- Research into spaced repetition and forgetting curves
- Debugging issues that span months
- User requests for their complete learning history
- Future features we haven't thought of yet

Why 300 Events in the Database?

We chose 300 events per user in the hot tier because:

- Covers 3-4 weeks of active studying (assuming ~10-15 questions/day)

- Provides enough data for statistically meaningful accuracy calculations
- Includes a buffer beyond the 200 needed for overall accuracy
- Keeps database queries fast (300 rows vs. potentially thousands)

If analysis shows we need more hot data, we can increase this. The archive ensures we never lose anything.

What About Coverage and Mastery?

Coverage (unique questions seen) and mastery (questions correct 2+ times) are cumulative metrics that span a user's entire history. We handle these by:

- Computing them incrementally when events arrive (via triggers or application code)
- Caching the results in user_readiness_cache or a similar table
- Not recomputing from raw events on every request

This means old events can be archived without losing coverage/mastery data—the aggregates persist in the cache. If we ever need to recompute from scratch, we can reload archives.

2.4 Why Content Hashing for Pool Transitions?

Amateur radio question pools are updated by the FCC approximately every 4 years. When a new pool is released, questions can be added, removed, moved between subelements, or have their NCVEC IDs reassigned.

The Problem

Consider this scenario:

- In the 2022-2026 pool, question T5A03 asks about Ohm's Law
- In the 2026-2030 pool, T5A03 is a completely different question about antenna theory
- The original Ohm's Law question moved to T6B07

If we only track by NCVEC ID (T5A03), a user's mastery data becomes meaningless after a pool transition—they'd get 'credit' for a question they've never seen.

Solutions Considered

Option A: Mapping Tables

Maintain explicit mappings between old and new question IDs.

- Pro: Explicit and auditable
- Con: Manual maintenance every 4 years; error-prone
- Con: Doesn't handle questions that are slightly modified

Option B: Content Hashing

Hash the question content (text + answers). Same content = same hash, regardless of NCVEC ID.

- Pro: Automatic—no manual mapping needed
- Pro: Handles moved questions automatically
- Pro: Detects when NCVEC reuses an ID for different content
- Con: Minor edits (typo fixes) create new hashes

Why We Chose Content Hashing

Content hashing is more robust and requires no manual intervention during pool transitions:

Scenario	Mapping Table	Content Hash
Question unchanged, same ID	Works (trivial)	Works (same hash)
Question moved to new ID	Requires manual mapping	Works automatically

NCVEC ID reused for new question	Requires manual mapping	Works (different hash)
Question slightly edited	Works (same mapping)	New hash (loses history)
Brand new question	N/A	N/A

The only downside is that minor edits create new hashes. In practice, this is rare and acceptable—if the FCC meaningfully edits a question, treating it as 'new' is arguably correct.

2.5 Why These Event Types?

We capture five event types. Here's why each exists:

question_attempt

The core event. Captures every question answered.

- Powers: Accuracy calculations, item statistics, mastery tracking
- Frequency: Very high (hundreds per active user per week)
- Why not use existing question_attempts table? Events include richer context (time spent, session ID, mode, content hash)

practice_test_completed

Captures completion of full practice exams.

- Powers: Practice test pass rate metric, readiness validation
- Frequency: Medium (a few per user per week)
- Why separate from question_attempt? Tests have aggregate data (total score, duration, subelement breakdown) that doesn't fit per-question events

session_start / session_end

Brackets study sessions for engagement analysis.

- Powers: Session duration metrics, days since last activity, study pattern analysis
- Frequency: Medium (a few per user per week)
- Why capture this? Understanding how users study (long sessions vs. short bursts) helps improve the product and may correlate with outcomes

exam_outcome_reported

Captures self-reported real exam results.

- Powers: Model calibration, readiness score validation
- Frequency: Low (once per user per exam attempt)
- Why this is critical: Without outcome data, we can't validate that our readiness predictions are accurate. This is the ground truth for model improvement.

The state_snapshot field captures the user's complete readiness state at exam time, enabling correlation analysis between preparation metrics and pass/fail outcomes.

3. Resource Calculations

This section provides detailed napkin math showing how the architecture fits within Supabase Pro plan limits. These calculations help contributors understand resource constraints and make informed decisions about changes.

3.1 Supabase Pro Plan Limits

OpenHamPrep runs on Supabase Pro ($25/month). Here are the relevant limits:

Resource	Included in Pro	Overage Cost	Notes
Database Storage	8 GB	$0.125/GB/month	Includes indexes
Edge Function Invocations	2,000,000/month	$2/million	Per request
Edge Function Compute	100 hours/month	$0.018/hour	Execution time
Bandwidth	250 GB/month	$0.09/GB	Data transfer
Realtime Connections	500 concurrent	—	Not used by events
Database Connections	60 direct + pooled	—	Rarely a limit

The binding constraint for the event system is database storage. Edge function limits and bandwidth are not concerns with our architecture.

3.2 Event Size Analysis

Let's calculate the storage cost of a single question_attempt event:

Payload Size Breakdown

```
{
  "session_id": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",  // 36 chars
  "question_id": "f47ac10b-58cc-4372-a567-0e02b2c3d479", // 36 chars
  "ncvec_id": "T5A03",                                   // 5 chars
  "content_hash": "e3b0c44298fc1c149afbf4c8996fb924...", // 64 chars
  "pool_version": "2022-2026",                           // 9 chars
  "subelement": "T5",                                    // 2 chars
  "group": "T5A",                                        // 3 chars
  "answer_selected": 2,                                  // 1 char
  "correct_answer": 1,                                   // 1 char
  "is_correct": false,                                   // 5 chars
  "time_spent_ms": 34500,                                // 5 chars
  "mode": "drill",                                       // 5 chars
  "practice_test_id": null                               // 4 chars
}

Payload JSON string: ~280 bytes (including keys, quotes, punctuation)
```

Row Overhead

```
Column                  | Type        | Size
------------------------|-------------|----------
id                      | UUID        | 16 bytes
event_type              | TEXT        | ~20 bytes (avg)
timestamp               | TIMESTAMPTZ | 8 bytes
user_id                 | UUID        | 16 bytes
payload                 | JSONB       | ~300 bytes (280 + JSONB overhead)
created_at              | TIMESTAMPTZ | 8 bytes
------------------------|-------------|----------
Row total (no indexes)  |             | ~370 bytes
```

Index Overhead

```
Index                       | Columns                       | Est. Size/Row
----------------------------|-------------------------------|---------------
PRIMARY KEY                 | id                            | ~20 bytes
idx_events_user_time        | user_id, timestamp            | ~30 bytes
idx_events_type_time        | event_type, timestamp         | ~35 bytes
idx_events_question_id      | payload->>'question_id'       | ~50 bytes
idx_events_content_hash     | payload->>'content_hash'      | ~80 bytes
idx_events_pool_version     | payload->>'pool_version'      | ~25 bytes
----------------------------|-------------------------------|---------------
Index overhead total        |                               | ~240 bytes
```

Total Per Event

```
Row data:       ~370 bytes
Index overhead: ~240 bytes
-------------------------
Total:          ~610 bytes per event

Rounded estimate: 600 bytes per event
```

3.3 Storage Projections

Using 600 bytes/event and 300 events/user retention:

Events Table Size

Total Users	Max Events	Calculation	Storage	% of 8 GB
1,000	300,000	300k × 600B	180 MB	2.2%
5,000	1,500,000	1.5M × 600B	900 MB	11%
10,000	3,000,000	3M × 600B	1.8 GB	22%
20,000	6,000,000	6M × 600B	3.6 GB	45%
30,000	9,000,000	9M × 600B	5.4 GB	67%
40,000	12,000,000	12M × 600B	7.2 GB	90%
50,000	15,000,000	15M × 600B	9.0 GB	112% ⚠️

The architecture supports approximately 40,000 users before hitting storage limits. Beyond that, options include reducing retention to 200 events/user or upgrading the plan.

Questions Table Size

The questions table grows slowly and is not a storage concern:

Time Horizon	Pool Versions	Total Questions	Est. Storage
Current	1	~1,600	~5 MB
4 years	2	~3,200	~10 MB
8 years	3	~4,800	~15 MB
16 years	5	~8,000	~25 MB

Each question record is approximately 2-3 KB (including question text, options, explanation). Even after 16 years and 5 pool versions, the questions table would be under 30 MB—negligible compared to events.

Other Tables

The event system adds one new table (question_pools) which is tiny:

- question_pools: ~10 rows × ~200 bytes = ~2 KB

Total storage impact of the event system is dominated by the events table.

3.4 Bandwidth Calculations

Bandwidth is charged for data transferred between Supabase and clients.

Per-Request Size

```
Event Insert Request:
  HTTP headers:        ~400 bytes
  Request body (JSON): ~350 bytes
  Total request:       ~750 bytes

Insert Response:
  HTTP headers:        ~300 bytes
  Response body:       ~100 bytes (success confirmation)
  Total response:      ~400 bytes

Round-trip total: ~1,150 bytes ≈ 1.2 KB per question attempt
```

Monthly Bandwidth by Scale

MAU	Questions/User	Total Requests	Bandwidth	% of 250 GB
1,000	500	500,000	0.6 GB	0.2%
5,000	500	2,500,000	3.0 GB	1.2%
10,000	500	5,000,000	6.0 GB	2.4%
25,000	500	12,500,000	15.0 GB	6.0%
50,000	500	25,000,000	30.0 GB	12.0%

Bandwidth is not a concern. Even at 50,000 MAU, we'd use only 12% of the included bandwidth.

3.5 Edge Function Usage

We use edge functions sparingly—only for operations requiring server-side logic.

Projected Edge Function Calls

Operation	Trigger	Calls/User/Month	At 10k MAU
Question attempts	Every question	0 (PostgREST)	0
Get readiness	Dashboard view (cached)	5	50,000
Report outcome	After real exam	0.05	500
Session tracking	Start/end session	0 (PostgREST)	0

Total: ~50,500 edge function calls/month at 10k MAU. This is 2.5% of the 2M included calls.

Compute Time Estimate

```
Get readiness (50,000 calls × 100ms avg): 5,000 seconds = 1.4 hours
Report outcome (500 calls × 500ms avg):   250 seconds   = 0.07 hours
                                          --------------------------
Total monthly compute:                                    ~1.5 hours

Supabase Pro includes 100 hours → Using 1.5% of limit ✓
```

3.6 Cost Summary

Bringing it all together (including archival storage):

Scale	DB Storage	Archive Storage	Bandwidth	Total Monthly
1,000 users	180 MB	~30 MB/yr	0.6 GB	$25
5,000 users	900 MB	~150 MB/yr	3 GB	$25
10,000 users	1.8 GB	~360 MB/yr	6 GB	$25
25,000 users	4.5 GB	~900 MB/yr	15 GB	$25
40,000 users	7.2 GB	~1.5 GB/yr	24 GB	$25 + ~$0.50

Archive storage costs are negligible: ~$0.02/GB/month. Even after 10 years at 40k users, archives would cost < $1/month while preserving complete history.

Key Findings

- Storage is the binding constraint—we hit 8 GB at ~42,000 users
- Bandwidth and edge functions are well under limits at all scales
- The $25/month Pro plan supports up to ~40,000 users comfortably
- Beyond 40k users: reduce retention to 200 events/user, or pay ~$2-3/month overage

Sensitivity Analysis

If our assumptions are wrong:

If...	Impact	Mitigation
Events are larger (800B)	Hit limit at ~30k users	Reduce retention to 225
Users more active (400 q/mo)	No impact (retention-based)	None needed
Index overhead higher	Hit limit at ~35k users	Drop less-used indexes
Need more history (500 events)	Hit limit at ~26k users	Archive old events to storage

4. Question Identity & Pool Versioning

4.1 The Problem

The FCC updates amateur radio question pools on a 4-year cycle. When a pool changes:

- New questions appear — new UUID records created in questions table
- Questions are removed — old UUID records become historical
- Questions move between subelements (T5A03 becomes T6B07)
- NCVEC IDs may be reused for completely different questions
- Exam structure and subelement weights may change

Without proper versioning, historical data becomes difficult to interpret across pool transitions.

4.2 Question Identifiers

Each question has three identifiers in the OpenHamPrep system:

Identifier	Source	Column	Purpose
UUID	OpenHamPrep	questions.id	Internal primary key, stable for this record
NCVEC ID	NCVEC/FCC	questions.display_name	Human-readable (T5A03), reused across pools
Content Hash	Computed	questions.content_hash	SHA-256 fingerprint of question content

4.3 How Identifiers Work Together

When a new question pool is released:

Scenario	UUID	NCVEC ID	Content Hash
New question	New UUID created	T5A15 (new)	xyz789 (new)
Question unchanged	New UUID created	T5A03 (same)	abc123 (same hash)
Question moved	New UUID created	T6B07 (different)	abc123 (same hash)
NCVEC ID reused	New UUID created	T5A03 (same)	def456 (different hash)
Question edited	New UUID created	T5A03 (same)	abc124 (similar hash)

The content_hash allows tracking the 'same' question across pools even when NCVEC reassigns IDs or moves questions between subelements.

4.4 Content Hash Generation

The content hash is a SHA-256 hash of the normalized question content:

```
// Hash is computed from question content (matching existing schema)
const hashInput = [
  normalize(question.question),           // questions.question
  normalize(question.options[0]),         // options jsonb array
  normalize(question.options[1]),
  normalize(question.options[2]),
  normalize(question.options[3]),
  question.correct_answer.toString()      // 0, 1, 2, or 3
].join('|');

const contentHash = sha256(hashInput);

// Normalization ensures consistent hashing
function normalize(text: string): string {
  return text
    .toLowerCase()
    .trim()
    .replace(/\s+/g, ' ')
    .replace(/['']/g, "'")
    .replace(/[""]/g, '"');
}
```

5. Database Schema

5.1 Questions Table Modifications

Add pool versioning and content hash to the existing questions table:

```
-- Add new columns to existing questions table
ALTER TABLE questions
ADD COLUMN IF NOT EXISTS content_hash TEXT,
ADD COLUMN IF NOT EXISTS pool_version TEXT DEFAULT '2022-2026';

-- Index for cross-pool content lookups
CREATE INDEX IF NOT EXISTS idx_questions_content_hash
  ON questions (content_hash);

-- Index for pool-specific queries
CREATE INDEX IF NOT EXISTS idx_questions_pool_version
  ON questions (pool_version);
```

5.2 Question Pools Reference Table

Track pool versions and their effective dates:

```
CREATE TABLE IF NOT EXISTS question_pools (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  pool_version TEXT NOT NULL,
  exam_type TEXT NOT NULL CHECK (exam_type IN ('technician', 'general', 'extra')),
  effective_date DATE NOT NULL,
  expiration_date DATE,
  question_count INT NOT NULL,
  is_current BOOLEAN DEFAULT false,
  created_at TIMESTAMPTZ DEFAULT NOW(),

  UNIQUE (pool_version, exam_type)
);

-- Example data
INSERT INTO question_pools (pool_version, exam_type, effective_date, question_count,
is_current)
VALUES
  ('2022-2026', 'technician', '2022-07-01', 412, true),
  ('2023-2027', 'general', '2023-07-01', 454, true),
  ('2024-2028', 'extra', '2024-07-01', 693, true);
```

5.3 Events Table

The events table is the core of the event system. It's append-only and stores all user interactions.

```
CREATE TABLE IF NOT EXISTS events (
  -- Primary key
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
```

```
  -- Event classification
  event_type TEXT NOT NULL,

  -- When it happened
  timestamp TIMESTAMPTZ NOT NULL DEFAULT NOW(),

  -- Who did it (references auth.users, same as profiles.id)
  user_id UUID NOT NULL REFERENCES auth.users(id),

  -- Full event data (flexible schema)
  payload JSONB NOT NULL,

  -- System metadata
  created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

-- Indexes for primary access patterns
CREATE INDEX idx_events_user_time
  ON events (user_id, timestamp DESC);

CREATE INDEX idx_events_type_time
  ON events (event_type, timestamp DESC);

-- Question-specific indexes
CREATE INDEX idx_events_question_id
  ON events ((payload->>'question_id'))
  WHERE event_type = 'question_attempt';

CREATE INDEX idx_events_content_hash
  ON events ((payload->>'content_hash'))
  WHERE event_type = 'question_attempt';

CREATE INDEX idx_events_pool_version
  ON events ((payload->>'pool_version'))
  WHERE event_type = 'question_attempt';
```

5.4 Row-Level Security

Users can only insert and read their own events:

```
ALTER TABLE events ENABLE ROW LEVEL SECURITY;

-- Users can insert their own events
CREATE POLICY "Users insert own events" ON events
  FOR INSERT WITH CHECK (auth.uid() = user_id);

-- Users can read their own events
CREATE POLICY "Users read own events" ON events
  FOR SELECT USING (auth.uid() = user_id);

-- Question pools readable by all authenticated users
ALTER TABLE question_pools ENABLE ROW LEVEL SECURITY;

CREATE POLICY "Authenticated read pools" ON question_pools
  FOR SELECT USING (auth.role() = 'authenticated');
```

6. Event Types

6.1 question_attempt

Captured every time a user answers a question. This is the most frequent event type.

Field	Type	Description
session_id	UUID	Groups attempts within a study session
question_id	UUID	Foreign key to questions.id
ncvec_id	String	NCVEC identifier (questions.display_name, e.g., 'T5A03')
content_hash	String	SHA-256 hash of question content
pool_version	String	Question pool version (e.g., '2022-2026')
subelement	String	Topic area (e.g., 'T5')
group	String	Question group (e.g., 'T5A')
answer_selected	Integer	User's answer (0, 1, 2, or 3)
correct_answer	Integer	Correct answer (0, 1, 2, or 3)
is_correct	Boolean	Whether user was correct
time_spent_ms	Integer	Milliseconds spent on question
mode	String	Context: 'drill', 'practice_test', 'review'
practice_test_id	UUID|null	If part of a practice test

```
// Example payload
{
  "session_id": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
  "question_id": "f47ac10b-58cc-4372-a567-0e02b2c3d479",
  "ncvec_id": "T5A03",
  "content_hash": "e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855",
  "pool_version": "2022-2026",
  "subelement": "T5",
  "group": "T5A",
  "answer_selected": 2,
  "correct_answer": 1,
  "is_correct": false,
  "time_spent_ms": 34500,
  "mode": "drill",
  "practice_test_id": null
}
```

6.2 practice_test_completed

Captured when a user finishes a full practice exam. Complements existing practice_test_results table.

Field	Type	Description
practice_test_id	UUID	Unique identifier for this test
test_result_id	UUID|null	FK to practice_test_results.id if exists
exam_type	String	'technician', 'general', or 'extra'
pool_version	String	Which question pool was used
score	Integer	Number answered correctly
total_questions	Integer	Total questions (35 or 50)
percentage	Number	Score as percentage
passed	Boolean	Score >= 74%
duration_seconds	Integer	Time to complete
subelement_breakdown	Object	Per-subelement results

6.3 session_start

Captured when a user begins a study session.

Field	Type	Description
session_id	UUID	Unique session identifier
platform	String	'web', 'ios', or 'android'
app_version	String	Client app version
exam_type	String	Which exam they're studying for
pool_version	String	Current active pool version
days_since_last_session	Integer|null	Gap since previous session

6.4 session_end

Captured when a study session ends (explicit logout, timeout, or app close).

Field	Type	Description
session_id	UUID	Matches session_start
duration_seconds	Integer	Total session length
questions_attempted	Integer	Questions answered this session
end_reason	String	'explicit', 'timeout', 'app_close'

6.5 exam_outcome_reported

Captured when a user reports their real exam result. Most valuable for model calibration.

Field	Type	Description
exam_type	String	'technician', 'general', or 'extra'
pool_version	String	Which pool the exam used
passed	Boolean	Whether they passed
attempt_number	Integer	Which attempt (1st, 2nd, etc.)
exam_date	Date	When they took the exam
self_reported_score	Integer|null	Their score if shared
confidence_level	String|null	How confident they felt

state_snapshot	Object	Full user state at exam time

7. Client Integration

7.1 Configuration

Store current pool version in a central config:

```
// lib/config.ts

export const POOL_CONFIG = {
  technician: {
    currentVersion: '2022-2026',
    effectiveDate: '2022-07-01',
  },
  general: {
    currentVersion: '2023-2027',
    effectiveDate: '2023-07-01',
  },
  extra: {
    currentVersion: '2024-2028',
    effectiveDate: '2024-07-01',
  }
} as const;

export type ExamType = keyof typeof POOL_CONFIG;

export function getCurrentPoolVersion(examType: ExamType): string {
  return POOL_CONFIG[examType].currentVersion;
}
```

7.2 Core Event Recording Function

Base function for inserting events:

```
// lib/events.ts
import { supabase } from './supabase';

type EventType =
  | 'question_attempt'
  | 'practice_test_completed'
  | 'session_start'
  | 'session_end'
  | 'exam_outcome_reported';

interface RecordEventParams {
  eventType: EventType;
  payload: Record<string, unknown>;
}

export async function recordEvent({ eventType, payload }: RecordEventParams) {
  const { data: { user } } = await supabase.auth.getUser();

  if (!user) {
    console.warn('Cannot record event: user not authenticated');
    return { error: 'Not authenticated' };
  }
```

```
  const { error } = await supabase
    .from('events')
    .insert({
      event_type: eventType,
      user_id: user.id,
      payload
    });

  if (error) {
    console.error('Failed to record event:', error);
    return { error: error.message };
  }

  return { success: true };
}
```

7.3 Question Attempt Helper

Records question attempts with full identity tracking. The question object should include content_hash and pool_version from the questions table.

```
// lib/events.ts (continued)

// Question type matching your existing schema
interface Question {
  id: string;                    // UUID
  display_name: string;          // NCVEC ID (T5A03)
  question: string;
  options: string[];
  correct_answer: number;        // 0, 1, 2, or 3
  subelement: string;
  question_group: string;
  content_hash: string;          // Added field
  pool_version: string;          // Added field
}

interface QuestionAttemptParams {
  sessionId: string;
  question: Question;
  answerSelected: number;        // 0, 1, 2, or 3
  timeSpentMs: number;
  mode: 'drill' | 'practice_test' | 'review';
  practiceTestId?: string;
}

export async function recordQuestionAttempt(params: QuestionAttemptParams) {
  const { question } = params;
  const isCorrect = params.answerSelected === question.correct_answer;

  return recordEvent({
    eventType: 'question_attempt',
    payload: {
      session_id: params.sessionId,

      // Primary reference (your internal UUID)
      question_id: question.id,

      // Human-readable NCVEC identifier
      ncvec_id: question.display_name,

      // Content identity for cross-pool tracking
      content_hash: question.content_hash,
      pool_version: question.pool_version,

      // Location in current pool
      subelement: question.subelement,
      group: question.question_group,

      // Response data
      answer_selected: params.answerSelected,
      correct_answer: question.correct_answer,
      is_correct: isCorrect,
      time_spent_ms: params.timeSpentMs,
      mode: params.mode,
      practice_test_id: params.practiceTestId || null
    }
  });
```

```
}
```

7.4 Usage Example

How events integrate into the existing question flow:

```
// components/QuestionCard.tsx
import { useState } from 'react';
import { recordQuestionAttempt } from '../lib/events';

export function QuestionCard({ question, sessionId, onNext }) {
  const [startTime] = useState(Date.now());
  const [selectedAnswer, setSelectedAnswer] = useState<number | null>(null);

  const handleSubmit = async () => {
    if (selectedAnswer === null) return;

    const timeSpentMs = Date.now() - startTime;

    // Record event (fire and forget)
    recordQuestionAttempt({
      sessionId,
      question,  // Pass full question object with content_hash, pool_version
      answerSelected: selectedAnswer,
      timeSpentMs,
      mode: 'drill'
    });

    // Existing logic continues...
    const isCorrect = selectedAnswer === question.correct_answer;
    onNext(isCorrect);
  };

  return (
    // ... existing render logic
  );
}
```

8. Data Retention

8.1 Retention Strategy

To keep database size within Supabase Pro limits (~8 GB), old events are archived to Supabase Storage rather than deleted. This gives us the best of both worlds:

- Fast queries: Only recent events (300 per user) in the database
- Complete history: All events preserved in cold storage forever
- Cost effective: Storage is 6x cheaper than database ($0.021/GB vs $0.125/GB)
- Future flexibility: Can reload archived events for deep analysis anytime

Retention Parameters

Parameter	Value	Rationale
Events per user (hot)	300	~3-4 weeks active study; sufficient for all real-time calculations
Minimum age before archive	7 days	Never archive events less than a week old
Archive frequency	Weekly	Sunday 4 AM UTC via pg_cron + edge function
Archive format	NDJSON (gzipped)	One JSON object per line; easy to process
Archive location	Supabase Storage	event-archive bucket, organized by date
Database target size	< 7 GB	Leave headroom under 8 GB limit
Archive retention	Forever	Storage is cheap; keep everything

8.2 Archive vs. Delete: Cost Comparison

Why archive instead of delete? Let's compare the costs:

Approach	Data Kept	Monthly Cost at 50k Users	Can Recover Old Data?
Delete old events	300/user in DB only	$25 (base plan)	No - gone forever
Archive to Storage	300/user in DB + all in Storage	$25 + ~$2 storage	Yes - reload anytime

For approximately $2/month extra, we retain complete history. This is valuable for:

- Training ML models on historical patterns
- Analyzing long-term user behavior
- Debugging issues that span months
- Research into learning patterns over time

- Regulatory compliance (if ever needed)

8.3 Archive Architecture

The archival system has three components:

1. Storage Bucket Structure

```
event-archive/
├── 2026/
│   ├── 01/
│   │   ├── 2026-01-05-events.ndjson.gz    # Week of Jan 5
│   │   ├── 2026-01-12-events.ndjson.gz    # Week of Jan 12
│   │   └── 2026-01-19-events.ndjson.gz    # Week of Jan 19
│   ├── 02/
│   │   └── ...
│   └── ...
└── manifests/
    └── archive-manifest.json               # Index of all archives
```

Each archive file contains all events archived that week, one JSON object per line (NDJSON format), gzip compressed.

2. Archive Job (Edge Function)

A weekly edge function handles the archive process:

```
// supabase/functions/archive-events/index.ts

import { createClient } from '@supabase/supabase-js';
import { gzip } from 'pako';

const EVENTS_TO_KEEP = 300;
const MIN_AGE_DAYS = 7;

Deno.serve(async (req) => {
  const supabase = createClient(
    Deno.env.get('SUPABASE_URL')!,
    Deno.env.get('SUPABASE_SERVICE_ROLE_KEY')!
  );

  // 1. Find events to archive (beyond 300 per user, older than 7 days)
  const { data: eventsToArchive, error: selectError } = await supabase.rpc(
    'get_events_to_archive',
    { keep_count: EVENTS_TO_KEEP, min_age_days: MIN_AGE_DAYS }
  );

  if (selectError || !eventsToArchive?.length) {
    return new Response(JSON.stringify({
      archived: 0,
      message: selectError?.message || 'No events to archive'
    }));
  }
```

```
  // 2. Convert to NDJSON and compress
  const ndjson = eventsToArchive
    .map((event: any) => JSON.stringify(event))
    .join('\n');
  const compressed = gzip(new TextEncoder().encode(ndjson));

  // 3. Upload to storage
  const date = new Date();
  const year = date.getFullYear();
  const month = String(date.getMonth() + 1).padStart(2, '0');
  const day = String(date.getDate()).padStart(2, '0');
  const filename = `${year}/${month}/${year}-${month}-${day}-events.ndjson.gz`;

  const { error: uploadError } = await supabase.storage
    .from('event-archive')
    .upload(filename, compressed, {
      contentType: 'application/gzip',
      upsert: true
    });

  if (uploadError) {
    return new Response(JSON.stringify({ error: uploadError.message }), {
      status: 500
    });
  }

  // 4. Delete archived events from database
  const archivedIds = eventsToArchive.map((e: any) => e.id);
  const { error: deleteError } = await supabase
    .from('events')
    .delete()
    .in('id', archivedIds);

  if (deleteError) {
    return new Response(JSON.stringify({ error: deleteError.message }), {
      status: 500
    });
  }

  // 5. Update manifest
  await updateManifest(supabase, filename, eventsToArchive.length);

  return new Response(JSON.stringify({
    archived: eventsToArchive.length,
    filename,
    compressedSize: compressed.length
  }));
});
```

3. Database Function to Find Archivable Events

```
-- Function to get events that should be archived
CREATE OR REPLACE FUNCTION get_events_to_archive(
  keep_count INTEGER DEFAULT 300,
  min_age_days INTEGER DEFAULT 7
)
RETURNS SETOF events AS $$
BEGIN
  RETURN QUERY
  WITH ranked_events AS (
    SELECT
      e.*,
```

```
      ROW_NUMBER() OVER (
        PARTITION BY e.user_id
        ORDER BY e.timestamp DESC
      ) as rn
    FROM events e
  )
  SELECT id, event_type, timestamp, user_id, payload, created_at
  FROM ranked_events
  WHERE rn > keep_count
    AND timestamp < NOW() - (min_age_days || ' days')::INTERVAL
  ORDER BY timestamp ASC
  LIMIT 50000;  -- Process in batches
END;
$$ LANGUAGE plpgsql SECURITY DEFINER;
```

8.4 Scheduling the Archive Job

Use pg_cron to trigger the edge function weekly:

```
-- Schedule: 4:00 AM UTC every Sunday
-- This calls the edge function via pg_net extension

SELECT cron.schedule(
  'archive-old-events',
  '0 4 * * 0',
  $$
  SELECT net.http_post(
    url := current_setting('app.settings.supabase_url') ||
'/functions/v1/archive-events',
    headers := jsonb_build_object(
      'Authorization', 'Bearer ' || current_setting('app.settings.service_role_key'),
      'Content-Type', 'application/json'
    ),
    body := '{}'::jsonb
  );
  $$
);

-- Alternative: Use Supabase's built-in cron if pg_net isn't available
-- Configure in supabase/config.toml or dashboard
```

8.5 Retrieving Archived Events

When you need to analyze historical data:

Option A: Download and Process Locally

```
// scripts/analyze-archived-events.ts

import { createClient } from '@supabase/supabase-js';
import { gunzip } from 'pako';
import * as readline from 'readline';

async function loadArchivedEvents(startDate: string, endDate: string) {
  const supabase = createClient(SUPABASE_URL, SUPABASE_SERVICE_KEY);
```

```
  // List archive files in date range
  const { data: files } = await supabase.storage
    .from('event-archive')
    .list('2026/01', { limit: 100 });

  const allEvents = [];

  for (const file of files || []) {
    // Download and decompress
    const { data } = await supabase.storage
      .from('event-archive')
      .download(`2026/01/${file.name}`);

    const compressed = await data?.arrayBuffer();
    const decompressed = gunzip(new Uint8Array(compressed!));
    const ndjson = new TextDecoder().decode(decompressed);

    // Parse NDJSON (one event per line)
    const events = ndjson.split('\n')
      .filter(line => line.trim())
      .map(line => JSON.parse(line));

    allEvents.push(...events);
  }

  return allEvents;
}
```

Option B: Reload into Temporary Table for SQL Analysis

```
-- Create temporary table for analysis
CREATE TEMP TABLE archived_events (LIKE events);

-- Load from application after downloading from storage
-- (Application parses NDJSON and inserts)

-- Now you can query across both current and archived events
SELECT
  DATE_TRUNC('month', timestamp) as month,
  COUNT(*) as attempts,
  AVG(CASE WHEN (payload->>'is_correct')::boolean THEN 1 ELSE 0 END) as accuracy
FROM (
  SELECT * FROM events
  UNION ALL
  SELECT * FROM archived_events
) all_events
WHERE user_id = $1
GROUP BY DATE_TRUNC('month', timestamp)
ORDER BY month;
```

8.6 Storage Bucket Setup

Create the archive bucket with appropriate permissions:

```
-- Storage bucket (create via Supabase dashboard or SQL)
INSERT INTO storage.buckets (id, name, public)
VALUES ('event-archive', 'event-archive', false);
```

```
-- RLS policy: Only service role can read/write
-- (No user access - this is backend-only storage)
CREATE POLICY "Service role only" ON storage.objects
  FOR ALL USING (bucket_id = 'event-archive' AND auth.role() = 'service_role');
```

8.7 Archive Storage Costs

Estimating long-term storage costs:

Timeframe	Events Archived	Raw Size	Compressed (~10:1)	Monthly Cost
1 month	~500k	300 MB	~30 MB	$0.001
1 year	~6M	3.6 GB	~360 MB	$0.008
5 years	~30M	18 GB	~1.8 GB	$0.04
10 years	~60M	36 GB	~3.6 GB	$0.08

At 10,000 MAU generating 500k events/month, 10 years of complete history costs less than $1/month in storage. This is negligible compared to the value of having complete historical data.

8.8 Questions Table Retention

The questions table is NOT archived or pruned. All pool versions are retained in the database permanently to support:

- Historical queries against archived events (question_id FK can be joined)
- Cross-pool question tracking via content_hash
- Audit trail of pool changes

The questions table grows slowly (~500 questions × ~4 pools = ~2000 rows over 16 years) and is negligible in size.

8.9 Capacity Planning

With archival, the database size stays bounded regardless of user count or time:

Total Users	Events in DB	DB Size	Archived (Year 1)	Archive Size
5,000	1.5M	~0.9 GB	~6M	~360 MB
10,000	3M	~1.8 GB	~12M	~720 MB
25,000	7.5M	~4.5 GB	~30M	~1.8 GB

40,000	12M	~7.2 GB	~48M	~2.9 GB
50,000	15M	~9 GB ⚠️	~60M	~3.6 GB

At 40,000 users, we stay within the 8 GB database limit while preserving complete history in storage. Beyond 40k users, reduce the hot retention from 300 to 200-250 events per user.

9. Querying Events

9.1 Access Patterns

Pattern	Use Case	Index Used
User's recent events	Readiness computation	idx_events_user_time
Events by question UUID	Per-question analysis	idx_events_question_id
Events by content hash	Cross-pool tracking	idx_events_content_hash
Events by pool version	Pool-specific analysis	idx_events_pool_version

9.2 Example Queries

User's accuracy in current pool

```
SELECT
  COUNT(*) as attempts,
  SUM(CASE WHEN (payload->>'is_correct')::boolean THEN 1 ELSE 0 END) as correct,
  AVG(CASE WHEN (payload->>'is_correct')::boolean THEN 1 ELSE 0 END) as accuracy
FROM events
WHERE user_id = $1
  AND event_type = 'question_attempt'
  AND payload->>'pool_version' = '2022-2026';
```

User's mastery of a question across all pools (using content_hash)

```
-- Find if user has seen this "same" question in any pool version
WITH target_hash AS (
  SELECT content_hash FROM questions WHERE id = $2
)
SELECT
  COUNT(*) as times_seen,
  SUM(CASE WHEN (payload->>'is_correct')::boolean THEN 1 ELSE 0 END) as times_correct
FROM events, target_hash
WHERE user_id = $1
  AND event_type = 'question_attempt'
  AND payload->>'content_hash' = target_hash.content_hash;
```

Questions that moved between pools

```
SELECT
  q1.display_name as old_ncvec_id,
  q1.subelement as old_subelement,
  q2.display_name as new_ncvec_id,
  q2.subelement as new_subelement
FROM questions q1
JOIN questions q2 ON q1.content_hash = q2.content_hash
WHERE q1.pool_version = '2022-2026'
  AND q2.pool_version = '2026-2030'
  AND (q1.display_name != q2.display_name
       OR q1.subelement != q2.subelement);
```

10. Migration Plan

10.1 Overview

The event system runs in parallel with the existing system during migration:

Phase	Duration	Description
1. Schema Updates	1 day	Add columns to questions, create events + question_pools tables
2. Backfill Hashes	1 day	Compute content_hash for existing questions
3. Dual Write	2 weeks	Capture events alongside existing question_attempts
4. Backfill Events	1 week	Migrate recent question_attempts to events
5. Validate	1 week	Verify data integrity
6. Enable Pruning	1 day	Start pg_cron job

10.2 Phase 1: Schema Updates

```
-- 1. Add columns to existing questions table
ALTER TABLE questions
ADD COLUMN IF NOT EXISTS content_hash TEXT,
ADD COLUMN IF NOT EXISTS pool_version TEXT DEFAULT '2022-2026';

-- 2. Create question_pools table
CREATE TABLE question_pools ( ... );

-- 3. Create events table
CREATE TABLE events ( ... );

-- 4. Create indexes and RLS policies
-- (See Appendix A for complete SQL)
```

10.3 Phase 4: Backfill Events from question_attempts

```
-- Migrate recent question_attempts to events
INSERT INTO events (event_type, timestamp, user_id, payload)
SELECT
  'question_attempt',
  qa.attempted_at,
  qa.user_id,
  jsonb_build_object(
    'question_id', qa.question_id,
    'ncvec_id', q.display_name,
    'content_hash', q.content_hash,
    'pool_version', COALESCE(q.pool_version, '2022-2026'),
    'subelement', q.subelement,
    'group', q.question_group,
    'answer_selected', qa.selected_answer,
    'correct_answer', q.correct_answer,
    'is_correct', qa.is_correct,
```

```
    'mode', 'backfill',
    'time_spent_ms', null,
    'session_id', null
  )
FROM question_attempts qa
JOIN questions q ON q.id = qa.question_id
WHERE qa.attempted_at > NOW() - INTERVAL '60 days'
ORDER BY qa.attempted_at
LIMIT 50000;
```

11. Pool Transitions

11.1 When a New Pool is Released

FCC releases new question pools approximately every 4 years:

7. Download the new pool from NCVEC
8. Run import script to create new question records with new UUIDs
9. Content hashes are computed automatically during import
10. Add new pool metadata to question_pools table
11. Update POOL_CONFIG in client code
12. Deploy client update

11.2 Key Points

- New pool = new question records with new UUIDs
- Old questions remain in database (historical FK references stay valid)
- content_hash links 'same' questions across pools
- Events from both pools coexist; filter by pool_version as needed

11.3 User Experience

During a pool transition:

- Users studying the old pool continue normally until cutoff date
- After cutoff, app switches to new pool_version
- Mastery of unchanged questions (same content_hash) can optionally carry over
- Coverage metrics reset (new UUIDs = new questions to discover)

12. Monitoring

12.1 Health Check Query

```
SELECT
  -- Event counts
  (SELECT COUNT(*) FROM events) as total_events,
  (SELECT COUNT(*) FROM events
   WHERE timestamp > NOW() - INTERVAL '24 hours') as events_24h,

  -- Active users
  (SELECT COUNT(DISTINCT user_id) FROM events
   WHERE timestamp > NOW() - INTERVAL '7 days') as active_users_7d,

  -- Storage
  (SELECT pg_size_pretty(pg_total_relation_size('events'))) as events_size,
  (SELECT pg_size_pretty(pg_total_relation_size('questions'))) as questions_size,

  -- Questions with hashes
  (SELECT COUNT(*) FROM questions WHERE content_hash IS NOT NULL) as
questions_with_hash,
  (SELECT COUNT(*) FROM questions WHERE content_hash IS NULL) as
questions_missing_hash,

  -- Pool distribution in recent events
  (SELECT jsonb_object_agg(pool_version, cnt) FROM (
    SELECT payload->>'pool_version' as pool_version, COUNT(*) as cnt
    FROM events
    WHERE event_type = 'question_attempt'
      AND timestamp > NOW() - INTERVAL '7 days'
    GROUP BY payload->>'pool_version'
  ) t) as events_by_pool_7d;
```

12.2 Alerts

Condition	Threshold	Action
Events table size	> 6 GB	Review pruning settings
Missing content_hash in events	Any	Fix client code
Missing content_hash in questions	Any	Run backfill script
Event volume drop	< 50% daily avg	Check client integration
Unknown pool_version	Any	Check configuration

Appendix A: Complete Schema SQL

Run this in Supabase SQL editor to set up the complete event system:

```
-- ============================================
-- OpenHamPrep Event System Schema
-- Version 1.4 (aligned with existing database)
-- ============================================

-- 1. UPDATE EXISTING QUESTIONS TABLE
ALTER TABLE questions
ADD COLUMN IF NOT EXISTS content_hash TEXT,
ADD COLUMN IF NOT EXISTS pool_version TEXT DEFAULT '2022-2026';

CREATE INDEX IF NOT EXISTS idx_questions_content_hash
  ON questions (content_hash);
CREATE INDEX IF NOT EXISTS idx_questions_pool_version
  ON questions (pool_version);

-- 2. QUESTION POOLS TABLE
CREATE TABLE IF NOT EXISTS question_pools (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  pool_version TEXT NOT NULL,
  exam_type TEXT NOT NULL CHECK (exam_type IN ('technician', 'general', 'extra')),
  effective_date DATE NOT NULL,
  expiration_date DATE,
  question_count INT NOT NULL,
  is_current BOOLEAN DEFAULT false,
  created_at TIMESTAMPTZ DEFAULT NOW(),
  UNIQUE (pool_version, exam_type)
);

ALTER TABLE question_pools ENABLE ROW LEVEL SECURITY;
CREATE POLICY "Authenticated read pools" ON question_pools
  FOR SELECT USING (auth.role() = 'authenticated');

-- 3. EVENTS TABLE
CREATE TABLE IF NOT EXISTS events (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  event_type TEXT NOT NULL,
  timestamp TIMESTAMPTZ NOT NULL DEFAULT NOW(),
  user_id UUID NOT NULL REFERENCES auth.users(id),
  payload JSONB NOT NULL,
  created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

CREATE INDEX idx_events_user_time ON events (user_id, timestamp DESC);
CREATE INDEX idx_events_type_time ON events (event_type, timestamp DESC);
CREATE INDEX idx_events_question_id ON events ((payload->>'question_id'))
  WHERE event_type = 'question_attempt';
CREATE INDEX idx_events_content_hash ON events ((payload->>'content_hash'))
  WHERE event_type = 'question_attempt';
CREATE INDEX idx_events_pool_version ON events ((payload->>'pool_version'))
  WHERE event_type = 'question_attempt';

ALTER TABLE events ENABLE ROW LEVEL SECURITY;
CREATE POLICY "Users insert own events" ON events
  FOR INSERT WITH CHECK (auth.uid() = user_id);
CREATE POLICY "Users read own events" ON events
  FOR SELECT USING (auth.uid() = user_id);
```

```
-- 4. ARCHIVE SUPPORT

-- Function to get events that should be archived
CREATE OR REPLACE FUNCTION get_events_to_archive(
  keep_count INTEGER DEFAULT 300,
  min_age_days INTEGER DEFAULT 7
)
RETURNS SETOF events AS $$
BEGIN
  RETURN QUERY
  WITH ranked_events AS (
    SELECT
      e.*,
      ROW_NUMBER() OVER (
        PARTITION BY e.user_id
        ORDER BY e.timestamp DESC
      ) as rn
    FROM events e
  )
  SELECT id, event_type, timestamp, user_id, payload, created_at
  FROM ranked_events
  WHERE rn > keep_count
    AND timestamp < NOW() - (min_age_days || ' days')::INTERVAL
  ORDER BY timestamp ASC
  LIMIT 50000;
END;
$$ LANGUAGE plpgsql SECURITY DEFINER;

-- Storage bucket for archives (run in dashboard or via CLI)
-- INSERT INTO storage.buckets (id, name, public)
-- VALUES ('event-archive', 'event-archive', false);

-- Schedule archive job (requires pg_cron and pg_net extensions)
-- The edge function handles: select events -> compress -> upload -> delete
SELECT cron.schedule(
  'archive-old-events',
  '0 4 * * 0',
  $$
  SELECT net.http_post(
    url := current_setting('app.settings.supabase_url') ||
'/functions/v1/archive-events',
    headers := jsonb_build_object(
      'Authorization', 'Bearer ' || current_setting('app.settings.service_role_key'),
      'Content-Type', 'application/json'
    ),
    body := '{}'::jsonb
  );
  $$
);

-- 5. INITIAL POOL DATA
INSERT INTO question_pools (pool_version, exam_type, effective_date, question_count,
is_current)
VALUES
  ('2022-2026', 'technician', '2022-07-01', 412, true),
  ('2023-2027', 'general', '2023-07-01', 454, true),
  ('2024-2028', 'extra', '2024-07-01', 693, true)
ON CONFLICT (pool_version, exam_type) DO NOTHING;
```

Appendix B: Existing Tables Reference

Key existing tables that the event system integrates with:

Table	Primary Key	Event System Relationship
questions	id (UUID)	Events store question_id in payload
profiles	id (UUID)	Events store user_id (same as auth.users.id)
question_attempts	id (UUID)	Events capture similar + richer data
question_mastery	id (UUID)	Downstream; can be rebuilt from events
practice_test_results	id (UUID)	Events link via test_result_id
user_readiness_cache	id (UUID)	Downstream; computed from events

Appendix C: Event Type Quick Reference

Event Type	Frequency	Key Payload Fields
question_attempt	Very High	question_id, ncvec_id, content_hash, is_correct
practice_test_completed	Medium	score, total_questions, passed, subelement_breakdown
session_start	Medium	session_id, platform, exam_type, pool_version
session_end	Medium	session_id, duration_seconds, questions_attempted
exam_outcome_reported	Low	passed, pool_version, state_snapshot