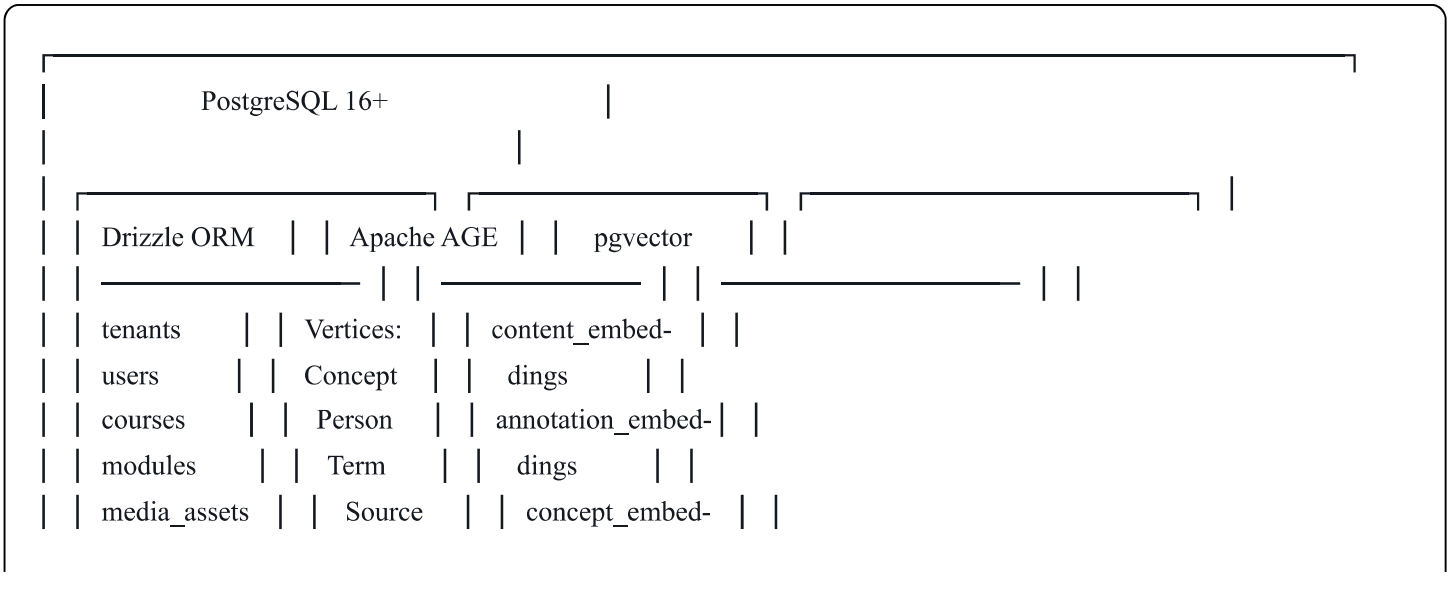
EduSphere Database Schema Definition

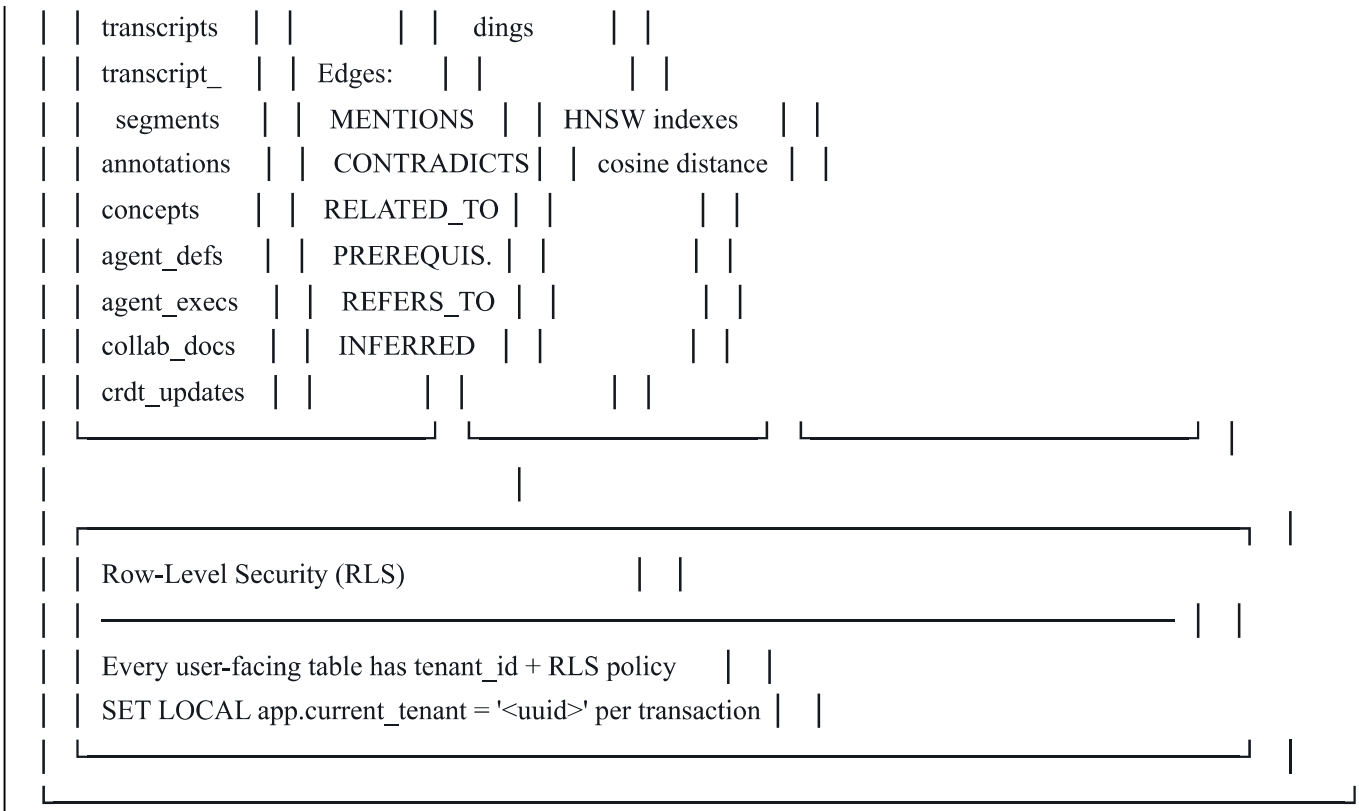
> **Reference document for Claude Code and all developers.** Every table, graph node, edge type, vector index, and RLS policy is defined here. Do NOT deviate from these definitions without updating this document first.

Table of Contents

---

1. Architecture Overview





Key Design Decisions:

- `uuid` for all primary keys (globally unique, no tenant collision)
- `tenant_id` on every user-facing table (RLS enforced)
- Timestamps use `timestamp with time zone` (always UTC)
- Soft deletes via `deleted_at` column (never hard delete user data)
- `jsonb` for flexible/schemaless fields (syllabus, config, metadata)
- Apache AGE graph name: `edusphere_graph` (one graph per database)
- pgvector dimensions: 768 (nomic-embed-text) — configurable via constant

---

2. PostgreSQL Extensions Setup

```sql
```

```sql
-- File: packages/db/src/migrations/0000_extensions.sql
-- Run ONCE as superuser before Drizzle migrations

-- Required extensions
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";      -- UUID generation
CREATE EXTENSION IF NOT EXISTS "pgcrypto";       -- Crypto functions
CREATE EXTENSION IF NOT EXISTS "age";            -- Apache AGE graph
CREATE EXTENSION IF NOT EXISTS "vector";         -- pgvector

-- Load AGE into search path (required per-connection)
-- This is handled in the connection pool initialization
LOAD 'age';
SET search_path = ag_catalog, "$user", public;

-- Create the knowledge graph
SELECT create_graph('edusphere_graph');

-- Create application role for RLS
DO $$
BEGIN
  IF NOT EXISTS (SELECT FROM pg_roles WHERE rolname = 'edusphere_app') THEN
    CREATE ROLE edusphere_app NOLOGIN;
  END IF;
END
$$;

-- Grant necessary permissions
GRANT USAGE ON SCHEMA ag_catalog TO edusphere_app;
GRANT SELECT ON ALL TABLES IN SCHEMA ag_catalog TO edusphere_app;
```

Custom PostgreSQL Docker Image (`infrastructure/docker/postgres-age/Dockerfile`):

dockerfile

```
FROM postgres:16-bookworm

# Install build dependencies
RUN apt-get update && apt-get install -y \
    build-essential libreadline-dev zlib1g-dev flex bison \
    git postgresql-server-dev-16 \
    && rm -rf /var/lib/apt/lists/*

# Install Apache AGE
RUN git clone --branch release/PG16/1.5.0 https://github.com/apache/age.git /tmp/age \
    && cd /tmp/age && make install && rm -rf /tmp/age

# Install pgvector
RUN git clone --branch v0.8.0 https://github.com/pgvector/pgvector.git /tmp/pgvector \
    && cd /tmp/pgvector && make install && rm -rf /tmp/pgvector

# Configure shared_preload_libraries
RUN echo "shared_preload_libraries = 'age'" >> /usr/share/postgresql/postgresql.conf.sample
```

---

3. Shared Definitions

typescript

```ts
// File: packages/db/src/schema/_shared.ts

import { sql } from 'drizzle-orm';
import {
  uuid,
  timestamp,
  pgRole,
  pgEnum,
} from 'drizzle-orm/pg-core';

// ─────────────────────────────────────────────────────────────
// Application Role (for RLS policies)
// ─────────────────────────────────────────────────────────────
export const appRole = pgRole('edusphere_app').existing();

// ─────────────────────────────────────────────────────────────
// Enums
// ─────────────────────────────────────────────────────────────

export const userRoleValues = ['super_admin', 'org_admin', 'instructor', 'student', 'researcher'] as const;
export type UserRole = typeof userRoleValues[number];
export const userRoleEnum = pgEnum('user_role', userRoleValues);

export const mediaTypeValues = ['video', 'audio', 'pdf', 'image', 'document'] as const;
export type MediaType = typeof mediaTypeValues[number];
export const mediaTypeEnum = pgEnum('media_type', mediaTypeValues);

export const transcriptionStatusValues = ['pending', 'processing', 'completed', 'failed'] as const;
export type TranscriptionStatus = typeof transcriptionStatusValues[number];
export const transcriptionStatusEnum = pgEnum('transcription_status', transcriptionStatusValues);

export const annotationTypeValues = ['text', 'sketch', 'link', 'bookmark', 'spatial_comment'] as const;
export type AnnotationType = typeof annotationTypeValues[number];
export const annotationTypeEnum = pgEnum('annotation_type', annotationTypeValues);

export const annotationLayerValues = ['personal', 'shared', 'instructor', 'ai_generated'] as const;
export type AnnotationLayer = typeof annotationLayerValues[number];
export const annotationLayerEnum = pgEnum('annotation_layer', annotationLayerValues);

export const agentTemplateValues = ['chavruta', 'summarizer', 'quiz_master', 'research_scout', 'explainer', 'custom'] as const;
export type AgentTemplate = typeof agentTemplateValues[number];
export const agentTemplateEnum = pgEnum('agent_template', agentTemplateValues);

export const agentTriggerValues = ['manual', 'on_annotation', 'on_new_content', 'scheduled'] as const;
export type AgentTrigger = typeof agentTriggerValues[number];
export const agentTriggerEnum = pgEnum('agent_trigger', agentTriggerValues);
```

```ts
export const agentOutputFormatValues = ['chat', 'overlay', 'side_panel', 'notification'] as const;
export type AgentOutputFormat = typeof agentOutputFormatValues[number];
export const agentOutputFormatEnum = pgEnum('agent_output_format', agentOutputFormatValues);

export const executionStatusValues = ['queued', 'running', 'completed', 'failed', 'cancelled'] as const;
export type ExecutionStatus = typeof executionStatusValues[number];
export const executionStatusEnum = pgEnum('execution_status', executionStatusValues);

export const tenantPlanValues = ['free', 'starter', 'professional', 'enterprise'] as const;
export type TenantPlan = typeof tenantPlanValues[number];
export const tenantPlanEnum = pgEnum('tenant_plan', tenantPlanValues);

// ─────────────────────────────────────────────────────────────
// Shared Column Helpers
// ─────────────────────────────────────────────────────────────

/** Standard primary key: UUID v4, auto-generated */
export const pk = () => uuid('id').primaryKey().defaultRandom();

/** Tenant isolation column — REQUIRED on all user-facing tables */
export const tenantId = () => uuid('tenant_id').notNull();

/** Standard timestamps */
export const timestamps = () => ({
  createdAt: timestamp('created_at', { withTimezone: true })
    .notNull()
    .defaultNow(),
  updatedAt: timestamp('updated_at', { withTimezone: true })
    .notNull()
    .defaultNow()
    .$onUpdate(() => new Date()),
});

/** Soft delete column */
export const softDelete = () => ({
  deletedAt: timestamp('deleted_at', { withTimezone: true }),
});

// ─────────────────────────────────────────────────────────────
// RLS Helper SQL fragments
// ─────────────────────────────────────────────────────────────

/** RLS USING clause: tenant_id matches current_setting */
export const rlsTenantCheck = sql`(tenant_id = current_setting('app.current_tenant', true)::uuid)`;

/** RLS WITH CHECK clause: same as using for inserts */
```

```typescript
export const rlsTenantInsertCheck = sql`(tenant_id = current_setting('app.current_tenant', true)::uuid)`;

// ─────────────────────────────────────────────────────────────
// Constants
// ─────────────────────────────────────────────────────────────

/** pgvector embedding dimensions (nomic-embed-text = 768) */
export const EMBEDDING_DIMENSIONS = 768;

/** Apache AGE graph name */
export const GRAPH_NAME = 'edusphere_graph';
```

---

4. Core Schema — Tenants & Users

typescript

```ts
// File: packages/db/src/schema/tenants.ts

import { pgTable, pgPolicy, varchar, jsonb, boolean, integer } from 'drizzle-orm/pg-core';
import { sql } from 'drizzle-orm';
import { pk, tenantId, timestamps, softDelete, tenantPlanEnum, appRole, rlsTenantCheck, rlsTenantInsertCheck } from './_sha

// ─────────────────────────────────────────────────────────────
// tenants — Top-level organizational units
// ─────────────────────────────────────────────────────────────
// NOTE: tenants table does NOT have tenant_id (it IS the tenant)
// RLS is based on the tenant's own id
export const tenants = pgTable('tenants', {
  id: pk(),
  name: varchar('name', { length: 255 }).notNull(),
  slug: varchar('slug', { length: 100 }).notNull().unique(),
  plan: tenantPlanEnum('plan').notNull().default('free'),
  isActive: boolean('is_active').notNull().default(true),
  maxUsers: integer('max_users').notNull().default(50),
  settings: jsonb('settings').$type<TenantSettings>().default({}),
  keycloakOrgId: varchar('keycloak_org_id', { length: 255 }).unique(),
  ...timestamps(),
  ...softDelete(),
}, (table) => [
  pgPolicy('tenants_select_own', {
    for: 'select',
    to: appRole,
    using: sql`(${table.id} = current_setting('app.current_tenant', true)::uuid)`,
  }),
  pgPolicy('tenants_update_own', {
    for: 'update',
    to: appRole,
    using: sql`(${table.id} = current_setting('app.current_tenant', true)::uuid)`,
    withCheck: sql`(${table.id} = current_setting('app.current_tenant', true)::uuid)`,
  }),
  // INSERT and DELETE managed by super_admin only (no RLS policy = denied)
]);

/** Tenant settings JSON structure */
export interface TenantSettings {
  defaultLanguage?: string;
  allowedLlmProviders?: string[];
  maxStorageGb?: number;
  customOntologyFields?: Array<{ name: string; type: string }>;
  brandingColors?: { primary: string; secondary: string };
}
```

typescript

```ts
// File: packages/db/src/schema/users.ts

import { pgTable, pgPolicy, varchar, jsonb, boolean, index } from 'drizzle-orm/pg-core';
import { sql } from 'drizzle-orm';
import {
  pk, tenantId, timestamps, softDelete,
  userRoleEnum, appRole, rlsTenantCheck, rlsTenantInsertCheck,
} from './_shared.ts';
import { tenants } from './tenants.ts';

// ─────────────────────────────────────────────────────────────
// users — Authenticated users (synced from Keycloak)
// ─────────────────────────────────────────────────────────────
export const users = pgTable('users', {
  id: pk(),
  tenantId: tenantId().references(() => tenants.id, { onDelete: 'restrict' }),
  keycloakUserId: varchar('keycloak_user_id', { length: 255 }).notNull().unique(),
  email: varchar('email', { length: 320 }).notNull(),
  displayName: varchar('display_name', { length: 255 }).notNull(),
  avatarUrl: varchar('avatar_url', { length: 2048 }),
  role: userRoleEnum('role').notNull().default('student'),
  isActive: boolean('is_active').notNull().default(true),
  preferences: jsonb('preferences').$type<UserPreferences>().default({}),
  offlineSyncToken: varchar('offline_sync_token', { length: 500 }),
  ...timestamps(),
  ...softDelete(),
}, (table) => [
  // Indexes
  index('idx_users_tenant').on(table.tenantId),
  index('idx_users_email').on(table.email),
  index('idx_users_keycloak').on(table.keycloakUserId),

  // RLS Policies
  pgPolicy('users_select_same_tenant', {
    for: 'select',
    to: appRole,
    using: rlsTenantCheck,
  }),
  pgPolicy('users_insert_same_tenant', {
    for: 'insert',
    to: appRole,
    withCheck: rlsTenantInsertCheck,
  }),
  pgPolicy('users_update_same_tenant', {
    for: 'update',
    to: appRole,
```

```
    using: rlsTenantCheck,
    withCheck: rlsTenantInsertCheck,
  }),
]);

export interface UserPreferences {
  language?: string;
  theme?: 'light' | 'dark' | 'system';
  defaultAnnotationLayer?: string;
  notificationsEnabled?: boolean;
  agentDefaults?: { difficulty?: string; personality?: string };
}
```

---

5. Content Schema — Courses, Media, Transcripts

typescript

```ts
// File: packages/db/src/schema/courses.ts

import { pgTable, pgPolicy, varchar, text, jsonb, integer, boolean, index } from 'drizzle-orm/pg-core';
import { sql } from 'drizzle-orm';
import { pk, tenantId, timestamps, softDelete, appRole, rlsTenantCheck, rlsTenantInsertCheck } from './_shared.ts';
import { tenants } from './tenants.ts';
import { users } from './users.ts';

// ─────────────────────────────────────────────────────────────
// courses
// ─────────────────────────────────────────────────────────────
export const courses = pgTable('courses', {
  id: pk(),
  tenantId: tenantId().references(() => tenants.id, { onDelete: 'restrict' }),
  creatorId: pk().references(() => users.id, { onDelete: 'set null' }),
  title: varchar('title', { length: 500 }).notNull(),
  description: text('description'),
  syllabusJson: jsonb('syllabus_json').$type<SyllabusItem[]>().default([]),
  version: integer('version').notNull().default(1),
  isPublished: boolean('is_published').notNull().default(false),
  forkedFromId: pk().references((): any => courses.id, { onDelete: 'set null' }),
  tags: jsonb('tags').$type<string[]>().default([]),
  ...timestamps(),
  ...softDelete(),
}, (table) => [
  index('idx_courses_tenant').on(table.tenantId),
  index('idx_courses_creator').on(table.creatorId),

  pgPolicy('courses_select_same_tenant', {
    for: 'select', to: appRole, using: rlsTenantCheck,
  }),
  pgPolicy('courses_insert_same_tenant', {
    for: 'insert', to: appRole, withCheck: rlsTenantInsertCheck,
  }),
  pgPolicy('courses_update_same_tenant', {
    for: 'update', to: appRole,
    using: rlsTenantCheck, withCheck: rlsTenantInsertCheck,
  }),
  pgPolicy('courses_delete_same_tenant', {
    for: 'delete', to: appRole, using: rlsTenantCheck,
  }),
]);

export interface SyllabusItem {
  title: string;
  description?: string;
```

```
  moduleIds?: string[];
  order: number;
}

// ─────────────────────────────────────────────────────────────
// modules — Subdivisions within a course
// ─────────────────────────────────────────────────────────────
export const modules = pgTable('modules', {
  id: pk(),
  tenantId: tenantId().references(() => tenants.id, { onDelete: 'restrict' }),
  courseId: pk().references(() => courses.id, { onDelete: 'cascade' }).notNull(),
  title: varchar('title', { length: 500 }).notNull(),
  description: text('description'),
  orderIndex: integer('order_index').notNull().default(0),
  ...timestamps(),
}, (table) => [
  index('idx_modules_tenant').on(table.tenantId),
  index('idx_modules_course').on(table.courseId),

  pgPolicy('modules_tenant_isolation', {
    for: 'all', to: appRole,
    using: rlsTenantCheck, withCheck: rlsTenantInsertCheck,
  }),
]);
```

typescript

```
// File: packages/db/src/schema/media-assets.ts

import {
  pgTable, pgPolicy, varchar, text, jsonb,
  integer, real, index, uniqueIndex,
} from 'drizzle-orm/pg-core';
import { sql } from 'drizzle-orm';
import {
  pk, tenantId, timestamps, softDelete,
  mediaTypeEnum, transcriptionStatusEnum,
  appRole, rlsTenantCheck, rlsTenantInsertCheck,
} from './_shared.ts';
import { tenants } from './tenants.ts';
import { users } from './users.ts';
import { modules } from './courses.ts';

// ─────────────────────────────────────────────────────────────
// media_assets — Videos, PDFs, audio files, etc.
// ─────────────────────────────────────────────────────────────
export const mediaAssets = pgTable('media_assets', {
  id: pk(),
  tenantId: tenantId().references(() => tenants.id, { onDelete: 'restrict' }),
  uploaderId: pk().references(() => users.id, { onDelete: 'set null' }),
  moduleId: pk().references(() => modules.id, { onDelete: 'set null' }),
  type: mediaTypeEnum('type').notNull(),
  title: varchar('title', { length: 500 }).notNull(),
  description: text('description'),
  s3Path: varchar('s3_path', { length: 2048 }).notNull(),
  s3Bucket: varchar('s3_bucket', { length: 255 }).notNull(),
  originalFilename: varchar('original_filename', { length: 500 }),
  mimeType: varchar('mime_type', { length: 255 }),
  fileSizeBytes: integer('file_size_bytes'),
  durationSeconds: real('duration_seconds'),
  transcriptionStatus: transcriptionStatusEnum('transcription_status').default('pending'),
  hlsManifestPath: varchar('hls_manifest_path', { length: 2048 }),
  thumbnailPath: varchar('thumbnail_path', { length: 2048 }),
  metadata: jsonb('metadata').$type<MediaMetadata>().default({}),
  ...timestamps(),
  ...softDelete(),
}, (table) => [
  index('idx_media_tenant').on(table.tenantId),
  index('idx_media_module').on(table.moduleId),
  index('idx_media_type').on(table.type),
  index('idx_media_transcription_status').on(table.transcriptionStatus),

  pgPolicy('media_tenant_isolation', {
```

```
    for: 'all', to: appRole,
    using: rlsTenantCheck, withCheck: rlsTenantInsertCheck,
  }),
]);

export interface MediaMetadata {
  width?: number;
  height?: number;
  codec?: string;
  bitrate?: number;
  language?: string;
  processingJobId?: string;
}

// ─────────────────────────────────────────────────────────────
// transcripts — Full transcription per media asset
// ─────────────────────────────────────────────────────────────
export const transcripts = pgTable('transcripts', {
  id: pk(),
  tenantId: tenantId().references(() => tenants.id, { onDelete: 'restrict' }),
  assetId: pk().references(() => mediaAssets.id, { onDelete: 'cascade' }).notNull(),
  fullText: text('full_text').notNull(),
  language: varchar('language', { length: 10 }).notNull().default('he'),
  modelUsed: varchar('model_used', { length: 100 }),  // e.g., 'whisper-large-v3'
  confidence: real('confidence'),                     // average confidence 0-1
  wordCount: integer('word_count'),
  ...timestamps(),
}, (table) => [
  index('idx_transcripts_tenant').on(table.tenantId),
  index('idx_transcripts_asset').on(table.assetId),

  pgPolicy('transcripts_tenant_isolation', {
    for: 'all', to: appRole,
    using: rlsTenantCheck, withCheck: rlsTenantInsertCheck,
  }),
]);

// ─────────────────────────────────────────────────────────────
// transcript_segments — Time-aligned segments
// ─────────────────────────────────────────────────────────────
export const transcriptSegments = pgTable('transcript_segments', {
  id: pk(),
  tenantId: tenantId().references(() => tenants.id, { onDelete: 'restrict' }),
  transcriptId: pk().references(() => transcripts.id, { onDelete: 'cascade' }).notNull(),
  assetId: pk().references(() => mediaAssets.id, { onDelete: 'cascade' }).notNull(),
  startTime: real('start_time').notNull(),            // seconds from beginning
  endTime: real('end_time').notNull(),                // seconds from beginning
```

```
  text: text('text').notNull(),
  speaker: varchar('speaker', { length: 255 }),       // speaker diarization
  confidence: real('confidence'),
  ...timestamps(),
}, (table) => [
  index('idx_segments_tenant').on(table.tenantId),
  index('idx_segments_transcript').on(table.transcriptId),
  index('idx_segments_asset').on(table.assetId),
  index('idx_segments_time_range').on(table.assetId, table.startTime, table.endTime),

  pgPolicy('segments_tenant_isolation', {
    for: 'all', to: appRole,
    using: rlsTenantCheck, withCheck: rlsTenantInsertCheck,
  }),
]);
```

---

6. Annotation Schema

typescript

```
// File: packages/db/src/schema/annotations.ts

import {
  pgTable, pgPolicy, varchar, text, jsonb,
  real, integer, index, uuid as uuidCol,
} from 'drizzle-orm/pg-core';
import { sql } from 'drizzle-orm';
import {
  pk, tenantId, timestamps, softDelete,
  annotationTypeEnum, annotationLayerEnum,
  appRole, rlsTenantCheck, rlsTenantInsertCheck,
} from './_shared.ts';
import { tenants } from './tenants.ts';
import { users } from './users.ts';
import { mediaAssets } from './media-assets.ts';

// ═══════════════════════════════════════════════════════════════
// annotations — All user/AI markings on content
// ═══════════════════════════════════════════════════════════════
export const annotations = pgTable('annotations', {
  id: pk(),
  tenantId: tenantId().references(() => tenants.id, { onDelete: 'restrict' }),
  userId: uuidCol('user_id').notNull().references(() => users.id, { onDelete: 'cascade' }),
  assetId: uuidCol('asset_id').notNull().references(() => mediaAssets.id, { onDelete: 'cascade' }),
  type: annotationTypeEnum('type').notNull(),
  layer: annotationLayerEnum('layer').notNull().default('personal'),

  // Content — depends on type
  content: text('content'),                    // text content or markdown
  sketchData: jsonb('sketch_data').$type<SketchData>(),  // Konva.js JSON for sketches

  // Temporal anchoring
  timestampStart: real('timestamp_start'),      // seconds (null for non-temporal)
  timestampEnd: real('timestamp_end'),          // seconds (null for point annotations)

  // Spatial anchoring (for spatial_comment on video frame)
  spatialX: real('spatial_x'),                  // 0-1 normalized X position
  spatialY: real('spatial_y'),                  // 0-1 normalized Y position

  // PDF-specific anchoring
  pageNumber: integer('page_number'),
  textRangeStart: integer('text_range_start'),  // character offset
  textRangeEnd: integer('text_range_end'),      // character offset

  // Reference to other annotation (reply chain)
  parentId: uuidCol('parent_id').references((): any => annotations.id, { onDelete: 'cascade' }),
```

```
  // AI-generated metadata
  agentId: uuidCol('agent_id'),                    // if created by an agent
  metadata: jsonb('metadata').$type<AnnotationMetadata>().default({}),

  ...timestamps(),
  ...softDelete(),
}, (table) => [
  index('idx_annotations_tenant').on(table.tenantId),
  index('idx_annotations_user').on(table.userId),
  index('idx_annotations_asset').on(table.assetId),
  index('idx_annotations_asset_time').on(table.assetId, table.timestampStart),
  index('idx_annotations_layer').on(table.layer),
  index('idx_annotations_parent').on(table.parentId),

  pgPolicy('annotations_select_tenant', {
    for: 'select', to: appRole, using: rlsTenantCheck,
  }),
  pgPolicy('annotations_insert_tenant', {
    for: 'insert', to: appRole, withCheck: rlsTenantInsertCheck,
  }),
  pgPolicy('annotations_update_own', {
    for: 'update', to: appRole,
    // Users can only update their own annotations within same tenant
    using: sql`(tenant_id = current_setting('app.current_tenant', true)::uuid
      AND user_id = current_setting('app.current_user', true)::uuid)`,
    withCheck: rlsTenantInsertCheck,
  }),
  pgPolicy('annotations_delete_own', {
    for: 'delete', to: appRole,
    using: sql`(tenant_id = current_setting('app.current_tenant', true)::uuid
      AND user_id = current_setting('app.current_user', true)::uuid)`,
  }),
]);

export interface SketchData {
  /** Konva.js stage JSON export */
  stage: Record<string, unknown>;
  /** Dimensions the sketch was created at */
  canvasWidth: number;
  canvasHeight: number;
  /** Video frame timestamp this sketch is anchored to */
  frameTimestamp?: number;
}

export interface AnnotationMetadata {
  color?: string;
```

```
  pinned?: boolean;
  resolvedAt?: string;
  linkedConceptIds?: string[];
  aiConfidence?: number;
}
```

---

7. Collaboration Schema — CRDT Persistence

typescript

```ts
// File: packages/db/src/schema/collaboration.ts

import {
  pgTable, pgPolicy, varchar, text, jsonb,
  integer, index, uuid as uuidCol, customType,
} from 'drizzle-orm/pg-core';
import { sql } from 'drizzle-orm';
import {
  pk, tenantId, timestamps,
  appRole, rlsTenantCheck, rlsTenantInsertCheck,
} from './_shared.ts';
import { tenants } from './tenants.ts';
import { users } from './users.ts';

// ─────────────────────────────────────────────────────────────
// Custom type: bytea for CRDT binary data
// ─────────────────────────────────────────────────────────────
const bytea = customType<{ data: Buffer; driverData: Buffer }>({
  dataType() {
    return 'bytea';
  },
});

// ─────────────────────────────────────────────────────────────
// collab_documents — Root document for Yjs collaboration
// ─────────────────────────────────────────────────────────────
export const collabDocuments = pgTable('collab_documents', {
  id: pk(),
  tenantId: tenantId().references(() => tenants.id, { onDelete: 'restrict' }),
  /** Hocuspocus document name — unique per tenant */
  documentName: varchar('document_name', { length: 500 }).notNull(),
  /** The compacted Yjs state vector (full snapshot) */
  stateVector: bytea('state_vector'),
  /** Number of incremental updates since last compaction */
  pendingUpdates: integer('pending_updates').notNull().default(0),
  /** Human-readable title */
  title: varchar('title', { length: 500 }),
  /** Associated entity (annotation layer, course notes, etc.) */
  entityType: varchar('entity_type', { length: 100 }),
  entityId: uuidCol('entity_id'),
  ...timestamps(),
}, (table) => [
  index('idx_collab_docs_tenant').on(table.tenantId),
  index('idx_collab_docs_name').on(table.tenantId, table.documentName),
  index('idx_collab_docs_entity').on(table.entityType, table.entityId),
```

```
  pgPolicy('collab_docs_tenant_isolation', {
    for: 'all', to: appRole,
    using: rlsTenantCheck, withCheck: rlsTenantInsertCheck,
  }),
]);

// ─────────────────────────────────────────────────────────────
// crdt_updates — Incremental Yjs binary updates
// ─────────────────────────────────────────────────────────────
// Append-only. Periodically compacted into collab_documents.state_vector
export const crdtUpdates = pgTable('crdt_updates', {
  id: pk(),
  tenantId: tenantId().references(() => tenants.id, { onDelete: 'restrict' }),
  documentId: uuidCol('document_id').notNull()
    .references(() => collabDocuments.id, { onDelete: 'cascade' }),
  /** The binary Yjs update */
  update: bytea('update').notNull(),
  /** Sequence number for ordering */
  seq: integer('seq').notNull(),
  /** Which user caused this update */
  userId: uuidCol('user_id').references(() => users.id, { onDelete: 'set null' }),
  ...timestamps(),
}, (table) => [
  index('idx_crdt_updates_doc').on(table.documentId),
  index('idx_crdt_updates_doc_seq').on(table.documentId, table.seq),

  pgPolicy('crdt_updates_tenant_isolation', {
    for: 'all', to: appRole,
    using: rlsTenantCheck, withCheck: rlsTenantInsertCheck,
  }),
]);

// ─────────────────────────────────────────────────────────────
// collab_sessions — Active collaboration sessions (ephemeral)
// ─────────────────────────────────────────────────────────────
export const collabSessions = pgTable('collab_sessions', {
  id: pk(),
  tenantId: tenantId().references(() => tenants.id, { onDelete: 'restrict' }),
  documentId: uuidCol('document_id').notNull()
    .references(() => collabDocuments.id, { onDelete: 'cascade' }),
  userId: uuidCol('user_id').notNull().references(() => users.id, { onDelete: 'cascade' }),
  cursorPosition: jsonb('cursor_position').$type<CursorPosition>(),
  connectedAt: timestamps().createdAt,
  lastSeenAt: timestamps().updatedAt,
}, (table) => [
  index('idx_collab_sessions_doc').on(table.documentId),
```

```
  pgPolicy('collab_sessions_tenant_isolation', {
    for: 'all', to: appRole,
    using: rlsTenantCheck, withCheck: rlsTenantInsertCheck,
  }),
]);

export interface CursorPosition {
  x?: number;
  y?: number;
  timestamp?: number;    // video timestamp if applicable
  pageNumber?: number;   // PDF page if applicable
  userName?: string;
  color?: string;
}
```

---

8. Agent Schema

typescript

```ts
// File: packages/db/src/schema/agents.ts

import {
  pgTable, pgPolicy, varchar, text, jsonb,
  integer, real, index, uuid as uuidCol,
  boolean,
} from 'drizzle-orm/pg-core';
import { sql } from 'drizzle-orm';
import {
  pk, tenantId, timestamps, softDelete,
  agentTemplateEnum, agentTriggerEnum, agentOutputFormatEnum,
  executionStatusEnum,
  appRole, rlsTenantCheck, rlsTenantInsertCheck,
} from './_shared.ts';
import { tenants } from './tenants.ts';
import { users } from './users.ts';

// ─────────────────────────────────────────────────────────────
// agent_definitions — User-created or built-in agents
// ─────────────────────────────────────────────────────────────
export const agentDefinitions = pgTable('agent_definitions', {
  id: pk(),
  tenantId: tenantId().references(() => tenants.id, { onDelete: 'restrict' }),
  creatorId: uuidCol('creator_id').notNull().references(() => users.id, { onDelete: 'cascade' }),
  name: varchar('name', { length: 255 }).notNull(),
  description: text('description'),
  templateType: agentTemplateEnum('template_type').notNull(),
  isBuiltIn: boolean('is_built_in').notNull().default(false),
  isPublic: boolean('is_public').notNull().default(false),

  // Agent configuration
  config: jsonb('config').$type<AgentConfig>().notNull(),

  // Usage stats
  executionCount: integer('execution_count').notNull().default(0),
  avgExecutionMs: real('avg_execution_ms'),
  lastExecutedAt: timestamps().updatedAt,

  ...timestamps(),
  ...softDelete(),
}, (table) => [
  index('idx_agent_defs_tenant').on(table.tenantId),
  index('idx_agent_defs_creator').on(table.creatorId),
  index('idx_agent_defs_template').on(table.templateType),

  pgPolicy('agent_defs_select_tenant', {
```

```
    for: 'select', to: appRole, using: rlsTenantCheck,
  }),
  pgPolicy('agent_defs_insert_tenant', {
    for: 'insert', to: appRole, withCheck: rlsTenantInsertCheck,
  }),
  pgPolicy('agent_defs_update_own', {
    for: 'update', to: appRole,
    using: sql`(tenant_id = current_setting('app.current_tenant', true)::uuid
      AND creator_id = current_setting('app.current_user', true)::uuid)`,
    withCheck: rlsTenantInsertCheck,
  }),
  pgPolicy('agent_defs_delete_own', {
    for: 'delete', to: appRole,
    using: sql`(tenant_id = current_setting('app.current_tenant', true)::uuid
      AND creator_id = current_setting('app.current_user', true)::uuid)`,
  }),
]);

export interface AgentConfig {
  dataScope: 'MY_NOTES' | 'COURSE' | 'ORGANIZATION' | 'CUSTOM_QUERY';
  customQuery?: string;              // Cypher query for CUSTOM_QUERY scope
  triggerType: AgentTrigger;
  outputFormat: AgentOutputFormat;
  personality?: string;              // e.g., "Strict Talmudic scholar"
  difficulty?: 'BEGINNER' | 'INTERMEDIATE' | 'ADVANCED';
  modelOverride?: string;            // e.g., 'phi4:14b'
  maxTokens?: number;
  temperature?: number;
  systemPrompt?: string;
  toolsEnabled?: string[];           // which MCP tools the agent can use
}

// ═══════════════════════════════════════════════════════════
// agent_executions — Log of every agent run
// ═══════════════════════════════════════════════════════════
export const agentExecutions = pgTable('agent_executions', {
  id: pk(),
  tenantId: tenantId().references(() => tenants.id, { onDelete: 'restrict' }),
  agentId: uuidCol('agent_id').notNull()
    .references(() => agentDefinitions.id, { onDelete: 'cascade' }),
  userId: uuidCol('user_id').notNull().references(() => users.id, { onDelete: 'cascade' }),
  status: executionStatusEnum('status').notNull().default('queued'),

  // Input/Output
  input: jsonb('input').$type<Record<string, unknown>>().notNull(),
  output: jsonb('output').$type<Record<string, unknown>>(),
  reasoning: text('reasoning'),        // chain-of-thought (debugging)
```

```
  errorMessage: text('error_message'),

  // Performance tracking
  tokensUsed: integer('tokens_used'),
  executionMs: integer('execution_ms'),
  modelUsed: varchar('model_used', { length: 100 }),

  ...timestamps(),
}, (table) => [
  index('idx_agent_execs_tenant').on(table.tenantId),
  index('idx_agent_execs_agent').on(table.agentId),
  index('idx_agent_execs_user').on(table.userId),
  index('idx_agent_execs_status').on(table.status),

  pgPolicy('agent_execs_tenant_isolation', {
    for: 'all', to: appRole,
    using: rlsTenantCheck, withCheck: rlsTenantInsertCheck,
  }),
]);
```

---

9. Row-Level Security Policies

typescript

```typescript
// File: packages/db/src/schema/_rls-policies.ts
//
// This file documents the RLS strategy. Actual policies are defined
// inline on each table (see above). This file provides the middleware
// that sets the tenant context per request.

/**
 * CRITICAL: Every database transaction MUST set the tenant context.
 * This is enforced by the `withTenantContext` wrapper.
 *
 * Pattern:
 *   SET LOCAL app.current_tenant = '<tenant-uuid>';
 *   SET LOCAL app.current_user = '<user-uuid>';
 *   SET LOCAL ROLE edusphere_app;
 *
 * SET LOCAL is transaction-scoped — auto-resets on commit/rollback.
 * This is safe with PgBouncer in transaction mode.
 */

import { sql } from 'drizzle-orm';
import { type PostgresJsDatabase } from 'drizzle-orm/postgres-js';

export interface TenantContext {
  tenantId: string;
  userId: string;
}

/**
 * Execute a callback within a tenant-isolated transaction.
 * ALL user-facing database operations MUST use this wrapper.
 *
 * @example
 * ```typescript
 * const result = await withTenantContext(db, ctx, async (tx) => {
 *   return tx.select().from(courses);
 *   // RLS automatically filters by tenant_id
 * });
 * ```
 */
export async function withTenantContext<T>(
  db: PostgresJsDatabase,
  ctx: TenantContext,
  callback: (tx: PostgresJsDatabase) => Promise<T>,
): Promise<T> {
  return db.transaction(async (tx) => {
    // Set tenant context for RLS
```

```
    await tx.execute(
      sql`SET LOCAL app.current_tenant = ${ctx.tenantId}`
    );
    await tx.execute(
      sql`SET LOCAL app.current_user = ${ctx.userId}`
    );
    // Switch to application role (RLS policies apply)
    await tx.execute(sql`SET LOCAL ROLE edusphere_app`);

    return callback(tx);
  });
}

/**
 * For service-to-service calls that need to bypass RLS
 * (e.g., NATS event handlers processing cross-tenant data).
 * Use SPARINGLY and document why.
 */
export async function withSuperuserContext<T>(
  db: PostgresJsDatabase,
  callback: (tx: PostgresJsDatabase) => Promise<T>,
): Promise<T> {
  // No SET LOCAL ROLE — runs as connection owner (superuser)
  return db.transaction(async (tx) => callback(tx));
}
```

RLS Policy Summary Table:

Table	SELECT	INSERT	UPDATE	DELETE
tenants	Own tenant only	Super admin	Own tenant	Super admin
users	Same tenant	Same tenant	Same tenant	—
courses	Same tenant	Same tenant	Same tenant	Same tenant
modules	Same tenant	Same tenant	Same tenant	Same tenant
media_assets	Same tenant	Same tenant	Same tenant	Same tenant
transcripts	Same tenant	Same tenant	Same tenant	Same tenant
transcript_segments	Same tenant	Same tenant	Same tenant	Same tenant
annotations	Same tenant	Same tenant	Own only + tenant	Own only + tenant
collab_documents	Same tenant	Same tenant	Same tenant	Same tenant
crdt_updates	Same tenant	Same tenant	Same tenant	Same tenant
agent_definitions	Same tenant	Same tenant	Own only + tenant	Own only + tenant
agent_executions	Same tenant	Same tenant	Same tenant	Same tenant

10. Apache AGE — Knowledge Graph Ontology

10.1 Graph Initialization

sql

```
-- File: packages/db/src/graph/init-graph.sql
-- Run after extensions are created

-- Create vertex labels (node types)
SELECT create_vlabel('edusphere_graph', 'Concept');
SELECT create_vlabel('edusphere_graph', 'Person');
SELECT create_vlabel('edusphere_graph', 'Term');
SELECT create_vlabel('edusphere_graph', 'Source');
SELECT create_vlabel('edusphere_graph', 'TopicCluster');

-- Create edge labels (relationship types)
SELECT create_elabel('edusphere_graph', 'MENTIONS');
SELECT create_elabel('edusphere_graph', 'REFERS_TO');
SELECT create_elabel('edusphere_graph', 'RELATED_TO');
SELECT create_elabel('edusphere_graph', 'CONTRADICTS');
SELECT create_elabel('edusphere_graph', 'PREREQUISITE_OF');
SELECT create_elabel('edusphere_graph', 'DERIVED_FROM');
SELECT create_elabel('edusphere_graph', 'AUTHORED_BY');
SELECT create_elabel('edusphere_graph', 'CITES');
SELECT create_elabel('edusphere_graph', 'INFERRED_RELATED');
SELECT create_elabel('edusphere_graph', 'BELONGS_TO');

-- Create indexes on vertex properties for fast lookups
CREATE INDEX idx_concept_tenant ON edusphere_graph."Concept" USING btree ((properties->>'tenant_id'));
CREATE INDEX idx_concept_label ON edusphere_graph."Concept" USING gin ((properties->'label'));
CREATE INDEX idx_person_name ON edusphere_graph."Person" USING gin ((properties->'name'));
CREATE INDEX idx_term_label ON edusphere_graph."Term" USING gin ((properties->'label'));
```

10.2 Vertex Property Schemas

typescript

```typescript
// File: packages/db/src/graph/ontology.ts

/**
 * Vertex (Node) property schemas for the knowledge graph.
 * These are stored as agtype JSON in Apache AGE.
 * Validated with Zod before insertion.
 */

import { z } from 'zod';

// ── Concept ──────────────────────────────────────────────
// A semantic concept (e.g., "Thermodynamics", "Rambam", "Polymers")
export const ConceptVertex = z.object({
  id: z.string().uuid(),          // Same UUID as concepts SQL table
  tenant_id: z.string().uuid(),
  label: z.string().min(1),       // Primary label (Hebrew/English)
  description: z.string().optional(),
  aliases: z.array(z.string()).default([]),  // Synonyms, abbreviations
  domain: z.string().optional(),   // e.g., "Physics", "Talmud", "Engineering"
  confidence: z.number().min(0).max(1).default(1), // 1 = manual, <1 = AI-extracted
  source_type: z.enum(['manual', 'ai_extracted', 'imported']).default('manual'),
  created_by: z.string().uuid().optional(),
});
export type ConceptVertex = z.infer<typeof ConceptVertex>;

// ── Person ───────────────────────────────────────────────
// A person referenced in content (author, speaker, historical figure)
export const PersonVertex = z.object({
  id: z.string().uuid(),
  tenant_id: z.string().uuid(),
  name: z.string(),
  role: z.string().optional(),     // e.g., "Speaker", "Author", "Historical"
  external_url: z.string().url().optional(),
});
export type PersonVertex = z.infer<typeof PersonVertex>;

// ── Term ─────────────────────────────────────────────────
// A domain-specific term (narrower than Concept)
export const TermVertex = z.object({
  id: z.string().uuid(),
  tenant_id: z.string().uuid(),
  label: z.string(),
  domain: z.string(),
  definition: z.string().optional(),
});
export type TermVertex = z.infer<typeof TermVertex>;
```

```typescript
// ── Source ──────────────────────────────────────────
// An external reference (book, paper, URL)
export const SourceVertex = z.object({
  id: z.string().uuid(),
  tenant_id: z.string().uuid(),
  title: z.string(),
  type: z.enum(['book', 'paper', 'url', 'lecture', 'internal_asset']),
  external_url: z.string().optional(),
  internal_asset_id: z.string().uuid().optional(),  // FK to media_assets
  isbn: z.string().optional(),
  doi: z.string().optional(),
});
export type SourceVertex = z.infer<typeof SourceVertex>;

// ── TopicCluster ────────────────────────────────────
// Auto-generated grouping of related concepts
export const TopicClusterVertex = z.object({
  id: z.string().uuid(),
  tenant_id: z.string().uuid(),
  label: z.string(),
  concept_count: z.number().int(),
  coherence_score: z.number().min(0).max(1),
});
export type TopicClusterVertex = z.infer<typeof TopicClusterVertex>;
```

10.3 Edge Property Schemas

typescript

```typescript
// File: packages/db/src/graph/ontology.ts (continued)

// ── Edge schemas ──────────────────────────────────────

export const MentionsEdge = z.object({
  segment_id: z.string().uuid(),     // FK to transcript_segments
  asset_id: z.string().uuid(),       // FK to media_assets
  start_time: z.number(),
  end_time: z.number(),
  confidence: z.number().min(0).max(1).default(1),
});

export const RelatedToEdge = z.object({
  strength: z.number().min(0).max(1),  // Semantic similarity score
  method: z.enum(['manual', 'embedding_similarity', 'co_occurrence', 'citation']),
  evidence: z.string().optional(),     // Human-readable explanation
});

export const ContradictsEdge = z.object({
  evidence: z.string(),              // Why these concepts contradict
  source_a: z.string().uuid(),       // Which media asset/annotation
  source_b: z.string().uuid(),
  severity: z.enum(['minor', 'moderate', 'major']).default('moderate'),
  detected_by: z.enum(['manual', 'chavruta_agent', 'ai_inference']),
});

export const PrerequisiteOfEdge = z.object({
  strength: z.enum(['required', 'recommended', 'optional']).default('recommended'),
});

export const InferredRelatedEdge = z.object({
  confidence: z.number().min(0).max(1),
  method: z.string(),                // Algorithm used
  inferred_at: z.string().datetime(),
  reviewed: z.boolean().default(false),
});

export const CitesEdge = z.object({
  page: z.number().optional(),
  timestamp: z.number().optional(),  // If citing a video moment
  context: z.string().optional(),    // Quote or surrounding text
});
```

10.4 Example Cypher Queries

typescript

```
// File: packages/db/src/graph/queries.ts

import { sql } from 'drizzle-orm';
import { GRAPH_NAME } from '../schema/_shared.ts';

/**
 * All Cypher queries are executed via SQL using Apache AGE's cypher() function.
 * Parameters are passed as the third argument (agtype) for safety.
 */

/** Create a new Concept vertex */
export const createConceptQuery = (concept: ConceptVertex) => sql`
  SELECT * FROM cypher(${GRAPH_NAME}, $$
    CREATE (c:Concept {
      id: '${sql.raw(concept.id)}',
      tenant_id: '${sql.raw(concept.tenant_id)}',
      label: '${sql.raw(concept.label)}',
      aliases: ${sql.raw(JSON.stringify(concept.aliases))},
      domain: '${sql.raw(concept.domain ?? '')}',
      confidence: ${sql.raw(String(concept.confidence))},
      source_type: '${sql.raw(concept.source_type)}'
    })
    RETURN c
  $$) AS (v agtype)
`;

/** Find all concepts related to a specific concept */
export const findRelatedConceptsQuery = (conceptId: string, tenantId: string) => sql`
  SELECT * FROM cypher(${GRAPH_NAME}, $$
    MATCH (c:Concept {id: '${sql.raw(conceptId)}', tenant_id: '${sql.raw(tenantId)}'})
          -[r:RELATED_TO|CONTRADICTS|PREREQUISITE_OF*1..3]-
          (related:Concept)
    WHERE related.tenant_id = '${sql.raw(tenantId)}'
    RETURN related.id, related.label, type(r) AS relation_type, r.strength
    ORDER BY r.strength DESC
    LIMIT 50
  $$) AS (id agtype, label agtype, relation_type agtype, strength agtype)
`;

/** Get semantic context for a video timestamp (for Contextual Copilot) */
export const getSemanticContextQuery = (assetId: string, timestamp: number, tenantId: string) => sql`
  SELECT * FROM cypher(${GRAPH_NAME}, $$
    MATCH (seg)-[:MENTIONS]->(concept:Concept)
    WHERE seg.asset_id = '${sql.raw(assetId)}'
      AND concept.tenant_id = '${sql.raw(tenantId)}'
      AND seg.start_time <= ${sql.raw(String(timestamp))}
```

```
    AND seg.end_time >= ${sql.raw(String(timestamp))}
  WITH concept
  OPTIONAL MATCH (concept)-[r:RELATED_TO|CONTRADICTS]->(related:Concept)
  WHERE related.tenant_id = '${sql.raw(tenantId)}'
  RETURN concept.id, concept.label, concept.domain,
         collect({
           related_id: related.id,
           related_label: related.label,
           relation: type(r),
           strength: r.strength
         }) AS connections
 $$) AS (id agtype, label agtype, domain agtype, connections agtype)
`;

/** Find contradictions for Chavruta agent */
export const findContradictionsQuery = (conceptId: string, tenantId: string) => sql`
 SELECT * FROM cypher(${GRAPH_NAME}, $$
  MATCH (a:Concept {id: '${sql.raw(conceptId)}', tenant_id: '${sql.raw(tenantId)}'})
        -[r:CONTRADICTS]-
        (b:Concept {tenant_id: '${sql.raw(tenantId)}'})
  RETURN b.id, b.label, r.evidence, r.severity, r.detected_by
 $$) AS (id agtype, label agtype, evidence agtype, severity agtype, detected_by agtype)
`;

/** Build concept prerequisite chain (learning path) */
export const getPrerequisiteChainQuery = (conceptId: string, tenantId: string) => sql`
 SELECT * FROM cypher(${GRAPH_NAME}, $$
  MATCH path = (target:Concept {id: '${sql.raw(conceptId)}', tenant_id: '${sql.raw(tenantId)}'})
               <-[:PREREQUISITE_OF*1..5]-
               (prereq:Concept {tenant_id: '${sql.raw(tenantId)}'})
  RETURN nodes(path) AS chain, length(path) AS depth
  ORDER BY depth ASC
 $$) AS (chain agtype, depth agtype)
`;
```

---

11. pgvector — Embedding Tables & Indexes

typescript

```ts
// File: packages/db/src/schema/embeddings.ts

import {
  pgTable, pgPolicy, varchar, text, jsonb,
  integer, index, uuid as uuidCol, real,
} from 'drizzle-orm/pg-core';
import { sql } from 'drizzle-orm';
import {
  pk, tenantId, timestamps,
  appRole, rlsTenantCheck, rlsTenantInsertCheck,
  EMBEDDING_DIMENSIONS,
} from './_shared.ts';
import { tenants } from './tenants.ts';
import { mediaAssets } from './media-assets.ts';
import { transcriptSegments } from './media-assets.ts';
import { annotations } from './annotations.ts';

/**
 * Custom vector column type for pgvector.
 * Drizzle does not have built-in vector support,
 * so we define a custom type.
 */
import { customType } from 'drizzle-orm/pg-core';

const vector = (name: string, dimensions: number) =>
  customType<{ data: number[]; driverData: string }>({
    dataType() {
      return `vector(${dimensions})`;
    },
    toDriver(value: number[]) {
      return `[${value.join(',')}]`;
    },
    fromDriver(value: string) {
      return value
        .replace('[', '')
        .replace(']', '')
        .split(',')
        .map(Number);
    },
  })(name);

// ─────────────────────────────────────────────────────────
// content_embeddings — Vectors for transcript segments
// ─────────────────────────────────────────────────────────
export const contentEmbeddings = pgTable('content_embeddings', {
  id: pk(),
```

```
  tenantId: tenantId().references(() => tenants.id, { onDelete: 'restrict' }),
  segmentId: uuidCol('segment_id')
    .references(() => transcriptSegments.id, { onDelete: 'cascade' }),
  assetId: uuidCol('asset_id').notNull()
    .references(() => mediaAssets.id, { onDelete: 'cascade' }),
  /** The text that was embedded */
  sourceText: text('source_text').notNull(),
  /** The embedding vector */
  embedding: vector('embedding', EMBEDDING_DIMENSIONS).notNull(),
  /** Which model generated this embedding */
  modelName: varchar('model_name', { length: 100 }).notNull(),
  /** Chunk index if text was split */
  chunkIndex: integer('chunk_index').notNull().default(0),
  ...timestamps(),
}, (table) => [
  index('idx_content_emb_tenant').on(table.tenantId),
  index('idx_content_emb_asset').on(table.assetId),

  pgPolicy('content_embeddings_tenant_isolation', {
    for: 'all', to: appRole,
    using: rlsTenantCheck, withCheck: rlsTenantInsertCheck,
  }),
]);

// ─────────────────────────────────────────────────────────────
// annotation_embeddings — Vectors for user annotations
// ─────────────────────────────────────────────────────────────
export const annotationEmbeddings = pgTable('annotation_embeddings', {
  id: pk(),
  tenantId: tenantId().references(() => tenants.id, { onDelete: 'restrict' }),
  annotationId: uuidCol('annotation_id').notNull()
    .references(() => annotations.id, { onDelete: 'cascade' }),
  sourceText: text('source_text').notNull(),
  embedding: vector('embedding', EMBEDDING_DIMENSIONS).notNull(),
  modelName: varchar('model_name', { length: 100 }).notNull(),
  ...timestamps(),
}, (table) => [
  index('idx_annotation_emb_tenant').on(table.tenantId),

  pgPolicy('annotation_embeddings_tenant_isolation', {
    for: 'all', to: appRole,
    using: rlsTenantCheck, withCheck: rlsTenantInsertCheck,
  }),
]);

// ─────────────────────────────────────────────────────────────
// concept_embeddings — Vectors for knowledge graph concepts
```

```
// ─────────────────────────────────────────────────────────────
export const conceptEmbeddings = pgTable('concept_embeddings', {
  id: pk(),
  tenantId: tenantId().references(() => tenants.id, { onDelete: 'restrict' }),
  /** UUID matching the Concept vertex in AGE */
  conceptId: uuidCol('concept_id').notNull(),
  /** label + description + aliases concatenated */
  sourceText: text('source_text').notNull(),
  embedding: vector('embedding', EMBEDDING_DIMENSIONS).notNull(),
  modelName: varchar('model_name', { length: 100 }).notNull(),
  ...timestamps(),
}, (table) => [
  index('idx_concept_emb_tenant').on(table.tenantId),
  index('idx_concept_emb_concept').on(table.conceptId),

  pgPolicy('concept_embeddings_tenant_isolation', {
    for: 'all', to: appRole,
    using: rlsTenantCheck, withCheck: rlsTenantInsertCheck,
  }),
]);
```

11.1 HNSW Indexes (Created via Migration)

```
sql

-- File: packages/db/src/migrations/0002_vector_indexes.sql
-- HNSW indexes for approximate nearest neighbor search
-- cosine distance is best for normalized embeddings (sentence-transformers, nomic-embed-text)

-- Content embeddings index
CREATE INDEX CONCURRENTLY idx_content_emb_hnsw
  ON content_embeddings
  USING hnsw (embedding vector_cosine_ops)
  WITH (m = 16, ef_construction = 200);

-- Annotation embeddings index
CREATE INDEX CONCURRENTLY idx_annotation_emb_hnsw
  ON annotation_embeddings
  USING hnsw (embedding vector_cosine_ops)
  WITH (m = 16, ef_construction = 200);

-- Concept embeddings index
CREATE INDEX CONCURRENTLY idx_concept_emb_hnsw
  ON concept_embeddings
  USING hnsw (embedding vector_cosine_ops)
  WITH (m = 16, ef_construction = 200);
```

11.2 Semantic Search Functions

sql

```sql
-- File: packages/db/src/migrations/0003_search_functions.sql

-- Semantic search across all content within a tenant
CREATE OR REPLACE FUNCTION search_content_semantic(
  query_embedding vector(768),
  p_tenant_id uuid,
  p_limit integer DEFAULT 10,
  p_min_similarity float DEFAULT 0.7
)
RETURNS TABLE (
  segment_id uuid,
  asset_id uuid,
  source_text text,
  similarity float
)
LANGUAGE sql STABLE
AS $$
  SELECT
    ce.segment_id,
    ce.asset_id,
    ce.source_text,
    1 - (ce.embedding <=> query_embedding) AS similarity
  FROM content_embeddings ce
  WHERE ce.tenant_id = p_tenant_id
    AND 1 - (ce.embedding <=> query_embedding) >= p_min_similarity
  ORDER BY ce.embedding <=> query_embedding
  LIMIT p_limit;
$$;

-- HybridRAG: combines vector search + graph traversal
-- Called from the semantic service, results merged in TypeScript
CREATE OR REPLACE FUNCTION search_concepts_semantic(
  query_embedding vector(768),
  p_tenant_id uuid,
  p_limit integer DEFAULT 10
)
RETURNS TABLE (
  concept_id uuid,
  source_text text,
  similarity float
)
LANGUAGE sql STABLE
AS $$
  SELECT
    ce.concept_id,
    ce.source_text,
```

```
  1 - (ce.embedding <=> query_embedding) AS similarity
  FROM concept_embeddings ce
  WHERE ce.tenant_id = p_tenant_id
  ORDER BY ce.embedding <=> query_embedding
  LIMIT p_limit;
$$;
```

---

12. Drizzle Relations (RQBv2)

typescript

```ts
// File: packages/db/src/relations.ts

import { defineRelations } from 'drizzle-orm';
import * as schema from './schema/index.ts';

export const relations = defineRelations(schema, (r) => ({
  // ── Tenants ──────────────────────────────────────────
  tenants: {
    users: r.many.users({ from: r.tenants.id, to: r.users.tenantId }),
    courses: r.many.courses({ from: r.tenants.id, to: r.courses.tenantId }),
  },

  // ── Users ────────────────────────────────────────────
  users: {
    tenant: r.one.tenants({ from: r.users.tenantId, to: r.tenants.id }),
    annotations: r.many.annotations({ from: r.users.id, to: r.annotations.userId }),
    agentDefinitions: r.many.agentDefinitions({ from: r.users.id, to: r.agentDefinitions.creatorId }),
    agentExecutions: r.many.agentExecutions({ from: r.users.id, to: r.agentExecutions.userId }),
  },

  // ── Courses ──────────────────────────────────────────
  courses: {
    tenant: r.one.tenants({ from: r.courses.tenantId, to: r.tenants.id }),
    modules: r.many.modules({ from: r.courses.id, to: r.modules.courseId }),
    forkedFrom: r.one.courses({ from: r.courses.forkedFromId, to: r.courses.id }),
  },

  // ── Modules ──────────────────────────────────────────
  modules: {
    course: r.one.courses({ from: r.modules.courseId, to: r.courses.id }),
    mediaAssets: r.many.mediaAssets({ from: r.modules.id, to: r.mediaAssets.moduleId }),
  },

  // ── Media Assets ─────────────────────────────────────
  mediaAssets: {
    module: r.one.modules({ from: r.mediaAssets.moduleId, to: r.modules.id }),
    transcripts: r.many.transcripts({ from: r.mediaAssets.id, to: r.transcripts.assetId }),
    annotations: r.many.annotations({ from: r.mediaAssets.id, to: r.annotations.assetId }),
    contentEmbeddings: r.many.contentEmbeddings({ from: r.mediaAssets.id, to: r.contentEmbeddings.assetId }),
  },

  // ── Transcripts ──────────────────────────────────────
  transcripts: {
    asset: r.one.mediaAssets({ from: r.transcripts.assetId, to: r.mediaAssets.id }),
    segments: r.many.transcriptSegments({ from: r.transcripts.id, to: r.transcriptSegments.transcriptId }),
  },
```

```
  // ── Transcript Segments ──────────────────────
  transcriptSegments: {
    transcript: r.one.transcripts({ from: r.transcriptSegments.transcriptId, to: r.transcripts.id }),
    asset: r.one.mediaAssets({ from: r.transcriptSegments.assetId, to: r.mediaAssets.id }),
  },

  // ── Annotations ──────────────────────────────
  annotations: {
    user: r.one.users({ from: r.annotations.userId, to: r.users.id }),
    asset: r.one.mediaAssets({ from: r.annotations.assetId, to: r.mediaAssets.id }),
    parent: r.one.annotations({ from: r.annotations.parentId, to: r.annotations.id }),
    replies: r.many.annotations({ from: r.annotations.id, to: r.annotations.parentId }),
  },

  // ── Agent Definitions ────────────────────────
  agentDefinitions: {
    creator: r.one.users({ from: r.agentDefinitions.creatorId, to: r.users.id }),
    executions: r.many.agentExecutions({ from: r.agentDefinitions.id, to: r.agentExecutions.agentId }),
  },

  // ── Agent Executions ─────────────────────────
  agentExecutions: {
    agent: r.one.agentDefinitions({ from: r.agentExecutions.agentId, to: r.agentDefinitions.id }),
    user: r.one.users({ from: r.agentExecutions.userId, to: r.users.id }),
  },

  // ── Collab Documents ─────────────────────────
  collabDocuments: {
    updates: r.many.crdtUpdates({ from: r.collabDocuments.id, to: r.crdtUpdates.documentId }),
    sessions: r.many.collabSessions({ from: r.collabDocuments.id, to: r.collabSessions.documentId }),
  },
}));
```

---

13. Migration Strategy

13.1 Drizzle Kit Configuration

typescript

```
// File: packages/db/drizzle.config.ts

import { defineConfig } from 'drizzle-kit';

export default defineConfig({
  schema: './src/schema/index.ts',
  out: './src/migrations',
  dialect: 'postgresql',
  dbCredentials: {
    url: process.env.DATABASE_URL!,
  },
  verbose: true,
  strict: true,
  // Manage RLS policies and roles
  entities: {
    roles: {
      provider: '',   // Empty = drizzle manages roles
      exclude: ['postgres', 'pg_*'],
    },
  },
});
```

13.2 Migration Order

```
1. 0000_extensions.sql          — PG extensions (manual, run as superuser)
2. 0001_initial_schema.sql      — Drizzle-generated tables + RLS policies
3. 0002_vector_indexes.sql      — HNSW indexes (manual, CONCURRENTLY)
4. 0003_search_functions.sql    — Semantic search PG functions (manual)
5. 0004_graph_init.sql          — Apache AGE graph + labels + indexes (manual)
```

13.3 Commands

```
bash

# Generate migration from schema changes
pnpm --filter @edusphere/db exec drizzle-kit generate

# Apply migrations
pnpm --filter @edusphere/db exec drizzle-kit migrate

# Push schema directly (dev only)
pnpm --filter @edusphere/db exec drizzle-kit push

# Open Drizzle Studio (visual DB browser)
pnpm --filter @edusphere/db exec drizzle-kit studio
```

14. Seed Data

typescript

```ts
// File: packages/db/src/seed/index.ts

import { db } from '../client.ts';
import { tenants, users, courses, modules, mediaAssets } from '../schema/index.ts';

const DEFAULT_TENANT_ID = '00000000-0000-0000-0000-000000000001';
const ADMIN_USER_ID = '00000000-0000-0000-0000-000000000010';
const STUDENT_USER_ID = '00000000-0000-0000-0000-000000000020';

export async function seed() {
  console.log('🌱 Seeding database...');

  // 1. Create default tenant
  await db.insert(tenants).values({
    id: DEFAULT_TENANT_ID,
    name: 'EduSphere Dev',
    slug: 'edusphere-dev',
    plan: 'enterprise',
    maxUsers: 1000,
    settings: {
      defaultLanguage: 'he',
      allowedLlmProviders: ['ollama', 'openai', 'anthropic'],
      maxStorageGb: 100,
    },
  }).onConflictDoNothing();

  // 2. Create test users
  await db.insert(users).values([
    {
      id: ADMIN_USER_ID,
      tenantId: DEFAULT_TENANT_ID,
      keycloakUserId: 'dev-admin-001',
      email: 'admin@edusphere.dev',
      displayName: 'Admin User',
      role: 'org_admin',
    },
    {
      id: STUDENT_USER_ID,
      tenantId: DEFAULT_TENANT_ID,
      keycloakUserId: 'dev-student-001',
      email: 'student@edusphere.dev',
      displayName: 'Test Student',
      role: 'student',
    },
  ]).onConflictDoNothing();
```

```
  // 3. Create sample course
  const [course] = await db.insert(courses).values({
    tenantId: DEFAULT_TENANT_ID,
    creatorId: ADMIN_USER_ID,
    title: 'Introduction to Semantic Learning',
    description: 'A sample course demonstrating EduSphere capabilities',
    isPublished: true,
    tags: ['demo', 'tutorial'],
  }).returning();

  // 4. Create sample module
  const [mod] = await db.insert(modules).values({
    tenantId: DEFAULT_TENANT_ID,
    courseId: course!.id,
    title: 'Module 1: Getting Started',
    orderIndex: 0,
  }).returning();

  console.log('✅ Seed complete');
  console.log(`   Tenant: ${DEFAULT_TENANT_ID}`);
  console.log(`   Admin:  ${ADMIN_USER_ID}`);
  console.log(`   Course: ${course!.id}`);
}

seed().catch(console.error);
```

---

15. TypeScript Graph Query Helpers

typescript

```typescript
// File: packages/db/src/graph/client.ts

import { sql, type PostgresJsDatabase } from 'drizzle-orm';
import { GRAPH_NAME } from '../schema/_shared.ts';

/**
 * Execute a Cypher query through Apache AGE.
 * Always loads AGE extension before execution.
 *
 * @example
 * ```typescript
 * const results = await executeCypher(db, {
 *   query: `MATCH (c:Concept {tenant_id: $tenant_id})
 *           WHERE c.label CONTAINS $search
 *           RETURN c.id, c.label LIMIT 10`,
 *   columns: ['id', 'label'],
 *   tenantId: ctx.tenantId,
 * });
 * ```
 */
export async function executeCypher<T extends Record<string, unknown>>(
  db: PostgresJsDatabase,
  options: {
    query: string;
    columns: string[];
    tenantId?: string;
  },
): Promise<T[]> {
  const { query, columns } = options;

  // Build RETURN type declaration
  const returnType = columns.map((col) => `${col} agtype`).join(', ');

  const result = await db.execute(sql`
    LOAD 'age';
    SET search_path = ag_catalog, "$user", public;
    SELECT * FROM cypher(${GRAPH_NAME}, $cypher$${sql.raw(query)}$cypher$)
    AS (${sql.raw(returnType)});
  `);

  // Parse agtype results into plain objects
  return (result.rows as any[]).map((row) => {
    const parsed: Record<string, unknown> = {};
    for (const col of columns) {
      const val = row[col];
      // agtype values come as strings, need JSON.parse for objects/arrays
```

```
      try {
        parsed[col] = typeof val === 'string' ? JSON.parse(val) : val;
      } catch {
        parsed[col] = val;
      }
    }
    return parsed as T;
  });
}

/**
 * Add a vertex to the knowledge graph.
 */
export async function addVertex(
  db: PostgresJsDatabase,
  label: string,
  properties: Record<string, unknown>,
): Promise<void> {
  const propsStr = Object.entries(properties)
    .map(([k, v]) => `${k}: ${JSON.stringify(v)}`)
    .join(', ');

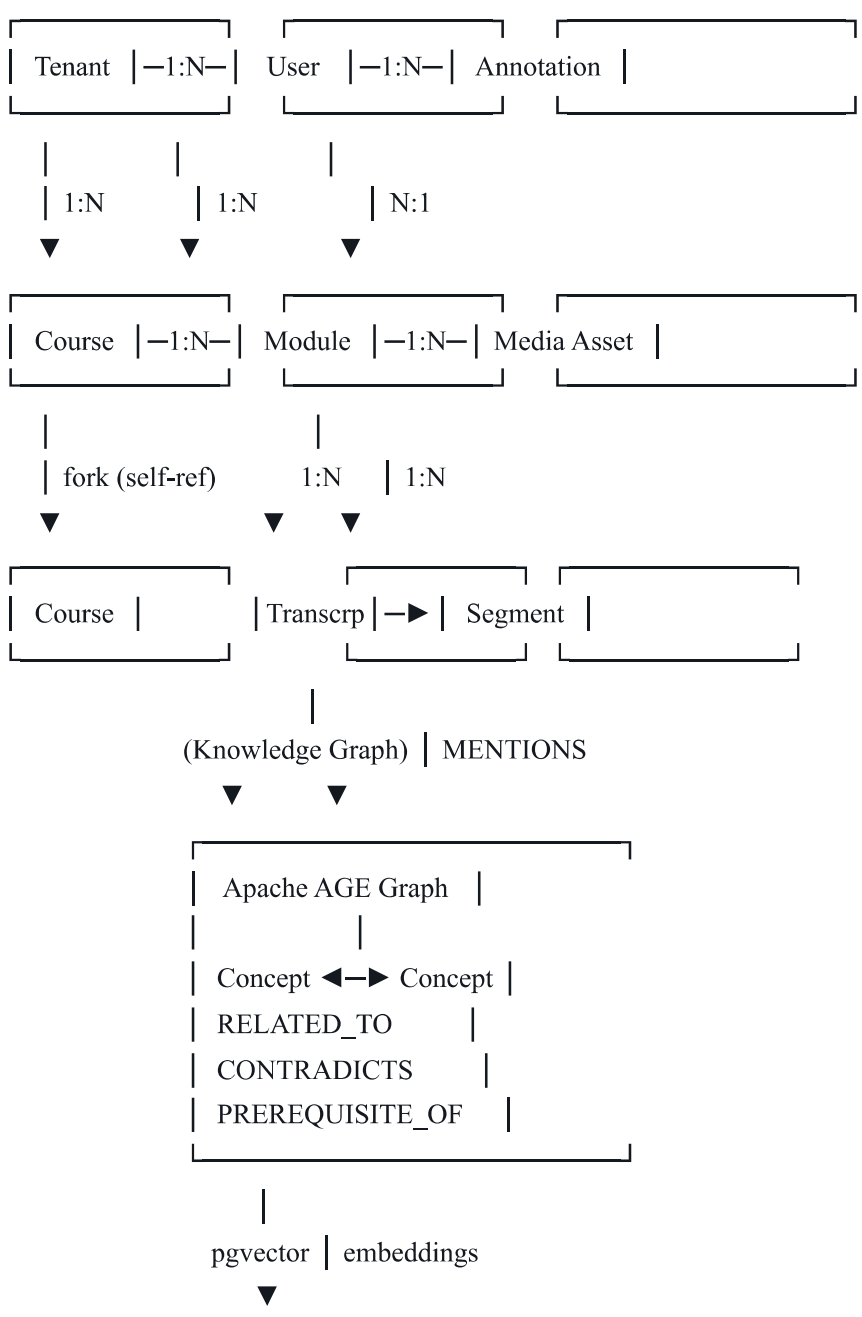
  await db.execute(sql`
    LOAD 'age';
    SET search_path = ag_catalog, "$user", public;
    SELECT * FROM cypher(${GRAPH_NAME}, $$
      CREATE (:${sql.raw(label)} {${sql.raw(propsStr)}})
    $$) AS (v agtype);
  `);
}

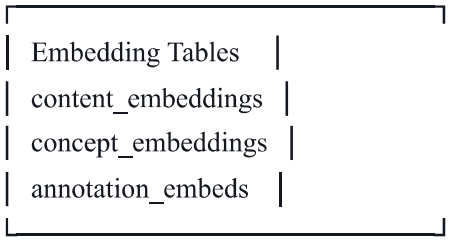
/**
 * Add an edge between two vertices.
 */
export async function addEdge(
  db: PostgresJsDatabase,
  fromLabel: string,
  fromId: string,
  toLabel: string,
  toId: string,
  edgeLabel: string,
  properties: Record<string, unknown> = {},
): Promise<void> {
  const propsStr = Object.entries(properties)
    .map(([k, v]) => `${k}: ${JSON.stringify(v)}`)
    .join(', ');
```

```
  const edgeProps = propsStr ? ` {${propsStr}}` : '';

  await db.execute(sql`
    LOAD 'age';
    SET search_path = ag_catalog, "$user", public;
    SELECT * FROM cypher(${GRAPH_NAME}, $$
      MATCH (a:${sql.raw(fromLabel)} {id: '${sql.raw(fromId)}'}),
            (b:${sql.raw(toLabel)} {id: '${sql.raw(toId)}'})
      CREATE (a)-[:${sql.raw(edgeLabel)}${sql.raw(edgeProps)}]->(b)
    $$) AS (e agtype);
  `);
}
```

Entity Relationship Summary





Total Table Count: 16

#	Table	Tenant-Isolated	RLS
1	tenants	N/A (IS tenant)	✅
2	users	✅	✅
3	courses	✅	✅
4	modules	✅	✅
5	media_assets	✅	✅
6	transcripts	✅	✅
7	transcript_segments	✅	✅
8	annotations	✅	✅
9	collab_documents	✅	✅
10	crdt_updates	✅	✅
11	collab_sessions	✅	✅
12	agent_definitions	✅	✅
13	agent_executions	✅	✅
14	content_embeddings	✅	✅
15	annotation_embeddings	✅	✅
16	concept_embeddings	✅	✅