```
-- ============================================================
-- OGS / TokenSpace / Lattice — Unified Schema (Idempotent)
-- ============================================================

-- ---------- Extensions ----------
CREATE EXTENSION IF NOT EXISTS vector;
CREATE EXTENSION IF NOT EXISTS pg_trgm;

-- ---------- Schemas ----------
CREATE SCHEMA IF NOT EXISTS content;
CREATE SCHEMA IF NOT EXISTS token;
CREATE SCHEMA IF NOT EXISTS cog;
CREATE SCHEMA IF NOT EXISTS lat;

-- ============================================================
-- CONTENT (RAG spine)
-- ============================================================
CREATE TABLE IF NOT EXISTS content.sources (
  source_id   BIGSERIAL PRIMARY KEY,
  kind        TEXT NOT NULL CHECK (kind IN ('web','file','manual','api','other')),
  uri         TEXT,
  fingerprint TEXT,
  meta        JSONB DEFAULT '{}'::jsonb,
  created_at  TIMESTAMPTZ NOT NULL DEFAULT now()
);

CREATE TABLE IF NOT EXISTS content.documents (
  doc_id      BIGSERIAL PRIMARY KEY,
  source_id   BIGINT REFERENCES content.sources(source_id) ON DELETE SET NULL,
  external_id TEXT,
  title       TEXT,
  authored_at TIMESTAMPTZ,
  meta        JSONB DEFAULT '{}'::jsonb,
  created_at  TIMESTAMPTZ NOT NULL DEFAULT now()
);

-- Chunks with embeddings and full-text similarity
CREATE TABLE IF NOT EXISTS content.chunks (
  chunk_id    BIGSERIAL PRIMARY KEY,
  doc_id      BIGINT NOT NULL REFERENCES content.documents(doc_id) ON DELETE
CASCADE,
  seq         INT NOT NULL,
  text        TEXT NOT NULL,
  token_count INT,
  embedding   VECTOR(768) NOT NULL,  -- Reduced to 768 dimensions
  lang        TEXT DEFAULT 'en',
  tags        TEXT[] DEFAULT '{}',
  meta        JSONB DEFAULT '{}'::jsonb,
  created_at  TIMESTAMPTZ NOT NULL DEFAULT now(),
  UNIQUE (doc_id, seq)
);
```

```
-- Indexes
CREATE INDEX IF NOT EXISTS documents_title_trgm ON content.documents USING GIN
((coalesce(title,'')) gin_trgm_ops);
CREATE INDEX IF NOT EXISTS chunks_text_trgm     ON content.chunks    USING GIN
((coalesce(text,'')) gin_trgm_ops);
CREATE INDEX IF NOT EXISTS chunks_embed_hnsw    ON content.chunks USING hnsw
(embedding vector_cosine_ops);
CREATE INDEX IF NOT EXISTS chunks_doc_seq_idx   ON content.chunks (doc_id, seq);
CREATE INDEX IF NOT EXISTS chunks_tags_idx      ON content.chunks USING GIN (tags);

-- ============================================================
-- TOKENSPACE / TOKENSENSE
-- ============================================================
CREATE TABLE IF NOT EXISTS token.forms (
  form_id     BIGSERIAL PRIMARY KEY,
  form_text   TEXT NOT NULL,
  norm        TEXT,
  df          BIGINT DEFAULT 0,
  meta        JSONB DEFAULT '{}'::jsonb,
  created_at  TIMESTAMPTZ NOT NULL DEFAULT now(),
  UNIQUE (form_text)
);

-- Senses as centroid-based interpretations
CREATE TABLE IF NOT EXISTS token.senses (
  sense_id    BIGSERIAL PRIMARY KEY,
  form_id     BIGINT NOT NULL REFERENCES token.forms(form_id) ON DELETE CASCADE,
  centroid    VECTOR(768) NOT NULL,  -- Reduced to 768 dimensions
  examples_n  INT DEFAULT 0,
  tags        TEXT[] DEFAULT '{}',
  meta        JSONB DEFAULT '{}'::jsonb,
  created_at  TIMESTAMPTZ NOT NULL DEFAULT now()
);
CREATE INDEX IF NOT EXISTS senses_form_idx      ON token.senses (form_id);
CREATE INDEX IF NOT EXISTS senses_centroid_hnsw ON token.senses USING hnsw (centroid
vector_cosine_ops);

-- Instances of tokens in chunks
CREATE TABLE IF NOT EXISTS token.instances (
  inst_id     BIGSERIAL PRIMARY KEY,
  sense_id    BIGINT REFERENCES token.senses(sense_id) ON DELETE SET NULL,
  form_id     BIGINT NOT NULL REFERENCES token.forms(form_id) ON DELETE CASCADE,
  chunk_id    BIGINT NOT NULL REFERENCES content.chunks(chunk_id) ON DELETE
CASCADE,
  span_start  INT NOT NULL,
  span_end    INT NOT NULL,
  ctx_embed   VECTOR(768) NOT NULL,  -- Reduced to 768 dimensions
  created_at  TIMESTAMPTZ NOT NULL DEFAULT now(),
  CONSTRAINT inst_span_ck CHECK (span_start >= 0 AND span_end > span_start)
);
CREATE INDEX IF NOT EXISTS instances_chunk_idx  ON token.instances (chunk_id);
CREATE INDEX IF NOT EXISTS instances_form_idx   ON token.instances (form_id);
```

```
CREATE INDEX IF NOT EXISTS instances_sense_idx  ON token.instances (sense_id);
CREATE INDEX IF NOT EXISTS instances_ctx_hnsw   ON token.instances USING hnsw
(ctx_embed vector_cosine_ops);

-- Co-occurrences (canonical order enforced)
CREATE TABLE IF NOT EXISTS token.cooc (
  form_id_a   BIGINT NOT NULL REFERENCES token.forms(form_id) ON DELETE
CASCADE,
  form_id_b   BIGINT NOT NULL REFERENCES token.forms(form_id) ON DELETE
CASCADE,
  weight      REAL NOT NULL,
  CONSTRAINT cooc_pk PRIMARY KEY (form_id_a, form_id_b),
  CONSTRAINT cooc_order_ck CHECK (form_id_a < form_id_b)
);

-- ============================================================
-- COGNITION
-- ============================================================
CREATE TABLE IF NOT EXISTS cog.conversations (
  convo_id    BIGSERIAL PRIMARY KEY,
  title       TEXT,
  started_at  TIMESTAMPTZ NOT NULL DEFAULT now(),
  meta        JSONB DEFAULT '{}'::jsonb
);

-- Turns in conversation
CREATE TABLE IF NOT EXISTS cog.turns (
  turn_id     BIGSERIAL PRIMARY KEY,
  convo_id    BIGINT NOT NULL REFERENCES cog.conversations(convo_id) ON DELETE
CASCADE,
  role        TEXT NOT NULL CHECK (role IN ('user','assistant','system','tool')),
  content     TEXT NOT NULL,
  embedding   VECTOR(768),  -- Reduced to 768 dimensions
  confidence  REAL,
  mode        TEXT CHECK (mode IN ('logical','philosophical','emotional','structural','unsure')),
  tags        TEXT[] DEFAULT '{}',
  created_at  TIMESTAMPTZ NOT NULL DEFAULT now()
);
CREATE INDEX IF NOT EXISTS turns_convo_time_idx ON cog.turns (convo_id, created_at);
CREATE INDEX IF NOT EXISTS turns_embed_hnsw     ON cog.turns USING hnsw (embedding
vector_cosine_ops);

-- Reflections — internal thoughts or hooks
CREATE TABLE IF NOT EXISTS cog.reflections (
  refl_id     BIGSERIAL PRIMARY KEY,
  convo_id    BIGINT REFERENCES cog.conversations(convo_id) ON DELETE CASCADE,
  turn_id     BIGINT REFERENCES cog.turns(turn_id) ON DELETE SET NULL,
  kind        TEXT NOT NULL CHECK (kind IN
('inner_thought','curiosity_hook','evaluation','memory_write')),
  content     TEXT NOT NULL,
  confidence  REAL,
  meta        JSONB DEFAULT '{}'::jsonb,
```

```
  created_at  TIMESTAMPTZ NOT NULL DEFAULT now()
);
CREATE INDEX IF NOT EXISTS refl_convo_time_idx  ON cog.reflections (convo_id,
created_at);

-- Memories
CREATE TABLE IF NOT EXISTS cog.memories (
  mem_id      BIGSERIAL PRIMARY KEY,
  scope       TEXT NOT NULL CHECK (scope IN ('fact','rule','plan','preference','identity','event')),
  text        TEXT NOT NULL,
  embedding   VECTOR(768) NOT NULL,  -- Reduced to 768 dimensions
  strength    REAL DEFAULT 0.5,
  source_ref  JSONB DEFAULT '{}'::jsonb,
  tags        TEXT[] DEFAULT '{}',
  created_at  TIMESTAMPTZ NOT NULL DEFAULT now()
);
CREATE INDEX IF NOT EXISTS memories_scope_idx   ON cog.memories (scope);
CREATE INDEX IF NOT EXISTS memories_embed_hnsw  ON cog.memories USING hnsw
(embedding vector_cosine_ops);

-- ============================================================
-- LATTICE: enums
-- ============================================================
DO $$
BEGIN
  IF NOT EXISTS (SELECT 1 FROM pg_type t JOIN pg_namespace n ON n.oid=t.typnamespace
                 WHERE t.typname='node_kind' AND n.nspname='lat') THEN
    CREATE TYPE lat.node_kind AS ENUM ('form','sense','instance','chunk','memory','turn','doc');
  END IF;

  IF NOT EXISTS (SELECT 1 FROM pg_type t JOIN pg_namespace n ON n.oid=t.typnamespace
                 WHERE t.typname='rel_kind' AND n.nspname='lat') THEN
    CREATE TYPE lat.rel_kind AS ENUM (

'cooccurs','synonym','antonym','entails','evokes','refers_to','supports','contradicts','quotes','hyperlink',
'derives_from',
      'initiates','stabilizes','closes'
    );
  END IF;

  IF NOT EXISTS (SELECT 1 FROM pg_type t JOIN pg_namespace n ON n.oid=t.typnamespace
                 WHERE t.typname='space_kind' AND n.nspname='lat') THEN
    CREATE TYPE lat.space_kind AS ENUM ('senses','contexts','memories','chunks');
  END IF;

  IF NOT EXISTS (SELECT 1 FROM pg_type t JOIN pg_namespace n ON n.oid=t.typnamespace
                 WHERE t.typname='metric_kind' AND n.nspname='lat') THEN
    CREATE TYPE lat.metric_kind AS ENUM ('cosine','l2','ip');
  END IF;

END$$;
```

```
-- ============================================================
-- LATTICE: topology, FoL shells, temporal dynamics
-- ============================================================

-- Edge connections between nodes
CREATE TABLE IF NOT EXISTS lat.edges (
  src_kind   lat.node_kind NOT NULL,
  src_id     BIGINT        NOT NULL,
  rel        lat.rel_kind  NOT NULL,
  dst_kind   lat.node_kind NOT NULL,
  dst_id     BIGINT        NOT NULL,
  weight     REAL NOT NULL DEFAULT 0.0,
  phase      REAL,  -- [-π..π], optional alignment/temporal angle
  evidence   JSONB DEFAULT '{}'::jsonb,
  created_at TIMESTAMPTZ NOT NULL DEFAULT now(),
  PRIMARY KEY (src_kind, src_id, rel, dst_kind, dst_id),
  CONSTRAINT lat_edges_weight_ck CHECK (weight >= 0),
  CONSTRAINT lat_edges_phase_ck  CHECK (phase IS NULL OR (phase >=
-3.141592653589793 AND phase <= 3.141592653589793))
);
CREATE INDEX IF NOT EXISTS lat_edges_by_src    ON lat.edges (src_kind, src_id, rel);
CREATE INDEX IF NOT EXISTS lat_edges_by_dst    ON lat.edges (dst_kind, dst_id, rel);
CREATE INDEX IF NOT EXISTS lat_edges_weight_ix ON lat.edges (rel, weight DESC);

-- Cell structure (FoL Shell Levels)
CREATE TABLE IF NOT EXISTS lat.cells (
  cell_id       BIGSERIAL PRIMARY KEY,
  space         lat.space_kind NOT NULL,
  level         INT NOT NULL,
  radial_index  INT DEFAULT 0,        -- FoL concentric layer (R in Φ^R)
  centroid      VECTOR(768) NOT NULL,  -- Reduced to 768 dimensions
  radius        REAL,
  spiral_angle  DOUBLE PRECISION,     -- radians
  radial_distance DOUBLE PRECISION,
  created_at    TIMESTAMPTZ NOT NULL DEFAULT now(),
  CONSTRAINT lat_cells_level_ck  CHECK (level >= 0),
  CONSTRAINT lat_cells_radial_ck CHECK (radial_index >= 0)
);
CREATE INDEX IF NOT EXISTS lat_cells_level_idx    ON lat.cells (space, level);
CREATE INDEX IF NOT EXISTS lat_cells_centroid_hnsw ON lat.cells USING hnsw (centroid
vector_cosine_ops);

-- Cell membership tracking
CREATE TABLE IF NOT EXISTS lat.memberships (
  space      lat.space_kind NOT NULL,
  entity_id  BIGINT NOT NULL,
  level      INT NOT NULL,
  cell_id    BIGINT NOT NULL REFERENCES lat.cells(cell_id) ON DELETE CASCADE,
  dist       REAL,
  PRIMARY KEY (space, entity_id, level)
);
CREATE INDEX IF NOT EXISTS lat_memberships_cell_idx ON lat.memberships (cell_id);
```

```
-- KNN lookup table
CREATE TABLE IF NOT EXISTS lat.neighbors (
  space       lat.space_kind NOT NULL,
  entity_id   BIGINT NOT NULL,
  neighbor_id BIGINT NOT NULL,
  metric      lat.metric_kind NOT NULL DEFAULT 'cosine',
  rank        INT NOT NULL,
  dist        REAL NOT NULL,
  PRIMARY KEY (space, entity_id, neighbor_id),
  CONSTRAINT lat_neighbors_rank_uniq UNIQUE (space, entity_id, rank)
);
CREATE INDEX IF NOT EXISTS lat_neighbors_rank_idx ON lat.neighbors (space, entity_id,
rank);

-- Activations over time
CREATE TABLE IF NOT EXISTS lat.activations (
  act_id     BIGSERIAL PRIMARY KEY,
  kind       lat.node_kind NOT NULL,
  node_id    BIGINT NOT NULL,
  source     TEXT,
  strength   REAL NOT NULL DEFAULT 1.0,
  phase      REAL,
  created_at TIMESTAMPTZ NOT NULL DEFAULT now(),
  CONSTRAINT lat_act_strength_ck CHECK (strength >= 0),
  CONSTRAINT lat_act_phase_ck    CHECK (phase IS NULL OR (phase >= -3.141592653589793
AND phase <= 3.141592653589793))
);
CREATE INDEX IF NOT EXISTS lat_activations_node_time_idx ON lat.activations (kind,
node_id, created_at);

-- Toroidal projections
CREATE TABLE IF NOT EXISTS lat.torus (
  space       lat.space_kind NOT NULL,
  entity_id   BIGINT NOT NULL,
  u           DOUBLE PRECISION NOT NULL CHECK (u >= 0 AND u < 1),
  v           DOUBLE PRECISION NOT NULL CHECK (v >= 0 AND v < 1),
  level       INT NOT NULL DEFAULT 0,
  updated_at  TIMESTAMPTZ NOT NULL DEFAULT now(),
  PRIMARY KEY (space, entity_id, level)
);

-- Spiral/topological projections
CREATE TABLE IF NOT EXISTS lat.projections (
  proj_id   BIGSERIAL PRIMARY KEY,
  kind      TEXT NOT NULL CHECK (kind IN ('spiral','toroid','force2d','force3d')),
  node_kind lat.node_kind NOT NULL,
  node_id   BIGINT NOT NULL,
  theta     DOUBLE PRECISION,
  radius    DOUBLE PRECISION,
  x         DOUBLE PRECISION,
  y         DOUBLE PRECISION,
```

```
  z         DOUBLE PRECISION,
  level     INT DEFAULT 0,
  created_at TIMESTAMPTZ NOT NULL DEFAULT now()
);
CREATE INDEX IF NOT EXISTS lat_proj_node_idx ON lat.projections (node_kind, node_id,
kind, level);

-- Topology change log
CREATE TABLE IF NOT EXISTS lat.topology_events (
  evt_id     BIGSERIAL PRIMARY KEY,
  evt_kind   TEXT NOT NULL CHECK (evt_kind IN
('edge_add','edge_update','edge_prune','cell_split','cell_merge','membership_move','neighbor_refres
h')),
  payload    JSONB NOT NULL DEFAULT '{}',
  created_at TIMESTAMPTZ NOT NULL DEFAULT now()
);

-- Lattice configuration (including energy formula tunables)
CREATE TABLE IF NOT EXISTS lat.config (
  key        TEXT PRIMARY KEY,
  value_text  TEXT,
  value_real  REAL,
  description TEXT,
  created_at  TIMESTAMPTZ NOT NULL DEFAULT now()
);
CREATE UNIQUE INDEX IF NOT EXISTS lat_config_key_idx ON lat.config (key);

INSERT INTO lat.config (key, value_real, description) VALUES
  ('golden_ratio_phi', 1.6180339887, 'Golden Ratio Φ'),
  ('damping_factor_k', 5.0,          'Damping factor k'),
  ('oscillatory_frequency_k', 0.1,    'Frequency factor k in sin(k·t)'),
  ('S_w_chunk',  0.34, 'weight of chunk density into S'),
  ('S_w_sense',  0.33, 'weight of sense support into S'),
  ('S_w_memory', 0.33, 'weight of memory strength into S')
ON CONFLICT (key) DO UPDATE
SET value_real = EXCLUDED.value_real,
    description = EXCLUDED.description;

-- ============================================================
-- VIEWS: Unified lattice logic and dynamics
-- ============================================================

-- Co-occurrence edges mapped to lattice edges
CREATE OR REPLACE VIEW lat.cooc_edges AS
SELECT
  'form'::lat.node_kind AS src_kind,
  c.form_id_a           AS src_id,
  'cooccurs'::lat.rel_kind AS rel,
  'form'::lat.node_kind AS dst_kind,
  c.form_id_b           AS dst_id,
  c.weight              AS weight,
  NULL::REAL            AS phase,
```

```
  jsonb_build_object('source','token.cooc') AS evidence,
  now()               AS created_at
FROM token.cooc c
UNION ALL
SELECT
  'form'::lat.node_kind,
  c.form_id_b,
  'cooccurs'::lat.rel_kind,
  'form'::lat.node_kind,
  c.form_id_a,
  c.weight,
  NULL::REAL,
  jsonb_build_object('source','token.cooc'),
  now()
FROM token.cooc c;

-- Unified node view across all kinds
CREATE OR REPLACE VIEW lat.nodes AS
SELECT 'form'::lat.node_kind AS kind, f.form_id AS node_id, f.form_text AS label, NULL::vector
AS embedding, f.created_at
FROM token.forms f
UNION ALL
SELECT 'sense'::lat.node_kind, s.sense_id, f.form_text||' · sense #'||s.sense_id::text, s.centroid,
s.created_at
FROM token.senses s JOIN token.forms f ON f.form_id=s.form_id
UNION ALL
SELECT 'chunk'::lat.node_kind, ch.chunk_id, 'chunk '||ch.chunk_id::text, ch.embedding,
ch.created_at
FROM content.chunks ch
UNION ALL
SELECT 'doc'::lat.node_kind, d.doc_id, coalesce(d.title,'doc '||d.doc_id::text), NULL::vector,
d.created_at
FROM content.documents d
UNION ALL
SELECT 'memory'::lat.node_kind, m.mem_id, left(m.text,80), m.embedding, m.created_at
FROM cog.memories m
UNION ALL
SELECT 'turn'::lat.node_kind, t.turn_id, t.role||' turn '||t.turn_id::text, t.embedding, t.created_at
FROM cog.turns t;

-- Cell view with φ^R calculations
CREATE OR REPLACE VIEW lat.cell_phi AS
SELECT
  c.cell_id, c.space, c.level, c.radial_index,
  c.centroid, c.radius, c.spiral_angle, c.radial_distance,
  (SELECT value_real FROM lat.config WHERE key='golden_ratio_phi') AS phi,
  power((SELECT value_real FROM lat.config WHERE key='golden_ratio_phi'), c.radial_index)
AS phi_pow_r
FROM lat.cells c;

-- Config weight view
CREATE OR REPLACE VIEW lat._cfg AS
```

```
SELECT
 (SELECT value_real FROM lat.config WHERE key='S_w_chunk')  AS w_chunk,
 (SELECT value_real FROM lat.config WHERE key='S_w_sense')  AS w_sense,
 (SELECT value_real FROM lat.config WHERE key='S_w_memory') AS w_memory;

-- Calculate S(r,t) derived value
CREATE OR REPLACE VIEW lat.sense_energy AS
WITH cfg AS (SELECT * FROM lat._cfg),
inst AS (
  SELECT s.sense_id,
         count(*)::float AS n_inst,
         avg(least(greatest(ch.token_count,0), 4096))::float AS avg_tokens
  FROM token.senses s
  LEFT JOIN token.instances i ON i.sense_id=s.sense_id
  LEFT JOIN content.chunks ch ON ch.chunk_id=i.chunk_id
  GROUP BY s.sense_id
),
mem AS (
  SELECT e.src_id AS sense_id, avg(m.strength)::float AS avg_mem_strength
  FROM lat.edges e
  JOIN cog.memories m ON (e.dst_kind = 'memory'::lat.node_kind AND e.dst_id = m.mem_id)
  WHERE e.src_kind = 'sense'::lat.node_kind
  GROUP BY e.src_id
)
SELECT
  s.sense_id,
  coalesce(inst.n_inst,0)                    AS n_inst,
  coalesce(inst.avg_tokens,0)                AS avg_tokens,
  coalesce(mem.avg_mem_strength,0)           AS avg_mem_strength,
  (1 - exp(-coalesce(inst.n_inst,0)/10.0))   AS n_inst_nz,
  least(coalesce(inst.avg_tokens,0)/2048.0, 1.0) AS tokens_nz,
  least(coalesce(mem.avg_mem_strength,0), 1.0)   AS mem_nz,
  (SELECT w_chunk FROM cfg)  * least(coalesce(inst.avg_tokens,0)/2048.0, 1.0) +
  (SELECT w_sense FROM cfg)  * (1 - exp(-coalesce(inst.n_inst,0)/10.0)) +
  (SELECT w_memory FROM cfg) * least(coalesce(mem.avg_mem_strength,0), 1.0) AS S
FROM token.senses s
LEFT JOIN inst USING (sense_id)
LEFT JOIN mem  USING (sense_id);

-- Unified edge influence score
CREATE OR REPLACE VIEW lat.edge_influence AS
WITH a_recent AS (
  SELECT kind, node_id, sum(strength) AS act24
  FROM lat.activations
  WHERE created_at > now() - interval '24 hours'
  GROUP BY 1,2
),
sense_S AS (SELECT sense_id, S FROM lat.sense_energy),
end_S AS (
  SELECT e.src_kind, e.src_id,
         CASE WHEN e.src_kind='sense'::lat.node_kind THEN s.S ELSE NULL END AS S_src
  FROM lat.edges e
```

```
  LEFT JOIN sense_S s ON (e.src_kind='sense'::lat.node_kind AND e.src_id=s.sense_id)
),
dst_S AS (
  SELECT e.dst_kind, e.dst_id,
        CASE WHEN e.dst_kind='sense'::lat.node_kind THEN s.S ELSE NULL END AS S_dst
  FROM lat.edges e
  LEFT JOIN sense_S s ON (e.dst_kind='sense'::lat.node_kind AND e.dst_id=s.sense_id)
)
SELECT
  e.*,
  coalesce(a1.act24,0) AS src_act24,
  coalesce(a2.act24,0) AS dst_act24,
  coalesce(es.S_src,0) AS S_src,
  coalesce(ds.S_dst,0) AS S_dst,
  (e.weight*0.6) + (least(coalesce(a1.act24,0) + coalesce(a2.act24,0), 10)/10.0)*0.2 +
  (least(coalesce(es.S_src,0)+coalesce(ds.S_dst,0),2)/2.0)*0.2 AS influence
FROM lat.edges e
LEFT JOIN a_recent a1 ON a1.kind=e.src_kind AND a1.node_id=e.src_id
LEFT JOIN a_recent a2 ON a2.kind=e.dst_kind AND a2.node_id=e.dst_id
LEFT JOIN end_S es    ON es.src_kind=e.src_kind AND es.src_id=e.src_id
LEFT JOIN dst_S ds    ON ds.dst_kind=e.dst_kind AND ds.dst_id=e.dst_id;

-- ============================================================
-- HYGIENE: cleanup edges/activations on delete (no FKs possible)
-- ============================================================
CREATE OR REPLACE FUNCTION lat._del_edges_for(k lat.node_kind, i BIGINT)
RETURNS void LANGUAGE sql AS $$
  DELETE FROM lat.edges WHERE (src_kind = k AND src_id = i) OR (dst_kind = k AND dst_id
= i);
$$;

CREATE OR REPLACE FUNCTION lat._del_acts_for(k lat.node_kind, i BIGINT)
RETURNS void LANGUAGE sql AS $$
  DELETE FROM lat.activations WHERE kind = k AND node_id = i;
$$;

-- Triggers
CREATE OR REPLACE FUNCTION lat._cleanup_after_form() RETURNS trigger LANGUAGE
plpgsql AS $$
BEGIN
  PERFORM lat._del_edges_for('form',   OLD.form_id);
  PERFORM lat._del_acts_for('form',    OLD.form_id);
  RETURN NULL;
END$$;

CREATE OR REPLACE FUNCTION lat._cleanup_after_sense() RETURNS trigger LANGUAGE
plpgsql AS $$
BEGIN
  PERFORM lat._del_edges_for('sense',  OLD.sense_id);
  PERFORM lat._del_acts_for('sense',   OLD.sense_id);
  RETURN NULL;
END$$;
```

```
CREATE OR REPLACE FUNCTION lat._cleanup_after_chunk() RETURNS trigger LANGUAGE
plpgsql AS $$
BEGIN
  PERFORM lat._del_edges_for('chunk',  OLD.chunk_id);
  PERFORM lat._del_acts_for('chunk',   OLD.chunk_id);
  RETURN NULL;
END$$;

CREATE OR REPLACE FUNCTION lat._cleanup_after_memory() RETURNS trigger
LANGUAGE plpgsql AS $$
BEGIN
  PERFORM lat._del_edges_for('memory', OLD.mem_id);
  PERFORM lat._del_acts_for('memory',  OLD.mem_id);
  RETURN NULL;
END$$;

CREATE OR REPLACE FUNCTION lat._cleanup_after_turn() RETURNS trigger LANGUAGE
plpgsql AS $$
BEGIN
  PERFORM lat._del_edges_for('turn',   OLD.turn_id);
  PERFORM lat._del_acts_for('turn',    OLD.turn_id);
  RETURN NULL;
END$$;

CREATE OR REPLACE FUNCTION lat._cleanup_after_doc() RETURNS trigger LANGUAGE
plpgsql AS $$
BEGIN
  PERFORM lat._del_edges_for('doc',    OLD.doc_id);
  PERFORM lat._del_acts_for('doc',     OLD.doc_id);
  RETURN NULL;
END$$;

-- Trigger bindings (idempotent)
DO $$
BEGIN
  IF NOT EXISTS (SELECT 1 FROM pg_trigger WHERE tgname = '_lat_cleanup_form') THEN
    CREATE TRIGGER _lat_cleanup_form   AFTER DELETE ON token.forms       FOR EACH
ROW EXECUTE FUNCTION lat._cleanup_after_form();
  END IF;
  IF NOT EXISTS (SELECT 1 FROM pg_trigger WHERE tgname = '_lat_cleanup_sense') THEN
    CREATE TRIGGER _lat_cleanup_sense  AFTER DELETE ON token.senses      FOR EACH
ROW EXECUTE FUNCTION lat._cleanup_after_sense();
  END IF;
  IF NOT EXISTS (SELECT 1 FROM pg_trigger WHERE tgname = '_lat_cleanup_chunk') THEN
    CREATE TRIGGER _lat_cleanup_chunk  AFTER DELETE ON content.chunks    FOR EACH
ROW EXECUTE FUNCTION lat._cleanup_after_chunk();
  END IF;
  IF NOT EXISTS (SELECT 1 FROM pg_trigger WHERE tgname = '_lat_cleanup_mem') THEN
    CREATE TRIGGER _lat_cleanup_mem    AFTER DELETE ON cog.memories      FOR EACH
ROW EXECUTE FUNCTION lat._cleanup_after_memory();
  END IF;
```

```
  IF NOT EXISTS (SELECT 1 FROM pg_trigger WHERE tgname = '_lat_cleanup_turn') THEN
    CREATE TRIGGER _lat_cleanup_turn   AFTER DELETE ON cog.turns         FOR EACH ROW
EXECUTE FUNCTION lat._cleanup_after_turn();
  END IF;
  IF NOT EXISTS (SELECT 1 FROM pg_trigger WHERE tgname = '_lat_cleanup_doc') THEN
    CREATE TRIGGER _lat_cleanup_doc    AFTER DELETE ON content.documents FOR EACH
ROW EXECUTE FUNCTION lat._cleanup_after_doc();
  END IF;
END$$;
```