```
--
╔═══════════════════════════════════════════════════════════════════════════════╗
-- ║  AEQUITAVERSE — MOTEUR CHAINQL                                                ║
-- ║  Rôle    : Requêtes relationnelles sur les 3 chaînes XML                      ║
-- ║  Syntaxe : SQL étendu avec fonctions XPath natives                            ║
-- ║  Accès   : Public (lecture) — authentifié (écriture/signalement)              ║
-- ║  Latence : < 100ms sur index primaires                                        ║
--
╚═══════════════════════════════════════════════════════════════════════════════╝



--
═══════════════════════════════════════════════════════════════════════════════
-- PARTIE 1 — SCHÉMA RELATIONNEL
-- Tables dérivées automatiquement depuis les 3 chaînes XML
-- Mises à jour en temps réel à chaque nouveau bloc validé
--
═══════════════════════════════════════════════════════════════════════════════


-- ─────────────────────────────────────────────────────────────────────────────
-- TABLE MAÎTRE : Tous les blocs des 3 chaînes
-- ─────────────────────────────────────────────────────────────────────────────
CREATE TABLE blocs (
  id              VARCHAR(30)  PRIMARY KEY,
  hash            CHAR(70)     NOT NULL UNIQUE,
  hash_precedent  CHAR(70)     NOT NULL,
  hauteur         BIGINT       NOT NULL,
  timestamp       TIMESTAMPTZ  NOT NULL,
  type            VARCHAR(20)  NOT NULL,
```

```
  chaine            VARCHAR(10)  NOT NULL,  -- PRINCIPALE | CASINO | PARTAGEE
  territoire        VARCHAR(10),           -- AURUM | NEXUS | CIVITAS | ...
  noeud_emetteur_id     VARCHAR(50)  NOT NULL,
  masse_monetaire_apres NUMERIC(25,6),
  -- Contrainte chaîne : hash_precedent doit exister dans la table
  CONSTRAINT fk_hash_precedent
    FOREIGN KEY (hash_precedent)
    REFERENCES blocs(hash)
    DEFERRABLE INITIALLY DEFERRED
);

-- Index critiques pour performance
CREATE INDEX idx_blocs_chaine_type     ON blocs(chaine, type);
CREATE INDEX idx_blocs_timestamp       ON blocs(timestamp DESC);
CREATE INDEX idx_blocs_hauteur_chaine   ON blocs(chaine, hauteur);
CREATE INDEX idx_blocs_territoire      ON blocs(territoire);

-- ─────────────────────────────────────────────────────────────────
-- CHAÎNE PRINCIPALE — Producteurs
-- ─────────────────────────────────────────────────────────────────
CREATE TABLE producteurs (
  id                  VARCHAR(80)   PRIMARY KEY,
  type                VARCHAR(40)   NOT NULL,
  pseudonyme              VARCHAR(100)  NOT NULL UNIQUE,
  territoire              VARCHAR(10),
  membre_depuis           DATE        NOT NULL,
  certification_organisme   VARCHAR(50),
  certification_numero      VARCHAR(50),
  certification_validite    DATE,
```

```
  fiabilite_score         NUMERIC(5,2)  DEFAULT 50.00,
  productions_validees    INTEGER       DEFAULT 0,
  incidents               INTEGER       DEFAULT 0,
  caution_montant_GU      NUMERIC(25,6) DEFAULT 0,
  caution_taux_pct        NUMERIC(5,2),
  portefeuille_adresse    VARCHAR(100),
  solde_disponible_GU     NUMERIC(25,6) DEFAULT 0,
  solde_bloque_GU         NUMERIC(25,6) DEFAULT 0,
  statut                  VARCHAR(20)   DEFAULT 'ACTIF',
  -- ACTIF | SUSPENDU | BANNI | EN_PROBATION
  date_suspension         DATE,
  date_fin_suspension     DATE
);

CREATE INDEX idx_producteurs_fiabilite  ON producteurs(fiabilite_score DESC);
CREATE INDEX idx_producteurs_territoire ON producteurs(territoire);
CREATE INDEX idx_producteurs_statut     ON producteurs(statut);

-- ═══════════════════════════════════════════════════════════════
-- CHAÎNE PRINCIPALE — Productions
-- ═══════════════════════════════════════════════════════════════
CREATE TABLE productions (
  id                  VARCHAR(50)   PRIMARY KEY,
  bloc_id             VARCHAR(30)   NOT NULL REFERENCES blocs(id),
  producteur_id       VARCHAR(80)   NOT NULL REFERENCES producteurs(id),
  statut              VARCHAR(20)   NOT NULL,
  cycle               VARCHAR(50),
  territoire          VARCHAR(10),
  -- Denrée
```

```
  denree_type        VARCHAR(30)   NOT NULL,
  denree_grade       VARCHAR(5)    NOT NULL,
  quantite_valeur    NUMERIC(20,3) NOT NULL,
  quantite_unite     VARCHAR(20)   NOT NULL,
  -- Localisation
  pays               VARCHAR(60),
  region             VARCHAR(80),
  lat                NUMERIC(9,6),
  lng                NUMERIC(9,6),
  silo_id            VARCHAR(50),
  -- Oracles
  oracle_satellite_ref  VARCHAR(100),
  oracle_sat_ndvi    NUMERIC(5,3),
  oracle_sat_superficie_ha NUMERIC(10,2),
  oracle_meteo_ref   VARCHAR(100),
  oracle_meteo_precipmm NUMERIC(8,2),
  oracle_meteo_temp_moy NUMERIC(5,2),
  oracle_marche_prix_ref NUMERIC(15,10),
  -- Timestamp
  timestamp          TIMESTAMPTZ   NOT NULL,
  -- Cohérence oracles : calculée à l'insertion
  divergence_max_pct   NUMERIC(5,2),
  -- > 5% → production bloquée automatiquement
  CONSTRAINT chk_divergence CHECK (divergence_max_pct <= 5.00
    OR statut IN ('REJETÉ', 'SUSPENDU', 'CONTESTÉ'))
);

CREATE INDEX idx_prod_producteur   ON productions(producteur_id);
CREATE INDEX idx_prod_denree_type  ON productions(denree_type);
```

```
CREATE INDEX idx_prod_timestamp     ON productions(timestamp DESC);
CREATE INDEX idx_prod_pays_region   ON productions(pays, region);
CREATE INDEX idx_prod_statut        ON productions(statut);
CREATE INDEX idx_prod_territoire    ON productions(territoire);

-- ═══════════════════════════════════════════════════════════════
-- CHAÎNE PRINCIPALE — Émissions
-- ═══════════════════════════════════════════════════════════════
CREATE TABLE emissions (
 id                VARCHAR(50)   PRIMARY KEY,
 bloc_id           VARCHAR(30)   NOT NULL REFERENCES blocs(id),
 production_id     VARCHAR(50)   NOT NULL REFERENCES productions(id),
 producteur_id     VARCHAR(80)   NOT NULL REFERENCES producteurs(id),
 -- Formule
 formule_version   VARCHAR(10)   NOT NULL,
 Q_grammes         NUMERIC(25,3),
 P_ref             NUMERIC(20,13),
 F_qualite         NUMERIC(5,4),
 F_region          NUMERIC(5,4),
 F_carbone         NUMERIC(5,4),
 F_temps           NUMERIC(5,4),
 -- Résultats
 micro_GU_bruts    NUMERIC(25,6) NOT NULL,
 taxe_creation_GU  NUMERIC(25,6) NOT NULL,
 micro_GU_nets     NUMERIC(25,6) NOT NULL,
 taxe_pct          NUMERIC(5,2)  NOT NULL DEFAULT 1.00,
 -- Statut paiement taxe
 taxe_statut       VARCHAR(20)   NOT NULL,
 taxe_tx_id        VARCHAR(60),
```

```
  taxe_timestamp        TIMESTAMPTZ,
  -- Masse monétaire
  masse_avant_GU        NUMERIC(25,6),
  masse_apres_GU        NUMERIC(25,6),
  -- Timestamp
  timestamp             TIMESTAMPTZ   NOT NULL,
  -- Règle absolue : émission impossible si taxe non payée
  CONSTRAINT chk_taxe_payee CHECK (
    taxe_statut = 'PAYÉE'
    OR micro_GU_nets = 0
  )
);

CREATE INDEX idx_emis_production    ON emissions(production_id);
CREATE INDEX idx_emis_producteur    ON emissions(producteur_id);
CREATE INDEX idx_emis_timestamp     ON emissions(timestamp DESC);
CREATE INDEX idx_emis_montant       ON emissions(micro_GU_nets DESC);

-- ═══════════════════════════════════════════════════════════════
-- CHAÎNE PRINCIPALE — Transactions
-- ═══════════════════════════════════════════════════════════════
CREATE TABLE transactions (
  id                VARCHAR(60)   PRIMARY KEY,
  bloc_id           VARCHAR(30)   NOT NULL REFERENCES blocs(id),
  type              VARCHAR(30)   NOT NULL,
  chaine            VARCHAR(10)   NOT NULL,
  expediteur_id     VARCHAR(80)   NOT NULL,
  destinataire_id   VARCHAR(80)   NOT NULL,
  montant_GU        NUMERIC(25,6) NOT NULL,
```

```
  frais_reseau_GU      NUMERIC(25,6) DEFAULT 0,
  -- Sous-jacent physique (si applicable)
  sous_jacent_desc     TEXT,
  sous_jacent_prod_ref  VARCHAR(50)   REFERENCES productions(id),
  bon_livraison        VARCHAR(60),
  livraison_prevue     DATE,
  -- Conversion fiat (si applicable)
  montant_fiat_valeur   NUMERIC(20,2),
  montant_fiat_devise   VARCHAR(10),
  taux_conversion      NUMERIC(15,8),
  facteur_penalite     NUMERIC(5,1),
  -- Soldes
  solde_expediteur_avant  NUMERIC(25,6),
  solde_expediteur_apres  NUMERIC(25,6),
  -- Validation
  signature_expediteur  VARCHAR(80),
  timestamp           TIMESTAMPTZ   NOT NULL,
  timestamp_confirmation TIMESTAMPTZ,
  -- Indexation productivité pour taxe modulée
  type_economique      VARCHAR(20)   DEFAULT 'COMMERCIAL',
  -- PRODUCTIF | COMMERCIAL | FINANCIER_PUR | SPECULATIF | SUSPECT
  multiplicateur_taxe   NUMERIC(4,2)  DEFAULT 1.00
);

CREATE INDEX idx_tx_expediteur   ON transactions(expediteur_id);
CREATE INDEX idx_tx_destinataire ON transactions(destinataire_id);
CREATE INDEX idx_tx_timestamp    ON transactions(timestamp DESC);
CREATE INDEX idx_tx_type         ON transactions(type);
CREATE INDEX idx_tx_type_eco     ON transactions(type_economique);
```

```
CREATE INDEX idx_tx_montant     ON transactions(montant_GU DESC);


-- ═══════════════════════════════════════════════════════════════════
-- CHAÎNE CASINO — Contrats d'options
-- ═══════════════════════════════════════════════════════════════════
CREATE TABLE contrats_option (
 id                 VARCHAR(50)   PRIMARY KEY,
 bloc_id            VARCHAR(30)   NOT NULL REFERENCES blocs(id),
 type_option        VARCHAR(5)    NOT NULL, -- CALL | PUT
 statut             VARCHAR(10)   NOT NULL,
 -- Acheteur / Vendeur (pseudonymes dans le Casino)
 acheteur_pseudonyme    VARCHAR(100)  NOT NULL,
 acheteur_adresse       VARCHAR(100),
 acheteur_pct_marche    NUMERIC(8,5),
 vendeur_pseudonyme     VARCHAR(100)  NOT NULL,
 vendeur_adresse        VARCHAR(100),
 vendeur_prod_ref       VARCHAR(50)   REFERENCES productions(id),
 -- Sous-jacent
 commodite          VARCHAR(30)   NOT NULL,
 quantite_kg        NUMERIC(20,3) NOT NULL,
 pct_production_couverte NUMERIC(5,2),
 -- Paramètres
 prix_exercice_GU       NUMERIC(15,8) NOT NULL,
 echeance           TIMESTAMPTZ   NOT NULL,
 prime_payee_GU         NUMERIC(25,6) NOT NULL,
 levier             VARCHAR(6)    NOT NULL,
 levier_valeur          INTEGER       NOT NULL,
 taux_taxe_pct          NUMERIC(5,2)  NOT NULL,
 -- Taxe Casino
```

```
  taxe_montant_GU       NUMERIC(25,6) NOT NULL,
  taxe_statut           VARCHAR(20)   NOT NULL,
  taxe_tx_id            VARCHAR(60),
  -- Mouvement net
  mouvement_net_GU      NUMERIC(25,6) NOT NULL,
  -- Résultat à l'échéance
  prix_marche_echeance_GU NUMERIC(15,8),
  gain_acheteur_GU      NUMERIC(25,6),
  -- Timestamps
  timestamp_emission    TIMESTAMPTZ   NOT NULL,
  timestamp_cloture     TIMESTAMPTZ,
  CONSTRAINT chk_taxe_casino CHECK (taxe_statut = 'PAYÉE')
);

CREATE INDEX idx_opt_commodite   ON contrats_option(commodite);
CREATE INDEX idx_opt_statut      ON contrats_option(statut);
CREATE INDEX idx_opt_echeance    ON contrats_option(echeance);
CREATE INDEX idx_opt_levier      ON contrats_option(levier_valeur);
CREATE INDEX idx_opt_acheteur    ON contrats_option(acheteur_pseudonyme);

-- ═══════════════════════════════════════════════════════════════
-- CHAÎNE CASINO — Passages membrane Casino → Réel
-- ═══════════════════════════════════════════════════════════════
CREATE TABLE passages_membrane (
  id                    VARCHAR(50)  PRIMARY KEY,
  bloc_casino_id        VARCHAR(30)  NOT NULL REFERENCES blocs(id),
  bloc_principal_genere_id  VARCHAR(30)  REFERENCES blocs(id),
  contrat_ref           VARCHAR(50)  REFERENCES contrats_option(id),
  acteur_pseudonyme     VARCHAR(100) NOT NULL,
```

```
 montant_transfere_GU       NUMERIC(25,6) NOT NULL,
 taxe_membrane_GU           NUMERIC(25,6) NOT NULL,
 montant_net_GU             NUMERIC(25,6) NOT NULL,
 livraison_physique_possible BOOLEAN     DEFAULT FALSE,
 silo_id                    VARCHAR(50),
 quantite_kg                NUMERIC(20,3),
 bon_livraison              VARCHAR(60),
 timestamp                  TIMESTAMPTZ  NOT NULL
);

CREATE INDEX idx_membrane_timestamp ON passages_membrane(timestamp DESC);
CREATE INDEX idx_membrane_acteur    ON passages_membrane(acteur_pseudonyme);

-- ═══════════════════════════════════════════════════════════════════════
-- CHAÎNE PARTAGÉE — Fonds Commun (journal comptable)
-- ═══════════════════════════════════════════════════════════════════════
CREATE TABLE fonds_commun_journal (
 id                 VARCHAR(60)   PRIMARY KEY,
 type_mouvement     VARCHAR(40)   NOT NULL,
 -- ENTREE_TAXE_CREATION | ENTREE_TAXE_CASINO | ENTREE_TAXE_MEMBRANE
 -- ENTREE_TAXE_BANCAIRE | ENTREE_FRAIS_RESEAU | ENTREE_RECUPERATION_FRAUDE
 -- SORTIE_INFRASTRUCTURE | SORTIE_AIDE_PRODUCTEUR | SORTIE_RECHERCHE
 -- SORTIE_STABILISATION | SORTIE_RECOMPENSE_AUDITEUR | SORTIE_URGENCE
 direction          VARCHAR(7)    NOT NULL, -- ENTREE | SORTIE
 montant_GU         NUMERIC(25,6) NOT NULL,
 poste_id           VARCHAR(5),
 source_bloc_id     VARCHAR(30)   REFERENCES blocs(id),
 source_tx_id       VARCHAR(60),
 beneficiaire_id    VARCHAR(80),
```

```
  solde_avant_GU     NUMERIC(25,6) NOT NULL,
  solde_apres_GU     NUMERIC(25,6) NOT NULL,
  exercice           INTEGER       NOT NULL,
  timestamp          TIMESTAMPTZ   NOT NULL,
  note               TEXT
);

CREATE INDEX idx_fc_type_mouvement ON fonds_commun_journal(type_mouvement);
CREATE INDEX idx_fc_direction      ON fonds_commun_journal(direction);
CREATE INDEX idx_fc_timestamp      ON fonds_commun_journal(timestamp DESC);
CREATE INDEX idx_fc_exercice       ON fonds_commun_journal(exercice);
CREATE INDEX idx_fc_poste          ON fonds_commun_journal(poste_id);

-- ═══════════════════════════════════════════════════════════════════════
-- CHAÎNE PARTAGÉE — Auditeurs et signalements
-- ═══════════════════════════════════════════════════════════════════════
CREATE TABLE auditeurs (
  id                 VARCHAR(60)   PRIMARY KEY,
  pseudonyme         VARCHAR(100)  NOT NULL UNIQUE,
  niveau             VARCHAR(15)   NOT NULL,
  territoire_principal  VARCHAR(10),
  membre_depuis      DATE          NOT NULL,
  audits_completes   INTEGER       DEFAULT 0,
  fraudes_detectees  INTEGER       DEFAULT 0,
  faux_positifs      INTEGER       DEFAULT 0,
  taux_confirmation_pct  NUMERIC(5,2)  DEFAULT 0,
  recompenses_cumulees_GU NUMERIC(25,6) DEFAULT 0,
  reputation_score   NUMERIC(5,2)  DEFAULT 50.00,
  caution_depositee_GU  NUMERIC(25,6) DEFAULT 0,
```

```
  statut              VARCHAR(20)   DEFAULT 'ACTIF',
  avertissements      INTEGER       DEFAULT 0
);

CREATE TABLE signalements (
  id                  VARCHAR(60)   PRIMARY KEY,
  auditeur_id         VARCHAR(60)   NOT NULL REFERENCES auditeurs(id),
  production_cible_id VARCHAR(50)   REFERENCES productions(id),
  bloc_cible_id       VARCHAR(30)   REFERENCES blocs(id),
  type_fraude_suspectee VARCHAR(40) NOT NULL,
  z_score_energie     NUMERIC(8,4),
  z_score_travail     NUMERIC(8,4),
  pattern_detecte     VARCHAR(40),
  preuve_chainql      TEXT,
  caution_auditeur_GU NUMERIC(25,6) NOT NULL,
  statut              VARCHAR(20)   NOT NULL DEFAULT 'EN_ATTENTE',
  verdict             VARCHAR(30),
  montant_fraude_GU   NUMERIC(25,6),
  recompense_versee_GU NUMERIC(25,6),
  timestamp_signalement TIMESTAMPTZ NOT NULL,
  timestamp_verdict   TIMESTAMPTZ
);

CREATE INDEX idx_signal_auditeur  ON signalements(auditeur_id);
CREATE INDEX idx_signal_statut    ON signalements(statut);
CREATE INDEX idx_signal_timestamp ON signalements(timestamp_signalement DESC);

-- ═══════════════════════════════════════════════════════════════

-- VUES MATÉRIALISÉES — Rafraîchies à chaque nouveau bloc
```

```
-- ─────────────────────────────────────────────────────────────────────

-- Vue : Production mondiale agrégée par région et denrée
CREATE MATERIALIZED VIEW mv_production_regionale AS
SELECT
  p.pays,
  p.region,
  p.territoire,
  p.denree_type,
  p.denree_grade,
  DATE_TRUNC('month', p.timestamp)      AS mois,
  COUNT(*)                      AS nb_productions,
  SUM(p.quantite_valeur)              AS quantite_totale,
  p.quantite_unite,
  AVG(p.oracle_marche_prix_ref)        AS prix_moyen_GU,
  SUM(e.micro_GU_nets)               AS GU_emis,
  SUM(e.taxe_creation_GU)             AS taxes_au_fonds_commun,
  AVG(p.divergence_max_pct)           AS divergence_oracle_moy,
  COUNT(*) FILTER (WHERE p.statut = 'REJETÉ') AS rejets
FROM productions p
JOIN emissions e ON e.production_id = p.id
GROUP BY
  p.pays, p.region, p.territoire, p.denree_type,
  p.denree_grade, mois, p.quantite_unite;

CREATE UNIQUE INDEX idx_mv_prod_reg
  ON mv_production_regionale(pays, region, denree_type, mois);

-- Vue : Tableau de bord Casino temps réel
```

```
CREATE MATERIALIZED VIEW mv_casino_dashboard AS
SELECT
  c.commodite,
  DATE_TRUNC('day', c.timestamp_emission) AS jour,
  COUNT(*)                        AS nb_contrats,
  SUM(c.prime_payee_GU)               AS volume_primes_GU,
  SUM(c.taxe_montant_GU)              AS taxes_fonds_commun_GU,
  AVG(c.levier_valeur)                AS levier_moyen,
  AVG(c.taux_taxe_pct)                AS taux_taxe_moyen,
  COUNT(*) FILTER (WHERE c.type_option = 'CALL') AS nb_calls,
  COUNT(*) FILTER (WHERE c.type_option = 'PUT')  AS nb_puts,
  COUNT(*) FILTER (WHERE c.statut = 'EXERCÉ')    AS nb_exerces,
  COUNT(*) FILTER (WHERE c.statut = 'EXPIRÉ')    AS nb_expires,
  SUM(c.gain_acheteur_GU) FILTER (WHERE c.gain_acheteur_GU > 0) AS gains_acheteurs,
  MAX(c.levier_valeur)                AS levier_max_utilise
FROM contrats_option c
GROUP BY c.commodite, jour;

-- Vue : Santé financière du Fonds Commun
CREATE MATERIALIZED VIEW mv_sante_fonds_commun AS
SELECT
  exercice,
  DATE_TRUNC('month', timestamp)        AS mois,
  SUM(montant_GU) FILTER (WHERE direction = 'ENTREE') AS revenus_GU,
  SUM(montant_GU) FILTER (WHERE direction = 'SORTIE') AS depenses_GU,
  SUM(montant_GU) FILTER (WHERE direction = 'ENTREE')
    - SUM(montant_GU) FILTER (WHERE direction = 'SORTIE') AS solde_net_GU,
  SUM(montant_GU) FILTER (WHERE type_mouvement = 'ENTREE_TAXE_CREATION')    AS
taxe_creation_GU,
```

```
  SUM(montant_GU) FILTER (WHERE type_mouvement = 'ENTREE_TAXE_CASINO')     AS taxe_casino_GU,
  SUM(montant_GU) FILTER (WHERE type_mouvement = 'ENTREE_TAXE_BANCAIRE')    AS taxe_bancaire_GU,
  SUM(montant_GU) FILTER (WHERE type_mouvement = 'SORTIE_RECOMPENSE_AUDITEUR') AS recompenses_auditeurs_GU
FROM fonds_commun_journal
GROUP BY exercice, mois;

--
═══════════════════════════════════════════════════════════════════════════════
-- PARTIE 2 — REQUÊTES CHAINQL FONDAMENTALES
-- Accessibles publiquement via API REST
--
═══════════════════════════════════════════════════════════════════════════════

-- ─────────────────────────────────────────────────────────────────────────────
-- REQUÊTE 1 : Tableau de bord mondial temps réel
-- Endpoint : GET /chainql/dashboard
-- Latence cible : < 50ms (vue matérialisée)
-- ─────────────────────────────────────────────────────────────────────────────

SELECT
  -- Économie réelle
  (SELECT SUM(micro_GU_nets) FROM emissions)          AS masse_monetaire_totale_GU,
  (SELECT COUNT(*) FROM productions WHERE statut = 'APPROUVÉ'
   AND timestamp > NOW() - INTERVAL '30 days')        AS productions_actives_30j,
  (SELECT COUNT(*) FROM producteurs WHERE statut = 'ACTIF') AS producteurs_actifs,
  (SELECT COUNT(*) FROM transactions
   WHERE timestamp > NOW() - INTERVAL '24h')          AS transactions_24h,
```

```
-- Casino
(SELECT COUNT(*) FROM contrats_option WHERE statut = 'ACTIF') AS options_actives,
(SELECT SUM(prime_payee_GU * levier_valeur)
 FROM contrats_option WHERE statut = 'ACTIF')        AS valeur_notionnelle_casino_GU,
ROUND(
  (SELECT SUM(prime_payee_GU * levier_valeur)
   FROM contrats_option WHERE statut = 'ACTIF')
  / NULLIF((SELECT SUM(micro_GU_nets) FROM emissions), 0) * 100
, 2)                                    AS ratio_casino_sur_reel_pct,

-- Fonds Commun
(SELECT MAX(solde_apres_GU) FROM fonds_commun_journal) AS fonds_commun_solde_GU,
(SELECT SUM(montant_GU) FROM fonds_commun_journal
 WHERE direction = 'ENTREE'
 AND timestamp > NOW() - INTERVAL '24h')            AS revenus_fonds_24h_GU,

-- Gouvernance
(SELECT COUNT(*) FROM auditeurs WHERE statut = 'ACTIF') AS auditeurs_actifs,
(SELECT COUNT(*) FROM signalements WHERE statut = 'EN_ATTENTE') AS signalements_en_cours,

-- Timestamp
NOW()                                   AS timestamp_requete;



-- ──────────────────────────────────────────────────────────────────────
-- REQUÊTE 2 : Production mondiale par région et denrée
-- Endpoint : GET /chainql/productions/regionales?denree=blé_dur&mois=2026-03
-- ──────────────────────────────────────────────────────────────────────
```

```
SELECT
  p.pays,
  p.region,
  p.denree_type,
  p.denree_grade,
  COUNT(*)                    AS nb_productions,
  SUM(p.quantite_valeur)          AS kg_produits,
  SUM(e.micro_GU_nets)            AS GU_emis,
  SUM(e.taxe_creation_GU)         AS taxes_fonds_commun_GU,
  AVG(e.F_region)                 AS facteur_region_moyen,
  AVG(e.F_carbone)                AS facteur_carbone_moyen,
  AVG(p.oracle_sat_ndvi)          AS ndvi_moyen,
  ROUND(
    SUM(e.micro_GU_nets)
    / NULLIF(SUM(p.quantite_valeur), 0)
  , 8)                        AS GU_par_kg,
  -- Indicateur de durabilité
  CASE
    WHEN AVG(e.F_carbone) >= 1.15 THEN '🌱 Exemplaire'
    WHEN AVG(e.F_carbone) >= 1.05 THEN '✅ Durable'
    WHEN AVG(e.F_carbone) >= 0.95 THEN '🟡 Standard'
    ELSE                       '⚠️ À améliorer'
  END                         AS bilan_carbone

FROM productions p
JOIN emissions e ON e.production_id = p.id
WHERE p.statut    = 'APPROUVÉ'
```

```
  AND p.denree_type = :denree       -- paramètre URL
  AND DATE_TRUNC('month', p.timestamp) = :mois  -- paramètre URL
GROUP BY p.pays, p.region, p.denree_type, p.denree_grade
ORDER BY GU_emis DESC;



-- ═══════════════════════════════════════════════════════════════
-- REQUÊTE 3 : Traçabilité complète d'une production
-- Endpoint : GET /chainql/traçabilite/:production_id
-- Du champ au consommateur — toutes les étapes
-- ═══════════════════════════════════════════════════════════════

WITH RECURSIVE parcours AS (

  -- Point de départ : la production
  SELECT
    p.id                AS event_id,
    'PRODUCTION'        AS type_event,
    p.producteur_id     AS acteur_id,
    pr.pseudonyme       AS acteur_nom,
    e.micro_GU_nets     AS montant_GU,
    p.quantite_valeur   AS quantite_physique,
    p.quantite_unite,
    p.timestamp,
    p.pays,
    p.denree_type,
    0                   AS profondeur,
    p.id                AS production_origine
```

```
FROM productions p
JOIN emissions e    ON e.production_id = p.id
JOIN producteurs pr ON pr.id = p.producteur_id
WHERE p.id = :production_id

UNION ALL

-- Récursion : chaque transaction liée
SELECT
  t.id,
  t.type,
  t.destinataire_id,
  t.destinataire_id,    -- pseudonyme si dispo
  t.montant_GU,
  NULL,
  NULL,
  t.timestamp,
  NULL,
  pa.denree_type,
  pa.profondeur + 1,
  pa.production_origine

FROM transactions t
JOIN parcours pa
  ON t.expediteur_id = pa.acteur_id
  AND t.timestamp > pa.timestamp
WHERE pa.profondeur < 15    -- max 15 transformations
  AND t.sous_jacent_prod_ref = pa.production_origine
```

```
)
SELECT
  profondeur          AS etape,
  type_event          AS evenement,
  acteur_nom          AS acteur,
  ROUND(montant_GU, 6) AS valeur_GU,
  quantite_physique,
  quantite_unite,
  timestamp,
  pays,
  denree_type         AS produit,
  production_origine

FROM parcours
ORDER BY timestamp ASC;



-- ──────────────────────────────────────────────────────────────────────────
-- REQUÊTE 4 : Détection d'anomalies statistiques (Anti-Fraude)
-- Lancée automatiquement à chaque nouveau bloc de production
-- Endpoint : POST /chainql/audit/scan (privé — auditeurs niveau 2+)
-- ──────────────────────────────────────────────────────────────────────────

WITH historique AS (
  -- Ratios historiques par producteur (90 derniers jours)
  SELECT
    p.producteur_id,
    p.denree_type,
    p.region,
```

```
  AVG(e.micro_GU_nets / NULLIF(p.quantite_valeur, 0))   AS ratio_GU_kg_moy,
  STDDEV(e.micro_GU_nets / NULLIF(p.quantite_valeur, 0)) AS ratio_GU_kg_std,
  AVG(p.oracle_sat_ndvi)                    AS ndvi_moy,
  STDDEV(p.oracle_sat_ndvi)                 AS ndvi_std,
  COUNT(*)                                  AS nb_obs
 FROM productions p
 JOIN emissions e ON e.production_id = p.id
 WHERE p.timestamp BETWEEN NOW() - INTERVAL '90 days'
                   AND NOW() - INTERVAL '1 day'
  AND p.statut = 'APPROUVÉ'
 GROUP BY p.producteur_id, p.denree_type, p.region
 HAVING COUNT(*) >= 3   -- minimum 3 obs pour baseline fiable
),

production_recente AS (
 -- Productions des dernières 24h en attente de validation
 SELECT
  p.id,
  p.producteur_id,
  p.denree_type,
  p.region,
  p.pays,
  p.quantite_valeur,
  p.oracle_sat_ndvi,
  p.oracle_sat_superficie_ha,
  p.divergence_max_pct,
  e.micro_GU_nets,
  e.micro_GU_bruts,
  e.F_region,
```

```
  e.F_carbone,
  p.timestamp,
  pr.fiabilite_score,
  pr.incidents
 FROM productions p
 JOIN emissions e    ON e.production_id = p.id
 JOIN producteurs pr ON pr.id = p.producteur_id
 WHERE p.timestamp > NOW() - INTERVAL '24h'
)

SELECT
 pr.id                          AS production_id,
 pr.producteur_id,
 pr.denree_type,
 pr.region,
 pr.fiabilite_score,

 -- Z-score ratio GU/kg (éloignement de la normale)
 ROUND(
  ABS(pr.micro_GU_nets / NULLIF(pr.quantite_valeur, 0)
     - h.ratio_GU_kg_moy)
  / NULLIF(h.ratio_GU_kg_std, 0)
 , 2)                           AS z_score_ratio_GU,

 -- Z-score NDVI (cohérence satellite)
 ROUND(
  ABS(pr.oracle_sat_ndvi - h.ndvi_moy)
  / NULLIF(h.ndvi_std, 0)
 , 2)                           AS z_score_ndvi,
```

```
-- Divergence oracles
pr.divergence_max_pct,

-- NDVI nul avec production déclarée = signal fort
CASE
  WHEN pr.oracle_sat_ndvi IS NULL
    OR pr.oracle_sat_ndvi < 0.1
  THEN TRUE ELSE FALSE
END                                 AS ghost_production_flag,

-- Score de risque composite
ROUND(
  (ABS(pr.micro_GU_nets / NULLIF(pr.quantite_valeur,0) - h.ratio_GU_kg_moy)
   / NULLIF(h.ratio_GU_kg_std, 0))
  * 0.40   -- 40% : ratio GU/kg
  + (ABS(COALESCE(pr.oracle_sat_ndvi,0) - COALESCE(h.ndvi_moy,0))
     / NULLIF(h.ndvi_std, 0.01))
  * 0.30   -- 30% : cohérence satellite
  + (pr.divergence_max_pct / 5.0)
  * 0.20   -- 20% : divergence oracles
  + (pr.incidents::NUMERIC / 10.0)
  * 0.10   -- 10% : historique incidents
, 3)                                AS score_risque,

-- Action automatique recommandée
CASE
  WHEN pr.oracle_sat_ndvi < 0.1
    THEN '🔴 BLOQUER — Ghost production suspectée'
```

```
    WHEN ABS(pr.micro_GU_nets / NULLIF(pr.quantite_valeur,0)
           - h.ratio_GU_kg_moy)
         / NULLIF(h.ratio_GU_kg_std, 0) > 4
     THEN '🔴 BLOQUER — Anomalie z-score > 4'
    WHEN ABS(pr.micro_GU_nets / NULLIF(pr.quantite_valeur,0)
           - h.ratio_GU_kg_moy)
         / NULLIF(h.ratio_GU_kg_std, 0) > 3
     THEN '🟠 SUSPENDRE 72h — Anomalie z-score > 3'
    WHEN pr.divergence_max_pct > 4.5
     THEN '🟠 SUSPENDRE — Divergence oracles critique'
    WHEN ABS(pr.micro_GU_nets / NULLIF(pr.quantite_valeur,0)
           - h.ratio_GU_kg_moy)
         / NULLIF(h.ratio_GU_kg_std, 0) > 2
     THEN '🟡 SIGNALER AUX AUDITEURS — Anomalie modérée'
    ELSE     '🟢 APPROUVER — Dans les normes'
  END                              AS action_recommandee,

  pr.timestamp

FROM production_recente pr
LEFT JOIN historique h
  ON h.producteur_id = pr.producteur_id
  AND h.denree_type  = pr.denree_type
ORDER BY score_risque DESC NULLS LAST;


-- ═══════════════════════════════════════════════════════════════════

-- REQUÊTE 5 : Analyse de graphe — Réseaux de collusion
```

```
-- Détecte les clusters de producteurs liés suspicieusement
-- Endpoint : GET /chainql/audit/graphe-collusion (privé — niveau 3)
-- ──────────────────────────────────────────────────────────────────────

WITH graphe AS (
  SELECT
    t.expediteur_id                 AS source,
    t.destinataire_id               AS cible,
    COUNT(*)                        AS nb_tx,
    SUM(t.montant_GU)               AS volume_GU,
    MIN(t.timestamp)                AS premiere_tx,
    MAX(t.timestamp)                AS derniere_tx,
    AVG(EXTRACT(EPOCH FROM
      t.timestamp - LAG(t.timestamp)
        OVER (PARTITION BY t.expediteur_id, t.destinataire_id
              ORDER BY t.timestamp)
    ))                              AS delai_moyen_secondes
  FROM transactions t
  WHERE t.timestamp > NOW() - INTERVAL '90 days'
    AND t.type NOT IN ('TAXE_CREATION', 'TAXE_CASINO',
                       'TAXE_MEMBRANE', 'FRAIS_RESEAU')
  GROUP BY t.expediteur_id, t.destinataire_id
),

concentration AS (
  -- Concentration des flux : si >80% vers même destination → suspect
  SELECT
    g.source,
    g.cible,
```

```
  g.volume_GU,
  g.nb_tx,
  g.delai_moyen_secondes,
  ROUND(
    g.volume_GU / NULLIF(
      SUM(g.volume_GU) OVER (PARTITION BY g.source)
    , 0) * 100
  , 2)                          AS pct_concentration
 FROM graphe g
)

SELECT
 c.source                       AS producteur_suspect,
 c.cible                        AS destination_concentrée,
 c.nb_tx,
 ROUND(c.volume_GU, 4)          AS volume_total_GU,
 c.pct_concentration,
 ROUND(c.delai_moyen_secondes, 1)  AS delai_moyen_sec,
 -- Signal : même silo partagé ?
 CASE
  WHEN p1.silo_id IS NOT NULL
   AND p1.silo_id = p2.silo_id
  THEN '⚠️ MÊME SILO PARTAGÉ'
  ELSE 'Silos distincts'
 END                            AS silo_partage,
 -- Niveau d'alerte
 CASE
  WHEN c.pct_concentration > 90
   AND c.delai_moyen_secondes < 10
```

```
    THEN '🔴 COLLUSION PROBABLE — Signaler Tribunal'
   WHEN c.pct_concentration > 80
    AND c.nb_tx > 50
    THEN '🟠 RÉSEAU SUSPECT — Enquête niveau 3'
   WHEN c.pct_concentration > 70
    THEN '🟡 À SURVEILLER'
   ELSE '🟢 Normal'
  END                          AS niveau_alerte

FROM concentration c
LEFT JOIN (
  SELECT DISTINCT ON (producteur_id) producteur_id, silo_id
  FROM productions WHERE silo_id IS NOT NULL
  ORDER BY producteur_id, timestamp DESC
) p1 ON p1.producteur_id = c.source
LEFT JOIN (
  SELECT DISTINCT ON (producteur_id) producteur_id, silo_id
  FROM productions WHERE silo_id IS NOT NULL
  ORDER BY producteur_id, timestamp DESC
) p2 ON p2.producteur_id = c.cible

WHERE c.pct_concentration > 70
ORDER BY c.pct_concentration DESC, c.volume_GU DESC;



-- ═══════════════════════════════════════════════════════════════════

-- REQUÊTE 6 : Stabilité des prix — Le Casino aide-t-il les producteurs ?

-- Mesure la corrélation entre liquidité options et volatilité réelle
```

```
-- Endpoint : GET /chainql/analyse/impact-casino-prix
-- ═══════════════════════════════════════════════════════════════════════

SELECT
  p.denree_type,
  DATE_TRUNC('month', p.timestamp)        AS mois,

  -- Prix réels des transactions
  ROUND(AVG(t.montant_GU
    / NULLIF(p.quantite_valeur, 0))
  , 8)                                    AS prix_moyen_reel_GU_par_kg,

  ROUND(STDDEV(t.montant_GU
    / NULLIF(p.quantite_valeur, 0))
  , 8)                                    AS volatilite_prix_reel,

  -- Activité du Casino sur cette commodité ce mois
  COUNT(DISTINCT c.id)                    AS nb_options_actives,
  SUM(c.prime_payee_GU)                   AS volume_options_GU,
  AVG(c.levier_valeur)                    AS levier_moyen,

  -- Taux de couverture : quelle fraction de la production est couverte ?
  ROUND(
    SUM(c.quantite_kg)
    / NULLIF(SUM(p.quantite_valeur), 0)
    * 100
  , 2)                                    AS pct_production_couverte,

  -- Corrélation options ↔ volatilité (valeur négative = stabilisant)
```

```
 ROUND(CORR(
  COUNT(c.id)::NUMERIC,
  STDDEV(t.montant_GU / NULLIF(p.quantite_valeur, 0))
 ) OVER (PARTITION BY p.denree_type)
 , 4)                          AS corr_options_volatilite,

 -- Interprétation
 CASE
  WHEN CORR(
   COUNT(c.id)::NUMERIC,
   STDDEV(t.montant_GU / NULLIF(p.quantite_valeur, 0))
  ) OVER (PARTITION BY p.denree_type) < -0.3
  THEN '✅ Le Casino STABILISE les prix'
  WHEN CORR(
   COUNT(c.id)::NUMERIC,
   STDDEV(t.montant_GU / NULLIF(p.quantite_valeur, 0))
  ) OVER (PARTITION BY p.denree_type) > 0.3
  THEN '⚠️ Le Casino DÉSTABILISE — surveiller ratio'
  ELSE '🟡 Effet neutre'
 END                           AS verdict_casino

FROM productions p
JOIN transactions t
 ON t.sous_jacent_prod_ref = p.id
LEFT JOIN contrats_option c
 ON c.commodite = p.denree_type
 AND DATE_TRUNC('month', c.timestamp_emission)
    = DATE_TRUNC('month', p.timestamp)
WHERE p.statut = 'APPROUVÉ'
```

```
GROUP BY p.denree_type, mois
ORDER BY p.denree_type, mois DESC;



-- ─────────────────────────────────────────────────────────────────────────
-- REQUÊTE 7 : Santé du Fonds Commun — Projection 12 mois
-- Endpoint : GET /chainql/fonds-commun/projection
-- ─────────────────────────────────────────────────────────────────────────

WITH historique_mensuel AS (
  SELECT
    DATE_TRUNC('month', timestamp)        AS mois,
    SUM(montant_GU) FILTER (WHERE direction = 'ENTREE') AS revenus,
    SUM(montant_GU) FILTER (WHERE direction = 'SORTIE') AS depenses
  FROM fonds_commun_journal
  WHERE timestamp > NOW() - INTERVAL '6 months'
  GROUP BY mois
),

tendances AS (
  SELECT
    AVG(revenus)                    AS revenu_mensuel_moy,
    STDDEV(revenus)                 AS revenu_mensuel_std,
    AVG(depenses)                   AS depense_mensuelle_moy,
    REGR_SLOPE(revenus,
      EXTRACT(EPOCH FROM mois))     AS tendance_revenus,
    REGR_SLOPE(depenses,
      EXTRACT(EPOCH FROM mois))     AS tendance_depenses
  FROM historique_mensuel
```

```
),

projection AS (
  SELECT
    generate_series(1, 12)              AS mois_futur,
    t.revenu_mensuel_moy
      + t.tendance_revenus
        * generate_series(1, 12)
        * 30 * 86400                    AS revenu_projete,
    t.depense_mensuelle_moy
      + t.tendance_depenses
        * generate_series(1, 12)
        * 30 * 86400                    AS depense_projetee,
    t.revenu_mensuel_std
  FROM tendances t
)

SELECT
  mois_futur,
  NOW() + (mois_futur || ' months')::INTERVAL AS date_projetee,
  ROUND(revenu_projete, 2)              AS revenu_projete_GU,
  ROUND(depense_projetee, 2)            AS depense_projetee_GU,
  ROUND(revenu_projete - depense_projetee, 2) AS surplus_projete_GU,
  -- Intervalle de confiance 90%
  ROUND(revenu_projete - 1.645 * revenu_mensuel_std, 2) AS revenu_pessimiste,
  ROUND(revenu_projete + 1.645 * revenu_mensuel_std, 2) AS revenu_optimiste,
  CASE
    WHEN revenu_projete - depense_projetee < 0
    THEN '🔴 DÉFICIT PROJETÉ — Réviser budget'
```

```
  WHEN (revenu_projete - 1.645 * revenu_mensuel_std)
      - depense_projetee < 0
  THEN '🟡 RISQUE DÉFICIT — Scénario pessimiste'
  ELSE '🟢 EXCÉDENT PROJETÉ'
 END                          AS statut_projection

FROM projection
ORDER BY mois_futur;



-- ═══════════════════════════════════════════════════════════════════
-- REQUÊTE 8 : Taux de conversion fiat → µGU en temps réel
-- Recalibré chaque mois selon données AIE + FAO
-- Endpoint : GET /chainql/taux/conversion-fiat
-- ═══════════════════════════════════════════════════════════════════

WITH calibration AS (
 SELECT
  -- Énergie de référence par gramme de blé grade B
  9500.0                       AS E_ref_kJ_par_g,

  -- Prix mondial moyen du blé en µGU (30 derniers jours)
  AVG(t.montant_GU / NULLIF(p.quantite_valeur, 0))
                               AS prix_ble_GU_par_kg,

  -- Émissions totales / production totale = calibration P_ref
  SUM(e.micro_GU_nets) / NULLIF(SUM(p.quantite_valeur), 0)
                               AS P_ref_GU_par_g,
```

```
  -- Taux de change implicite : combien de kJ = 1 µGU ?
  9500.0 / NULLIF(
    SUM(e.micro_GU_nets) / NULLIF(SUM(p.quantite_valeur / 1000.0), 0)
  , 0)                          AS kJ_par_GU

 FROM productions p
 JOIN emissions e ON e.production_id = p.id
 JOIN transactions t ON t.sous_jacent_prod_ref = p.id
 WHERE p.denree_type = 'blé_dur'
  AND p.denree_grade IN ('B', 'A2')
  AND p.timestamp > NOW() - INTERVAL '30 days'
),

taux_par_montant AS (
 SELECT
  -- Tranches de conversion avec facteur pénalité progressif
  tranche,
  montant_min_USD,
  montant_max_USD,
  facteur_penalite,
  ROUND(
    (SELECT P_ref_GU_par_g FROM calibration) * 1000
    / facteur_penalite
  , 8)                          AS taux_GU_par_USD
 FROM (VALUES
  ('MICRO',   0,        10000,     2.0,  NULL),
  ('PETIT',   10000,    100000,    5.0,  NULL),
  ('MOYEN',   100000,   1000000,   10.0, NULL),
  ('GRAND',   1000000,  1000000000, 50.0, NULL),
```

```
   ('MEGA',    1000000000, NULL,      200.0,NULL)
  ) AS tranches(tranche, montant_min_USD, montant_max_USD,
                facteur_penalite, _dummy)
)

SELECT
  t.tranche,
  t.montant_min_USD,
  t.montant_max_USD,
  t.facteur_penalite                AS penalite_appliquee,
  t.taux_GU_par_USD,
  -- Exemples concrets
  ROUND(1000 * t.taux_GU_par_USD, 6)      AS GU_pour_1000_USD,
  ROUND(1000000 * t.taux_GU_par_USD, 2)    AS GU_pour_1M_USD,
  ROUND(1000000000 * t.taux_GU_par_USD, 0) AS GU_pour_1Mds_USD,
  -- Calibration courante
  (SELECT ROUND(P_ref_GU_par_g, 13) FROM calibration) AS P_ref_courant,
  (SELECT ROUND(prix_ble_GU_par_kg, 6) FROM calibration) AS prix_ble_ref_GU,
  NOW()                             AS timestamp_calibration

FROM taux_par_montant t
ORDER BY t.montant_min_USD;



-- ═══════════════════════════════════════════════════════════════════
-- REQUÊTE 9 : Performance d'un auditeur citoyen
-- Endpoint : GET /chainql/auditeurs/:auditeur_id/performance
-- ═══════════════════════════════════════════════════════════════════
```

```
SELECT
 a.id,
 a.pseudonyme,
 a.niveau,
 a.territoire_principal,
 a.reputation_score,

 -- Activité
 a.audits_completes,
 a.fraudes_detectees,
 a.faux_positifs,
 ROUND(a.taux_confirmation_pct, 2)       AS taux_confirmation_pct,

 -- Financier
 ROUND(a.recompenses_cumulees_GU, 6)     AS recompenses_totales_GU,
 ROUND(a.recompenses_cumulees_GU
  / NULLIF(a.fraudes_detectees, 0)
 , 6)                          AS recompense_moyenne_par_fraude_GU,
 a.caution_depositee_GU,

 -- ROI auditeur
 ROUND(
  (a.recompenses_cumulees_GU
   - a.caution_depositee_GU
   - (a.faux_positifs * a.caution_depositee_GU / NULLIF(a.audits_completes,1)))
  / NULLIF(a.caution_depositee_GU, 0)
  * 100
 , 2)                          AS roi_auditeur_pct,
```

```
-- Signalements en cours
COUNT(s.id) FILTER (WHERE s.statut = 'EN_ATTENTE') AS signalements_actifs,

-- Spécialités les plus rentables
(SELECT type_fraude_suspectee
 FROM signalements
 WHERE auditeur_id = a.id
  AND verdict = 'FRAUDE_DÉLIBÉRÉE'
 GROUP BY type_fraude_suspectee
 ORDER BY COUNT(*) DESC
 LIMIT 1)                    AS specialite_la_plus_rentable,

-- Rang parmi auditeurs de même niveau
RANK() OVER (
 PARTITION BY a.niveau
 ORDER BY a.reputation_score DESC
)                            AS rang_niveau,

-- Recommandation upgrade niveau
CASE
 WHEN a.niveau = 'OBSERVATEUR'
  AND a.audits_completes >= 50
  AND a.faux_positifs <= 3
 THEN '✅ Éligible au niveau ANALYSTE'
 WHEN a.niveau = 'ANALYSTE'
  AND a.fraudes_detectees >= 5
  AND a.taux_confirmation_pct >= 85
 THEN '✅ Éligible à la certification CERTIFIÉ'
 ELSE '📋 Continuer à accumuler de l expérience'
```

```
  END                       AS recommandation_upgrade

FROM auditeurs a
LEFT JOIN signalements s ON s.auditeur_id = a.id
WHERE a.id = :auditeur_id
GROUP BY a.id;



-- ─────────────────────────────────────────────────────────────────────────
-- REQUÊTE 10 : Évolution masse monétaire — Honnêteté systémique
-- Prouve que chaque µGU créé correspond à une production réelle
-- Endpoint : GET /chainql/masse-monetaire/audit-physique
-- ─────────────────────────────────────────────────────────────────────────

WITH masse_theorique AS (
  -- Ce que la masse monétaire DEVRAIT être
  -- si chaque µGU correspond exactement à sa production
  SELECT
    DATE_TRUNC('week', e.timestamp)       AS semaine,
    SUM(e.micro_GU_nets)                  AS GU_emis_semaine,
    SUM(p.quantite_valeur)                AS kg_produits_semaine,
    -- GU théoriques selon formule pure
    SUM(p.quantite_valeur * 1000          -- en grammes
        * e.P_ref
        * e.F_qualite
        * e.F_region
        * e.F_carbone
        * e.F_temps
        * 0.99)                           AS GU_theoriques_formule
```

```
  FROM emissions e
  JOIN productions p ON p.id = e.production_id
  GROUP BY semaine
),

masse_reelle AS (
  -- Ce que la masse monétaire EST réellement (à partir des blocs)
  SELECT
    DATE_TRUNC('week', b.timestamp)       AS semaine,
    MAX(b.masse_monetaire_apres)          AS masse_fin_semaine,
    MIN(b.masse_monetaire_apres)          AS masse_debut_semaine
  FROM blocs b
  WHERE b.chaine = 'PRINCIPALE'
    AND b.masse_monetaire_apres IS NOT NULL
  GROUP BY semaine
)

SELECT
  mt.semaine,
  ROUND(mt.GU_emis_semaine, 6)            AS GU_emis_semaine,
  ROUND(mt.GU_theoriques_formule, 6)      AS GU_theoriques_formule,
  ROUND(mr.masse_fin_semaine
      - mr.masse_debut_semaine, 6)        AS variation_masse_reelle,

  -- Écart entre théorique et réel (doit être 0 ou très proche)
  ROUND(ABS(
    mt.GU_theoriques_formule
    - (mr.masse_fin_semaine - mr.masse_debut_semaine)
  ), 6)                                   AS ecart_absolu_GU,
```

```
 -- En pourcentage
 ROUND(ABS(
  mt.GU_theoriques_formule
  - (mr.masse_fin_semaine - mr.masse_debut_semaine)
 ) / NULLIF(mt.GU_theoriques_formule, 0) * 100
 , 4)                          AS ecart_pct,

 -- Verdict de cohérence
 CASE
  WHEN ABS(mt.GU_theoriques_formule
          - (mr.masse_fin_semaine - mr.masse_debut_semaine))
      / NULLIF(mt.GU_theoriques_formule, 0) < 0.001
  THEN '✅ INTÈGRE — Masse 100% adossée production réelle'
  WHEN ABS(mt.GU_theoriques_formule
          - (mr.masse_fin_semaine - mr.masse_debut_semaine))
      / NULLIF(mt.GU_theoriques_formule, 0) < 0.01
  THEN '🟡 ACCEPTABLE — Écart < 1% (arrondi formule)'
  ELSE '🔴 ANOMALIE — Création hors production détectée'
 END                           AS verdict_integrite,

 mt.kg_produits_semaine

FROM masse_theorique mt
JOIN masse_reelle mr USING (semaine)
ORDER BY mt.semaine DESC;
```

```
--
═══════════════════════════════════════════════════════════════════════════
═══

-- PARTIE 3 — PROCÉDURES STOCKÉES (Actions ChainQL)

-- Ces procédures modifient l'état — requièrent authentification

--
═══════════════════════════════════════════════════════════════════════════
═══


-- ──────────────────────────────────────────────────────────────────────

-- PROCÉDURE : Soumettre un signalement de fraude

-- Appelée par un auditeur avec sa signature cryptographique

-- ──────────────────────────────────────────────────────────────────────

CREATE OR REPLACE PROCEDURE soumettre_signalement(

  p_auditeur_id       VARCHAR,

  p_production_id     VARCHAR,

  p_type_fraude       VARCHAR,

  p_z_score_energie    NUMERIC,

  p_z_score_travail    NUMERIC,

  p_pattern           VARCHAR,

  p_preuve_chainql    TEXT,

  p_signature         VARCHAR   -- signature cryptographique de l'auditeur

)

LANGUAGE plpgsql AS $$

DECLARE

  v_auditeur      auditeurs%ROWTYPE;

  v_caution       NUMERIC;

  v_signalement_id VARCHAR;

BEGIN

  -- Vérifier que l'auditeur existe et est actif
```

```
SELECT * INTO v_auditeur FROM auditeurs
WHERE id = p_auditeur_id AND statut = 'ACTIF';

IF NOT FOUND THEN
  RAISE EXCEPTION 'Auditeur % non trouvé ou inactif', p_auditeur_id;
END IF;

-- Caution anti-spam selon le niveau
v_caution := CASE v_auditeur.niveau
  WHEN 'OBSERVATEUR' THEN 100.0
  WHEN 'ANALYSTE'    THEN 500.0
  WHEN 'CERTIFIÉ'    THEN 1000.0
  ELSE 100.0
END;

-- Vérifier que la caution est suffisante
IF v_auditeur.caution_deposee_GU < v_caution THEN
  RAISE EXCEPTION
    'Caution insuffisante. Requis : % µGU, Disponible : % µGU',
    v_caution, v_auditeur.caution_deposee_GU;
END IF;

-- Générer l'identifiant du signalement
v_signalement_id := 'SIG-' || TO_CHAR(NOW(), 'YYYY-MM-DD') || '-'
                 || LPAD(NEXTVAL('seq_signalements')::TEXT, 8, '0');

-- Insérer le signalement
INSERT INTO signalements (
  id, auditeur_id, production_cible_id,
```

```
  type_fraude_suspectee, z_score_energie, z_score_travail,
  pattern_detecte, preuve_chainql,
  caution_auditeur_GU, statut, timestamp_signalement
) VALUES (
  v_signalement_id, p_auditeur_id, p_production_id,
  p_type_fraude, p_z_score_energie, p_z_score_travail,
  p_pattern, p_preuve_chainql,
  v_caution, 'EN_ATTENTE', NOW()
);

-- Bloquer la caution
UPDATE auditeurs
SET caution_depositee_GU = caution_depositee_GU - v_caution
WHERE id = p_auditeur_id;

-- Déclencher la suspension préventive de la production si z-score critique
IF p_z_score_energie > 4 OR p_z_score_travail > 4 THEN
  UPDATE productions
  SET statut = 'SUSPENDU'
  WHERE id = p_production_id AND statut = 'APPROUVÉ';
END IF;

-- Log dans le Fonds Commun (la caution bloquée est tracée)
INSERT INTO fonds_commun_journal (
  id, type_mouvement, direction, montant_GU,
  source_tx_id, beneficiaire_id, solde_avant_GU,
  solde_apres_GU, exercice, timestamp, note
) VALUES (
  'FC-' || v_signalement_id,
```

```
    'CAUTION_AUDITEUR_BLOQUÉE', 'ENTREE', v_caution,
    v_signalement_id, p_auditeur_id,
    (SELECT MAX(solde_apres_GU) FROM fonds_commun_journal),
    (SELECT MAX(solde_apres_GU) FROM fonds_commun_journal) + v_caution,
    EXTRACT(YEAR FROM NOW())::INTEGER, NOW(),
    'Caution bloquée pour signalement ' || v_signalement_id
  );

  RAISE NOTICE 'Signalement % créé avec succès. Caution % µGU bloquée.',
    v_signalement_id, v_caution;

END;
$$;



-- ═══════════════════════════════════════════════════════════════
-- PROCÉDURE : Clôturer un signalement (verdict du Tribunal)
-- Distribue automatiquement les récompenses ou libère la caution
-- ═══════════════════════════════════════════════════════════════
CREATE OR REPLACE PROCEDURE cloturer_signalement(
  p_signalement_id     VARCHAR,
  p_verdict            VARCHAR,   -- FRAUDE_CONFIRMÉE | FAUSSE_ALERTE
  p_montant_fraude_GU  NUMERIC,
  p_niveau_sanction    VARCHAR,
  p_arbitre_id         VARCHAR,
  p_signature_tribunal VARCHAR
)
LANGUAGE plpgsql AS $$
DECLARE
```

```
 v_sig        signalements%ROWTYPE;
 v_recompense   NUMERIC;
 v_fonds_part   NUMERIC;
 v_burn_part    NUMERIC;
BEGIN
 SELECT * INTO v_sig FROM signalements WHERE id = p_signalement_id;
 IF NOT FOUND THEN
  RAISE EXCEPTION 'Signalement % non trouvé', p_signalement_id;
 END IF;

 IF p_verdict = 'FRAUDE_CONFIRMÉE' THEN
  -- Distribution : 50% auditeur, 25% Fonds, 15% victimes, 10% burn
  v_recompense := p_montant_fraude_GU * 0.50;
  v_fonds_part := p_montant_fraude_GU * 0.25;
  v_burn_part  := p_montant_fraude_GU * 0.10;

  -- Récompenser l'auditeur
  UPDATE auditeurs SET
   fraudes_detectees   = fraudes_detectees + 1,
   recompenses_cumulees_GU = recompenses_cumulees_GU + v_recompense,
   caution_depositee_GU = caution_depositee_GU
                + v_sig.caution_auditeur_GU  -- remboursé × 2
                + v_sig.caution_auditeur_GU,
   audits_completes    = audits_completes + 1,
   reputation_score    = LEAST(100, reputation_score + 2.5)
  WHERE id = v_sig.auditeur_id;

  -- Verser au Fonds Commun
  INSERT INTO fonds_commun_journal VALUES (
```

```
  'FC-RECUP-' || p_signalement_id,
  'ENTREE_RECUPERATION_FRAUDE', 'ENTREE', v_fonds_part,
  'B5', v_sig.auditeur_id,
  (SELECT MAX(solde_apres_GU) FROM fonds_commun_journal),
  (SELECT MAX(solde_apres_GU) FROM fonds_commun_journal) + v_fonds_part,
  EXTRACT(YEAR FROM NOW())::INTEGER, NOW(),
  'Récupération fraude ' || p_signalement_id || ' — part Fonds Commun'
);

-- Sanctionner le producteur fraudeur
UPDATE producteurs SET
  statut        = 'SUSPENDU',
  date_suspension = NOW()::DATE,
  date_fin_suspension = CASE p_niveau_sanction
    WHEN 'NÉGLIGENCE'       THEN (NOW() + INTERVAL '30 days')::DATE
    WHEN 'FRAUDE_DÉLIBÉRÉE'  THEN (NOW() + INTERVAL '24 months')::DATE
    WHEN 'FRAUDE_SYSTÉMIQUE' THEN NULL  -- bannissement permanent
    ELSE (NOW() + INTERVAL '7 days')::DATE
  END,
  incidents      = incidents + 1,
  fiabilite_score = GREATEST(0, fiabilite_score - 20)
WHERE id = (
  SELECT producteur_id FROM productions
  WHERE id = v_sig.production_cible_id
);

-- Mettre à jour le signalement
UPDATE signalements SET
  statut = 'FRAUDE_CONFIRMÉE',
```

```
    verdict = p_verdict,
    montant_fraude_GU = p_montant_fraude_GU,
    recompense_versee_GU = v_recompense,
    timestamp_verdict = NOW()
  WHERE id = p_signalement_id;

ELSIF p_verdict = 'FAUSSE_ALERTE' THEN
  -- Fausse accusation : caution saisie + pénalité réputation
  UPDATE auditeurs SET
    faux_positifs      = faux_positifs + 1,
    audits_completes   = audits_completes + 1,
    reputation_score   = GREATEST(0, reputation_score - 5),
    statut = CASE
      WHEN faux_positifs + 1 >= 5 THEN 'SUSPENDU'
      ELSE statut
    END
  WHERE id = v_sig.auditeur_id;

  -- Caution versée au Fonds Commun (pénalité)
  INSERT INTO fonds_commun_journal VALUES (
    'FC-PENALITE-' || p_signalement_id,
    'ENTREE_RECUPERATION_FRAUDE', 'ENTREE', v_sig.caution_auditeur_GU,
    'B5', v_sig.auditeur_id,
    (SELECT MAX(solde_apres_GU) FROM fonds_commun_journal),
    (SELECT MAX(solde_apres_GU) FROM fonds_commun_journal)
      + v_sig.caution_auditeur_GU,
    EXTRACT(YEAR FROM NOW())::INTEGER, NOW(),
    'Caution saisie — fausse alerte ' || p_signalement_id
  );
```

```
  -- Réhabiliter la production
  UPDATE productions SET statut = 'APPROUVÉ'
  WHERE id = v_sig.production_cible_id
    AND statut = 'SUSPENDU';

  UPDATE signalements SET
    statut = 'FAUSSE_ALERTE',
    verdict = p_verdict,
    montant_fraude_GU = 0,
    recompense_versee_GU = 0,
    timestamp_verdict = NOW()
  WHERE id = p_signalement_id;

 END IF;

 RAISE NOTICE 'Signalement % clôturé. Verdict : %', p_signalement_id, p_verdict;
END;
$$;



--
═══════════════════════════════════════════════════════════════════════════
═══
-- PARTIE 4 — CONFIGURATION API CHAINQL
-- Points d'accès REST exposés publiquement
--
═══════════════════════════════════════════════════════════════════════════
═══
```

/*

ENDPOINTS PUBLICS (sans authentification) :

_____

GET /chainql/dashboard

GET /chainql/productions/regionales

GET /chainql/traçabilite/:production_id

GET /chainql/analyse/impact-casino-prix

GET /chainql/fonds-commun/projection

GET /chainql/taux/conversion-fiat

GET /chainql/masse-monetaire/audit-physique

ENDPOINTS AUDITEURS (auth niveau 2+) :

_____

POST /chainql/audit/scan

GET /chainql/auditeurs/:id/performance

ENDPOINTS AUDITEURS CERTIFIÉS (auth niveau 3) :

_____

GET /chainql/audit/graphe-collusion

POST /chainql/signalements/soumettre

ENDPOINTS TRIBUNAL (auth Tribunal) :

_____

POST /chainql/signalements/cloturer

FORMAT DE RÉPONSE : JSON | XML | CSV

RATE LIMIT public : 1000 req/min

RATE LIMIT auditeur : 10 000 req/min

RATE LIMIT tribunal : illimité

```
*/
```