AIA Bloc_02 : stripe

Modèle de données OLTP & SQL

Le modèle OLTP conçu pour Stripe répond aux exigences critiques d'une plateforme FinTech à très haut débit, avec un accent sur la conformité, l'intégrité des données et la scalabilité.

Respect strict de la 3e forme normale (3NF)

- Élimination des redondances (ex: country_name stocké une seule fois dans COUNTRY)
- Réduction des anomalies de mise à jour
- Cohérence maximale des données critiques (montants, statuts, devises)

Relations explicites au lieu de many-to-many implicites

- Toute association complexe devient une entité métier à part entière (ex: SUBSCRIPTION, TRANSACTION_EVENT, FRAUD_SCORE)
- Permet de capturer des attributs temporels, des métadonnées et un historique (ex: événements de statut, dates de remboursement)

Traçabilité et auditabilité native

- Chaque entité comporte created_at / updated_at
- Les changements d'état sont historisés via TRANSACTION_EVENT
- Essentiel pour la conformité PCI-DSS, GDPR et les enquêtes de fraude

Extensibilité via jsonb sans compromis structurel

- Champs comme metadata, risk_indicators, evidence permettent d'ajouter des données sans modifier le schéma
- Idéal pour les intégrations rapides (nouveaux PSP, réglementations locales) tout en gardant les colonnes critiques typées (ex: amount, currency_code)

Séparation claire des responsabilités

- CUSTOMER vs CUSTOMER_PROFILE : les données d'identité sont séparées des agrégats analytiques
- Permet des accès différenciés (ex: support accède au client, ML accède au profil)
- Réduit la surface d'attaque (moins de données sensibles exposées)

Préparation native pour le CDC (Change Data Capture)

- Clés primaires UUID, versioning (version dans TRANSACTION)
- Modèle idéal pour alimenter Kafka via Debezium sans transformation complexe
- Base solide pour l'architecture Lambda (batch + streaming)

Alignement avec les exigences métier FinTech

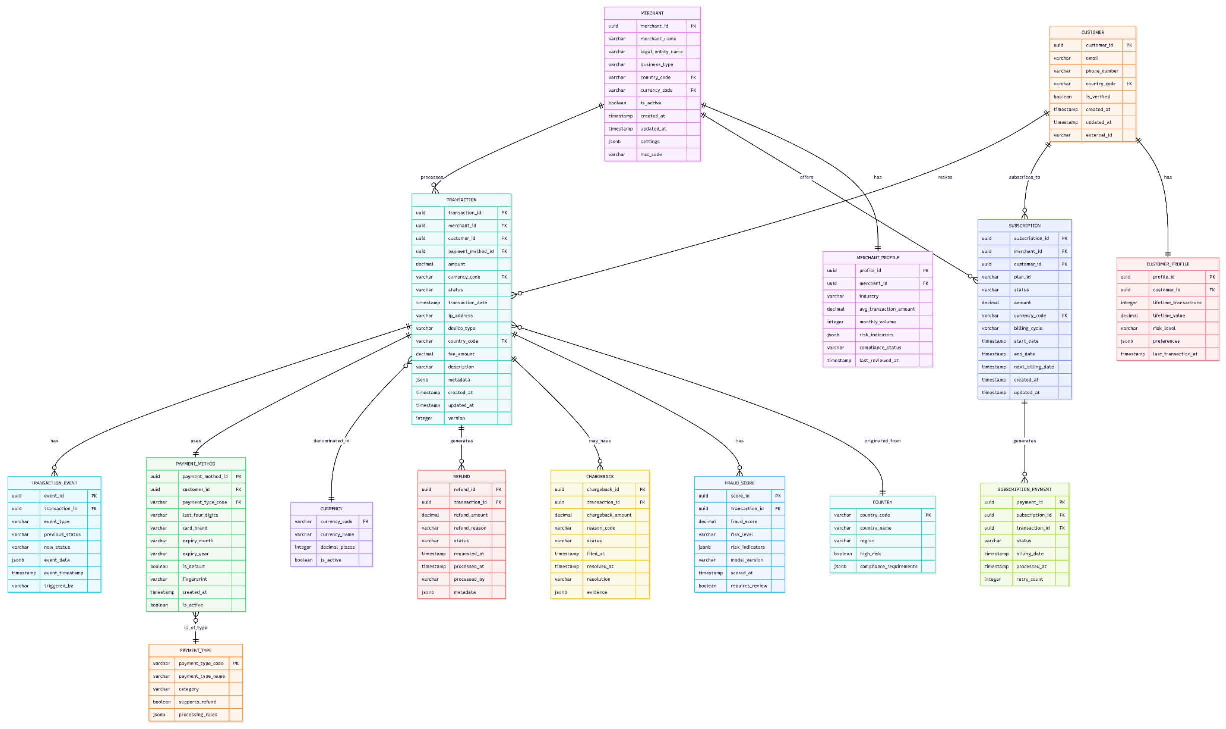
- Modélisation fine des concepts clés : remboursements, chargebacks, abonnements, fraude
- Chaque opération financière est traçable, annulable (logiquement) et justifiable — crucial pour les audits

Performance transactionnelle optimisée

- Indexation implicite via les clés étrangères
- Tables étroites (pas de colonnes inutiles dans TRANSACTION)
- Faible latence même sous forte charge (OLTP distribué compatible avec CockroachDB/PostgreSQL)

```
erDiagram
    MERCHANT ||--o{ TRANSACTION : processes
    CUSTOMER ||--o{ TRANSACTION : makes
    TRANSACTION ||--o{ TRANSACTION_EVENT : has
    TRANSACTION ||--|| PAYMENT_METHOD : uses
    TRANSACTION }o--|| CURRENCY : denominated_in
    TRANSACTION ||--o{ REFUND : generates
    TRANSACTION ||--o{ CHARGEBACK : may_have
    MERCHANT ||--o{ SUBSCRIPTION : offers
    CUSTOMER ||--o{ SUBSCRIPTION : subscribes_to
    SUBSCRIPTION ||--o{ SUBSCRIPTION_PAYMENT : generates
    TRANSACTION ||--o{ FRAUD_SCORE : has
    CUSTOMER ||--|| CUSTOMER_PROFILE : has
    MERCHANT ||--|| MERCHANT_PROFILE : has
    TRANSACTION }o--|| COUNTRY : originated_from
    PAYMENT_METHOD }o--|| PAYMENT_TYPE : is_of_type

    MERCHANT {
        uuid merchant_id PK
        varchar merchant_name
        varchar legal_entity_name
        varchar business_type
        varchar country_code FK
        varchar currency_code FK
        boolean is_active
        timestamp created_at
        timestamp updated_at
        jsonb settings
        varchar mcc_code
    }

    MERCHANT_PROFILE {
        uuid profile_id PK
        uuid merchant_id FK
        varchar industry
        decimal avg_transaction_amount
        integer monthly_volume
        jsonb risk_indicators
        varchar compliance_status
        timestamp last_reviewed_at
    }

    CUSTOMER {
        uuid customer_id PK
        varchar email
        varchar phone_number
        varchar country_code FK
        boolean is_verified
        timestamp created_at
        timestamp updated_at
        varchar external_id
    }

    CUSTOMER_PROFILE {
        uuid profile_id PK
        uuid customer_id FK
        integer lifetime_transactions
        decimal lifetime_value
        varchar risk_level
        jsonb preferences
        timestamp last_transaction_at
    }

    TRANSACTION {
        uuid transaction_id PK
        uuid merchant_id FK
        uuid customer_id FK
        uuid payment_method_id FK
        decimal amount
        varchar currency_code FK
        varchar status
        timestamp transaction_date
        varchar ip_address
        varchar device_type
        varchar country_code FK
        decimal fee_amount
        varchar description
        jsonb metadata
        timestamp created_at
        timestamp updated_at
        integer version
    }

    TRANSACTION_EVENT {
        uuid event_id PK
        uuid transaction_id FK
        varchar event_type
        varchar previous_status
        varchar new_status
        jsonb event_data
        timestamp event_timestamp
        varchar triggered_by
    }

    PAYMENT_METHOD {
        uuid payment_method_id PK
        uuid customer_id FK
        varchar payment_type_code FK
        varchar last_four_digits
        varchar card_brand
        varchar expiry_month
        varchar expiry_year
        boolean is_default
        varchar fingerprint
        timestamp created_at
        boolean is_active
    }

    PAYMENT_TYPE {
        varchar payment_type_code PK
        varchar payment_type_name
        varchar category
        boolean supports_refund
        jsonb processing_rules
    }

    REFUND {
        uuid refund_id PK
        uuid transaction_id FK
        decimal refund_amount
        varchar refund_reason
        varchar status
        timestamp requested_at
        timestamp processed_at
        varchar processed_by
        jsonb metadata
    }

    CHARGEBACK {
        uuid chargeback_id PK
        uuid transaction_id FK
        decimal chargeback_amount
        varchar reason_code
        varchar status
        timestamp filed_at
        timestamp resolved_at
        varchar resolution
        jsonb evidence
    }

    SUBSCRIPTION {
        uuid subscription_id PK
        uuid merchant_id FK
        uuid customer_id FK
        varchar plan_id
        varchar status
        decimal amount
        varchar currency_code FK
        varchar billing_cycle
        timestamp start_date
        timestamp end_date
        timestamp next_billing_date
        timestamp created_at
        timestamp updated_at
    }

    SUBSCRIPTION_PAYMENT {
        uuid payment_id PK
        uuid subscription_id FK
        uuid transaction_id FK
        varchar status
        timestamp billing_date
        timestamp processed_at
        integer retry_count
    }

    FRAUD_SCORE {
        uuid score_id PK
        uuid transaction_id FK
        decimal fraud_score
        varchar risk_level
        jsonb risk_indicators
        varchar model_version
        timestamp scored_at
        boolean requires_review
    }

    CURRENCY {
        varchar currency_code PK
        varchar currency_name
        integer decimal_places
        boolean is_active
    }

    COUNTRY {
        varchar country_code PK
        varchar country_name
        varchar region
        boolean high_risk
        jsonb compliance_requirements
    }
```

Diagramme ERD OLTP STRIPE



Script SQL de Création OLTP

```
-- ============================================================
-- STRIPE OLTP DATABASE - PostgreSQL 15+
-- Architecture: Multi-tenant, Sharded, Partitioned
-- ============================================================

-- Extensions nécessaires
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
CREATE EXTENSION IF NOT EXISTS "pgcrypto";
CREATE EXTENSION IF NOT EXISTS "pg_trgm"; -- Pour recherche full-text
CREATE EXTENSION IF NOT EXISTS "btree_gin"; -- Pour indexation composite



-- ============================================================
-- TABLES DE RÉFÉRENCE (Slowly Changing Dimensions)
-- ============================================================

CREATE TABLE country (
    country_code VARCHAR(3) PRIMARY KEY,
    country_name VARCHAR(100) NOT NULL,
    region VARCHAR(50) NOT NULL,
```

```sql
    high_risk BOOLEAN DEFAULT FALSE,
    compliance_requirements JSONB DEFAULT '{}',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE currency (
    currency_code VARCHAR(3) PRIMARY KEY,
    currency_name VARCHAR(50) NOT NULL,
    decimal_places INTEGER DEFAULT 2,
    is_active BOOLEAN DEFAULT TRUE,
    exchange_rate_to_usd DECIMAL(18, 6),
    last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE payment_type (
    payment_type_code VARCHAR(20) PRIMARY KEY,
    payment_type_name VARCHAR(50) NOT NULL,
    category VARCHAR(30) NOT NULL, -- card, bank_transfer, wallet, crypto
    supports_refund BOOLEAN DEFAULT TRUE,
    processing_rules JSONB DEFAULT '{}',
    is_active BOOLEAN DEFAULT TRUE
);

-- -----------------------------------------------------
-- TABLES PRINCIPALES
-- -----------------------------------------------------

-- Table MERCHANT avec audit trail
CREATE TABLE merchant (
    merchant_id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    merchant_name VARCHAR(255) NOT NULL,
    legal_entity_name VARCHAR(255) NOT NULL,
    business_type VARCHAR(50) NOT NULL,
    country_code VARCHAR(3) NOT NULL REFERENCES country(country_code),
    currency_code VARCHAR(3) NOT NULL REFERENCES currency(currency_code),
    is_active BOOLEAN DEFAULT TRUE,
    mcc_code VARCHAR(4), -- Merchant Category Code
    tax_id VARCHAR(50),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    settings JSONB DEFAULT '{}',

    CONSTRAINT chk_business_type CHECK (business_type IN ('individual',
'company', 'non_profit', 'government'))
);
```

```sql
CREATE TABLE merchant_profile (
    profile_id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    merchant_id UUID NOT NULL REFERENCES merchant(merchant_id) ON DELETE
CASCADE,
    industry VARCHAR(100),
    avg_transaction_amount DECIMAL(18, 2),
    monthly_volume INTEGER,
    risk_indicators JSONB DEFAULT '{}',
    compliance_status VARCHAR(20) DEFAULT 'pending',
    kyc_verified_at TIMESTAMP,
    last_reviewed_at TIMESTAMP,

    CONSTRAINT chk_compliance_status CHECK (compliance_status IN
('pending', 'approved', 'rejected', 'under_review')),
    CONSTRAINT uk_merchant_profile UNIQUE (merchant_id)
);

-- Table CUSTOMER
CREATE TABLE customer (
    customer_id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    email VARCHAR(255) NOT NULL,
    phone_number VARCHAR(20),
    country_code VARCHAR(3) REFERENCES country(country_code),
    is_verified BOOLEAN DEFAULT FALSE,
    external_id VARCHAR(100), -- ID from merchant's system
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    CONSTRAINT uk_customer_email UNIQUE (email)
);

CREATE INDEX idx_customer_external_id ON customer(external_id) WHERE
external_id IS NOT NULL;

CREATE TABLE customer_profile (
    profile_id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    customer_id UUID NOT NULL REFERENCES customer(customer_id) ON DELETE
CASCADE,
    lifetime_transactions INTEGER DEFAULT 0,
    lifetime_value DECIMAL(18, 2) DEFAULT 0,
    risk_level VARCHAR(20) DEFAULT 'low',
    preferences JSONB DEFAULT '{}',
    last_transaction_at TIMESTAMP,

    CONSTRAINT chk_risk_level CHECK (risk_level IN ('low', 'medium',
'high', 'blocked')),
    CONSTRAINT uk_customer_profile UNIQUE (customer_id)
);

-- Table PAYMENT_METHOD
CREATE TABLE payment_method (
    payment_method_id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    customer_id UUID NOT NULL REFERENCES customer(customer_id) ON DELETE
CASCADE,
    payment_type_code VARCHAR(20) NOT NULL REFERENCES
payment_type(payment_type_code),
    last_four_digits VARCHAR(4),
    card_brand VARCHAR(20), -- visa, mastercard, amex, etc.
    expiry_month VARCHAR(2),
    expiry_year VARCHAR(4),
    is_default BOOLEAN DEFAULT FALSE,
    fingerprint VARCHAR(64), -- Hash for duplicate detection
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_payment_method_customer ON payment_method(customer_id)
WHERE is_active = TRUE;
CREATE INDEX idx_payment_method_fingerprint ON
payment_method(fingerprint);

-- =====================================================
-- TABLE TRANSACTION (PARTITIONNÉE PAR DATE)
-- =====================================================

CREATE TABLE transaction (
    transaction_id UUID NOT NULL DEFAULT uuid_generate_v4(),
    merchant_id UUID NOT NULL REFERENCES merchant(merchant_id),
    customer_id UUID NOT NULL REFERENCES customer(customer_id),
    payment_method_id UUID NOT NULL REFERENCES
payment_method(payment_method_id),
    amount DECIMAL(18, 2) NOT NULL CHECK (amount >= 0),
    currency_code VARCHAR(3) NOT NULL REFERENCES currency(currency_code),
    status VARCHAR(20) NOT NULL DEFAULT 'pending',
    transaction_date TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    ip_address INET,
    device_type VARCHAR(20),
    country_code VARCHAR(3) REFERENCES country(country_code),
    fee_amount DECIMAL(18, 2) DEFAULT 0,
    net_amount DECIMAL(18, 2) GENERATED ALWAYS AS (amount - fee_amount)
STORED,
```

```sql
    description TEXT,
    metadata JSONB DEFAULT '{}',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    version INTEGER DEFAULT 1, -- Pour optimistic locking

    PRIMARY KEY (transaction_id, transaction_date),
    CONSTRAINT chk_status CHECK (status IN ('pending', 'processing',
'succeeded', 'failed', 'refunded', 'partially_refunded', 'disputed'))
) PARTITION BY RANGE (transaction_date);

-- Partitions mensuelles (exemple pour 2025)
CREATE TABLE transaction_2025_01 PARTITION OF transaction
    FOR VALUES FROM ('2025-01-01') TO ('2025-02-01');

CREATE TABLE transaction_2025_02 PARTITION OF transaction
    FOR VALUES FROM ('2025-02-01') TO ('2025-03-01');

CREATE TABLE transaction_2025_03 PARTITION OF transaction
    FOR VALUES FROM ('2025-03-01') TO ('2025-04-01');

-- Function pour créer automatiquement les partitions
CREATE OR REPLACE FUNCTION create_transaction_partition()
RETURNS void AS $$
DECLARE
    start_date DATE;
    end_date DATE;
    partition_name TEXT;
BEGIN
    start_date := DATE_TRUNC('month', CURRENT_DATE + INTERVAL '1 month');
    end_date := start_date + INTERVAL '1 month';
    partition_name := 'transaction_' || TO_CHAR(start_date, 'YYYY_MM');

    EXECUTE format('CREATE TABLE IF NOT EXISTS %I PARTITION OF
transaction FOR VALUES FROM (%L) TO (%L)',
                   partition_name, start_date, end_date);
END;
$$ LANGUAGE plpgsql;

-- Index sur les partitions
CREATE INDEX idx_transaction_merchant ON transaction(merchant_id,
transaction_date DESC);
CREATE INDEX idx_transaction_customer ON transaction(customer_id,
transaction_date DESC);
CREATE INDEX idx_transaction_status ON transaction(status,
transaction_date DESC);
```

```sql
CREATE INDEX idx_transaction_amount ON transaction(amount) WHERE status =
'succeeded';
CREATE INDEX idx_transaction_metadata ON transaction USING GIN(metadata);

-- Table d'événements de transaction (Event Sourcing pattern)
CREATE TABLE transaction_event (
    event_id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    transaction_id UUID NOT NULL,
    event_type VARCHAR(50) NOT NULL,
    previous_status VARCHAR(20),
    new_status VARCHAR(20),
    event_data JSONB DEFAULT '{}',
    event_timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    triggered_by VARCHAR(100), -- user_id or system

    CONSTRAINT fk_transaction FOREIGN KEY (transaction_id,
event_timestamp)
        REFERENCES transaction(transaction_id, transaction_date) ON
DELETE CASCADE
);

CREATE INDEX idx_transaction_event_txn ON
transaction_event(transaction_id, event_timestamp DESC);
CREATE INDEX idx_transaction_event_type ON transaction_event(event_type,
event_timestamp DESC);

-- ─────────────────────────────────────────────────────
-- TABLES LIÉES AUX TRANSACTIONS
-- ─────────────────────────────────────────────────────

CREATE TABLE refund (
    refund_id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    transaction_id UUID NOT NULL,
    refund_amount DECIMAL(18, 2) NOT NULL CHECK (refund_amount > 0),
    refund_reason VARCHAR(100),
    status VARCHAR(20) DEFAULT 'pending',
    requested_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    processed_at TIMESTAMP,
    processed_by VARCHAR(100),
    metadata JSONB DEFAULT '{}',

    CONSTRAINT chk_refund_status CHECK (status IN ('pending',
'processing', 'succeeded', 'failed', 'cancelled'))
);

CREATE INDEX idx_refund_transaction ON refund(transaction_id);
CREATE INDEX idx_refund_status ON refund(status, requested_at DESC);

CREATE TABLE chargeback (
    chargeback_id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    transaction_id UUID NOT NULL,
    chargeback_amount DECIMAL(18, 2) NOT NULL,
    reason_code VARCHAR(10) NOT NULL,
    status VARCHAR(20) DEFAULT 'filed',
    filed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    resolved_at TIMESTAMP,
    resolution VARCHAR(20),
    evidence JSONB DEFAULT '{}',

    CONSTRAINT chk_chargeback_status CHECK (status IN ('filed',
'under_review', 'won', 'lost', 'withdrawn')),
    CONSTRAINT chk_chargeback_resolution CHECK (resolution IN
('merchant_wins', 'customer_wins', 'split', NULL))
);

CREATE INDEX idx_chargeback_transaction ON chargeback(transaction_id);
CREATE INDEX idx_chargeback_status ON chargeback(status, filed_at DESC);

-- -----------------------------------------------------
-- SUBSCRIPTION MANAGEMENT
-- -----------------------------------------------------

CREATE TABLE subscription (
    subscription_id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    merchant_id UUID NOT NULL REFERENCES merchant(merchant_id),
    customer_id UUID NOT NULL REFERENCES customer(customer_id),
    plan_id VARCHAR(50) NOT NULL,
    status VARCHAR(20) DEFAULT 'active',
    amount DECIMAL(18, 2) NOT NULL,
    currency_code VARCHAR(3) NOT NULL REFERENCES currency(currency_code),
    billing_cycle VARCHAR(20) NOT NULL, -- daily, weekly, monthly, yearly
    start_date TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    end_date TIMESTAMP,
    next_billing_date TIMESTAMP NOT NULL,
    trial_end_date TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    metadata JSONB DEFAULT '{}',

    CONSTRAINT chk_subscription_status CHECK (status IN ('active',
'trialing', 'past_due', 'cancelled', 'unpaid', 'paused')),
    CONSTRAINT chk_billing_cycle CHECK (billing_cycle IN ('daily',
'weekly', 'monthly', 'quarterly', 'yearly'))
);

CREATE INDEX idx_subscription_merchant ON subscription(merchant_id,
status);
CREATE INDEX idx_subscription_customer ON subscription(customer_id,
status);
CREATE INDEX idx_subscription_next_billing ON
subscription(next_billing_date) WHERE status IN ('active', 'trialing');

CREATE TABLE subscription_payment (
    payment_id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    subscription_id UUID NOT NULL REFERENCES
subscription(subscription_id) ON DELETE CASCADE,
    transaction_id UUID,
    status VARCHAR(20) DEFAULT 'pending',
    billing_date TIMESTAMP NOT NULL,
    processed_at TIMESTAMP,
    retry_count INTEGER DEFAULT 0,
    next_retry_at TIMESTAMP,

    CONSTRAINT chk_subscription_payment_status CHECK (status IN
('pending', 'succeeded', 'failed', 'skipped'))
);

CREATE INDEX idx_subscription_payment_sub ON
subscription_payment(subscription_id, billing_date DESC);

-- —----------------------------------------------------
-- FRAUD DETECTION
-- —----------------------------------------------------

CREATE TABLE fraud_score (
    score_id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    transaction_id UUID NOT NULL,
    fraud_score DECIMAL(5, 4) NOT NULL CHECK (fraud_score BETWEEN 0 AND
1),
    risk_level VARCHAR(20) NOT NULL,
    risk_indicators JSONB DEFAULT '{}',
    model_version VARCHAR(20) NOT NULL,
    scored_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    requires_review BOOLEAN DEFAULT FALSE,
    reviewed_by VARCHAR(100),
    review_decision VARCHAR(20),

    CONSTRAINT chk_risk_level CHECK (risk_level IN ('very_low', 'low',
'medium', 'high', 'very_high')),
    CONSTRAINT chk_review_decision CHECK (review_decision IN ('approve',
'decline', 'needs_more_info', NULL))
);

CREATE INDEX idx_fraud_score_transaction ON fraud_score(transaction_id);
CREATE INDEX idx_fraud_score_high_risk ON fraud_score(fraud_score DESC,
scored_at DESC)
    WHERE risk_level IN ('high', 'very_high');
CREATE INDEX idx_fraud_score_review ON fraud_score(requires_review,
scored_at)
    WHERE requires_review = TRUE AND review_decision IS NULL;

-- ––––––––––––––––––––––––––––––––––––––––––––––––––––––
-- TRIGGERS ET FONCTIONS
-- ––––––––––––––––––––––––––––––––––––––––––––––––––––––

-- Trigger pour mettre à jour updated_at
CREATE OR REPLACE FUNCTION update_updated_at_column()
RETURNS TRIGGER AS $$
BEGIN
    NEW.updated_at = CURRENT_TIMESTAMP;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_merchant_updated_at BEFORE UPDATE ON merchant
    FOR EACH ROW EXECUTE FUNCTION update_updated_at_column();

CREATE TRIGGER update_customer_updated_at BEFORE UPDATE ON customer
    FOR EACH ROW EXECUTE FUNCTION update_updated_at_column();

CREATE TRIGGER update_transaction_updated_at BEFORE UPDATE ON transaction
    FOR EACH ROW EXECUTE FUNCTION update_updated_at_column();

CREATE TRIGGER update_subscription_updated_at BEFORE UPDATE ON
subscription
    FOR EACH ROW EXECUTE FUNCTION update_updated_at_column();

-- Trigger pour enregistrer les changements de statut de transaction
CREATE OR REPLACE FUNCTION log_transaction_status_change()
RETURNS TRIGGER AS $$
BEGIN
    IF OLD.status IS DISTINCT FROM NEW.status THEN
        INSERT INTO transaction_event (
            transaction_id,
            event_type,
            previous_status,
            new_status,
            event_data,
            triggered_by
        ) VALUES (
            NEW.transaction_id,
            'status_change',
            OLD.status,
            NEW.status,
            jsonb_build_object('version', NEW.version),
            current_user
        );
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER log_transaction_status AFTER UPDATE ON transaction
    FOR EACH ROW EXECUTE FUNCTION log_transaction_status_change();

-- Fonction pour calculer les frais
CREATE OR REPLACE FUNCTION calculate_transaction_fee(
    p_amount DECIMAL,
    p_currency VARCHAR,
    p_payment_type VARCHAR,
    p_country VARCHAR
) RETURNS DECIMAL AS $$
DECLARE
    base_fee DECIMAL := 0.029; -- 2.9%
    fixed_fee DECIMAL := 0.30;
    total_fee DECIMAL;
BEGIN
    -- Logique simplifiée de calcul des frais
    total_fee := (p_amount * base_fee) + fixed_fee;

    -- Ajustements selon le type de paiement
    IF p_payment_type = 'card_international' THEN
        total_fee := total_fee + (p_amount * 0.01); -- +1% pour
international
    END IF;

    RETURN ROUND(total_fee, 2);
END;
$$ LANGUAGE plpgsql IMMUTABLE;
```

```
-- ---------------------------------------------------
-- VUES MATÉRIALISÉES POUR PERFORMANCE
-- ---------------------------------------------------

CREATE MATERIALIZED VIEW mv_merchant_daily_summary AS
SELECT
    merchant_id,
    DATE(transaction_date) AS transaction_day,
    currency_code,
    COUNT(*) AS transaction_count,
    COUNT(*) FILTER (WHERE status = 'succeeded') AS successful_count,
    SUM(amount) FILTER (WHERE status = 'succeeded') AS total_amount,
    SUM(fee_amount) FILTER (WHERE status = 'succeeded') AS total_fees,
    SUM(net_amount) FILTER (WHERE status = 'succeeded') AS net_revenue,
    AVG(amount) FILTER (WHERE status = 'succeeded') AS
avg_transaction_amount,
    COUNT(DISTINCT customer_id) AS unique_customers
FROM transaction
WHERE transaction_date >= CURRENT_DATE - INTERVAL '90 days'
GROUP BY merchant_id, DATE(transaction_date), currency_code;

CREATE UNIQUE INDEX idx_mv_merchant_daily_pk ON
mv_merchant_daily_summary(merchant_id, transaction_day, currency_code);
CREATE INDEX idx_mv_merchant_daily_date ON
mv_merchant_daily_summary(transaction_day DESC);

-- Refresh automatique quotidien
CREATE OR REPLACE FUNCTION refresh_daily_summaries()
RETURNS void AS $$
BEGIN
    REFRESH MATERIALIZED VIEW CONCURRENTLY mv_merchant_daily_summary;
END;
$$ LANGUAGE plpgsql;

-- ---------------------------------------------------
-- GRANTS ET SÉCURITÉ
-- ---------------------------------------------------

-- Role pour l'application (lecture/écriture)
CREATE ROLE stripe_app_user WITH LOGIN PASSWORD
'change_me_in_production';
GRANT CONNECT ON DATABASE stripe_oltp TO stripe_app_user;
GRANT USAGE ON SCHEMA public TO stripe_app_user;
GRANT SELECT, INSERT, UPDATE ON ALL TABLES IN SCHEMA public TO
stripe_app_user;
GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA public TO stripe_app_user;
```

```sql
-- Role pour analytics (lecture seule)
CREATE ROLE stripe_analytics_user WITH LOGIN PASSWORD
'change_me_in_production';
GRANT CONNECT ON DATABASE stripe_oltp TO stripe_analytics_user;
GRANT USAGE ON SCHEMA public TO stripe_analytics_user;
GRANT SELECT ON ALL TABLES IN SCHEMA public TO stripe_analytics_user;

-- Row Level Security pour multi-tenancy
ALTER TABLE transaction ENABLE ROW LEVEL SECURITY;

CREATE POLICY merchant_isolation ON transaction
    FOR ALL
    USING (merchant_id =
current_setting('app.current_merchant_id')::UUID);

-- ——————————————————————————————————————————————————————
-- COMMENTAIRES POUR DOCUMENTATION
-- ——————————————————————————————————————————————————————

COMMENT ON TABLE transaction IS 'Table principale des transactions,
partitionnée par mois pour optimisation';
COMMENT ON COLUMN transaction.version IS 'Version pour optimistic
locking, incrémenté à chaque update';
COMMENT ON TABLE transaction_event IS 'Event sourcing pour audit complet
des changements de transaction';
COMMENT ON TABLE fraud_score IS 'Scores de fraude calculés par ML en
temps réel';
COMMENT ON MATERIALIZED VIEW mv_merchant_daily_summary IS 'Agrégation
quotidienne par merchant, refresh automatique';
```