YTEMPIRE Enterprise Data Architecture

Version 1.0 - PostgreSQL-Centric Design

Table of Contents

Executive Summary

This Enterprise Data Architecture document defines the comprehensive data management strategy for YTEMPIRE, leveraging PostgreSQL 15 as the primary database system. The architecture is designed to support autonomous content generation at scale, from 2 initial channels to 1000+ channels, while maintaining sub-second query performance and 99.99% data availability.

Key Architectural Decisions:

- **PostgreSQL 15** as the primary OLTP database with advanced features (partitioning, JSONB, full-text search)
- **Hybrid Storage Model**: Relational tables for structured data, JSONB for flexible schemas
- **Time-Series Optimization**: Partitioned tables for metrics and analytics data
- **Multi-Layer Caching**: Redis for hot data, PostgreSQL for warm data, S3 for cold storage
- **Event-Driven Architecture**: Change Data Capture (CDC) for real-time data synchronization

Design Goals:

- Support 100,000+ videos and 10M+ analytics events daily
- Sub-100ms query response for operational queries
- 5-minute data freshness for analytics
- Zero data loss with point-in-time recovery
- GDPR and CCPA compliance built-in

Data Architecture Overview

Core Design Principles

1. Data as a Strategic Asset

```yaml
principles:
  data_driven_decisions:
    - Every feature backed by data insights
    - A/B testing infrastructure built-in
    - ML-ready data structures

  single_source_of_truth:
    - Authoritative data sources clearly defined
    - No duplicate data storage
    - Clear data lineage

  scalability_first:
    - Design for 100x current load
    - Horizontal scaling capabilities
    - Cost-effective growth path
```

2. Performance by Design

```yaml
performance_targets:
  operational_queries:
    latency_p50: 10ms
    latency_p99: 100ms
    throughput: 10000_qps

  analytical_queries:
    latency_p50: 500ms
    latency_p99: 5000ms
    throughput: 100_qps

  data_ingestion:
    events_per_second: 50000
    batch_size: 10000
    latency: <1_second
```

3. Security and Compliance

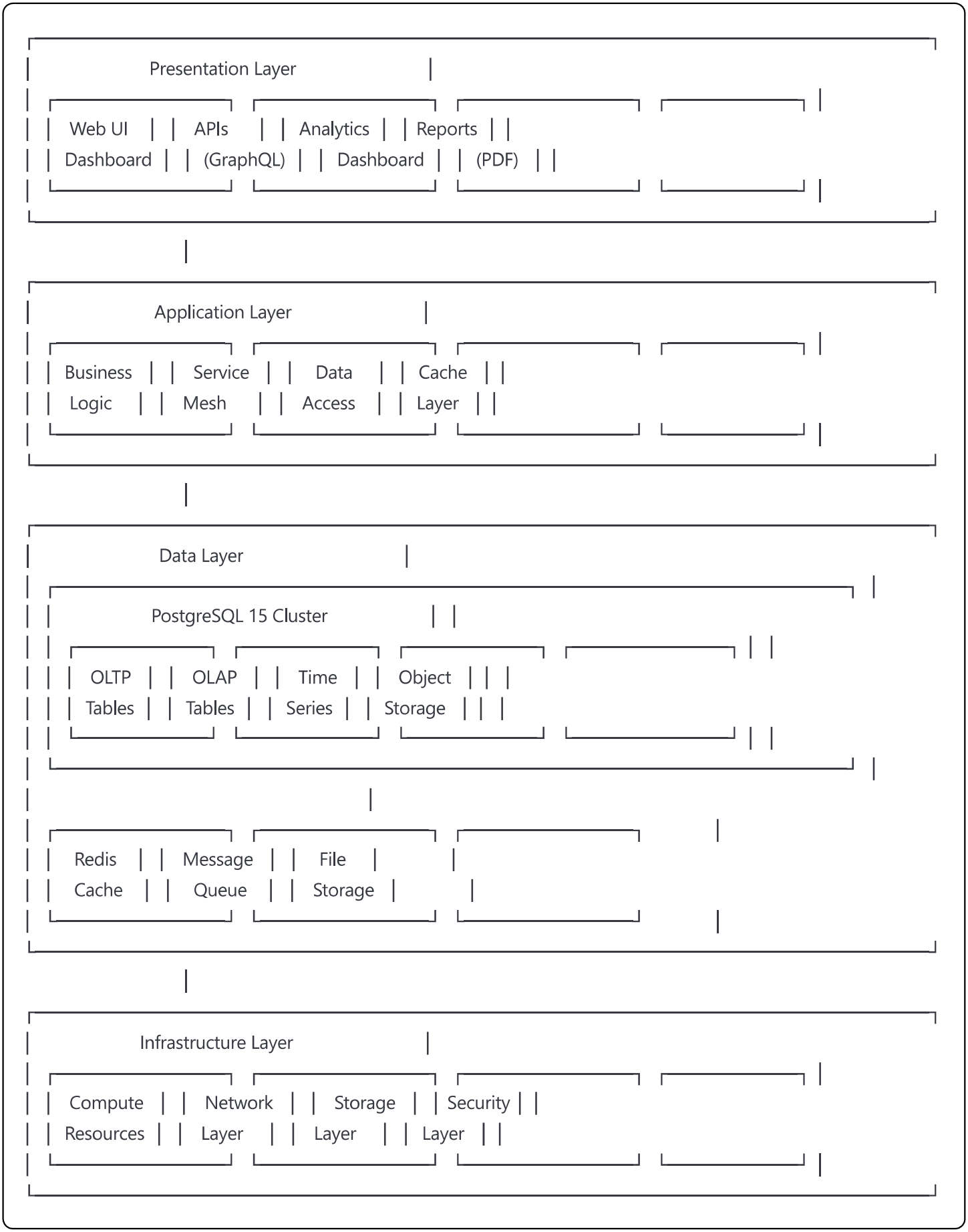
```yaml
```

```yaml
security_framework:
  encryption:
    at_rest: AES-256
    in_transit: TLS_1.3
    key_management: HashiCorp_Vault

  access_control:
    authentication: OAuth2_JWT
    authorization: RBAC
    audit_logging: comprehensive

  compliance:
    gdpr: full_compliance
    ccpa: full_compliance
    sox: audit_ready
```

Data Architecture Layers



Technology Stack

Core Database Technologies

yaml

```yaml
postgresql_stack:
  core:
    version: 15.5
    deployment: master_slave_replication
    connection_pooling: pgbouncer
    monitoring: pg_stat_statements

  extensions:
    - pgcrypto  # Encryption
    - uuid-ossp  # UUID generation
    - pg_trgm  # Trigram similarity
    - btree_gist # Advanced indexing
    - pg_stat_statements # Query analysis
    - postgres_fdw # Foreign data wrapper
    - timescaledb # Time-series optimization

  tools:
    backup: pgbackrest
    migration: flyway
    monitoring: prometheus_postgres_exporter
    query_analysis: pgbadger
```

Supporting Technologies

yaml

```yaml
data_ecosystem:
  caching:
    primary: redis_7
    cdn: cloudflare
    application: memcached

  streaming:
    platform: apache_kafka
    processing: apache_flink
    cdc: debezium

  analytics:
    olap: apache_druid
    visualization: grafana
    ml_feature_store: feast

  storage:
    object: minio_s3_compatible
    file: nfs_v4
    archive: aws_glacier
```

PostgreSQL Database Design

Schema Architecture

Multi-Schema Strategy

sql

```sql
-- Core schemas for logical separation
CREATE SCHEMA IF NOT EXISTS core;      -- Core business entities
CREATE SCHEMA IF NOT EXISTS analytics; -- Analytics and reporting
CREATE SCHEMA IF NOT EXISTS staging;   -- ETL staging area
CREATE SCHEMA IF NOT EXISTS archive;   -- Historical data
CREATE SCHEMA IF NOT EXISTS audit;     -- Audit trails
CREATE SCHEMA IF NOT EXISTS ml;        -- Machine learning features

-- Schema permissions
GRANT USAGE ON SCHEMA core TO ytempire_app;
GRANT USAGE ON SCHEMA analytics TO ytempire_analytics;
GRANT ALL ON SCHEMA staging TO ytempire_etl;

-- Search path configuration
ALTER DATABASE ytempire SET search_path TO core, public;
```

Schema Design Principles

```yaml
schema_design:
  core:
    purpose: Operational data for application
    characteristics:
      - Normalized to 3NF
      - ACID compliance critical
      - Real-time access patterns
      - Row-level security

  analytics:
    purpose: Denormalized for reporting
    characteristics:
      - Star/snowflake schemas
      - Materialized views
      - Columnar storage friendly
      - Batch update patterns

  staging:
    purpose: ETL processing area
    characteristics:
      - Temporary tables
      - Minimal constraints
      - Bulk operations optimized
      - Auto-cleanup policies
```

Table Structures

Core Business Entities

```sql
```

```sql
-- Channels table with comprehensive metadata
CREATE TABLE core.channels (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    youtube_channel_id VARCHAR(255) UNIQUE NOT NULL,
    name VARCHAR(255) NOT NULL,
    description TEXT,
    niche VARCHAR(100) NOT NULL,
    status VARCHAR(50) NOT NULL DEFAULT 'active',

    -- Channel configuration
    config JSONB NOT NULL DEFAULT '{}',
    personality_profile JSONB NOT NULL DEFAULT '{}',
    content_strategy JSONB NOT NULL DEFAULT '{}',

    -- Monetization settings
    monetization JSONB NOT NULL DEFAULT '{
        "adsense_enabled": true,
        "sponsorships_enabled": false,
        "affiliates_enabled": false
    }',

    -- Performance metrics
    metrics JSONB NOT NULL DEFAULT '{}',

    -- Audit fields
    created_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,
    created_by UUID REFERENCES core.users(id),
    updated_by UUID REFERENCES core.users(id),

    -- Constraints
    CONSTRAINT channels_status_check CHECK (
        status IN ('active', 'paused', 'suspended', 'archived')
    ),
    CONSTRAINT channels_niche_check CHECK (
        niche IN ('tech', 'education', 'entertainment', 'lifestyle', 'gaming', 'other')
    )
);

-- Indexes for channels
CREATE INDEX idx_channels_status ON core.channels(status) WHERE status = 'active';
CREATE INDEX idx_channels_niche ON core.channels(niche);
CREATE INDEX idx_channels_youtube_id ON core.channels(youtube_channel_id);
CREATE INDEX idx_channels_config ON core.channels USING GIN (config);
CREATE INDEX idx_channels_created_at ON core.channels(created_at DESC);
```

```
-- Triggers
CREATE TRIGGER update_channels_updated_at
    BEFORE UPDATE ON core.channels
    FOR EACH ROW
    EXECUTE FUNCTION core.update_updated_at_column();

-- Videos table with content and performance tracking
CREATE TABLE core.videos (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    channel_id UUID NOT NULL REFERENCES core.channels(id) ON DELETE CASCADE,
    youtube_video_id VARCHAR(255) UNIQUE,

    -- Content metadata
    title VARCHAR(255) NOT NULL,
    description TEXT,
    tags TEXT[],
    category VARCHAR(100),
    language VARCHAR(10) NOT NULL DEFAULT 'en',
    duration INTEGER NOT NULL, -- seconds

    -- Generation metadata
    script TEXT NOT NULL,
    script_metadata JSONB NOT NULL DEFAULT '{}',
    generation_params JSONB NOT NULL DEFAULT '{}',
    quality_scores JSONB NOT NULL DEFAULT '{}',

    -- Publishing metadata
    status VARCHAR(50) NOT NULL DEFAULT 'draft',
    scheduled_publish_at TIMESTAMPTZ,
    published_at TIMESTAMPTZ,
    visibility VARCHAR(50) DEFAULT 'public',

    -- Performance tracking
    performance_metrics JSONB NOT NULL DEFAULT '{}',
    analytics_snapshot JSONB NOT NULL DEFAULT '{}',

    -- File references
    video_file_path VARCHAR(500),
    thumbnail_path VARCHAR(500),
    captions_path VARCHAR(500),

    -- Audit fields
    created_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,
    workflow_id UUID REFERENCES core.workflows(id),

    -- Constraints
```

```
    CONSTRAINT videos_status_check CHECK (
        status IN ('draft', 'generating', 'processing', 'scheduled', 'published', 'failed', 'archived')
    ),
    CONSTRAINT videos_visibility_check CHECK (
        visibility IN ('public', 'unlisted', 'private')
    )
);

-- Comprehensive indexing for videos
CREATE INDEX idx_videos_channel_id ON core.videos(channel_id);
CREATE INDEX idx_videos_status ON core.videos(status);
CREATE INDEX idx_videos_published_at ON core.videos(published_at DESC) WHERE published_at IS NOT NULL;
CREATE INDEX idx_videos_scheduled ON core.videos(scheduled_publish_at) WHERE status = 'scheduled';
CREATE INDEX idx_videos_tags ON core.videos USING GIN (tags);
CREATE INDEX idx_videos_performance ON core.videos USING GIN (performance_metrics);
CREATE INDEX idx_videos_composite ON core.videos(channel_id, status, published_at DESC);

-- Trends tracking table
CREATE TABLE core.trends (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    topic VARCHAR(500) NOT NULL,
    topic_normalized VARCHAR(500) NOT NULL, -- Lowercase, cleaned

    -- Trend metadata
    category VARCHAR(100),
    subcategory VARCHAR(100),
    keywords TEXT[],
    related_topics TEXT[],

    -- Source tracking
    source VARCHAR(50) NOT NULL,
    source_metadata JSONB NOT NULL DEFAULT '{}',
    source_url VARCHAR(1000),

    -- Scoring and analysis
    viral_score DECIMAL(3,2) NOT NULL CHECK (viral_score >= 0 AND viral_score <= 1),
    competition_score DECIMAL(3,2) NOT NULL CHECK (competition_score >= 0 AND competition_score <= 1),
    opportunity_score DECIMAL(3,2) GENERATED ALWAYS AS (
        viral_score * (1 - competition_score)
    ) STORED,

    -- Temporal data
    first_detected TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,
    peak_time TIMESTAMPTZ,
    expiry_time TIMESTAMPTZ,

    -- Analytics
```

```
    search_volume INTEGER,
    growth_rate DECIMAL(5,2), -- Percentage
    geographic_data JSONB DEFAULT '{}',
    demographic_data JSONB DEFAULT '{}',

    -- Processing status
    status VARCHAR(50) NOT NULL DEFAULT 'new',
    processed_at TIMESTAMPTZ,
    action_taken VARCHAR(100),

    -- Audit
    created_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,

    -- Constraints
    CONSTRAINT trends_source_check CHECK (
        source IN ('youtube', 'google_trends', 'reddit', 'twitter', 'tiktok', 'custom')
    ),
    CONSTRAINT trends_status_check CHECK (
        status IN ('new', 'analyzing', 'qualified', 'rejected', 'expired')
    )
);

-- Trend indexes
CREATE INDEX idx_trends_topic ON core.trends(topic_normalized);
CREATE INDEX idx_trends_viral_score ON core.trends(viral_score DESC);
CREATE INDEX idx_trends_opportunity ON core.trends(opportunity_score DESC);
CREATE INDEX idx_trends_status ON core.trends(status) WHERE status IN ('new', 'analyzing');
CREATE INDEX idx_trends_temporal ON core.trends(first_detected DESC, expiry_time);
CREATE INDEX idx_trends_source ON core.trends(source, viral_score DESC);

-- Workflows table for process tracking
CREATE TABLE core.workflows (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    type VARCHAR(50) NOT NULL,
    status VARCHAR(50) NOT NULL DEFAULT 'pending',
    priority INTEGER NOT NULL DEFAULT 5 CHECK (priority BETWEEN 1 AND 10),

    -- Workflow context
    channel_id UUID REFERENCES core.channels(id),
    trend_id UUID REFERENCES core.trends(id),
    video_id UUID REFERENCES core.videos(id),

    -- Execution metadata
    config JSONB NOT NULL DEFAULT '{}',
    state JSONB NOT NULL DEFAULT '{}',
    steps JSONB NOT NULL DEFAULT '[]',
```

```sql
    -- Timing
    scheduled_at TIMESTAMPTZ,
    started_at TIMESTAMPTZ,
    completed_at TIMESTAMPTZ,

    -- Error handling
    retry_count INTEGER NOT NULL DEFAULT 0,
    max_retries INTEGER NOT NULL DEFAULT 3,
    error_details JSONB,

    -- Performance
    execution_time_ms INTEGER,
    resource_usage JSONB DEFAULT '{}',

    -- Audit
    created_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,
    created_by UUID REFERENCES core.users(id),

    -- Constraints
    CONSTRAINT workflows_type_check CHECK (
        type IN ('content_generation', 'video_processing', 'publishing', 'analytics_update', 'maintenance')
    ),
    CONSTRAINT workflows_status_check CHECK (
        status IN ('pending', 'queued', 'running', 'completed', 'failed', 'cancelled', 'retrying')
    )
);

-- Workflow indexes
CREATE INDEX idx_workflows_status ON core.workflows(status) WHERE status IN ('pending', 'queued', 'running');
CREATE INDEX idx_workflows_channel ON core.workflows(channel_id, status);
CREATE INDEX idx_workflows_scheduled ON core.workflows(scheduled_at) WHERE status = 'pending';
CREATE INDEX idx_workflows_priority ON core.workflows(priority DESC, created_at) WHERE status IN ('pending', 'queue
CREATE INDEX idx_workflows_type_status ON core.workflows(type, status);
```

Analytics Tables

sql

```sql
-- Time-series analytics table (partitioned)
CREATE TABLE analytics.video_metrics (
    video_id UUID NOT NULL,
    timestamp TIMESTAMPTZ NOT NULL,

    -- YouTube Analytics metrics
    views INTEGER NOT NULL DEFAULT 0,
    likes INTEGER NOT NULL DEFAULT 0,
    dislikes INTEGER NOT NULL DEFAULT 0,
    comments INTEGER NOT NULL DEFAULT 0,
    shares INTEGER NOT NULL DEFAULT 0,
    subscribers_gained INTEGER NOT NULL DEFAULT 0,
    subscribers_lost INTEGER NOT NULL DEFAULT 0,

    -- Watch time metrics
    watch_time_minutes DECIMAL(10,2) NOT NULL DEFAULT 0,
    average_view_duration DECIMAL(10,2) NOT NULL DEFAULT 0,
    average_percentage_viewed DECIMAL(5,2) NOT NULL DEFAULT 0,

    -- Revenue metrics
    estimated_revenue_usd DECIMAL(10,2) NOT NULL DEFAULT 0,
    estimated_ad_revenue_usd DECIMAL(10,2) NOT NULL DEFAULT 0,
    estimated_red_partner_revenue_usd DECIMAL(10,2) NOT NULL DEFAULT 0,

    -- Engagement metrics
    click_through_rate DECIMAL(5,2),
    engagement_rate DECIMAL(5,2),

    -- Traffic sources
    traffic_sources JSONB NOT NULL DEFAULT '{}',

    -- Demographics
    demographics JSONB NOT NULL DEFAULT '{}',

    -- Devices
    device_breakdown JSONB NOT NULL DEFAULT '{}',

    PRIMARY KEY (video_id, timestamp)
) PARTITION BY RANGE (timestamp);

-- Create monthly partitions
CREATE TABLE analytics.video_metrics_2024_01 PARTITION OF analytics.video_metrics
    FOR VALUES FROM ('2024-01-01') TO ('2024-02-01');

CREATE TABLE analytics.video_metrics_2024_02 PARTITION OF analytics.video_metrics
    FOR VALUES FROM ('2024-02-01') TO ('2024-03-01');
```

```sql
-- Automated partition creation function
CREATE OR REPLACE FUNCTION analytics.create_monthly_partition()
RETURNS void AS $$
DECLARE
    start_date date;
    end_date date;
    partition_name text;
BEGIN
    -- Calculate next month's dates
    start_date := date_trunc('month', CURRENT_DATE + interval '1 month');
    end_date := start_date + interval '1 month';
    partition_name := 'video_metrics_' || to_char(start_date, 'YYYY_MM');

    -- Create partition if it doesn't exist
    IF NOT EXISTS (
        SELECT 1 FROM pg_tables
        WHERE schemaname = 'analytics'
        AND tablename = partition_name
    ) THEN
        EXECUTE format(
            'CREATE TABLE analytics.%I PARTITION OF analytics.video_metrics
            FOR VALUES FROM (%L) TO (%L)',
            partition_name, start_date, end_date
        );
    END IF;
END;
$$ LANGUAGE plpgsql;

-- Schedule monthly partition creation
CREATE EXTENSION IF NOT EXISTS pg_cron;
SELECT cron.schedule('create-monthly-partitions', '0 0 25 * *',
    'SELECT analytics.create_monthly_partition()');

-- Aggregated daily metrics
CREATE TABLE analytics.daily_channel_metrics (
    channel_id UUID NOT NULL,
    date DATE NOT NULL,

    -- Video metrics
    videos_published INTEGER NOT NULL DEFAULT 0,
    total_views INTEGER NOT NULL DEFAULT 0,
    total_likes INTEGER NOT NULL DEFAULT 0,
    total_comments INTEGER NOT NULL DEFAULT 0,
    total_watch_time_hours DECIMAL(10,2) NOT NULL DEFAULT 0,

    -- Subscriber metrics
```

```
    subscriber_count INTEGER NOT NULL DEFAULT 0,
    subscribers_gained INTEGER NOT NULL DEFAULT 0,
    subscribers_lost INTEGER NOT NULL DEFAULT 0,

    -- Revenue metrics
    estimated_revenue_usd DECIMAL(10,2) NOT NULL DEFAULT 0,
    revenue_per_mille DECIMAL(10,2), -- RPM

    -- Performance indicators
    average_view_duration DECIMAL(10,2),
    average_ctr DECIMAL(5,2),
    engagement_rate DECIMAL(5,2),

    -- Computed metrics
    metrics_summary JSONB NOT NULL DEFAULT '{}',

    PRIMARY KEY (channel_id, date)
) PARTITION BY RANGE (date);

-- Create indexes on analytics tables
CREATE INDEX idx_video_metrics_timestamp ON analytics.video_metrics(timestamp DESC);
CREATE INDEX idx_video_metrics_video ON analytics.video_metrics(video_id);
CREATE INDEX idx_daily_metrics_channel ON analytics.daily_channel_metrics(channel_id);
CREATE INDEX idx_daily_metrics_date ON analytics.daily_channel_metrics(date DESC);
```

ML Feature Store Tables

sql

```sql
-- ML feature storage for trend prediction
CREATE TABLE ml.trend_features (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    trend_id UUID NOT NULL REFERENCES core.trends(id),
    feature_set VARCHAR(50) NOT NULL,

    -- Feature vectors
    text_embedding VECTOR(768), -- BERT embeddings
    numerical_features REAL[],
    categorical_features TEXT[],

    -- Feature metadata
    feature_names JSONB NOT NULL,
    feature_importance JSONB,

    -- Model predictions
    predictions JSONB NOT NULL DEFAULT '{}',
    confidence_scores JSONB NOT NULL DEFAULT '{}',

    -- Versioning
    model_version VARCHAR(50) NOT NULL,
    computed_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,

    -- Quality tracking
    feature_quality_score DECIMAL(3,2),
    is_valid BOOLEAN NOT NULL DEFAULT true,

    CONSTRAINT trend_features_unique UNIQUE (trend_id, feature_set, model_version)
);

-- Enable pgvector for embeddings
CREATE EXTENSION IF NOT EXISTS vector;

-- Create indexes for ML features
CREATE INDEX idx_ml_features_trend ON ml.trend_features(trend_id);
CREATE INDEX idx_ml_features_embedding ON ml.trend_features USING ivfflat (text_embedding vector_cosine_ops);
CREATE INDEX idx_ml_features_computed ON ml.trend_features(computed_at DESC);

-- Content generation features
CREATE TABLE ml.content_features (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    video_id UUID NOT NULL REFERENCES core.videos(id),

    -- Content analysis
    script_embedding VECTOR(768),
    title_embedding VECTOR(768),
```

```
    thumbnail_features JSONB NOT NULL DEFAULT '{}',

    -- Quality predictions
    predicted_ctr DECIMAL(5,2),
    predicted_retention DECIMAL(5,2),
    predicted_engagement DECIMAL(5,2),
    viral_probability DECIMAL(3,2),

    -- A/B test results
    variant_performance JSONB NOT NULL DEFAULT '{}',

    -- Model metadata
    model_version VARCHAR(50) NOT NULL,
    computed_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP
);
```

Indexing Strategy

Comprehensive Indexing Framework

```sql
```

```sql
-- Function to analyze and suggest indexes
CREATE OR REPLACE FUNCTION core.analyze_index_usage()
RETURNS TABLE (
    schemaname text,
    tablename text,
    indexname text,
    index_size text,
    index_scans bigint,
    tuples_read bigint,
    tuples_fetched bigint,
    recommendation text
) AS $$
BEGIN
    RETURN QUERY
    SELECT
        s.schemaname::text,
        s.tablename::text,
        s.indexname::text,
        pg_size_pretty(pg_relation_size(s.indexrelid))::text as index_size,
        s.idx_scan as index_scans,
        s.idx_tup_read as tuples_read,
        s.idx_tup_fetch as tuples_fetched,
        CASE
            WHEN s.idx_scan = 0 THEN 'UNUSED - Consider dropping'
            WHEN s.idx_scan < 100 THEN 'RARELY USED - Review necessity'
            WHEN s.idx_tup_read / NULLIF(s.idx_scan, 0) > 1000 THEN 'INEFFICIENT - Consider redesign'
            ELSE 'HEALTHY'
        END::text as recommendation
    FROM pg_stat_user_indexes s
    JOIN pg_indexes i ON s.schemaname = i.schemaname
        AND s.tablename = i.tablename
        AND s.indexname = i.indexname
    WHERE s.schemaname NOT IN ('pg_catalog', 'information_schema')
    ORDER BY s.idx_scan;
END;
$$ LANGUAGE plpgsql;

-- Automated index maintenance
CREATE OR REPLACE FUNCTION core.maintain_indexes()
RETURNS void AS $$
DECLARE
    idx RECORD;
BEGIN
    -- Rebuild bloated indexes
    FOR idx IN
        SELECT schemaname, tablename, indexname
```

```
        FROM pg_stat_user_indexes
        WHERE pg_relation_size(indexrelid) > 100 * 1024 * 1024 -- 100MB
        AND idx_scan > 0
    LOOP
        EXECUTE format('REINDEX INDEX CONCURRENTLY %I.%I',
            idx.schemaname, idx.indexname);
    END LOOP;

    -- Update statistics
    ANALYZE;
END;
$$ LANGUAGE plpgsql;

-- Schedule index maintenance
SELECT cron.schedule('index-maintenance', '0 3 * * 0',
    'SELECT core.maintain_indexes()');
```

Index Design Patterns

yaml

```yaml
index_patterns:
  primary_keys:
    type: btree
    characteristics:
      - UUID for distributed systems
      - Natural keys where applicable
      - Covering indexes for common queries

  foreign_keys:
    type: btree
    automatic: true
    naming: fk_{table}_{column}

  search_indexes:
    text_search:
      type: gin
      columns: [title, description, tags]
      operator_class: pg_trgm

    json_search:
      type: gin
      columns: [config, metadata]
      operator_class: jsonb_path_ops

  temporal_indexes:
    type: brin
    columns: [created_at, updated_at, timestamp]
    pages_per_range: 128

  composite_indexes:
    pattern: most_selective_first
    examples:
      - (channel_id, status, created_at DESC)
      - (video_id, timestamp DESC)
      - (trend_id, viral_score DESC)
```

Partitioning Strategy

Partition Management Framework

sql

```
-- Automated partition management for time-series data
CREATE OR REPLACE FUNCTION analytics.auto_partition_management()
RETURNS void AS $$
DECLARE
    table_name text;
    retention_days integer;
BEGIN
    -- Define retention policies
    FOR table_name, retention_days IN VALUES
        ('video_metrics', 730),      -- 2 years
        ('event_log', 90),           -- 3 months
        ('api_requests', 30)         -- 1 month
    LOOP
        -- Create future partitions
        PERFORM analytics.create_future_partitions(table_name, 3); -- 3 months ahead

        -- Drop old partitions
        PERFORM analytics.drop_old_partitions(table_name, retention_days);
    END LOOP;
END;
$$ LANGUAGE plpgsql;

-- Partition pruning optimization
ALTER TABLE analytics.video_metrics SET (autovacuum_enabled = true);
SET enable_partition_pruning = on;
SET constraint_exclusion = partition;
```

Partitioning Best Practices

yaml

```
partitioning_strategy:
  time_series_data:
    partition_by: range(timestamp)
    interval: monthly
    retention: 2_years
    maintenance: automated

  large_tables:
    threshold: 100GB
    partition_by:
      - range (temporal data)
      - list (categorical data)
      - hash (evenly distributed)

  performance_optimization:
    - Partition pruning enabled
    - Parallel query execution
    - Partition-wise joins
    - Constraint exclusion
```

Performance Optimization

Query Optimization Framework

sql

```sql
-- Query performance monitoring
CREATE OR REPLACE VIEW core.slow_queries AS
SELECT
    query,
    calls,
    total_exec_time,
    mean_exec_time,
    stddev_exec_time,
    rows,
    100.0 * shared_blks_hit / nullif(shared_blks_hit + shared_blks_read, 0) AS hit_percent
FROM pg_stat_statements
WHERE mean_exec_time > 100 -- milliseconds
ORDER BY mean_exec_time DESC
LIMIT 50;

-- Automated query optimization suggestions
CREATE OR REPLACE FUNCTION core.suggest_query_optimizations()
RETURNS TABLE (
    query_pattern text,
    optimization_suggestion text,
    estimated_improvement text
) AS $$
BEGIN
    RETURN QUERY
    WITH query_analysis AS (
        SELECT
            query,
            calls,
            mean_exec_time,
            shared_blks_read,
            shared_blks_hit
        FROM pg_stat_statements
        WHERE calls > 100
        AND mean_exec_time > 50
    )
    SELECT
        left(query, 100) || '...' as query_pattern,
        CASE
            WHEN shared_blks_read > shared_blks_hit THEN 'Add covering index'
            WHEN query LIKE '%JOIN%' AND mean_exec_time > 1000 THEN 'Review join strategy'
            WHEN query LIKE '%ORDER BY%' AND mean_exec_time > 500 THEN 'Add sorting index'
            ELSE 'Review execution plan'
        END as optimization_suggestion,
        CASE
            WHEN shared_blks_read > shared_blks_hit THEN '50-80% improvement'
            WHEN mean_exec_time > 1000 THEN '60-90% improvement'
```

```
            ELSE '20-40% improvement'
        END as estimated_improvement
    FROM query_analysis
    ORDER BY mean_exec_time * calls DESC
    LIMIT 20;
END;
$$ LANGUAGE plpgsql;
```

Performance Configuration

```yaml
postgresql_performance_config:
  memory:
    shared_buffers: 32GB          # 25% of RAM
    effective_cache_size: 96GB    # 75% of RAM
    work_mem: 256MB               # Per operation
    maintenance_work_mem: 2GB     # For maintenance

  checkpoint:
    checkpoint_timeout: 30min
    checkpoint_completion_target: 0.9
    max_wal_size: 16GB
    min_wal_size: 2GB

  query_planning:
    random_page_cost: 1.1         # SSD optimized
    effective_io_concurrency: 200 # SSD optimized
    parallel_workers: 16
    max_parallel_workers_per_gather: 8

  connections:
    max_connections: 1000
    connection_pooling:
      pgbouncer:
        pool_mode: transaction
        default_pool_size: 25
        max_client_conn: 10000
```

Data Models

Conceptual Data Model

Entity Relationship Overview

mermaid

```
erDiagram
    CHANNEL ||--o{ VIDEO : creates
    CHANNEL ||--o{ WORKFLOW : executes
    CHANNEL ||--o{ ANALYTICS : tracks

    TREND ||--o{ VIDEO : inspires
    TREND ||--o{ TREND_ANALYSIS : analyzed_by

    VIDEO ||--o{ VIDEO_METRICS : generates
    VIDEO ||--o{ CONTENT_VERSION : has
    VIDEO ||--|| WORKFLOW : created_by

    WORKFLOW ||--o{ WORKFLOW_STEP : contains
    WORKFLOW ||--o{ WORKFLOW_LOG : logs

    USER ||--o{ CHANNEL : owns
    USER ||--o{ AUDIT_LOG : creates

    ML_MODEL ||--o{ PREDICTION : generates
    ML_MODEL ||--o{ FEATURE_SET : uses
```

Business Entity Definitions

yaml

```
entities:
  channel:
    description: YouTube channel managed by YTEMPIRE
    attributes:
      - channel_id (identifier)
      - channel_metadata
      - personality_profile
      - content_strategy
      - monetization_settings

  video:
    description: Individual video content piece
    attributes:
      - video_id (identifier)
      - content_metadata
      - generation_parameters
      - performance_metrics
      - file_references

  trend:
    description: Identified content opportunity
    attributes:
      - trend_id (identifier)
      - topic_data
      - scoring_metrics
      - temporal_data
      - action_recommendations

  workflow:
    description: Automated process execution
    attributes:
      - workflow_id (identifier)
      - workflow_type
      - execution_state
      - resource_usage
      - outcome_data
```

Logical Data Model

Normalized Structure

sql

```
-- Entity Relationship Constraints
ALTER TABLE core.videos
    ADD CONSTRAINT fk_video_channel
    FOREIGN KEY (channel_id)
    REFERENCES core.channels(id)
    ON DELETE CASCADE
    DEFERRABLE INITIALLY DEFERRED;

ALTER TABLE core.workflows
    ADD CONSTRAINT fk_workflow_video
    FOREIGN KEY (video_id)
    REFERENCES core.videos(id)
    ON DELETE SET NULL;

-- Referential Integrity
CREATE OR REPLACE FUNCTION core.validate_video_workflow()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.workflow_id IS NOT NULL THEN
        IF NOT EXISTS (
            SELECT 1 FROM core.workflows
            WHERE id = NEW.workflow_id
            AND video_id = NEW.id
        ) THEN
            RAISE EXCEPTION 'Invalid workflow reference';
        END IF;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER validate_video_workflow_trigger
    BEFORE INSERT OR UPDATE ON core.videos
    FOR EACH ROW
    EXECUTE FUNCTION core.validate_video_workflow();
```

Data Integrity Rules

yaml

```yaml
integrity_rules:
  referential:
    - Videos must belong to active channels
    - Workflows must reference existing entities
    - Analytics must link to valid videos

  business:
    - One active workflow per video at a time
    - Published videos cannot be deleted
    - Channel suspension cascades to videos

  temporal:
    - created_at <= updated_at
    - scheduled_at >= current_timestamp
    - published_at requires status='published'
```

Physical Data Model

Storage Optimization

```sql
-- Table storage parameters
ALTER TABLE core.videos SET (
    fillfactor = 90,           -- Leave space for updates
    autovacuum_vacuum_scale_factor = 0.1,
    autovacuum_analyze_scale_factor = 0.05
);

ALTER TABLE analytics.video_metrics SET (
    fillfactor = 100,          -- Append-only table
    autovacuum_enabled = false    -- Manual vacuum for partitions
);

-- TOAST optimization for large text fields
ALTER TABLE core.videos ALTER COLUMN script SET STORAGE EXTERNAL;
ALTER TABLE core.videos ALTER COLUMN description SET STORAGE EXTENDED;

-- Compression for JSONB columns
ALTER TABLE core.channels ALTER COLUMN config SET STORAGE EXTENDED;
ALTER TABLE core.channels ALTER COLUMN metrics SET COMPRESSION lz4;
```

Physical Layout Strategy

```yaml
```

```yaml
storage_layout:
  tablespaces:
    fast_ssd:
      tables: [videos, channels, active_workflows]
      indexes: all_btree_indexes

    standard_ssd:
      tables: [analytics_tables, ml_features]
      indexes: gin_gist_indexes

    archive_storage:
      tables: [archived_videos, old_analytics]
      compression: enabled

  data_distribution:
    hot_data:      # Accessed frequently
      storage: fast_ssd
      cache_ratio: 0.9

    warm_data:     # Accessed occasionally
      storage: standard_ssd
      cache_ratio: 0.5

    cold_data:     # Rarely accessed
      storage: archive_storage
      cache_ratio: 0.1
```

Data Flow Architecture

Ingestion Patterns

Real-time Data Ingestion

```python
```

```
class DataIngestionPipeline:
    """Real-time data ingestion architecture"""

    def __init__(self):
        self.ingestion_streams = {
            'youtube_analytics': {
                'type': 'pull',
                'frequency': '5_minutes',
                'batch_size': 1000,
                'processing': 'micro_batch'
            },
            'trend_detection': {
                'type': 'push',
                'source': 'kafka',
                'topic': 'trends',
                'processing': 'stream'
            },
            'user_events': {
                'type': 'push',
                'source': 'webhook',
                'rate_limit': 10000,
                'processing': 'async'
            }
        }
```

Batch Processing Pipeline

yaml

```yaml
batch_processing:
  daily_aggregations:
    schedule: "0 2 * * *"
    steps:
      - extract_raw_metrics
      - validate_data_quality
      - transform_to_aggregates
      - load_to_analytics
      - update_materialized_views

  ml_feature_generation:
    schedule: "0 */6 * * *"
    steps:
      - extract_content_data
      - generate_embeddings
      - compute_features
      - store_in_feature_store
      - trigger_model_inference
```

Processing Pipelines

ETL Architecture

```sql
```

```sql
-- Staging tables for ETL
CREATE SCHEMA IF NOT EXISTS staging;

CREATE TABLE staging.youtube_analytics_raw (
    import_id UUID DEFAULT gen_random_uuid(),
    import_timestamp TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
    channel_id VARCHAR(255),
    data JSONB NOT NULL,
    processed BOOLEAN DEFAULT FALSE
);

-- ETL processing function
CREATE OR REPLACE FUNCTION staging.process_youtube_analytics()
RETURNS void AS $$
DECLARE
    record RECORD;
    processed_count INTEGER := 0;
BEGIN
    -- Process unprocessed records
    FOR record IN
        SELECT * FROM staging.youtube_analytics_raw
        WHERE processed = FALSE
        ORDER BY import_timestamp
        LIMIT 1000
    LOOP
        -- Parse and insert into analytics tables
        INSERT INTO analytics.video_metrics (
            video_id,
            timestamp,
            views,
            likes,
            watch_time_minutes
        )
        SELECT
            (record.data->>'videoId')::uuid,
            (record.data->>'date')::timestamptz,
            (record.data->>'views')::integer,
            (record.data->>'likes')::integer,
            (record.data->>'watchTime')::decimal / 60
        ON CONFLICT (video_id, timestamp)
        DO UPDATE SET
            views = EXCLUDED.views,
            likes = EXCLUDED.likes,
            watch_time_minutes = EXCLUDED.watch_time_minutes;

        -- Mark as processed
```

```
        UPDATE staging.youtube_analytics_raw
        SET processed = TRUE
        WHERE import_id = record.import_id;

        processed_count := processed_count + 1;
    END LOOP;

    RAISE NOTICE 'Processed % records', processed_count;
END;
$$ LANGUAGE plpgsql;
```

Stream Processing

```yaml
stream_processing:
  architecture:
    ingestion:
      - Apache Kafka (event streaming)
      - Debezium (CDC from PostgreSQL)

    processing:
      - Apache Flink (complex event processing)
      - ksqlDB (stream queries)

    output:
      - PostgreSQL (persistent storage)
      - Redis (real-time cache)
      - Elasticsearch (search/analytics)

  data_flows:
    trend_detection:
      source: social_media_apis
      transformation: trend_scoring_model
      destination: core.trends
      latency: <1_minute

    real_time_analytics:
      source: youtube_api
      transformation: metric_aggregation
      destination: analytics.video_metrics
      latency: <5_minutes
```

Data Integration

API Integration Layer

python

```python
class DataIntegrationLayer:
    """Unified data integration architecture"""

    def __init__(self):
        self.integrations = {
            'youtube_data_api': {
                'endpoint': 'https://www.googleapis.com/youtube/v3',
                'auth': 'oauth2',
                'rate_limit': 10000,  # daily quota
                'retry_strategy': 'exponential_backoff'
            },
            'youtube_analytics_api': {
                'endpoint': 'https://youtubeanalytics.googleapis.com/v2',
                'auth': 'oauth2',
                'rate_limit': 100,    # per minute
                'data_freshness': '3-5 hours delay'
            },
            'internal_apis': {
                'content_generator': 'http://localhost:8002',
                'media_processor': 'http://localhost:8003',
                'publisher': 'http://localhost:8004'
            }
        }
```

Data Synchronization

sql

```sql
-- Change Data Capture setup
CREATE PUBLICATION ytempire_cdc FOR TABLE
    core.channels,
    core.videos,
    core.trends,
    analytics.video_metrics;

-- Materialized view for real-time dashboards
CREATE MATERIALIZED VIEW analytics.channel_performance AS
SELECT
    c.id as channel_id,
    c.name as channel_name,
    COUNT(DISTINCT v.id) as total_videos,
    COUNT(DISTINCT CASE WHEN v.published_at > NOW() - INTERVAL '30 days'
        THEN v.id END) as videos_last_30d,
    SUM(vm.views) as total_views,
    SUM(vm.likes) as total_likes,
    AVG(vm.engagement_rate) as avg_engagement_rate,
    SUM(vm.estimated_revenue_usd) as total_revenue
FROM core.channels c
LEFT JOIN core.videos v ON c.id = v.channel_id
LEFT JOIN analytics.video_metrics vm ON v.id = vm.video_id
GROUP BY c.id, c.name
WITH DATA;

-- Refresh strategy
CREATE OR REPLACE FUNCTION analytics.refresh_materialized_views()
RETURNS void AS $$
BEGIN
    REFRESH MATERIALIZED VIEW CONCURRENTLY analytics.channel_performance;
    REFRESH MATERIALIZED VIEW CONCURRENTLY analytics.trend_performance;
    REFRESH MATERIALIZED VIEW CONCURRENTLY analytics.daily_summary;
END;
$$ LANGUAGE plpgsql;

-- Schedule refresh
SELECT cron.schedule('refresh-analytics-views', '*/15 * * * *',
    'SELECT analytics.refresh_materialized_views()');
```

Data Governance

Data Quality Framework

Quality Dimensions

sql

```sql
-- Data quality monitoring
CREATE TABLE audit.data_quality_metrics (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    table_schema VARCHAR(255) NOT NULL,
    table_name VARCHAR(255) NOT NULL,
    check_type VARCHAR(50) NOT NULL,
    check_name VARCHAR(255) NOT NULL,

    -- Quality metrics
    total_records BIGINT,
    failed_records BIGINT,
    quality_score DECIMAL(5,2),

    -- Check details
    check_sql TEXT,
    failure_details JSONB,

    -- Execution metadata
    executed_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,
    execution_time_ms INTEGER,

    CONSTRAINT quality_check_type CHECK (
        check_type IN ('completeness', 'accuracy', 'consistency', 'timeliness', 'validity', 'uniqueness')
    )
);

-- Quality check procedures
CREATE OR REPLACE FUNCTION audit.check_data_quality(
    p_schema_name VARCHAR,
    p_table_name VARCHAR
) RETURNS void AS $$
DECLARE
    quality_score DECIMAL;
    check_result RECORD;
BEGIN
    -- Completeness checks
    FOR check_result IN
        SELECT
            column_name,
            COUNT(*) as total_rows,
            COUNT(*) - COUNT(column_name) as null_count
        FROM information_schema.columns
        WHERE table_schema = p_schema_name
        AND table_name = p_table_name
        AND is_nullable = 'NO'
    LOOP
```

```
        INSERT INTO audit.data_quality_metrics (
            table_schema, table_name, check_type, check_name,
            total_records, failed_records, quality_score
        ) VALUES (
            p_schema_name, p_table_name, 'completeness',
            format('NOT NULL check for %s', check_result.column_name),
            check_result.total_rows, check_result.null_count,
            CASE
                WHEN check_result.total_rows = 0 THEN 100
                ELSE (1 - check_result.null_count::decimal / check_result.total_rows) * 100
            END
        );
    END LOOP;

    -- Additional quality checks...
END;
$$ LANGUAGE plpgsql;
```

Quality Rules Engine

```yaml
data_quality_rules:
  completeness:
    - All required fields must be populated
    - No critical data can be NULL
    - Referential integrity maintained

  accuracy:
    - Metrics within expected ranges
    - Timestamps logically consistent
    - Calculated fields match source

  consistency:
    - Cross-table relationships valid
    - Business rules enforced
    - No duplicate primary keys

  timeliness:
    - Data freshness within SLA
    - No stale data in cache
    - Timely synchronization
```

Data Security

Security Implementation

sql

```sql
-- Row Level Security
ALTER TABLE core.channels ENABLE ROW LEVEL SECURITY;

CREATE POLICY channel_isolation ON core.channels
    FOR ALL
    TO ytempire_app
    USING (
        current_setting('app.current_user_id')::uuid = created_by
        OR
        EXISTS (
            SELECT 1 FROM core.user_permissions
            WHERE user_id = current_setting('app.current_user_id')::uuid
            AND resource_type = 'channel'
            AND resource_id = channels.id
            AND permission IN ('read', 'write', 'admin')
        )
    );

-- Column encryption for sensitive data
CREATE EXTENSION IF NOT EXISTS pgcrypto;

-- Encrypted credentials table
CREATE TABLE core.encrypted_credentials (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    service_name VARCHAR(255) NOT NULL,
    encrypted_data BYTEA NOT NULL,
    key_version INTEGER NOT NULL DEFAULT 1,
    created_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,
    expires_at TIMESTAMPTZ,

    CONSTRAINT unique_service_version UNIQUE (service_name, key_version)
);

-- Encryption functions
CREATE OR REPLACE FUNCTION core.encrypt_sensitive_data(
    p_data TEXT,
    p_key TEXT
) RETURNS BYTEA AS $$
BEGIN
    RETURN pgp_sym_encrypt(p_data, p_key);
END;
$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE OR REPLACE FUNCTION core.decrypt_sensitive_data(
    p_encrypted BYTEA,
    p_key TEXT
```

```
) RETURNS TEXT AS $$
BEGIN
    RETURN pgp_sym_decrypt(p_encrypted, p_key);
END;
$$ LANGUAGE plpgsql SECURITY DEFINER;
```

Access Control Matrix

```yaml
```

```yaml
access_control:
  roles:
    admin:
      permissions: [all]
      data_access: unrestricted

    operator:
      permissions: [read, write, execute]
      data_access: channel_scoped

    analyst:
      permissions: [read]
      data_access: anonymized

    service_account:
      permissions: [read, write]
      data_access: api_scoped

  data_classification:
    public:
      - channel_names
      - video_titles
      - published_metrics

    internal:
      - performance_metrics
      - quality_scores
      - workflow_status

    confidential:
      - revenue_data
      - api_credentials
      - user_information

    restricted:
      - encryption_keys
      - audit_logs
      - security_events
```

Compliance and Privacy

GDPR Implementation

```sql
```

```sql
-- Privacy compliance tables
CREATE TABLE audit.privacy_requests (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    request_type VARCHAR(50) NOT NULL,
    subject_identifier VARCHAR(255) NOT NULL,

    -- Request details
    requested_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,
    requested_by VARCHAR(255),

    -- Processing
    status VARCHAR(50) NOT NULL DEFAULT 'pending',
    processed_at TIMESTAMPTZ,
    processed_by UUID REFERENCES core.users(id),

    -- Results
    affected_records JSONB,
    completion_report TEXT,

    CONSTRAINT request_type_check CHECK (
        request_type IN ('access', 'rectification', 'erasure', 'portability', 'restriction')
    ),
    CONSTRAINT status_check CHECK (
        status IN ('pending', 'in_progress', 'completed', 'rejected')
    )
);

-- Data anonymization functions
CREATE OR REPLACE FUNCTION audit.anonymize_personal_data(
    p_table_name VARCHAR,
    p_identifier_column VARCHAR,
    p_identifier_value VARCHAR
) RETURNS void AS $$
BEGIN
    -- Implementation for each table type
    CASE p_table_name
        WHEN 'users' THEN
            UPDATE core.users SET
                email = 'anonymized_' || id || '@example.com',
                name = 'Anonymized User',
                phone = NULL,
                address = NULL
            WHERE p_identifier_column = p_identifier_value;

        WHEN 'channels' THEN
            UPDATE core.channels SET
```

```
            name = 'Anonymized Channel ' || left(id::text, 8),
            description = 'Content has been anonymized'
        WHERE p_identifier_column = p_identifier_value;
    END CASE;

    -- Log anonymization
    INSERT INTO audit.privacy_requests (
        request_type, subject_identifier, status, processed_at
    ) VALUES (
        'erasure', p_identifier_value, 'completed', CURRENT_TIMESTAMP
    );
END;
$$ LANGUAGE plpgsql;
```

Compliance Monitoring

```yaml
compliance_monitoring:
  data_retention:
    operational_data: 2_years
    analytics_data: 5_years
    audit_logs: 7_years
    backups: 1_year

  automated_checks:
    - Data retention policy enforcement
    - Access pattern anomaly detection
    - Cross-border data transfer logging
    - Consent management validation

  reporting:
    - Monthly compliance dashboard
    - Quarterly audit reports
    - Annual compliance certification
    - Incident response documentation
```

Scalability and Performance

Scaling Strategies

Horizontal Scaling Architecture

```yaml
```

```yaml
horizontal_scaling:
  read_replicas:
    count: 3
    distribution: geographic
    lag_threshold: 1_second
    failover: automatic

  sharding_strategy:
    shard_key: channel_id
    shard_count: 16
    distribution: consistent_hash
    rebalancing: automatic

  connection_pooling:
    pgbouncer:
      pool_mode: transaction
      default_pool_size: 25
      reserve_pool_size: 5
      max_client_conn: 10000

  load_balancing:
    strategy: least_connections
    health_check: 5_seconds
    failover_time: <10_seconds
```

Vertical Scaling Optimization

```sql
```

```sql
-- Resource usage monitoring
CREATE OR REPLACE VIEW core.resource_usage AS
SELECT
    current_timestamp as snapshot_time,
    numbackends as active_connections,
    xact_commit as transactions_committed,
    xact_rollback as transactions_rolled_back,
    blks_read as blocks_read,
    blks_hit as blocks_hit,
    tup_returned as tuples_returned,
    tup_fetched as tuples_fetched,
    tup_inserted as tuples_inserted,
    tup_updated as tuples_updated,
    tup_deleted as tuples_deleted
FROM pg_stat_database
WHERE datname = current_database();

-- Auto-scaling triggers
CREATE OR REPLACE FUNCTION core.check_scaling_needs()
RETURNS void AS $$
DECLARE
    cpu_usage DECIMAL;
    memory_usage DECIMAL;
    connection_usage DECIMAL;
BEGIN
    -- Get current metrics
    SELECT
        (SELECT COUNT(*) FROM pg_stat_activity)::decimal /
        current_setting('max_connections')::decimal * 100
    INTO connection_usage;

    -- Alert if scaling needed
    IF connection_usage > 80 THEN
        INSERT INTO audit.system_alerts (
            alert_type, severity, message
        ) VALUES (
            'scaling_needed', 'high',
            format('Connection usage at %.1f%% - consider scaling', connection_usage)
        );
    END IF;
END;
$$ LANGUAGE plpgsql;
```

Caching Architecture

Multi-Layer Cache Strategy

python

```python
class CacheArchitecture:
    """Multi-layer caching implementation"""

    def __init__(self):
        self.cache_layers = {
            'l1_application': {
                'technology': 'in_memory_lru',
                'size': '4GB',
                'ttl': 300,  # 5 minutes
                'hit_ratio_target': 0.9
            },
            'l2_redis': {
                'technology': 'redis_cluster',
                'size': '32GB',
                'ttl': 3600,  # 1 hour
                'persistence': 'rdb_snapshot',
                'hit_ratio_target': 0.8
            },
            'l3_postgresql': {
                'technology': 'materialized_views',
                'refresh': '15_minutes',
                'indexes': 'covering',
                'hit_ratio_target': 0.95
            },
            'l4_cdn': {
                'technology': 'cloudflare',
                'ttl': 86400,  # 24 hours
                'invalidation': 'tag_based',
                'hit_ratio_target': 0.7
            }
        }
```

Cache Warming Strategy

sql

```sql
-- Proactive cache warming
CREATE OR REPLACE FUNCTION core.warm_cache()
RETURNS void AS $$
BEGIN
    -- Warm frequently accessed data
    PERFORM COUNT(*) FROM core.channels WHERE status = 'active';
    PERFORM COUNT(*) FROM core.videos WHERE published_at > NOW() - INTERVAL '7 days';

    -- Pre-compute expensive aggregations
    PERFORM * FROM analytics.channel_performance;
    PERFORM * FROM analytics.daily_summary WHERE date >= CURRENT_DATE - 7;

    -- Cache hot queries
    EXECUTE 'SELECT * FROM core.get_trending_videos(10)';
    EXECUTE 'SELECT * FROM analytics.get_channel_metrics($1, $2)'
        USING CURRENT_DATE - 30, CURRENT_DATE;
END;
$$ LANGUAGE plpgsql;

-- Schedule cache warming
SELECT cron.schedule('warm-cache', '0 */4 * * *', 'SELECT core.warm_cache()');
```

Query Optimization

Query Performance Patterns

sql

```sql
-- Optimized query patterns
-- Pattern 1: Covering index usage
CREATE INDEX idx_videos_channel_performance
ON core.videos(channel_id, status, published_at DESC)
INCLUDE (title, views, likes);

-- Pattern 2: Partial indexes for common filters
CREATE INDEX idx_active_videos
ON core.videos(published_at DESC)
WHERE status = 'published' AND visibility = 'public';

-- Pattern 3: JSONB indexing
CREATE INDEX idx_video_metrics_gin
ON core.videos USING gin (performance_metrics jsonb_path_ops);

-- Pattern 4: Optimized aggregation
CREATE OR REPLACE FUNCTION analytics.get_channel_summary(
    p_channel_id UUID,
    p_days INTEGER DEFAULT 30
) RETURNS TABLE (
    total_videos BIGINT,
    total_views BIGINT,
    avg_engagement DECIMAL,
    revenue_estimate DECIMAL
) AS $$
BEGIN
    RETURN QUERY
    WITH video_stats AS (
        SELECT
            v.id,
            vm.views,
            vm.engagement_rate,
            vm.estimated_revenue_usd
        FROM core.videos v
        JOIN analytics.video_metrics vm ON v.id = vm.video_id
        WHERE v.channel_id = p_channel_id
        AND v.published_at > CURRENT_DATE - p_days * INTERVAL '1 day'
        AND v.status = 'published'
    )
    SELECT
        COUNT(DISTINCT id)::BIGINT as total_videos,
        COALESCE(SUM(views), 0)::BIGINT as total_views,
        COALESCE(AVG(engagement_rate), 0)::DECIMAL as avg_engagement,
        COALESCE(SUM(estimated_revenue_usd), 0)::DECIMAL as revenue_estimate
    FROM video_stats;
```

```
END;
$$ LANGUAGE plpgsql STABLE;
```

Query Plan Analysis

```
sql
```

```sql
-- Automated query plan analysis
CREATE OR REPLACE FUNCTION core.analyze_query_performance(
    p_query TEXT
) RETURNS TABLE (
    node_type TEXT,
    total_cost NUMERIC,
    execution_time NUMERIC,
    optimization_suggestion TEXT
) AS $$
DECLARE
    plan_json JSONB;
BEGIN
    -- Get execution plan
    EXECUTE format('EXPLAIN (ANALYZE, BUFFERS, FORMAT JSON) %s', p_query)
    INTO plan_json;

    -- Analyze plan nodes
    RETURN QUERY
    WITH RECURSIVE plan_nodes AS (
        SELECT
            plan_json->'Plan' as node,
            0 as level
        UNION ALL
        SELECT
            unnest(CASE
                WHEN jsonb_typeof(node->'Plans') = 'array'
                THEN jsonb_array_elements(node->'Plans')
                ELSE '{}'::jsonb[]
            END),
            level + 1
        FROM plan_nodes
        WHERE node IS NOT NULL
    )
    SELECT
        node->>'Node Type' as node_type,
        (node->>'Total Cost')::numeric as total_cost,
        (node->>'Actual Total Time')::numeric as execution_time,
        CASE
            WHEN node->>'Node Type' = 'Seq Scan'
                AND (node->>'Actual Total Time')::numeric > 100
            THEN 'Consider adding index'
            WHEN node->>'Node Type' = 'Nested Loop'
                AND (node->>'Actual Rows')::numeric > 1000
            THEN 'Consider hash join'
            ELSE 'OK'
        END as optimization_suggestion
```

```
    FROM plan_nodes
    WHERE node->>'Node Type' IS NOT NULL;
END;
$$ LANGUAGE plpgsql;
```

Disaster Recovery and High Availability

Backup Strategies

Comprehensive Backup Architecture

```yaml
backup_strategy:
  continuous_archiving:
    method: pg_basebackup + WAL
    wal_archiving:
      archive_mode: on
      archive_command: 'pgbackrest archive-push %p'
      archive_timeout: 300  # 5 minutes

  backup_types:
    full_backup:
      frequency: weekly
      retention: 4_weeks
      storage:
        - local_fast_storage (1 week)
        - remote_s3 (4 weeks)
        - glacier (1 year)

    incremental_backup:
      frequency: daily
      retention: 7_days
      method: pgbackrest_delta

    continuous_backup:
      method: wal_streaming
      lag_threshold: 1_minute
      compression: lz4

  point_in_time_recovery:
    granularity: 1_second
    retention: 7_days
    testing_frequency: monthly
```

Backup Implementation

```sql
```

```
-- Backup configuration with pgBackRest
CREATE OR REPLACE FUNCTION core.configure_backup()
RETURNS void AS $
BEGIN
    -- Set archive configuration
    ALTER SYSTEM SET archive_mode = 'on';
    ALTER SYSTEM SET archive_command = 'pgbackrest --stanza=ytempire archive-push %p';
    ALTER SYSTEM SET archive_timeout = '5min';
    ALTER SYSTEM SET wal_level = 'replica';
    ALTER SYSTEM SET max_wal_senders = 10;
    ALTER SYSTEM SET wal_keep_size = '1GB';

    -- Create backup status tracking
    CREATE TABLE IF NOT EXISTS audit.backup_history (
        id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
        backup_type VARCHAR(50) NOT NULL,
        backup_label VARCHAR(255) NOT NULL,
        started_at TIMESTAMPTZ NOT NULL,
        completed_at TIMESTAMPTZ,
        size_bytes BIGINT,
        wal_start_lsn PG_LSN,
        wal_stop_lsn PG_LSN,
        status VARCHAR(50) NOT NULL DEFAULT 'running',
        error_message TEXT,
        metadata JSONB DEFAULT '{}'
    );

    -- Backup monitoring function
    CREATE OR REPLACE FUNCTION audit.monitor_backup_health()
    RETURNS TABLE (
        check_name TEXT,
        status TEXT,
        details TEXT
    ) AS $func$
    BEGIN
        -- Check last successful backup
        RETURN QUERY
        SELECT
            'last_full_backup'::TEXT,
            CASE
                WHEN MAX(completed_at) > NOW() - INTERVAL '8 days' THEN 'OK'
                ELSE 'WARNING'
            END,
            'Last backup: ' || COALESCE(MAX(completed_at)::TEXT, 'Never')
        FROM audit.backup_history
        WHERE backup_type = 'full'
```

```
        AND status = 'completed';

        -- Check WAL archiving
        RETURN QUERY
        SELECT
            'wal_archiving'::TEXT,
            CASE
                WHEN last_archived_time > NOW() - INTERVAL '10 minutes' THEN 'OK'
                ELSE 'CRITICAL'
            END,
            'Last WAL archive: ' || last_archived_time::TEXT
        FROM pg_stat_archiver;

        -- Check backup size growth
        RETURN QUERY
        WITH backup_growth AS (
            SELECT
                date_trunc('day', completed_at) as backup_date,
                SUM(size_bytes) as daily_size
            FROM audit.backup_history
            WHERE status = 'completed'
            AND completed_at > NOW() - INTERVAL '30 days'
            GROUP BY 1
        )
        SELECT
            'backup_size_trend'::TEXT,
            CASE
                WHEN AVG(daily_size) < 100 * 1024^3 THEN 'OK'  -- 100GB
                ELSE 'WARNING'
            END,
            'Average daily backup size: ' ||
            pg_size_pretty(AVG(daily_size)::BIGINT)
        FROM backup_growth;
    END;
    $func$ LANGUAGE plpgsql;
END;
$ LANGUAGE plpgsql;

-- Automated backup verification
CREATE OR REPLACE FUNCTION audit.verify_backup_integrity(
    p_backup_label VARCHAR
) RETURNS BOOLEAN AS $
DECLARE
    verification_result BOOLEAN;
BEGIN
    -- Perform backup verification
    EXECUTE format(
```

```
        'pgbackrest --stanza=ytempire --type=full verify %s',
        p_backup_label
    );

    -- Log verification
    INSERT INTO audit.backup_history (
        backup_type, backup_label, started_at, completed_at, status
    ) VALUES (
        'verification', p_backup_label, NOW(), NOW(), 'verified'
    );

    RETURN TRUE;
EXCEPTION
    WHEN OTHERS THEN
        INSERT INTO audit.backup_history (
            backup_type, backup_label, started_at, completed_at,
            status, error_message
        ) VALUES (
            'verification', p_backup_label, NOW(), NOW(),
            'failed', SQLERRM
        );
        RETURN FALSE;
END;
$ LANGUAGE plpgsql;
```

Backup Testing Procedures

```
yaml
```

```yaml
backup_testing:
  test_scenarios:
    full_restore:
      frequency: monthly
      procedure:
        - Restore to test server
        - Verify data integrity
        - Test application connectivity
        - Measure recovery time

    point_in_time:
      frequency: weekly
      test_points:
        - Random timestamp (last 7 days)
        - Before major operation
        - After batch processing

    partial_restore:
      frequency: bi_weekly
      scenarios:
        - Single table restore
        - Schema restore
        - Specific time range

  validation_checks:
    - Row count verification
    - Checksum validation
    - Foreign key integrity
    - Application functionality
    - Performance baseline
```

Replication Architecture

Multi-Region Replication

sql

```sql
-- Replication setup
CREATE OR REPLACE FUNCTION core.setup_replication()
RETURNS void AS $
BEGIN
    -- Primary configuration
    ALTER SYSTEM SET wal_level = 'logical';
    ALTER SYSTEM SET max_replication_slots = 10;
    ALTER SYSTEM SET max_logical_replication_workers = 10;
    ALTER SYSTEM SET logical_decoding_work_mem = '256MB';

    -- Create publication for replication
    CREATE PUBLICATION ytempire_main FOR ALL TABLES;

    -- Create replication slots
    SELECT pg_create_logical_replication_slot(
        'ytempire_replica_1',
        'pgoutput'
    );

    -- Replication monitoring
    CREATE OR REPLACE VIEW core.replication_status AS
    SELECT
        slot_name,
        active,
        restart_lsn,
        confirmed_flush_lsn,
        pg_size_pretty(
            pg_wal_lsn_diff(pg_current_wal_lsn(), confirmed_flush_lsn)
        ) as lag_size,
        CASE
            WHEN active THEN 'Active'
            WHEN pg_wal_lsn_diff(pg_current_wal_lsn(), confirmed_flush_lsn) > 1024^3
            THEN 'Lagging'
            ELSE 'Inactive'
        END as status
    FROM pg_replication_slots
    WHERE slot_type = 'logical';
END;
$ LANGUAGE plpgsql;

-- Replication lag monitoring
CREATE OR REPLACE FUNCTION core.check_replication_lag()
RETURNS TABLE (
    replica_name TEXT,
    lag_bytes BIGINT,
    lag_time INTERVAL,
```

```
    status TEXT
) AS $
BEGIN
    RETURN QUERY
    SELECT
        client_addr::TEXT as replica_name,
        pg_wal_lsn_diff(pg_current_wal_lsn(), replay_lsn) as lag_bytes,
        NOW() - pg_last_xact_replay_timestamp() as lag_time,
        CASE
            WHEN pg_wal_lsn_diff(pg_current_wal_lsn(), replay_lsn) < 1024^2 THEN 'Healthy'
            WHEN pg_wal_lsn_diff(pg_current_wal_lsn(), replay_lsn) < 1024^3 THEN 'Warning'
            ELSE 'Critical'
        END as status
    FROM pg_stat_replication;
END;
$ LANGUAGE plpgsql;
```

High Availability Configuration

yaml

```yaml
high_availability:
  architecture:
    primary:
      location: us_east_1a
      role: read_write

    synchronous_standby:
      location: us_east_1b
      role: read_only
      promotion_priority: 1
      max_lag: 1_second

    async_replicas:
      - location: us_west_2
        role: read_only
        promotion_priority: 2
        max_lag: 5_minutes

      - location: eu_west_1
        role: read_only
        promotion_priority: 3
        max_lag: 15_minutes

  failover_configuration:
    method: automatic
    detection_time: 30_seconds
    promotion_time: 60_seconds
    vip_management: keepalived

  connection_routing:
    write_connections: primary_only
    read_connections: least_loaded_replica
    failover_retry: exponential_backoff
```

Failover Procedures

Automated Failover System

```python
```

```
class FailoverOrchestrator:
    """Automated database failover orchestration"""

    def __init__(self):
        self.failover_config = {
            'health_check_interval': 5,  # seconds
            'failure_threshold': 3,      # consecutive failures
            'promotion_timeout': 60,     # seconds
            'notification_channels': ['slack', 'pagerduty', 'email']
        }

        self.failover_steps = [
            'detect_primary_failure',
            'verify_failure',
            'select_new_primary',
            'fence_old_primary',
            'promote_replica',
            'redirect_traffic',
            'verify_new_primary',
            'update_dns',
            'notify_completion'
        ]

    async def execute_failover(self):
        """Execute automated failover procedure"""

        failover_log = {
            'id': str(uuid.uuid4()),
            'started_at': datetime.utcnow(),
            'steps': []
        }

        try:
            # Step 1: Detect and verify failure
            if not await self.verify_primary_failure():
                return False

            # Step 2: Select best replica for promotion
            new_primary = await self.select_promotion_candidate()

            # Step 3: Fence old primary (STONITH)
            await self.fence_old_primary()

            # Step 4: Promote replica
            await self.promote_replica(new_primary)
```

```
        # Step 5: Update connection routing
        await self.update_connection_routing(new_primary)

        # Step 6: Verify new primary
        if not await self.verify_new_primary(new_primary):
            raise FailoverError("New primary verification failed")

        # Step 7: Complete failover
        failover_log['completed_at'] = datetime.utcnow()
        failover_log['status'] = 'success'
        failover_log['new_primary'] = new_primary

        await self.notify_failover_complete(failover_log)

    except Exception as e:
        failover_log['status'] = 'failed'
        failover_log['error'] = str(e)
        await self.initiate_manual_intervention(failover_log)
        raise

    finally:
        await self.log_failover_event(failover_log)
```

Manual Failover Procedures

sql

```
-- Manual failover execution
CREATE OR REPLACE FUNCTION core.execute_manual_failover(
    p_new_primary VARCHAR
) RETURNS void AS $
DECLARE
    v_current_lsn pg_lsn;
    v_replica_lsn pg_lsn;
BEGIN
    -- Pre-flight checks
    IF NOT EXISTS (
        SELECT 1 FROM pg_stat_replication
        WHERE client_addr::TEXT = p_new_primary
    ) THEN
        RAISE EXCEPTION 'Replica % not found', p_new_primary;
    END IF;

    -- Get current positions
    SELECT pg_current_wal_lsn() INTO v_current_lsn;
    SELECT replay_lsn INTO v_replica_lsn
    FROM pg_stat_replication
    WHERE client_addr::TEXT = p_new_primary;

    -- Check lag
    IF pg_wal_lsn_diff(v_current_lsn, v_replica_lsn) > 1024^3 THEN
        RAISE WARNING 'Replica lag is %, proceed with caution',
            pg_size_pretty(pg_wal_lsn_diff(v_current_lsn, v_replica_lsn));
    END IF;

    -- Create failover checkpoint
    CHECKPOINT;

    -- Log failover initiation
    INSERT INTO audit.failover_events (
        event_type, old_primary, new_primary, initiated_by, initiated_at
    ) VALUES (
        'manual', inet_server_addr()::TEXT, p_new_primary,
        current_user, CURRENT_TIMESTAMP
    );

    -- Signal for external orchestration
    NOTIFY failover_initiated, p_new_primary;
END;
$ LANGUAGE plpgsql;

-- Post-failover validation
CREATE OR REPLACE FUNCTION core.validate_failover()
```

```
RETURNS TABLE (
    check_name TEXT,
    result BOOLEAN,
    details TEXT
) AS $
BEGIN
    -- Check if we're the primary
    RETURN QUERY
    SELECT
        'is_primary'::TEXT,
        NOT pg_is_in_recovery(),
        CASE
            WHEN pg_is_in_recovery() THEN 'Still in recovery mode'
            ELSE 'Primary mode confirmed'
        END;

    -- Check replication slots
    RETURN QUERY
    SELECT
        'replication_slots'::TEXT,
        COUNT(*) > 0,
        format('%s active slots', COUNT(*))
    FROM pg_replication_slots
    WHERE active;

    -- Check connections
    RETURN QUERY
    SELECT
        'client_connections'::TEXT,
        COUNT(*) > 0,
        format('%s active connections', COUNT(*))
    FROM pg_stat_activity
    WHERE state = 'active'
    AND pid != pg_backend_pid();

    -- Check write capability
    RETURN QUERY
    SELECT
        'write_test'::TEXT,
        core.test_write_capability(),
        'Write test ' || CASE
            WHEN core.test_write_capability() THEN 'passed'
            ELSE 'failed'
        END;
END;
$ LANGUAGE plpgsql;
```

Analytics and Reporting

OLAP Architecture

Real-time OLAP Implementation

```sql
```

```sql
-- OLAP cube definitions
CREATE SCHEMA IF NOT EXISTS olap;

-- Video performance cube
CREATE MATERIALIZED VIEW olap.video_performance_cube AS
WITH date_dimension AS (
    SELECT generate_series(
        DATE '2024-01-01',
        CURRENT_DATE,
        INTERVAL '1 day'
    )::DATE as date
),
channel_dimension AS (
    SELECT
        id as channel_id,
        name as channel_name,
        niche
    FROM core.channels
),
video_facts AS (
    SELECT
        v.id as video_id,
        v.channel_id,
        DATE(v.published_at) as publish_date,
        vm.views,
        vm.likes,
        vm.comments,
        vm.watch_time_minutes,
        vm.estimated_revenue_usd,
        vm.engagement_rate
    FROM core.videos v
    JOIN analytics.video_metrics vm ON v.id = vm.video_id
    WHERE v.status = 'published'
)
SELECT
    d.date,
    c.channel_id,
    c.channel_name,
    c.niche,
    COUNT(DISTINCT vf.video_id) as video_count,
    SUM(vf.views) as total_views,
    SUM(vf.likes) as total_likes,
    SUM(vf.comments) as total_comments,
    SUM(vf.watch_time_minutes) as total_watch_time,
    SUM(vf.estimated_revenue_usd) as total_revenue,
    AVG(vf.engagement_rate) as avg_engagement_rate,
```

```
    -- Calculated metrics
    SUM(vf.views) / NULLIF(COUNT(DISTINCT vf.video_id), 0) as avg_views_per_video,
    SUM(vf.estimated_revenue_usd) / NULLIF(SUM(vf.views), 0) * 1000 as rpm,

    -- Period comparisons
    LAG(SUM(vf.views), 7) OVER (
        PARTITION BY c.channel_id ORDER BY d.date
    ) as views_7d_ago,
    LAG(SUM(vf.views), 30) OVER (
        PARTITION BY c.channel_id ORDER BY d.date
    ) as views_30d_ago

FROM date_dimension d
CROSS JOIN channel_dimension c
LEFT JOIN video_facts vf ON d.date = vf.publish_date AND c.channel_id = vf.channel_id
GROUP BY d.date, c.channel_id, c.channel_name, c.niche
WITH DATA;

-- Create indexes for OLAP queries
CREATE INDEX idx_olap_cube_date ON olap.video_performance_cube(date DESC);
CREATE INDEX idx_olap_cube_channel ON olap.video_performance_cube(channel_id);
CREATE INDEX idx_olap_cube_niche ON olap.video_performance_cube(niche);

-- Trend analysis cube
CREATE MATERIALIZED VIEW olap.trend_performance_cube AS
WITH trend_facts AS (
    SELECT
        t.id as trend_id,
        t.topic_normalized,
        t.category,
        t.source,
        t.viral_score,
        t.competition_score,
        t.opportunity_score,
        DATE(t.first_detected) as detection_date,
        COUNT(DISTINCT v.id) as videos_created,
        SUM(vm.views) as total_views,
        SUM(vm.estimated_revenue_usd) as total_revenue
    FROM core.trends t
    LEFT JOIN core.videos v ON v.generation_params->>'trend_id' = t.id::TEXT
    LEFT JOIN analytics.video_metrics vm ON v.id = vm.video_id
    GROUP BY t.id
)
SELECT
    tf.*,
    -- Success metrics
```

```sql
    CASE
        WHEN tf.videos_created > 0 THEN tf.total_views / tf.videos_created
        ELSE 0
    END as avg_views_per_video,
    CASE
        WHEN tf.videos_created > 0 THEN tf.total_revenue / tf.videos_created
        ELSE 0
    END as avg_revenue_per_video,
    -- ROI calculation
    CASE
        WHEN tf.viral_score > 0 THEN
            (tf.total_revenue / tf.viral_score) / NULLIF(tf.videos_created, 0)
        ELSE 0
    END as roi_score
FROM trend_facts tf
WITH DATA;
```

OLAP Query Patterns

```sql
```

```sql
-- High-performance OLAP queries
-- Query 1: Channel performance over time
CREATE OR REPLACE FUNCTION olap.get_channel_performance(
    p_channel_id UUID,
    p_start_date DATE,
    p_end_date DATE,
    p_granularity VARCHAR DEFAULT 'day'
) RETURNS TABLE (
    period_start DATE,
    metrics JSONB
) AS $
BEGIN
    RETURN QUERY
    SELECT
        CASE p_granularity
            WHEN 'day' THEN date
            WHEN 'week' THEN date_trunc('week', date)::DATE
            WHEN 'month' THEN date_trunc('month', date)::DATE
        END as period_start,
        jsonb_build_object(
            'videos_published', SUM(video_count),
            'total_views', SUM(total_views),
            'total_revenue', SUM(total_revenue),
            'avg_engagement', AVG(avg_engagement_rate),
            'rpm', SUM(total_revenue) / NULLIF(SUM(total_views), 0) * 1000,
            'growth_rate',
                (SUM(total_views) - SUM(views_30d_ago)) /
                NULLIF(SUM(views_30d_ago), 0) * 100
        ) as metrics
    FROM olap.video_performance_cube
    WHERE channel_id = p_channel_id
    AND date BETWEEN p_start_date AND p_end_date
    GROUP BY 1
    ORDER BY 1;
END;
$ LANGUAGE plpgsql STABLE;

-- Query 2: Trend success analysis
CREATE OR REPLACE FUNCTION olap.analyze_trend_success(
    p_min_viral_score DECIMAL DEFAULT 0.7,
    p_limit INTEGER DEFAULT 100
) RETURNS TABLE (
    trend_topic VARCHAR,
    category VARCHAR,
    viral_score DECIMAL,
    videos_created BIGINT,
```

```
    avg_performance JSONB,
    success_rate DECIMAL
) AS $
BEGIN
    RETURN QUERY
    SELECT
        topic_normalized,
        t.category,
        t.viral_score,
        t.videos_created,
        jsonb_build_object(
            'avg_views', t.avg_views_per_video,
            'avg_revenue', t.avg_revenue_per_video,
            'roi_score', t.roi_score
        ) as avg_performance,
        CASE
            WHEN t.videos_created > 0 AND t.avg_views_per_video > 10000
            THEN 1.0
            WHEN t.videos_created > 0 AND t.avg_views_per_video > 5000
            THEN 0.5
            ELSE 0.0
        END as success_rate
    FROM olap.trend_performance_cube t
    WHERE t.viral_score >= p_min_viral_score
    ORDER BY t.roi_score DESC NULLS LAST
    LIMIT p_limit;
END;
$ LANGUAGE plpgsql STABLE;
```

Real-time Analytics

Streaming Analytics Architecture

python

```python
class RealTimeAnalytics:
    """Real-time analytics processing system"""

    def __init__(self):
        self.stream_processors = {
            'view_counter': {
                'source': 'youtube_api_webhook',
                'window': '1_minute',
                'aggregation': 'sum',
                'output': 'redis_counter'
            },
            'engagement_tracker': {
                'source': 'youtube_analytics_stream',
                'window': '5_minutes',
                'aggregation': 'weighted_average',
                'output': 'postgres_update'
            },
            'anomaly_detector': {
                'source': 'all_metrics',
                'algorithm': 'isolation_forest',
                'threshold': 0.95,
                'output': 'alert_system'
            }
        }

    async def process_metric_stream(self, metric_type: str, data: dict):
        """Process incoming metric data in real-time"""

        # Update real-time counters
        await self.update_redis_counter(metric_type, data)

        # Check for anomalies
        if await self.detect_anomaly(metric_type, data):
            await self.trigger_alert(metric_type, data)

        # Update materialized views if threshold reached
        if self.should_update_materialized_view(metric_type):
            await self.refresh_materialized_view(metric_type)
```

Real-time Dashboard Queries

```sql
```

```sql
-- Real-time metrics view
CREATE OR REPLACE VIEW analytics.realtime_metrics AS
WITH current_hour_metrics AS (
    SELECT
        video_id,
        SUM(views) as hour_views,
        SUM(likes) as hour_likes,
        AVG(engagement_rate) as hour_engagement
    FROM analytics.video_metrics
    WHERE timestamp > NOW() - INTERVAL '1 hour'
    GROUP BY video_id
),
trending_videos AS (
    SELECT
        v.id,
        v.title,
        v.channel_id,
        chm.hour_views,
        chm.hour_views / EXTRACT(epoch FROM (NOW() - v.published_at)) * 3600
            as views_per_hour_lifetime,
        ROW_NUMBER() OVER (ORDER BY chm.hour_views DESC) as trend_rank
    FROM core.videos v
    JOIN current_hour_metrics chm ON v.id = chm.video_id
    WHERE v.published_at > NOW() - INTERVAL '7 days'
)
SELECT
    tv.*,
    c.name as channel_name,
    CASE
        WHEN tv.trend_rank <= 3 THEN 'viral'
        WHEN tv.trend_rank <= 10 THEN 'trending'
        WHEN tv.views_per_hour_lifetime > 1000 THEN 'performing'
        ELSE 'normal'
    END as performance_tier
FROM trending_videos tv
JOIN core.channels c ON tv.channel_id = c.id;

-- Real-time alerting function
CREATE OR REPLACE FUNCTION analytics.check_realtime_alerts()
RETURNS void AS $
DECLARE
    alert RECORD;
BEGIN
    -- Check for viral videos
    FOR alert IN
        SELECT
```

```
            v.id,
            v.title,
            rm.hour_views,
            rm.performance_tier
        FROM analytics.realtime_metrics rm
        JOIN core.videos v ON rm.id = v.id
        WHERE rm.performance_tier = 'viral'
        AND NOT EXISTS (
            SELECT 1 FROM audit.alerts
            WHERE entity_id = v.id
            AND alert_type = 'viral_video'
            AND created_at > NOW() - INTERVAL '1 hour'
        )
    LOOP
        INSERT INTO audit.alerts (
            alert_type, severity, entity_id, message, metadata
        ) VALUES (
            'viral_video', 'info', alert.id,
            format('Video "%s" is going viral with %s views in the last hour',
                alert.title, alert.hour_views),
            jsonb_build_object(
                'hour_views', alert.hour_views,
                'performance_tier', alert.performance_tier
            )
        );

        -- Trigger webhook
        PERFORM pg_notify('viral_alert', row_to_json(alert)::text);
    END LOOP;
END;
$ LANGUAGE plpgsql;

-- Schedule real-time checks
SELECT cron.schedule('realtime-alerts', '* * * * *',
    'SELECT analytics.check_realtime_alerts()');
```

Data Warehouse Design

Star Schema Implementation

sql

```sql
-- Fact table for video performance
CREATE TABLE IF NOT EXISTS warehouse.fact_video_performance (
    date_key INTEGER NOT NULL,
    video_key INTEGER NOT NULL,
    channel_key INTEGER NOT NULL,

    -- Metrics
    views INTEGER NOT NULL DEFAULT 0,
    likes INTEGER NOT NULL DEFAULT 0,
    dislikes INTEGER NOT NULL DEFAULT 0,
    comments INTEGER NOT NULL DEFAULT 0,
    shares INTEGER NOT NULL DEFAULT 0,
    watch_time_minutes DECIMAL(12,2) NOT NULL DEFAULT 0,

    -- Revenue
    estimated_revenue_cents INTEGER NOT NULL DEFAULT 0,
    ad_revenue_cents INTEGER NOT NULL DEFAULT 0,

    -- Derived metrics
    engagement_rate DECIMAL(5,4),
    retention_rate DECIMAL(5,4),
    click_through_rate DECIMAL(5,4),

    -- Audit
    etl_timestamp TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,

    PRIMARY KEY (date_key, video_key)
) PARTITION BY RANGE (date_key);

-- Dimension tables
CREATE TABLE warehouse.dim_date (
    date_key INTEGER PRIMARY KEY,
    full_date DATE NOT NULL UNIQUE,
    year INTEGER NOT NULL,
    quarter INTEGER NOT NULL,
    month INTEGER NOT NULL,
    week INTEGER NOT NULL,
    day_of_month INTEGER NOT NULL,
    day_of_week INTEGER NOT NULL,
    day_name VARCHAR(20) NOT NULL,
    month_name VARCHAR(20) NOT NULL,
    is_weekend BOOLEAN NOT NULL,
    is_holiday BOOLEAN NOT NULL DEFAULT FALSE,
    fiscal_year INTEGER NOT NULL,
    fiscal_quarter INTEGER NOT NULL
);
```

```
CREATE TABLE warehouse.dim_video (
    video_key SERIAL PRIMARY KEY,
    video_id UUID NOT NULL UNIQUE,
    title VARCHAR(255) NOT NULL,
    description TEXT,
    duration_seconds INTEGER NOT NULL,
    category VARCHAR(100),
    tags TEXT[],
    language VARCHAR(10) NOT NULL,

    -- SCD Type 2 attributes
    valid_from DATE NOT NULL,
    valid_to DATE,
    is_current BOOLEAN NOT NULL DEFAULT TRUE,
    version INTEGER NOT NULL DEFAULT 1
);

CREATE TABLE warehouse.dim_channel (
    channel_key SERIAL PRIMARY KEY,
    channel_id UUID NOT NULL,
    channel_name VARCHAR(255) NOT NULL,
    niche VARCHAR(100) NOT NULL,

    -- SCD Type 2
    valid_from DATE NOT NULL,
    valid_to DATE,
    is_current BOOLEAN NOT NULL DEFAULT TRUE,
    version INTEGER NOT NULL DEFAULT 1,

    UNIQUE(channel_id, version)
);

-- ETL procedures for warehouse
CREATE OR REPLACE FUNCTION warehouse.etl_load_daily_facts(
    p_date DATE DEFAULT CURRENT_DATE - 1
) RETURNS void AS $
DECLARE
    v_date_key INTEGER;
    v_record_count INTEGER := 0;
BEGIN
    -- Get date key
    v_date_key := TO_CHAR(p_date, 'YYYYMMDD')::INTEGER;

    -- Clear existing data for the date
    DELETE FROM warehouse.fact_video_performance
    WHERE date_key = v_date_key;
```

```sql
    -- Load fact data
    INSERT INTO warehouse.fact_video_performance (
        date_key,
        video_key,
        channel_key,
        views,
        likes,
        comments,
        watch_time_minutes,
        estimated_revenue_cents,
        engagement_rate,
        retention_rate
    )
    SELECT
        v_date_key,
        dv.video_key,
        dc.channel_key,
        SUM(vm.views),
        SUM(vm.likes),
        SUM(vm.comments),
        SUM(vm.watch_time_minutes),
        SUM(vm.estimated_revenue_usd * 100)::INTEGER,
        AVG(vm.engagement_rate),
        AVG(vm.average_percentage_viewed / 100)
    FROM analytics.video_metrics vm
    JOIN warehouse.dim_video dv ON vm.video_id = dv.video_id AND dv.is_current
    JOIN core.videos v ON vm.video_id = v.id
    JOIN warehouse.dim_channel dc ON v.channel_id = dc.channel_id AND dc.is_current
    WHERE DATE(vm.timestamp) = p_date
    GROUP BY dv.video_key, dc.channel_key;

    GET DIAGNOSTICS v_record_count = ROW_COUNT;

    -- Log ETL execution
    INSERT INTO audit.etl_log (
        process_name, target_date, records_processed, status
    ) VALUES (
        'daily_fact_load', p_date, v_record_count, 'completed'
    );

    -- Update statistics
    ANALYZE warehouse.fact_video_performance;
END;
$ LANGUAGE plpgsql;

-- Schedule daily ETL
```

```
SELECT cron.schedule('warehouse-etl', '0 3 * * *',
    'SELECT warehouse.etl_load_daily_facts()');
```

Implementation Guidelines

Development Best Practices

Database Development Standards

yaml

```yaml
development_standards:
  naming_conventions:
    tables: lowercase_snake_case
    columns: lowercase_snake_case
    indexes: idx_{table}_{columns}
    constraints: {table}_{column}_{type}
    functions: verb_noun_pattern

  code_organization:
    schemas:
      - core (business logic)
      - analytics (reporting)
      - audit (logging/compliance)
      - staging (ETL)
      - ml (machine learning)

  version_control:
    migrations: flyway/liquibase
    branching: gitflow
    reviews: required
    testing: mandatory
```

Performance Testing Framework

sql

```
-- Performance baseline establishment
CREATE OR REPLACE FUNCTION core.establish_performance_baseline()
RETURNS TABLE (
    query_name TEXT,
    execution_time_ms NUMERIC,
    rows_returned BIGINT,
    memory_used_mb NUMERIC
) AS $
BEGIN
    -- Test query 1: Channel dashboard
    RETURN QUERY
    WITH timing AS (
        SELECT clock_timestamp() as start_time
    )
    SELECT
        'channel_dashboard'::TEXT,
        EXTRACT(epoch FROM (clock_timestamp() - start_time)) * 1000,
        COUNT(*)::BIGINT,
        pg_size_pretty(pg_total_relation_size('core.channels'))::NUMERIC
    FROM timing, core.channels
    WHERE status = 'active';

    -- Test query 2: Video analytics
    RETURN QUERY
    WITH timing AS (
        SELECT clock_timestamp() as start_time
    )
    SELECT
        'video_analytics_30d'::TEXT,
        EXTRACT(epoch FROM (clock_timestamp() - start_time)) * 1000,
        COUNT(*)::BIGINT,
        pg_size_pretty(pg_total_relation_size('analytics.video_metrics'))::NUMERIC
    FROM timing, analytics.video_metrics
    WHERE timestamp > NOW() - INTERVAL '30 days';

    -- Additional baseline queries...
END;
$ LANGUAGE plpgsql;
```

Deployment Guidelines

Production Deployment Checklist

```
yaml
```

```
deployment_checklist:
  pre_deployment:
    - Run full backup
    - Verify backup integrity
    - Test rollback procedure
    - Review migration scripts
    - Performance baseline capture

  deployment_steps:
    - Enable maintenance mode
    - Run schema migrations
    - Deploy stored procedures
    - Update permissions
    - Rebuild indexes
    - Update statistics
    - Verify data integrity

  post_deployment:
    - Run smoke tests
    - Verify performance
    - Check replication lag
    - Monitor error logs
    - Update documentation

  rollback_triggers:
    - Migration failure
    - Performance degradation >20%
    - Data integrity issues
    - Replication failures
```

Monitoring and Maintenance

```sql
```

```sql
-- Automated maintenance procedures
CREATE OR REPLACE FUNCTION core.perform_maintenance()
RETURNS void AS $
BEGIN
    -- Update table statistics
    ANALYZE;

    -- Vacuum tables
    VACUUM (ANALYZE, VERBOSE) core.videos;
    VACUUM (ANALYZE, VERBOSE) analytics.video_metrics;

    -- Reindex if needed
    PERFORM core.maintain_indexes();

    -- Clean up old data
    DELETE FROM staging.youtube_analytics_raw
    WHERE import_timestamp < NOW() - INTERVAL '7 days'
    AND processed = TRUE;

    -- Archive old audit logs
    INSERT INTO archive.audit_log
    SELECT * FROM audit.audit_log
    WHERE created_at < NOW() - INTERVAL '90 days';

    DELETE FROM audit.audit_log
    WHERE created_at < NOW() - INTERVAL '90 days';

    -- Log maintenance completion
    INSERT INTO audit.maintenance_log (
        maintenance_type, completed_at, details
    ) VALUES (
        'routine', NOW(), 'All maintenance tasks completed successfully'
    );
END;
$ LANGUAGE plpgsql;

-- Schedule maintenance
SELECT cron.schedule('nightly-maintenance', '0 4 * * *',
    'SELECT core.perform_maintenance()');
```

Conclusion

This Enterprise Data Architecture provides a comprehensive, scalable foundation for YTEMPIRE's data management needs. The PostgreSQL-centric design leverages advanced features while maintaining

simplicity and performance.

Key Architectural Achievements:

1. **Scalability**: Designed to handle 100,000+ videos and billions of analytics events
2. **Performance**: Sub-100ms operational queries with intelligent caching
3. **Reliability**: Comprehensive backup and high availability architecture
4. **Security**: Multi-layer security with encryption and compliance built-in
5. **Analytics**: Real-time and batch analytics with OLAP capabilities

Implementation Priorities:

1. **Phase 1**: Core schema deployment and basic operations
2. **Phase 2**: Analytics and reporting infrastructure
3. **Phase 3**: ML integration and advanced features
4. **Phase 4**: Scale-out architecture and optimization

The architecture provides a solid foundation that can evolve with YTEMPIRE's growth while maintaining data integrity, performance, and operational excellence.

Document Version: 1.0
Last Updated: [Current Date]
Author: Saad T. - Solution Architect