Supabase Cloud Configuration Guide

Complete guide for setting up PostgreSQL database and vector store on Supabase Cloud for the CX-Catalyst AI Support System.

Table of Contents

---

1. Create Supabase Project

Step 1.1: Sign Up / Log In

1. Go to supabase.com
2. Click **Start your project** or **Sign In**
3. Sign up with GitHub, GitLab, or email

Step 1.2: Create New Project

1. Click **New Project**
2. Select your organization (or create one)
3. Enter project details:
   - **Name**: `cx-catalyst` (or your preferred name)
   - **Database Password**: Generate a strong password and **save it securely**
   - **Region**: Select closest to your n8n instance
   - **Pricing Plan**: Free tier works for development; Pro recommended for production
4. Click **Create new project**
5. Wait 2-3 minutes for provisioning to complete

Step 1.3: Get Connection Details

Once the project is ready, navigate to **Settings** > **Database**:

Copy and save these values:

Setting	Location	Example
Host	Connection string	`db.abcdefghijkl.supabase.co`
Port	Connection info	`5432` (or `6543` for connection pooling)
Database	Connection info	`postgres`

User	Connection info	`postgres`
Password	The one you set	(your saved password)

Connection String Format:

```
postgresql://postgres:[YOUR-PASSWORD]@db.[PROJECT-REF].supabase.co:5432/postgres
```

---

2. Configure Database

Step 2.1: Access SQL Editor

1. In your Supabase dashboard, click **SQL Editor** in the left sidebar
2. Click **New query**

Step 2.2: Enable Required Extensions

Run this SQL first to enable necessary extensions:

```sql
-- Enable UUID generation
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

-- Enable pgcrypto for gen_random_uuid()
CREATE EXTENSION IF NOT EXISTS pgcrypto;

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

-- Verify extensions are enabled
SELECT extname, extversion FROM pg_extension;
```

Click **Run** to execute.

---

3. Run Schema Migrations

Run the following SQL statements in the SQL Editor. Execute each section separately for easier debugging.

Step 3.1: Core Tables

```sql
-- =====================================================
-- CX-Catalyst Core Database Schema
-- =====================================================

-- Customers table
CREATE TABLE IF NOT EXISTS customers (
    customer_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    email VARCHAR(255) UNIQUE NOT NULL,
    name VARCHAR(255),
    account_tier VARCHAR(50) DEFAULT 'standard',
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW(),
    metadata JSONB DEFAULT '{}'
);

-- Cases table
CREATE TABLE IF NOT EXISTS cases (
    case_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    customer_id UUID REFERENCES customers(customer_id),
    channel VARCHAR(50) NOT NULL,
    description TEXT NOT NULL,
    category VARCHAR(100),
    subcategory VARCHAR(100),
    priority VARCHAR(20),
    confidence_score DECIMAL(3,2),
    status VARCHAR(50) DEFAULT 'new',
    resolution_type VARCHAR(50),
    escalated BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT NOW(),
    triaged_at TIMESTAMP,
    resolved_at TIMESTAMP,
    closed_at TIMESTAMP,
    resolution_time_minutes INTEGER,
    satisfaction_score INTEGER,
    metadata JSONB DEFAULT '{}'
);

-- Case interactions
CREATE TABLE IF NOT EXISTS case_interactions (
    interaction_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    case_id UUID REFERENCES cases(case_id) ON DELETE CASCADE,
    actor_type VARCHAR(50),
    actor_id VARCHAR(255),
    message TEXT,
    interaction_type VARCHAR(50),
    created_at TIMESTAMP DEFAULT NOW(),
    metadata JSONB DEFAULT '{}'
);

-- Knowledge base articles
CREATE TABLE IF NOT EXISTS kb_articles (
    article_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    title VARCHAR(500) NOT NULL,
    category VARCHAR(100),
    subcategory VARCHAR(100),
    tags TEXT[],
    auto_generated BOOLEAN DEFAULT FALSE,
    human_validated BOOLEAN DEFAULT FALSE,
    view_count INTEGER DEFAULT 0,
    success_count INTEGER DEFAULT 0,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW(),
    vector_store_id VARCHAR(255),
    confluence_page_id VARCHAR(255),
    metadata JSONB DEFAULT '{}'
);

-- Error codes
CREATE TABLE IF NOT EXISTS error_codes (
    error_code VARCHAR(100) PRIMARY KEY,
    product VARCHAR(100),
    description TEXT,
    severity VARCHAR(20),
    diagnostic_steps TEXT[],
    resolution_steps TEXT[],
    automated_fix_available BOOLEAN DEFAULT FALSE,
    automation_script TEXT,
    kb_article_id UUID REFERENCES kb_articles(article_id),
    occurrence_count INTEGER DEFAULT 0,
    last_seen TIMESTAMP,
    created_at TIMESTAMP DEFAULT NOW(),
    metadata JSONB DEFAULT '{}'
);

-- Health metrics
CREATE TABLE IF NOT EXISTS health_metrics (
    metric_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    metric_name VARCHAR(100),
    metric_value DECIMAL,
    unit VARCHAR(50),
    threshold_warning DECIMAL,
    threshold_critical DECIMAL,
    status VARCHAR(20),
    recorded_at TIMESTAMP DEFAULT NOW(),
    metadata JSONB DEFAULT '{}'
);

-- Proactive alerts
CREATE TABLE IF NOT EXISTS proactive_alerts (
    alert_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    alert_type VARCHAR(100),
    severity VARCHAR(20),
    description TEXT,
    affected_customers UUID[],
    root_cause_hypothesis TEXT,
    actions_taken TEXT[],
    prevented_cases_estimate INTEGER,
    jira_ticket_id VARCHAR(100),
    created_at TIMESTAMP DEFAULT NOW(),
    resolved_at TIMESTAMP,
    metadata JSONB DEFAULT '{}'
);

-- Review queue
CREATE TABLE IF NOT EXISTS review_queue (
    review_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    case_id UUID REFERENCES cases(case_id),
    ai_solution TEXT,
    ai_confidence DECIMAL(3,2),
    status VARCHAR(50) DEFAULT 'pending',
    assigned_to VARCHAR(255),
    created_at TIMESTAMP DEFAULT NOW(),
    reviewed_at TIMESTAMP,
    reviewer_comments TEXT,
    final_solution TEXT,
    metadata JSONB DEFAULT '{}'
);

-- Workflow executions
CREATE TABLE IF NOT EXISTS workflow_executions (
    execution_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    workflow_name VARCHAR(255),
    case_id UUID REFERENCES cases(case_id),
    status VARCHAR(50),
    start_time TIMESTAMP DEFAULT NOW(),
    end_time TIMESTAMP,
    duration_ms INTEGER,
    error_message TEXT,
    metadata JSONB DEFAULT '{}'
);

-- Agent feedback
CREATE TABLE IF NOT EXISTS agent_feedback (
    feedback_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    case_id UUID REFERENCES cases(case_id),
    agent_name VARCHAR(100),
    agent_output TEXT,
    human_correction TEXT,
    feedback_type VARCHAR(50),
    created_at TIMESTAMP DEFAULT NOW(),
    metadata JSONB DEFAULT '{}'
);
```

Step 3.2: Indexes for Core Tables

```
-- Create indexes for performance
CREATE INDEX IF NOT EXISTS idx_customers_email ON customers(email);
CREATE INDEX IF NOT EXISTS idx_customers_tier ON customers(account_tier);
CREATE INDEX IF NOT EXISTS idx_cases_customer ON cases(customer_id);
CREATE INDEX IF NOT EXISTS idx_cases_status ON cases(status);
CREATE INDEX IF NOT EXISTS idx_cases_category ON cases(category);
CREATE INDEX IF NOT EXISTS idx_cases_created_at ON cases(created_at DESC);
CREATE INDEX IF NOT EXISTS idx_cases_priority ON cases(priority);
CREATE INDEX IF NOT EXISTS idx_interactions_case ON case_interactions(case_id);
CREATE INDEX IF NOT EXISTS idx_kb_category ON kb_articles(category);
CREATE INDEX IF NOT EXISTS idx_kb_tags ON kb_articles USING GIN(tags);
```

```sql
CREATE INDEX IF NOT EXISTS idx_error_codes_product ON error_codes(product);
CREATE INDEX IF NOT EXISTS idx_health_metrics_name ON health_metrics(metric_name);
CREATE INDEX IF NOT EXISTS idx_alerts_created_at ON proactive_alerts(created_at
DESC);
CREATE INDEX IF NOT EXISTS idx_review_queue_status ON review_queue(status);
CREATE INDEX IF NOT EXISTS idx_workflow_exec_workflow ON
workflow_executions(workflow_name);
CREATE INDEX IF NOT EXISTS idx_agent_feedback_agent ON agent_feedback(agent_name);
```

Step 3.3: Token Usage & Dashboard Tables

```sql
-- =====================================================
-- Token Usage & Dashboard Metrics Schema
-- =====================================================

-- Token Usage Tracking
CREATE TABLE IF NOT EXISTS token_usage (
    usage_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    workflow_name VARCHAR(100) NOT NULL,
    workflow_execution_id VARCHAR(255),
    case_id UUID REFERENCES cases(case_id),
    provider VARCHAR(50) NOT NULL,
    model VARCHAR(100) NOT NULL,
    operation_type VARCHAR(50),
    input_tokens INTEGER NOT NULL DEFAULT 0,
    output_tokens INTEGER NOT NULL DEFAULT 0,
    total_tokens INTEGER GENERATED ALWAYS AS (input_tokens + output_tokens) STORED,
    cost_usd DECIMAL(10, 6),
    latency_ms INTEGER,
    success BOOLEAN DEFAULT TRUE,
    error_message TEXT,
    recorded_at TIMESTAMP DEFAULT NOW(),
    metadata JSONB DEFAULT '{}'
);

CREATE INDEX IF NOT EXISTS idx_token_usage_workflow ON token_usage(workflow_name);
CREATE INDEX IF NOT EXISTS idx_token_usage_provider ON token_usage(provider);
CREATE INDEX IF NOT EXISTS idx_token_usage_recorded_at ON token_usage(recorded_at
DESC);
CREATE INDEX IF NOT EXISTS idx_token_usage_case ON token_usage(case_id);

-- Token Budget Configuration
CREATE TABLE IF NOT EXISTS token_budgets (
    budget_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    budget_name VARCHAR(100) NOT NULL,
    provider VARCHAR(50) NOT NULL,
    budget_type VARCHAR(20) NOT NULL,
    token_limit BIGINT NOT NULL,
    cost_limit_usd DECIMAL(10, 2),
    warning_threshold DECIMAL(3, 2) DEFAULT 0.80,
    critical_threshold DECIMAL(3, 2) DEFAULT 0.95,
    period_start DATE NOT NULL,
    period_end DATE NOT NULL,
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT NOW(),
    metadata JSONB DEFAULT '{}'
);

CREATE INDEX IF NOT EXISTS idx_token_budgets_active ON token_budgets(is_active,
period_start, period_end);

-- Insert default budgets
INSERT INTO token_budgets (budget_name, provider, budget_type, token_limit,
cost_limit_usd, period_start, period_end)
VALUES
    ('Anthropic Monthly', 'anthropic', 'monthly', 10000000, 3000.00,
DATE_TRUNC('month', CURRENT_DATE), DATE_TRUNC('month', CURRENT_DATE) + INTERVAL '1
month' - INTERVAL '1 day'),
    ('OpenAI Monthly', 'openai', 'monthly', 5000000, 500.00, DATE_TRUNC('month',
CURRENT_DATE), DATE_TRUNC('month', CURRENT_DATE) + INTERVAL '1 month' - INTERVAL '1
day'),
    ('Anthropic Daily', 'anthropic', 'daily', 500000, 150.00, CURRENT_DATE,
CURRENT_DATE),
    ('OpenAI Daily', 'openai', 'daily', 250000, 25.00, CURRENT_DATE, CURRENT_DATE)
ON CONFLICT DO NOTHING;

-- Daily Metrics Snapshots
CREATE TABLE IF NOT EXISTS daily_metrics (
    metric_date DATE PRIMARY KEY,
    total_cases INTEGER DEFAULT 0,
    resolved_cases INTEGER DEFAULT 0,
    escalated_cases INTEGER DEFAULT 0,
    self_service_cases INTEGER DEFAULT 0,
    collaborative_cases INTEGER DEFAULT 0,
    avg_resolution_time_minutes DECIMAL(10, 2),
    median_resolution_time_minutes DECIMAL(10, 2),
    first_contact_resolution_count INTEGER DEFAULT 0,
    avg_satisfaction_score DECIMAL(3, 2),
    satisfaction_responses INTEGER DEFAULT 0,
    avg_confidence_score DECIMAL(3, 2),
    ai_approvals INTEGER DEFAULT 0,
    ai_edits INTEGER DEFAULT 0,
    ai_rejections INTEGER DEFAULT 0,
    anthropic_input_tokens BIGINT DEFAULT 0,
    anthropic_output_tokens BIGINT DEFAULT 0,
    anthropic_cost_usd DECIMAL(10, 2) DEFAULT 0,
    openai_input_tokens BIGINT DEFAULT 0,
    openai_output_tokens BIGINT DEFAULT 0,
    openai_cost_usd DECIMAL(10, 2) DEFAULT 0,
    estimated_time_saved_minutes INTEGER DEFAULT 0,
    estimated_cost_saved_usd DECIMAL(10, 2) DEFAULT 0,
    workflow_executions INTEGER DEFAULT 0,
    workflow_failures INTEGER DEFAULT 0,
    avg_workflow_duration_ms INTEGER,
    created_at TIMESTAMP DEFAULT NOW(),
    metadata JSONB DEFAULT '{}'
);

-- API Pricing Configuration
CREATE TABLE IF NOT EXISTS api_pricing (
    pricing_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    provider VARCHAR(50) NOT NULL,
    model VARCHAR(100) NOT NULL,
    input_price_per_million DECIMAL(10, 4) NOT NULL,
    output_price_per_million DECIMAL(10, 4) NOT NULL,
    effective_from DATE NOT NULL,
    effective_to DATE,
    is_current BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT NOW()
);

CREATE INDEX IF NOT EXISTS idx_api_pricing_current ON api_pricing(provider, model,
is_current);

-- Insert current pricing (January 2026 rates)
INSERT INTO api_pricing (provider, model, input_price_per_million,
output_price_per_million, effective_from)
VALUES
    ('anthropic', 'claude-sonnet-4', 3.00, 15.00, '2026-01-01'),
    ('anthropic', 'claude-haiku-3', 0.25, 1.25, '2026-01-01'),
    ('openai', 'gpt-4-turbo', 10.00, 30.00, '2026-01-01'),
    ('openai', 'gpt-4o', 5.00, 15.00, '2026-01-01'),
    ('openai', 'text-embedding-3-large', 0.13, 0.00, '2026-01-01'),
    ('openai', 'text-embedding-3-small', 0.02, 0.00, '2026-01-01')
ON CONFLICT DO NOTHING;

-- Baseline Configuration for ROI calculations
CREATE TABLE IF NOT EXISTS baseline_config (
    config_key VARCHAR(100) PRIMARY KEY,
    config_value DECIMAL(10, 2) NOT NULL,
    description TEXT,
    updated_at TIMESTAMP DEFAULT NOW()
);

INSERT INTO baseline_config (config_key, config_value, description)
VALUES
    ('manual_cost_per_case', 20.00, 'Average cost for fully manual case handling'),
    ('manual_avg_resolution_minutes', 45, 'Average resolution time without AI'),
    ('hourly_support_cost', 35.00, 'Fully-loaded hourly cost of support staff'),
    ('self_service_cost_per_case', 2.00, 'Cost for AI-automated case resolution'),
    ('collaborative_cost_per_case', 10.00, 'Cost for human-reviewed AI resolution'),
    ('escalated_cost_per_case', 25.00, 'Cost for fully escalated cases'),
    ('target_self_service_rate', 0.85, 'Target self-service resolution rate'),
    ('target_csat_score', 4.5, 'Target customer satisfaction score')
ON CONFLICT (config_key) DO NOTHING;
```

```sql
-- KB Article Performance Tracking
CREATE TABLE IF NOT EXISTS kb_article_performance (
    performance_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    article_id UUID REFERENCES kb_articles(article_id),
    metric_date DATE NOT NULL,
    retrievals INTEGER DEFAULT 0,
    successful_resolutions INTEGER DEFAULT 0,
    partial_resolutions INTEGER DEFAULT 0,
    failed_resolutions INTEGER DEFAULT 0,
    avg_relevance_score DECIMAL(3, 2),
    created_at TIMESTAMP DEFAULT NOW(),
    UNIQUE(article_id, metric_date)
);

CREATE INDEX IF NOT EXISTS idx_kb_perf_date ON kb_article_performance(metric_date
DESC);

-- Improvement Tracking
CREATE TABLE IF NOT EXISTS improvement_opportunities (
    opportunity_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    opportunity_type VARCHAR(50) NOT NULL,
    title VARCHAR(500) NOT NULL,
    description TEXT,
    category VARCHAR(100),
    frequency INTEGER DEFAULT 1,
    impact_score DECIMAL(3, 2),
    status VARCHAR(50) DEFAULT 'identified',
    resolution_notes TEXT,
    first_detected DATE DEFAULT CURRENT_DATE,
    last_detected DATE DEFAULT CURRENT_DATE,
    resolved_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT NOW(),
    metadata JSONB DEFAULT '{}'
);

CREATE INDEX IF NOT EXISTS idx_improvement_type ON
improvement_opportunities(opportunity_type, status);
CREATE INDEX IF NOT EXISTS idx_improvement_impact ON
improvement_opportunities(impact_score DESC);
```

---

4. Enable Vector Store (pgvector)

Step 4.1: Create Confluence KB Table with Vector Support

```sql
-- ======================================================
-- Vector Store for Knowledge Base (Confluence)
-- ======================================================

-- Confluence Knowledge Base with embeddings
```

```sql
CREATE TABLE IF NOT EXISTS confluence_kb (
    id BIGSERIAL PRIMARY KEY,
    page_id TEXT UNIQUE NOT NULL,
    space_key TEXT NOT NULL,
    title TEXT NOT NULL,
    content TEXT NOT NULL,
    url TEXT NOT NULL,
    embedding VECTOR(1536),  -- OpenAI text-embedding-3-small dimension
    metadata JSONB,
    created_at TIMESTAMPTZ DEFAULT NOW(),
    updated_at TIMESTAMPTZ DEFAULT NOW(),
    last_indexed_at TIMESTAMPTZ DEFAULT NOW()
);

-- Create index for vector similarity search
-- Note: ivfflat is good for datasets < 1M rows
CREATE INDEX IF NOT EXISTS idx_confluence_kb_embedding
ON confluence_kb USING ivfflat (embedding vector_cosine_ops)
WITH (lists = 100);

-- Create index for page_id lookups
CREATE INDEX IF NOT EXISTS idx_confluence_page_id ON confluence_kb(page_id);

-- Create index for space filtering
CREATE INDEX IF NOT EXISTS idx_confluence_space_key ON confluence_kb(space_key);
```

Step 4.2: Create Vector Search Function

```sql
-- Function to search similar documents
CREATE OR REPLACE FUNCTION match_confluence_pages(
    query_embedding VECTOR(1536),
    match_threshold FLOAT DEFAULT 0.7,
    match_count INT DEFAULT 5
)
RETURNS TABLE (
    page_id TEXT,
    title TEXT,
    content TEXT,
    url TEXT,
    similarity FLOAT
)
LANGUAGE SQL STABLE
AS $$
    SELECT
        page_id,
        title,
        content,
        url,
        1 - (embedding <=> query_embedding) AS similarity
    FROM confluence_kb
    WHERE 1 - (embedding <=> query_embedding) > match_threshold
    ORDER BY embedding <=> query_embedding
    LIMIT match_count;
$$;
```

Step 4.3: Create Token Cost Calculation Trigger

```
-- Function to calculate cost for token usage
CREATE OR REPLACE FUNCTION calculate_token_cost()
RETURNS TRIGGER AS $$
DECLARE
    input_price DECIMAL(10, 4);
    output_price DECIMAL(10, 4);
BEGIN
    -- Get current pricing
    SELECT input_price_per_million, output_price_per_million
    INTO input_price, output_price
    FROM api_pricing
    WHERE provider = NEW.provider
      AND model = NEW.model
      AND is_current = TRUE
    LIMIT 1;

    -- Calculate cost if pricing found
    IF input_price IS NOT NULL THEN
        NEW.cost_usd := (NEW.input_tokens * input_price / 1000000) +
                        (NEW.output_tokens * output_price / 1000000);
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Create trigger for automatic cost calculation
DROP TRIGGER IF EXISTS trg_calculate_token_cost ON token_usage;
CREATE TRIGGER trg_calculate_token_cost
    BEFORE INSERT ON token_usage
    FOR EACH ROW
    EXECUTE FUNCTION calculate_token_cost();
```

Step 4.4: Verify Vector Store Setup

```
-- Verify vector extension is enabled
SELECT * FROM pg_extension WHERE extname = 'vector';

-- Verify table structure
SELECT column_name, data_type
FROM information_schema.columns
WHERE table_name = 'confluence_kb';
```

```sql
-- Test the vector search function (will return empty until you add data)
SELECT * FROM match_confluence_pages(
    ARRAY_FILL(0::float, ARRAY[1536])::vector,
    0.5,
    5
);
```

---

5. Configure n8n Credentials

Step 5.1: Get Supabase API Keys

1. In Supabase dashboard, go to **Settings** > **API**
2. Copy these values:
   - **Project URL**: `https://[project-ref].supabase.co`
   - **anon public key**: For client-side requests (if needed)
   - **service_role secret key**: For server-side/n8n requests

Step 5.2: Configure PostgreSQL Credential in n8n

1. In n8n, go to **Settings** > **Credentials**
2. Click **Add Credential** > **PostgreSQL**
3. Enter:

Field	Value
Credential Name	`Supabase PostgreSQL`
Host	`db.[project-ref].supabase.co`
Port	`5432` (direct) or `6543` (pooling)
Database	`postgres`
User	`postgres`
Password	Your database password
SSL	Enable (Require)

4. Click **Test Connection** to verify
5. Click **Save**

Step 5.3: Configure Supabase API Credential in n8n

1. Click **Add Credential** > **Supabase API**
2. Enter:

Field	Value
Credential Name	`Supabase API`
Host	`https://[project-ref].supabase.co`
Service Role Secret	Your service_role key

3. Click **Save**

Step 5.4: Set Environment Variables in n8n

Go to **Settings** > **Environment Variables** and add:

```
SUPABASE_URL=https://[project-ref].supabase.co
SUPABASE_SERVICE_KEY=your-service-role-key
DATABASE_URL=postgresql://postgres:[password]@db.[project-
ref].supabase.co:5432/postgres
```

---

6. Test Connection

Step 6.1: Test PostgreSQL from n8n

Create a simple test workflow:

1. Add **Manual Trigger** node
2. Add **PostgreSQL** node with:
    - Operation: **Execute Query**
    - Query: `SELECT NOW() as current_time, version() as pg_version;`
3. Execute and verify results

Step 6.2: Test Vector Search

Add another PostgreSQL node:

```sql
-- Test embedding insert
INSERT INTO confluence_kb (page_id, space_key, title, content, url, embedding)
VALUES (
    'test-page-001',
    'TEST',
    'Test Article',
    'This is a test knowledge base article for verifying vector search.',
    'https://example.com/test',
    ARRAY_FILL(0.1::float, ARRAY[1536])::vector
);

-- Test retrieval
SELECT page_id, title,
       1 - (embedding <=> ARRAY_FILL(0.1::float, ARRAY[1536])::vector) as similarity
FROM confluence_kb
WHERE page_id = 'test-page-001';
```

Step 6.3: Clean Up Test Data

```sql
DELETE FROM confluence_kb WHERE page_id = 'test-page-001';
```

---

7. Security Best Practices

Row Level Security (RLS)

Enable RLS for production environments:

```sql
-- Enable RLS on sensitive tables
ALTER TABLE customers ENABLE ROW LEVEL SECURITY;
ALTER TABLE cases ENABLE ROW LEVEL SECURITY;
ALTER TABLE case_interactions ENABLE ROW LEVEL SECURITY;

-- Create policy for service role (full access)
CREATE POLICY "Service role has full access to customers"
ON customers FOR ALL
TO service_role
USING (true)
WITH CHECK (true);

CREATE POLICY "Service role has full access to cases"
ON cases FOR ALL
TO service_role
USING (true)
WITH CHECK (true);
```

API Key Security

- **Never** expose `service_role` key in client-side code
- Use `anon` key for client applications with RLS
- Rotate keys periodically
- Store keys in n8n credentials, not in workflow nodes

Network Security

1. In Supabase dashboard, go to **Settings** > **Database**
2. Under **Network restrictions**, add allowed IPs if your n8n has a static IP
3. Consider using **Connection Pooling** for production (port 6543)

---

8. Troubleshooting

Connection Issues

Error: "connection refused"

- Verify host is `db.[project-ref].supabase.co` (not just the project ref)
- Check if project is paused (free tier pauses after inactivity)
- Ensure SSL is enabled

Error: "password authentication failed"

- Reset database password in Supabase dashboard
- Update credential in n8n

Error: "too many connections"

- Use connection pooling (port 6543)

- Reduce concurrent workflow executions

Vector Search Issues

Error: "operator does not exist: vector <=> vector"

- Run: `CREATE EXTENSION IF NOT EXISTS vector;`

Slow searches

- Ensure index exists: Check with `\d confluence_kb`
- Rebuild index: `REINDEX INDEX idx_confluence_kb_embedding;`
- For large datasets (>1M rows), consider switching to HNSW index

Schema Issues

Error: "relation does not exist"

- Tables may not be created; re-run schema migrations
- Check you're connected to the correct database

Error: "duplicate key"

- Use `ON CONFLICT DO NOTHING` or `DO UPDATE` clauses
- Check for existing data before inserts

Performance Optimization

```sql
-- Analyze tables after bulk inserts
ANALYZE customers;
ANALYZE cases;
ANALYZE confluence_kb;

-- Check table sizes
SELECT
    relname as table_name,
    pg_size_pretty(pg_total_relation_size(relid)) as total_size
FROM pg_catalog.pg_statio_user_tables
ORDER BY pg_total_relation_size(relid) DESC;

-- Check index usage
SELECT
    indexrelname as index_name,
    idx_scan as times_used
FROM pg_stat_user_indexes
WHERE schemaname = 'public'
ORDER BY idx_scan DESC;
```

Quick Reference

Connection String

```
postgresql://postgres:[PASSWORD]@db.[PROJECT-REF].supabase.co:5432/postgres
```

Supabase Dashboard Links

- **SQL Editor**: https://supabase.com/dashboard/project/[PROJECT-REF]/sql
- **Table Editor**: https://supabase.com/dashboard/project/[PROJECT-REF]/editor
- **API Settings**: https://supabase.com/dashboard/project/[PROJECT-REF]/settings/api
- **Database Settings**: https://supabase.com/dashboard/project/[PROJECT-REF]/settings/database

Key Tables

Table	Purpose
`customers`	Customer records
`cases`	Support cases
`case_interactions`	Case history
`kb_articles`	Knowledge base metadata
`confluence_kb`	Vector embeddings for search
`token_usage`	AI API usage tracking
`token_budgets`	Usage limits and alerts
`daily_metrics`	Historical metrics

Next Steps

1. **Import Workflows**: Import n8n workflow JSON files
2. **Configure Credentials**: Connect workflows to Supabase
3. **Set Up Confluence**: Create KB space and run indexer
4. **Test End-to-End**: Submit test support request
5. **Monitor**: Set up alerts for budget thresholds

Supabase Cloud Setup Guide v1.0 - January 2026