PostgreSQL Cluster with pg_textsearch and pgvector

Complete OCI-compliant setup with 1 primary (write) and 2+ replicas (read-only) using PostgreSQL 18.

File Structure

Create this directory structure:

```
postgres-cluster/
├── Dockerfile
├── docker-compose.yml
├── podman-compose.yml
├── .env
├── README.md
├── conf/
│   ├── postgresql-primary.conf
│   └── postgresql-replica.conf
└── init-scripts/
    ├── 01-init-primary.sh
    └── init-replica.sh
```

---

1. Dockerfile

dockerfile

```
FROM postgres:18-bookworm

LABEL org.opencontainers.image.title="PostgreSQL with pg_textsearch and pgvector"
LABEL org.opencontainers.image.description="PostgreSQL 18 with pg_textsearch and pgvector extensions"
LABEL org.opencontainers.image.version="18.0"
LABEL org.opencontainers.image.licenses="PostgreSQL"

# Install build dependencies
RUN apt-get update && apt-get install -y \
    build-essential \
    git \
    postgresql-server-dev-18 \
    libicu-dev \
    pkg-config \
    && rm -rf /var/lib/apt/lists/*

# Install pgvector extension (using main branch for PostgreSQL 18 compatibility)
RUN cd /tmp && \
    git clone https://github.com/pgvector/pgvector.git && \
    cd pgvector && \
    make && \
    make install && \
    cd / && \
    rm -rf /tmp/pgvector

# Install pg_textsearch extension
RUN cd /tmp && \
    git clone https://github.com/timescale/pg_textsearch.git && \
    cd pg_textsearch && \
    make && \
    make install && \
    cd / && \
    rm -rf /tmp/pg_textsearch

# Create directories
RUN mkdir -p /docker-entrypoint-initdb.d

# Copy initialization scripts
COPY ./init-scripts/ /docker-entrypoint-initdb.d/

# Copy custom configuration
COPY ./conf/ /etc/postgresql/
```

```
# Expose PostgreSQL port
EXPOSE 5432

# Health check
HEALTHCHECK --interval=30s --timeout=10s --start-period=40s --retries=3 \
    CMD pg_isready -U postgres || exit 1

ENTRYPOINT ["docker-entrypoint.sh"]
CMD ["postgres"]
```

---

2. conf/postgresql-primary.conf

```
conf
```

```
# Primary PostgreSQL Configuration (Write Instance)

listen_addresses = '*'
max_connections = 200
superuser_reserved_connections = 3

# Memory Settings
shared_buffers = 256MB
effective_cache_size = 1GB
maintenance_work_mem = 64MB
work_mem = 4MB

# WAL Settings for Replication
wal_level = replica
max_wal_senders = 10
max_replication_slots = 10
wal_keep_size = 1GB
hot_standby = on

# Archive Settings
archive_mode = on
archive_command = 'test ! -f /var/lib/postgresql/archive/%f && cp %p /var/lib/postgresql/archive/%f'

# Checkpoint Settings
checkpoint_completion_target = 0.9
wal_compression = on

# Logging
logging_collector = on
log_directory = 'log'
log_filename = 'postgresql-%Y-%m-%d_%H%M%S.log'
log_rotation_age = 1d
log_rotation_size = 100MB
log_line_prefix = '%t [%p]: user=%u,db=%d,app=%a,client=%h '
log_timezone = 'UTC'
log_statement = 'mod'
log_replication_commands = on

# Extensions
shared_preload_libraries = 'pg_stat_statements'
# Note: vector extension is loaded per-database, not preloaded

# Performance
```

```
random_page_cost = 1.1
effective_io_concurrency = 200

timezone = 'UTC'
```

---

3. conf/postgresql-replica.conf

```
conf
```

```
# Replica PostgreSQL Configuration (Read-Only Instance)

listen_addresses = '*'
max_connections = 200
superuser_reserved_connections = 3

# Memory Settings
shared_buffers = 256MB
effective_cache_size = 1GB
maintenance_work_mem = 64MB
work_mem = 4MB

# WAL Settings
wal_level = replica
max_wal_senders = 10
max_replication_slots = 10
hot_standby = on
hot_standby_feedback = on

# Standby Settings
max_standby_streaming_delay = 30s
wal_receiver_status_interval = 10s
wal_retrieve_retry_interval = 5s

# Logging
logging_collector = on
log_directory = 'log'
log_filename = 'postgresql-%Y-%m-%d_%H%M%S.log'
log_rotation_age = 1d
log_rotation_size = 100MB
log_line_prefix = '%t [%p]: user=%u,db=%d,app=%a,client=%h '
log_timezone = 'UTC'
log_statement = 'none'
log_replication_commands = on

# Extensions
shared_preload_libraries = 'pg_stat_statements'

# Performance
random_page_cost = 1.1
effective_io_concurrency = 200
```

```
timezone = 'UTC'
```

---

4. init-scripts/01-init-primary.sh

```
bash
```

```bash
#!/bin/bash
set -e

echo "Initializing Primary PostgreSQL instance..."

# Create replication user
psql -v ON_ERROR_STOP=1 --username "$POSTGRES_USER" --dbname "$POSTGRES_DB" <<-EOSQL
    -- Create replication user
    DO \$\$
    BEGIN
        IF NOT EXISTS (SELECT FROM pg_catalog.pg_roles WHERE rolname = 'replicator') THEN
            CREATE ROLE replicator WITH REPLICATION LOGIN PASSWORD '${REPLICATION_PASSWORD:-replicator
        END IF;
    END
    \$\$;

    -- Create replication slots
    SELECT pg_create_physical_replication_slot('replica_slot_1')
    WHERE NOT EXISTS (SELECT 1 FROM pg_replication_slots WHERE slot_name = 'replica_slot_1');

    SELECT pg_create_physical_replication_slot('replica_slot_2')
    WHERE NOT EXISTS (SELECT 1 FROM pg_replication_slots WHERE slot_name = 'replica_slot_2');

    SELECT pg_create_physical_replication_slot('replica_slot_3')
    WHERE NOT EXISTS (SELECT 1 FROM pg_replication_slots WHERE slot_name = 'replica_slot_3');
EOSQL

# Configure pg_hba.conf for replication
cat >> "${PGDATA}/pg_hba.conf" <<EOF

# Replication connections
host    replication     replicator      0.0.0.0/0               scram-sha-256
host    replication     replicator      ::/0                    scram-sha-256
EOF

# Install extensions in default database
psql -v ON_ERROR_STOP=1 --username "$POSTGRES_USER" --dbname "$POSTGRES_DB" <<-EOSQL
    -- Install extensions
    CREATE EXTENSION IF NOT EXISTS vector;
    CREATE EXTENSION IF NOT EXISTS pg_stat_statements;

    -- Grant usage
    GRANT USAGE ON SCHEMA public TO replicator;
```

```
EOSQL

# Also install in template1 so new databases get the extensions
psql -v ON_ERROR_STOP=1 --username "$POSTGRES_USER" --dbname "template1" <<-EOSQL
    CREATE EXTENSION IF NOT EXISTS vector;
    CREATE EXTENSION IF NOT EXISTS pg_stat_statements;
EOSQL

# Create archive directory
mkdir -p /var/lib/postgresql/archive
chown postgres:postgres /var/lib/postgresql/archive

echo "Primary PostgreSQL initialization complete."
```

---

5. init-scripts/init-replica.sh

```
bash
```

```bash
#!/bin/bash
set -e

PRIMARY_HOST="${PRIMARY_HOST:-postgres-primary}"
PRIMARY_PORT="${PRIMARY_PORT:-5432}"
REPLICATION_USER="${REPLICATION_USER:-replicator}"
REPLICATION_PASSWORD="${REPLICATION_PASSWORD:-replicator_password}"
REPLICA_SLOT="${REPLICA_SLOT:-replica_slot_1}"

echo "Initializing PostgreSQL Replica..."

# Wait for primary
until PGPASSWORD="${REPLICATION_PASSWORD}" psql -h "${PRIMARY_HOST}" -p "${PRIMARY_PORT}" -U "$
    echo "Waiting for primary..."
    sleep 2
done

echo "Primary ready. Starting base backup..."

# Check if already initialized
if [ -f "${PGDATA}/standby.signal" ]; then
    echo "Replica already initialized."
    exit 0
fi

# Clean data directory
if [ -d "${PGDATA}" ] && [ "$(ls -A ${PGDATA})" ]; then
    rm -rf "${PGDATA}"/*
fi

# Perform base backup
PGPASSWORD="${REPLICATION_PASSWORD}" pg_basebackup \
    -h "${PRIMARY_HOST}" \
    -p "${PRIMARY_PORT}" \
    -U "${REPLICATION_USER}" \
    -D "${PGDATA}" \
    -P \
    -Xs \
    -c fast \
    -R \
    -S "${REPLICA_SLOT}"

# Create standby signal
```

```
touch "${PGDATA}/standby.signal"

# Update connection info
cat >> "${PGDATA}/postgresql.auto.conf" <<EOF
primary_conninfo = 'host=${PRIMARY_HOST} port=${PRIMARY_PORT} user=${REPLICATION_USER} password=${
primary_slot_name = '${REPLICA_SLOT}'
EOF

echo "Replica initialization complete."
```

---

6. docker-compose.yml

yaml

```yaml
version: '3.9'

services:
  postgres-primary:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: postgres-primary
    hostname: postgres-primary
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD:-postgres_password}
      POSTGRES_DB: ${POSTGRES_DB:-mydb}
      REPLICATION_PASSWORD: ${REPLICATION_PASSWORD:-replicator_password}
      PGDATA: /pgdata
    volumes:
      - postgres-primary-data:/pgdata
      - ./conf/postgresql-primary.conf:/etc/postgresql/postgresql.conf
      - ./init-scripts/01-init-primary.sh:/docker-entrypoint-initdb.d/01-init-primary.sh
    command: postgres -c config_file=/etc/postgresql/postgresql.conf
    ports:
      - "5432:5432"
    networks:
      - postgres-cluster
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U postgres"]
      interval: 10s
      timeout: 5s
      retries: 5
    restart: unless-stopped

  postgres-replica-1:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: postgres-replica-1
    hostname: postgres-replica-1
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD:-postgres_password}
      PRIMARY_HOST: postgres-primary
      PRIMARY_PORT: 5432
      REPLICATION_USER: replicator
```

```
    REPLICATION_PASSWORD: ${REPLICATION_PASSWORD:-replicator_password}
    REPLICA_SLOT: replica_slot_1
    PGDATA: /pgdata
  volumes:
    - postgres-replica-1-data:/pgdata
    - ./conf/postgresql-replica.conf:/etc/postgresql/postgresql.conf
    - ./init-scripts/init-replica.sh:/usr/local/bin/init-replica.sh
  command: bash -c "chmod +x /usr/local/bin/init-replica.sh && /usr/local/bin/init-replica.sh && postgres -c config_file=/etc
  ports:
    - "5433:5432"
  networks:
    - postgres-cluster
  depends_on:
    postgres-primary:
      condition: service_healthy
  healthcheck:
    test: ["CMD-SHELL", "pg_isready -U postgres"]
    interval: 10s
    timeout: 5s
    retries: 5
  restart: unless-stopped

postgres-replica-2:
  build:
    context: .
    dockerfile: Dockerfile
  container_name: postgres-replica-2
  hostname: postgres-replica-2
  environment:
    POSTGRES_USER: postgres
    POSTGRES_PASSWORD: ${POSTGRES_PASSWORD:-postgres_password}
    PRIMARY_HOST: postgres-primary
    PRIMARY_PORT: 5432
    REPLICATION_USER: replicator
    REPLICATION_PASSWORD: ${REPLICATION_PASSWORD:-replicator_password}
    REPLICA_SLOT: replica_slot_2
    PGDATA: /pgdata
  volumes:
    - postgres-replica-2-data:/pgdata
    - ./conf/postgresql-replica.conf:/etc/postgresql/postgresql.conf
    - ./init-scripts/init-replica.sh:/usr/local/bin/init-replica.sh
  command: bash -c "chmod +x /usr/local/bin/init-replica.sh && /usr/local/bin/init-replica.sh && postgres -c config_file=/etc
  ports:
    - "5434:5432"
```

```
    networks:
      - postgres-cluster
    depends_on:
      postgres-primary:
        condition: service_healthy
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U postgres"]
      interval: 10s
      timeout: 5s
      retries: 5
    restart: unless-stopped

networks:
  postgres-cluster:
    driver: bridge

volumes:
  postgres-primary-data:
  postgres-replica-1-data:
  postgres-replica-2-data:
```

---

7. .env

```
bash

# Database Configuration
POSTGRES_USER=postgres
POSTGRES_PASSWORD=change_me_secure_password
POSTGRES_DB=mydb

# Replication Configuration
REPLICATION_PASSWORD=change_me_replication_password
```

Quick Start

1. **Create directory structure**:

```
bash
```

```
mkdir -p postgres-cluster/{conf,init-scripts}
cd postgres-cluster
```

2. **Create all files** (copy content from above)
3. **Make scripts executable**:

```
bash

chmod +x init-scripts/*.sh
```

4. **Edit .env** with secure passwords
5. **Start cluster**:

```
bash

docker-compose up -d
```

6. **Verify replication**:

```
bash

docker exec -it postgres-primary psql -U postgres -c "SELECT * FROM pg_stat_replication;"
```

Connection Details

- **Primary (write)**: `localhost:5432`
- **Replica 1 (read)**: `localhost:5433`
- **Replica 2 (read)**: `localhost:5434`

Testing Extensions

Verify Extensions are Installed

First, check which extensions are available:

```
sql
```

```sql
-- List all available extensions
SELECT * FROM pg_available_extensions WHERE name LIKE '%vector%' OR name LIKE '%text%';

-- List installed extensions
SELECT extname, extversion FROM pg_extension;
```

If the extensions are not showing up, manually create them:

```sql
-- Create extensions
CREATE EXTENSION IF NOT EXISTS vector;
CREATE EXTENSION IF NOT EXISTS textsearch;
CREATE EXTENSION IF NOT EXISTS pg_stat_statements;

-- Verify they're installed
\dx
```

pgvector Example

```sql
-- Create a table with vector column
CREATE TABLE vectors (id serial PRIMARY KEY, embedding vector(3));
INSERT INTO vectors (embedding) VALUES ('[1,2,3]'), ('[4,5,6]');
SELECT * FROM vectors ORDER BY embedding <-> '[3,1,2]' LIMIT 1;
```

pg_textsearch Example

The pg_textsearch extension enhances PostgreSQL's full-text search capabilities with additional features.

```sql
```

```sql
-- Create a test table
CREATE TABLE documents (
    id SERIAL PRIMARY KEY,
    title TEXT,
    content TEXT
);

-- Insert sample data
INSERT INTO documents (title, content) VALUES
    ('PostgreSQL Tutorial', 'Learn about PostgreSQL database management and SQL queries'),
    ('Vector Search Guide', 'How to use pgvector for similarity search in databases'),
    ('Full Text Search', 'PostgreSQL provides powerful full-text search capabilities'),
    ('Database Performance', 'Optimizing PostgreSQL performance with indexes and queries');

-- Create a text search configuration (if using pg_textsearch features)
-- Note: Basic PostgreSQL text search works without pg_textsearch extension

-- Add a tsvector column for search
ALTER TABLE documents ADD COLUMN search_vector tsvector;

-- Update the search vector with content
UPDATE documents
SET search_vector = to_tsvector('english', coalesce(title, '') || ' ' || coalesce(content, ''));

-- Create a GIN index for faster searching
CREATE INDEX idx_documents_search ON documents USING GIN(search_vector);

-- Perform text searches
-- Simple search
SELECT id, title, content
FROM documents
WHERE search_vector @@ to_tsquery('english', 'postgresql');

-- Search with ranking
SELECT id, title,
       ts_rank(search_vector, query) AS rank
FROM documents,
     to_tsquery('english', 'postgresql & search') query
WHERE search_vector @@ query
ORDER BY rank DESC;

-- Search with highlighting
SELECT id, title,
```

```
    ts_headline('english', content, query) AS highlighted
FROM documents,
    to_tsquery('english', 'vector | search') query
WHERE search_vector @@ query;

-- Phrase search
SELECT id, title
FROM documents
WHERE search_vector @@ phraseto_tsquery('english', 'full text search');

-- Automatically update search_vector on insert/update using trigger
CREATE OR REPLACE FUNCTION documents_search_trigger() RETURNS trigger AS $
BEGIN
    NEW.search_vector := to_tsvector('english',
        coalesce(NEW.title, '') || ' ' || coalesce(NEW.content, '')
    );
    RETURN NEW;
END;
$ LANGUAGE plpgsql;

CREATE TRIGGER tsvector_update
BEFORE INSERT OR UPDATE ON documents
FOR EACH ROW EXECUTE FUNCTION documents_search_trigger();

-- Test the trigger
INSERT INTO documents (title, content)
VALUES ('New Document', 'Testing automatic search vector generation');

SELECT title, search_vector FROM documents WHERE title = 'New Document';
```

Advanced pg_textsearch Features

If pg_textsearch extension provides additional functions, test them:

```
sql
```

```
-- Check what functions pg_textsearch provides
SELECT proname, prosrc
FROM pg_proc
WHERE pronamespace = (
    SELECT oid FROM pg_namespace WHERE nspname = 'public'
)
AND proname LIKE '%text%';

-- Test any specific pg_textsearch functions
-- (The exact functions depend on what pg_textsearch provides)
```

Manual Extension Installation

If extensions aren't automatically loading, install them manually:

```
bash

# Connect to primary
docker exec -it postgres-primary psql -U postgres -d mydb

# Inside psql, run:
CREATE EXTENSION vector;
CREATE EXTENSION textsearch;
CREATE EXTENSION pg_stat_statements;
```

Check Extension Files

Verify extension files are present in the container:

```
bash

# Check if vector extension files exist
docker exec -it postgres-primary ls -la /usr/share/postgresql/18/extension/ | grep vector

# Check if textsearch extension files exist
docker exec -it postgres-primary ls -la /usr/share/postgresql/18/extension/ | grep text

# Should show files like:
# vector--0.8.0.sql
# vector.control
# textsearch--*.sql
# textsearch.control
```

Complete Test Script

Here's a complete test script you can run:

```sql
```

```sql
-- Test both extensions together
DROP TABLE IF EXISTS articles CASCADE;

CREATE TABLE articles (
    id SERIAL PRIMARY KEY,
    title TEXT,
    content TEXT,
    embedding vector(384),  -- Example: for sentence embeddings
    search_vector tsvector
);

-- Insert test data
INSERT INTO articles (title, content) VALUES
    ('AI and Machine Learning', 'Artificial intelligence is transforming technology'),
    ('Database Systems', 'PostgreSQL is a powerful relational database'),
    ('Search Technologies', 'Full-text search enables finding relevant content');

-- Update search vectors
UPDATE articles
SET search_vector = to_tsvector('english', title || ' ' || content);

-- Create indexes
CREATE INDEX ON articles USING GIN(search_vector);
CREATE INDEX ON articles USING hnsw (embedding vector_cosine_ops);

-- Test text search
SELECT title, ts_rank(search_vector, query) as rank
FROM articles, to_tsquery('english', 'database') query
WHERE search_vector @@ query
ORDER BY rank DESC;

-- Verify both extensions work
SELECT
    'vector' as extension_name,
    COUNT(*) as tables_using
FROM information_schema.columns
WHERE data_type = 'USER-DEFINED' AND udt_name = 'vector'
UNION ALL
SELECT
    'textsearch' as extension_name,
    COUNT(*) as tables_using
```

```
FROM information_schema.columns
WHERE data_type = 'tsvector';
```

ElasticSearch-Style JSON Document Search in PostgreSQL

PostgreSQL provides powerful JSON/JSONB support that can replicate most ElasticSearch functionality. Here's how to implement ElasticSearch-style search patterns:

1. JSON Storage and Indexing

sql

```
-- Create a table with JSONB (binary JSON, faster and indexable)
CREATE TABLE documents (
    id SERIAL PRIMARY KEY,
    created_at TIMESTAMP DEFAULT NOW(),
    doc JSONB NOT NULL,
    search_vector tsvector GENERATED ALWAYS AS (
        to_tsvector('english', doc::text)
    ) STORED
);

-- Create GIN indexes for fast JSON queries
CREATE INDEX idx_doc_gin ON documents USING GIN (doc);
CREATE INDEX idx_doc_jsonb_path ON documents USING GIN (doc jsonb_path_ops);
CREATE INDEX idx_doc_search ON documents USING GIN (search_vector);

-- Insert sample documents (ElasticSearch-style)
INSERT INTO documents (doc) VALUES
('{"title": "Introduction to PostgreSQL", "author": "John Doe", "tags": ["database", "sql"], "content": "PostgreSQL is a powerf
('{"title": "Advanced Indexing", "author": "Jane Smith", "tags": ["performance", "optimization"], "content": "Learn how to op
('{"title": "JSON in PostgreSQL", "author": "John Doe", "tags": ["json", "database"], "content": "Working with JSON docume
```

2. ElasticSearch Query Patterns in PostgreSQL

sql

```sql
-- Match Query (similar to ES match query)
-- Find documents where title contains "PostgreSQL"
SELECT id, doc->>'title' as title, doc->>'content' as content
FROM documents
WHERE doc->>'title' ILIKE '%PostgreSQL%';

-- Boolean Query (similar to ES bool query with must, should, must_not)
SELECT id, doc->>'title' as title
FROM documents
WHERE
    doc->>'author' = 'John Doe'  -- must
    AND doc @> '{"tags": ["database"]}'  -- must contain tag
    AND NOT doc @> '{"tags": ["deprecated"]}';  -- must_not

-- Range Query (similar to ES range query)
SELECT id, doc->>'title' as title, (doc->>'rating')::float as rating
FROM documents
WHERE (doc->>'rating')::float >= 4.5
    AND (doc->>'published')::date >= '2024-02-01';

-- Term Query (exact match, similar to ES term query)
SELECT id, doc->>'title' as title
FROM documents
WHERE doc @> '{"author": "John Doe"}';

-- Terms Query (match any of multiple values)
SELECT id, doc->>'title' as title
FROM documents
WHERE doc->'tags' ?| array['database', 'performance'];

-- Exists Query (check if field exists)
SELECT id, doc->>'title' as title
FROM documents
WHERE doc ? 'rating';

-- Wildcard/Pattern Matching
SELECT id, doc->>'title' as title
FROM documents
WHERE doc->>'author' SIMILAR TO '%(Doe|Smith)%';
```

3. Full-Text Search (similar to ES text analysis)

```sql
```

```sql
-- Full-text search with ranking (similar to ES _score)
SELECT
    id,
    doc->>'title' as title,
    ts_rank(search_vector, query) as score
FROM documents,
    to_tsquery('english', 'postgresql & database') as query
WHERE search_vector @@ query
ORDER BY score DESC;

-- Phrase search
SELECT id, doc->>'title' as title
FROM documents
WHERE search_vector @@ phraseto_tsquery('english', 'open source database');

-- Multi-field search (search across multiple JSON fields)
SELECT
    id,
    doc->>'title' as title,
    ts_rank(
        to_tsvector('english',
            coalesce(doc->>'title', '') || ' ' ||
            coalesce(doc->>'content', '') || ' ' ||
            coalesce(doc->>'author', '')
        ),
        query
    ) as score
FROM documents,
    to_tsquery('english', 'indexing & optimization') as query
WHERE to_tsvector('english',
        coalesce(doc->>'title', '') || ' ' ||
        coalesce(doc->>'content', '')
    ) @@ query
ORDER BY score DESC;
```

4. Aggregations (similar to ES aggregations)

```sql
```

```sql
-- Terms Aggregation (count by field value)
SELECT
    doc->>'author' as author,
    COUNT(*) as doc_count
FROM documents
GROUP BY doc->>'author'
ORDER BY doc_count DESC;

-- Nested aggregation (tags frequency)
SELECT
    jsonb_array_elements_text(doc->'tags') as tag,
    COUNT(*) as frequency
FROM documents
GROUP BY tag
ORDER BY frequency DESC;

-- Stats Aggregation (min, max, avg)
SELECT
    AVG((doc->>'rating')::float) as avg_rating,
    MIN((doc->>'rating')::float) as min_rating,
    MAX((doc->>'rating')::float) as max_rating,
    COUNT(*) as total_docs
FROM documents;

-- Date Histogram (group by time intervals)
SELECT
    DATE_TRUNC('month', (doc->>'published')::date) as month,
    COUNT(*) as doc_count
FROM documents
GROUP BY month
ORDER BY month;

-- Bucket Aggregation with filters
SELECT
    CASE
        WHEN (doc->>'rating')::float >= 4.5 THEN 'excellent'
        WHEN (doc->>'rating')::float >= 4.0 THEN 'good'
        ELSE 'average'
    END as rating_bucket,
    COUNT(*) as count
FROM documents
GROUP BY rating_bucket;
```

5. Advanced Features

sql

```sql
-- Highlighting (similar to ES highlight)
SELECT
    id,
    doc->>'title' as title,
    ts_headline(
        'english',
        doc->>'content',
        to_tsquery('english', 'database'),
        'StartSel=<mark>, StopSel=</mark>'
    ) as highlighted_content
FROM documents
WHERE search_vector @@ to_tsquery('english', 'database');

-- Nested Object Queries
-- First, insert a document with nested structure
INSERT INTO documents (doc) VALUES
('{"title": "Blog Post", "author": {"name": "John Doe", "email": "john@example.com"}, "comments": [{"user": "Jane", "text"

-- Query nested objects
SELECT id, doc->'author'->>'name' as author_name
FROM documents
WHERE doc->'author'->>'email' = 'john@example.com';

-- Query array elements
SELECT
    id,
    doc->>'title' as title,
    jsonb_array_elements(doc->'comments')->>'user' as commenter
FROM documents
WHERE doc ? 'comments';

-- More Like This (MLT) using vector similarity
-- This requires pgvector extension
ALTER TABLE documents ADD COLUMN embedding vector(384);

-- Then you can do similarity search
-- (Assuming you have embeddings generated)
SELECT id, doc->>'title' as title
FROM documents
ORDER BY embedding <-> (
    SELECT embedding FROM documents WHERE id = 1
```

```
)
LIMIT 5;
```

6. Combined Search (Full-Text + Filters + Aggregations)

sql

```sql
-- ElasticSearch-style query with filtering, full-text search, and aggregations
WITH search_results AS (
    SELECT
        id,
        doc,
        ts_rank(search_vector, query) as score
    FROM documents,
        to_tsquery('english', 'database | indexing') as query
    WHERE
        search_vector @@ query
        AND (doc->>'rating')::float >= 4.0
        AND doc @> '{"tags": ["database"]}'
)
SELECT
    -- Results
    id,
    doc->>'title' as title,
    doc->>'author' as author,
    (doc->>'rating')::float as rating,
    score,
    -- Aggregations
    (SELECT COUNT(*) FROM search_results) as total_hits,
    (SELECT AVG((doc->>'rating')::float) FROM search_results) as avg_rating
FROM search_results
ORDER BY score DESC
LIMIT 10;
```

7. Performance Optimization Tips

sql

```sql
-- Partial indexes for frequently queried subsets
CREATE INDEX idx_high_rated ON documents
USING GIN (doc)
WHERE (doc->>'rating')::float >= 4.5;

-- Expression indexes for computed values
CREATE INDEX idx_author_lower ON documents
((lower(doc->>'author')));

-- Composite indexes for common query patterns
CREATE INDEX idx_author_rating ON documents
((doc->>'author'), ((doc->>'rating')::float));

-- Statistics for better query planning
ANALYZE documents;

-- Check index usage
SELECT
    schemaname,
    tablename,
    indexname,
    idx_scan,
    idx_tup_read,
    idx_tup_fetch
FROM pg_stat_user_indexes
WHERE tablename = 'documents';
```

8. Bulk Operations (similar to ES _bulk API)

```sql
```

```sql
-- Bulk insert with COPY or multi-row INSERT
INSERT INTO documents (doc) VALUES
('{"title": "Doc 1", "content": "Content 1"}'),
('{"title": "Doc 2", "content": "Content 2"}'),
('{"title": "Doc 3", "content": "Content 3"}');

-- Bulk update
UPDATE documents
SET doc = jsonb_set(doc, '{updated_at}', to_jsonb(NOW()))
WHERE (doc->>'rating')::float < 4.0;

-- Bulk delete
DELETE FROM documents
WHERE (doc->>'published')::date < '2024-01-01';
```

Key Differences from ElasticSearch

Advantages of PostgreSQL:

- ACID transactions
- Strong consistency
- Complex joins with relational data
- No separate system to maintain
- Better for structured data with some JSON

When to use ElasticSearch instead:

- Extremely high-volume logging/time-series data
- Need for distributed search across clusters
- Complex text analysis with multiple languages
- Real-time analytics at massive scale
- Fuzzy matching and typo tolerance are critical

Hybrid Approach:

- Use PostgreSQL as primary database
- Sync to ElasticSearch for advanced search features
- Use pg_textsearch + pgvector for semantic search

- Keep transactional data in PostgreSQL