POSTGRESQL, BASE DE DONNÉES VECTORIELLES AVEC PGVECTOR

CAPITOLE DU LIBRE 2024

Philippe VIEGAS

LOXODATA

QUI SUIS-JE ?

Philippe Viegas

- PostgreSQL depuis 2008
- Rejoint LOXODATA en 2022
- Consultant et formateur PostgreSQL

LOXODATA

Entreprise disposant de 3 piliers d'expertises

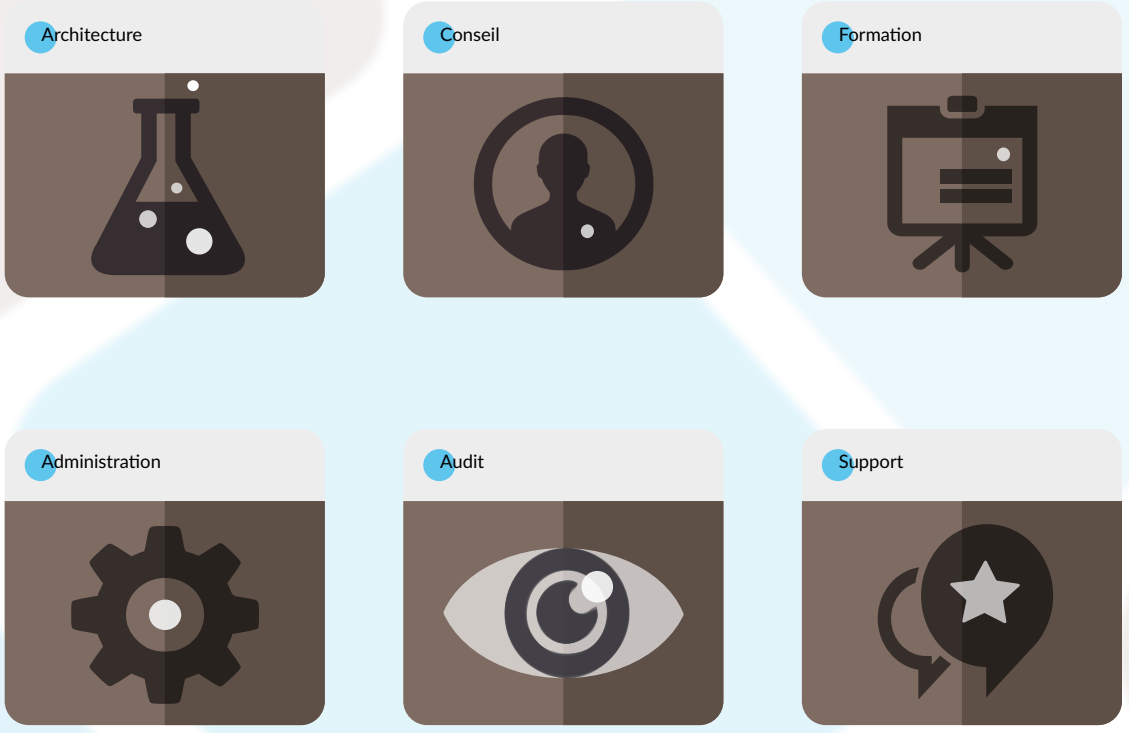
PostgreSQL

DevOps

Cloud

LOXODATA

Une large palette de services



POSTGRESQL

PRÉSENTATION PROJET

- Origine à l'université de Berkeley, initié par Michael Stonebraker
- Licence PostgreSQL, type BSD
- Projet communautaire porté par le PGDG

VERSIONS

- Versions majeurs maintenues pour cinq ans
- Versions mineurs tous les trimestres
- Versions actuelles : 17 à 13

BREF HISTORIQUE

- Postgres, Postgres95, PostgreSQL en 1996 en version 6.0
- Portage sur Windows en 8.0
- Performances et nouvelles fonctionnalités en 8.3
- Réplication physique en 9.0
- Parallélisation en 9.6
- Réplication logique et partionnement en 10
- Amélioration des performances et facilité d'administration

FONCTIONNALITÉS

- MVCC
- ACID
- Respect standard SQL
- Moteur personnalisable
- Nombreuses API clientes
- Extensibilité
- Sécurité
- Feature matrix

COMMUNAUTÉ

- Core Team
- Contributeurs
- CommitFest
- Documentation
- Liste de diffusion
- Wiki PostgreSQL
- Conférences

POSTGRESQL ET EXTENSIONS

EXTENSIONS

- Extensibilité éprouvée
- Un écosystème très large
- Contrib
- PGXN

CATALOGUE

- Vue pg_available_extensions
- Vue pg_available_extension_versions
- Vue pg_extension ou commande `\dx`

INSTALLATION

- Installer les binaires (si besoin)

```
sudo apt install postgresql-17-pgvector
```

- CREATE EXTENSION

```
CREATE EXTENSION vector;
```

- ALTER EXTENSION

```
ALTER EXTENSION vector UPDATE TO '0.8';
```

VECTEURS

VECTEUR ?

- Représentation de données hétérogènes (image, audio, texte)
- Tableau de données numériques (réels à virgule flottante)

```
[-0.07, -0.53, -0.02, …, -0.61, 0.59]
```

- Espace vectoriel de dimension finie (n dimensions)
- Produits par des modèles de langages (LLM)

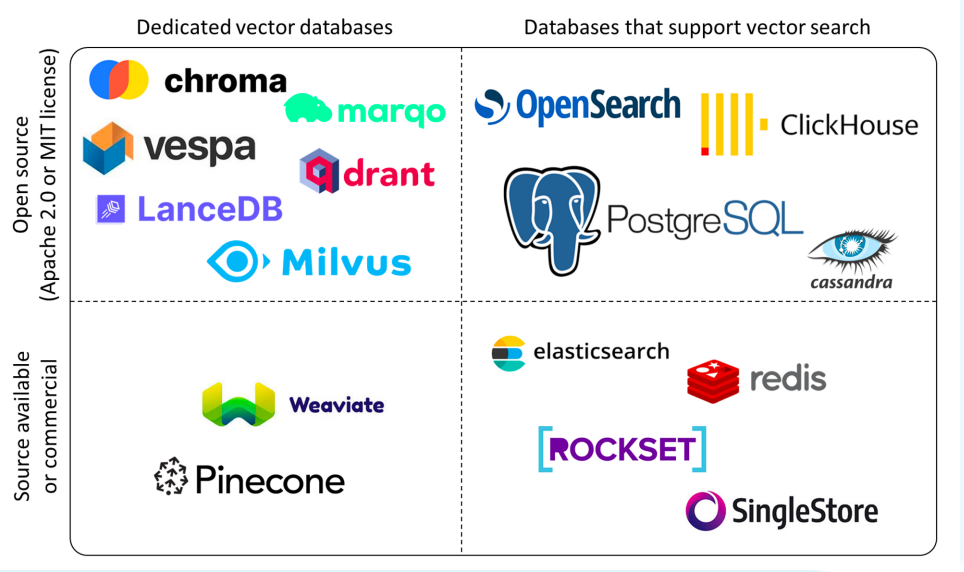
BASES DE DONNÉES VECTORIELLES

- Stockage et recherche de vecteurs (embeddings)
- Indexation performante
- Recherche de similarité et proximité efficace
- Calcul de distance (Nearest Neighbor Search)

FONCTIONNALITÉS ATTENDUES

- Précision
- Transactionnel
- Scalabilité
- Sécurité
- Haute disponibilité
- Recherche hybride

MARCHÉ



© Yingjun Wu

CAS D'USAGE

- Système de recommendation
- Recherche d'images
- Traitement du langage naturel (classification de texte, analyse de sentiment)
- Détection d'anomalies (finance, sécurité, maintenance)
- Bioinformatique
- Chatbots (RAG)

CHALLENGES

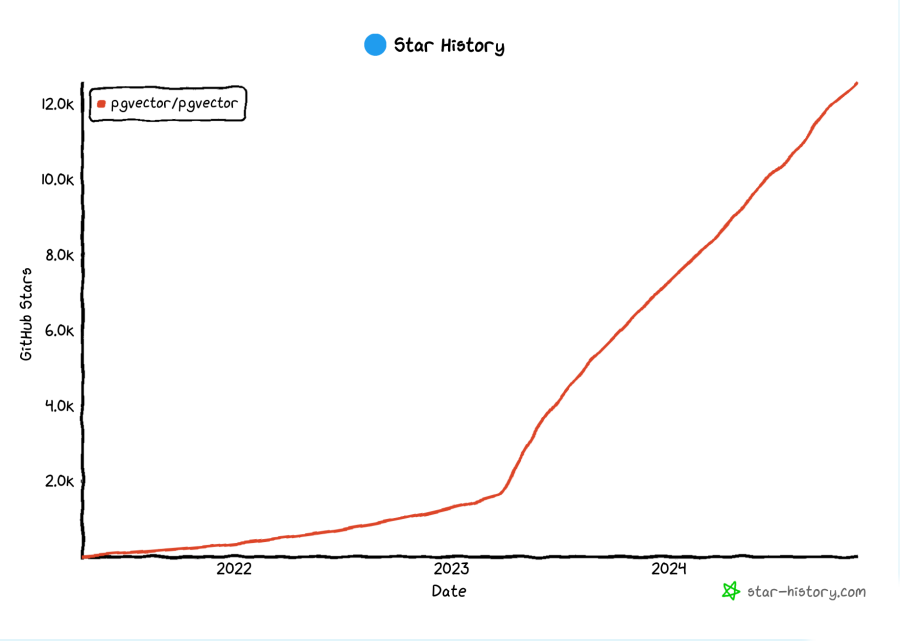
- Toujours plus de dimensions
- Précision versus approximation (ANN)
- Stockage et mémoire

PGVECTOR

EXTENSION PGVECTOR

- pgvector
- Andrew Kane
- Version 0.1.0 (21 avril 2021)
- Version courante 0.8.0 (30 octobre 2024)

POPULARITÉ



OBJECTIFS

- Recherche et stockage de données de type vecteurs
- Recherche exact et approximative (ANN)
- Type vecteur : précision simple, demi précision, binaire et sparse
- Opérations de distance : L2 distance, inner product, cosine distance, L1 distance, Hamming distance et Jaccard distance
- Nombreux langages clients
- Fonctionnalités éprouvées de PostgreSQL

DÉMARRER

- Stocker

```
CREATE TABLE documents (
    id serial PRIMARY KEY,
    embedding vector(3)
);

INSERT INTO documents (embedding) VALUES ('[1,2,3]'), ('[4,5,6]');
```

- Rechercher

```
SELECT * FROM documents ORDER BY embedding <-> '[3,1,2]' LIMIT 5;
```

OPÉRATEURS DE DISTANCE

- <-> distance L2 ou Euclidienne (vector_l2_ops)
- <#> produit scalaire (vector_ip_ops)
- <=> distance cosinus (vector_cosine_ops)
- <+> distance L1 ou Manhattan (vector_l1_ops)
- <~> distance Hamming (bit_hamming_ops)
- <%> distance Jaccard (bit_jaccard_ops)

OPÉRATEURS DE DISTANCE

- Similarité cosinus (1 - distance cosinus)

```
SELECT 1 - (embedding <=> '[3,1,2]') AS cosine_similarity FROM documents;
```

- Distance Euclidienne

```
SELECT * FROM documents WHERE embedding <-> '[3,1,2]' < 5;
```

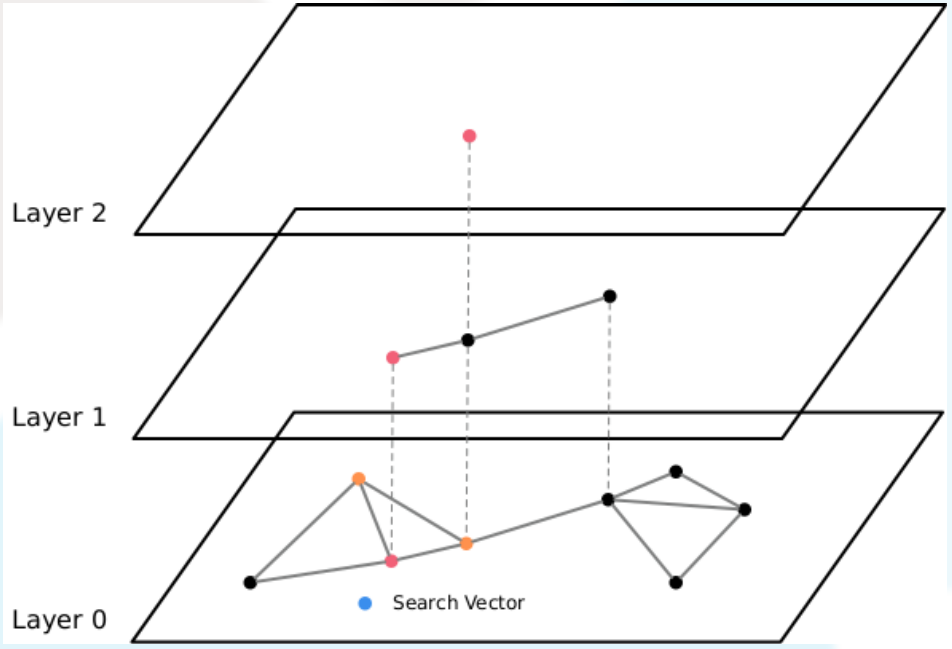
INDEXATION

- Recherche exacte (KNN)
  - Recherche séquentielle
- Recherche approximative (ANN)
  - `IVFFlat`
  - `HNSW`
- Compromis entre performance et rappel

INDEX HNSW

- Basé sur un graphe multi-couches
- Chaque couche étant plus ou moins dense (voisinage)
- Pas besoin de données (insertion itérative)
- Durée d'insertion augmente avec le nombre de vecteurs présents

INDEX HNSW



© Github @ skyzh

HNSW - PARAMÈTRES

- Construction :
  - `m` (défaut à 16) nombre maximum de connexions entre vecteurs (par couche)
  - `ef_construction` (défaut à 64) nombre de vecteurs candidats par voisinage pour construction

```
CREATE INDEX ON documents USING hnsw (embedding vector_cosine_ops) WITH (m = 16, ef_construct
```

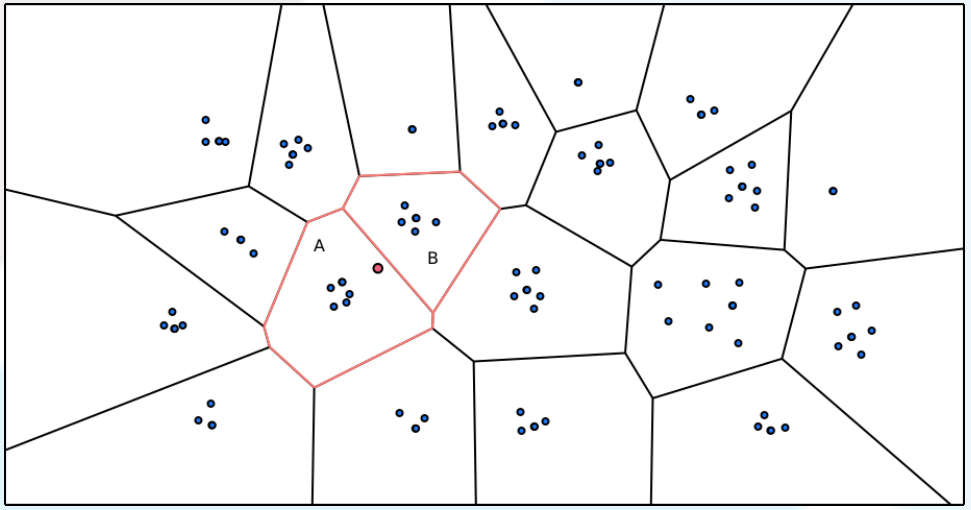
- Recherche :
  - `ef_search` (défaut à 40) nombre de vecteurs candidats par voisinage pour requête

```
SET hnsw.ef_search = 100;
SELECT * FROM documents ORDER BY embedding <=> '[3,1,2]' LIMIT 10;
```

INDEX IVFFLAT

- Création de clusters (listes) de vecteurs
- Basé sur algorithme K-means (recherche de centroïdes)
- Données pré-populées sur la table avant construction
- Durée d'insertion liée au nombre de listes

INDEX IVFFLAT



© Github @ skyzh

IVFFLAT - PARAMÈTRES

- Construction
  - `lists` nombre de cluster (liste)

```
CREATE INDEX ON documents
USING ivfflat (embedding vector_cosine_ops)
WITH (lists = 100);
```

- Recherche
  - `probes` (1 par défaut) nombre de liste dans lesquelles chercher

```
SET ivfflat_probes = 2;
SELECT * FROM documents ORDER BY embedding <=> '[3,1,2]' LIMIT 3;
```

FILTRES

- Clause `WHERE`
- Index pour recherche exacte (B-tree, hash, GiST, SP-GiST, GIN et BRIN)
- Index partiels, multicolonnes ou partitionnement

```
CREATE INDEX ON documents (isbn, category_id);
```

SCAN D'INDEX ITÉRATIF

- version 0.8.0 (par défaut à `off`)
- Scan de l'index approximatif
- Filtres appliqués après le scan d'index
- Si résultats insuffisants, on continue le scan de l'index

```
SET hnsw.iterative_scan = strict_order|relaxed_order;
SET ivfflat.iterative_scan = relaxed_order;
```

- Configuration du nombre maximum de résultats scannés

```
SET hnsw.max_scan_tuples = 20000; (par défaut)
SET hnsw.scan_mem_multiplier = 2; (x `work_mem`)
SET ivfflat.max_probes = 100;
```

QUANTIFICATION (QUANTIZATION)

- Type vecteur par défaut : 32 bits floating point (4 octets)
- Optimisation du stockage et requêtage de vecteurs
- Réduire la taille d'index (disque et mémoire)
- Type `halfvec` (16 bit), `bit` (1 bit)

```
CREATE INDEX ON documents USING hnsw ((embedding::halfvec(1536)) halfvec_l2_ops);
```

```
CREATE INDEX ON documents
USING hnsw ((binary_quantize(embedding)::bit(3072)) bit_hamming_ops);

SELECT id FROM documents
ORDER BY binary_quantize(embedding)::bit(3072) <~> binary_quantize($1)
LIMIT 10;
```

CONCLUSION

EN RÉSUMÉ

- Projet très actif grâce à la communauté
- Nombreux challenges encore à relever

DES QUESTIONS ?

LOXODATA
p.viegas@loxodata.com