FR

Guide Pratique

OpenClaw + Bases de donnees
PostgreSQL, RAG et recherche semantique

Integration base de donnees pour agents IA en production

Mars 2026 — Serie OpenClaw B10/10

Sommaire

Les fichiers plats ont leur utilite. Mais quand un reseau d'agents doit lire, ecrire et raisonner sur des donnees structurees a l'echelle — les bases de donnees changent la donne. Ce guide clos la serie technique OpenClaw sur une note concrete : PostgreSQL, RAG et recherche semantique.

1. Au-dela des fichiers : pourquoi connecter a une vraie BDD

Le workspace Git couvre 80% des scenarios d'automatisation. Mais certaines situations necessitent plus :

• Volumes importants — 50 000 transactions ne se lisent pas en Markdown

• Requetes dynamiques — filtrer par date, statut, revenu : SQL > grep

• Concurrence — plusieurs agents ecrivant dans le meme fichier = race conditions

• Recherche semantique — par sens, pas par mot-cle exact

■ *Gartner 2025 : 61% des deployements agents IA en production ont besoin d'une couche persistance structuree dans les 6 premiers mois.*

2. PostgreSQL + OpenClaw : connexion directe

L'integration la plus directe : un script Python dans le workspace. L'agent l'execute via son skill exec, le script se connecte via psycopg2, retourne les resultats dans un fichier JSON.

```
conn = psycopg2.connect(
host=os.environ['PG_HOST'],
dbname=os.environ['PG_DB'],
user=os.environ['PG_USER'],
password=os.environ['PG_PASS']
)
cur.execute("SELECT client, SUM(hours) FROM timesheets
WHERE month=%s GROUP BY client", ('2026-03',))
```

Regle absolue : les credentials ne transitent jamais dans le workspace Git. Variables d'environnement injectees par le vault uniquement.

3. Cas d'usage concrets

Timesheet automatique

L'agent LEDGER execute chaque vendredi un script qui agregge les timesheets depuis PostgreSQL et genere un rapport Markdown. Zero intervention humaine.

CRM mis a jour par un agent

Apres chaque appel client logue en messagerie, l'agent NEXUS extrait les infos cles et met a jour la ligne correspondante dans la table contacts. Le CRM se tient a jour en temps reel.

Logs structures d'operations

Chaque action significative des agents est loggee dans agent_logs (timestamp, agent source, type, statut). Audit complet — impossible a faire de facon fiable avec des fichiers texte.

4. RAG : pourquoi ca change tout

Un LLM a une fenetre de contexte limitee. Impossible d'injecter 10 000 pages. Le RAG resout ca : plutot que tout injecter, on recupere seulement les passages les plus pertinents a la question posee.

Architecture simple :

```
1. Indexation (une fois)
Documents -> Embeddings (vecteurs 1536 dims) -> pgvector

2. Requete (chaque question)
Question -> Embedding -> Similarite -> Top-K passages -> LLM
```

> *“La pertinence est calculee par similarite semantique — proximite de sens dans l'espace vectoriel, pas correspondance exacte de mots-cles.”*

5. Implementer un RAG basique

Generer des embeddings

```
def get_embedding(text: str) -> list[float]:
response = client.embeddings.create(
model='text-embedding-3-small',
input=text
)
return response.data[0].embedding
```

pgvector (production)

```
CREATE EXTENSION IF NOT EXISTS vector;
CREATE TABLE documents (
id SERIAL PRIMARY KEY,
content TEXT,
embedding vector(1536)
);
CREATE INDEX ON documents USING hnsw (embedding vector_cosine_ops);
```

Requete de similarite

```
SELECT content, 1-(embedding <=> %s::vector) AS score
FROM documents ORDER BY embedding <=> %s::vector LIMIT 5;
```

■ *Pour les volumes < 10 000 docs : JSON + cosine en Python suffisent. Pour la production : pgvector.*

6. Recherche semantique sur vos donnees

Cas d'usage BOTUM :

• Retrouver des decisions passees : 'quelle etait la decision sur le backup ?' -> passage exact dans les notes de reunion

• Trouver les billets similaires avant publication

• Chercher dans un CRM : 'quels clients ont mentionne des problemes de securite ?'

• Knowledge base interne : les agents trouvent la procedure standard applicable

7. Limites et realite terrain

Qualite des embeddings

Un corpus de notes cryptiques produit des embeddings inutiles. La qualite du corpus determine la qualite des resultats.

Cout

text-embedding-3-small : $0.02/million tokens. 10 000 docs x 500 tokens = ~$0.10 pour l'indexation initiale.

Latence

200-500ms par requete (API embedding + recherche vectorielle). Negligeable pour un agent interactif.

Quand le RAG est inutile

Donnees < 50 000 tokens : injecter directement dans le contexte. Claude 3.5 gere 200K tokens.

8. Architecture recommandee

Situation	Solution	Pourquoi
Donnees < 50K tokens	Injection directe	Zero latence
Donnees structurees	PostgreSQL psycopg2	SQL > grep
< 10K docs, proto	JSON + cosine Python	Simple, 0 dep.
10K-1M docs, Postgres	pgvector	Naturel, performant
> 1M docs	Qdrant/Weaviate	Vector stores dedies
Logs operations	PostgreSQL agent_logs	Audit, dashboards

9. Checklist integration DB

■ Credentials via vault uniquement — jamais dans le workspace Git

■ Utilisateur PostgreSQL dedie — droits minimaux (SELECT/INSERT sur tables necessaires)

■ Pool de connexions — pgBouncer ou pool applicatif en production

■ Timeout sur les requetes — statement_timeout obligatoire

■ Requetes parametrees uniquement — jamais de f-string dans un SELECT

■ Resultats ecrits dans data/ — l'agent lit un fichier, pas la connexion DB

■ Index sur les colonnes frequentes — eviter les full scans

■ Seuil de similarite RAG — filtrer avec score > 0.65-0.7

■ Reindexation en batch — pas en temps reel a chaque ecriture

■ Tests de regression — verifier les requetes de reference apres chaque modif du corpus

10. Conclusion de la serie

Ce dixieme billet marque la fin de la serie technique OpenClaw. En dix episodes, BOTUM a documente un deploiement reel : de l'installation jusqu'a l'integration base de donnees et la recherche semantique.

B1-B2 : concept, installation. B3-B4 : securite, vault. B5 : reseau d'agents. B6-B7 : choix LLM. B8-B9 : automatisation, memoire. B10 : persistance structuree.

La serie suivante explorera les dimensions business : ROI, PME vs Enterprise, edge. Articles sur botum.ca/blog.

Article complet : blog.botum.ca/openclaw-bases-donnees-postgresql-rag-recherche-semantique

Site web : www.botum.ca • contact@botum.ca