GESTION INCIDENTS DBA POSTGRESQL

PostgreSQL

IDEO·LAB

Ideo-Lab

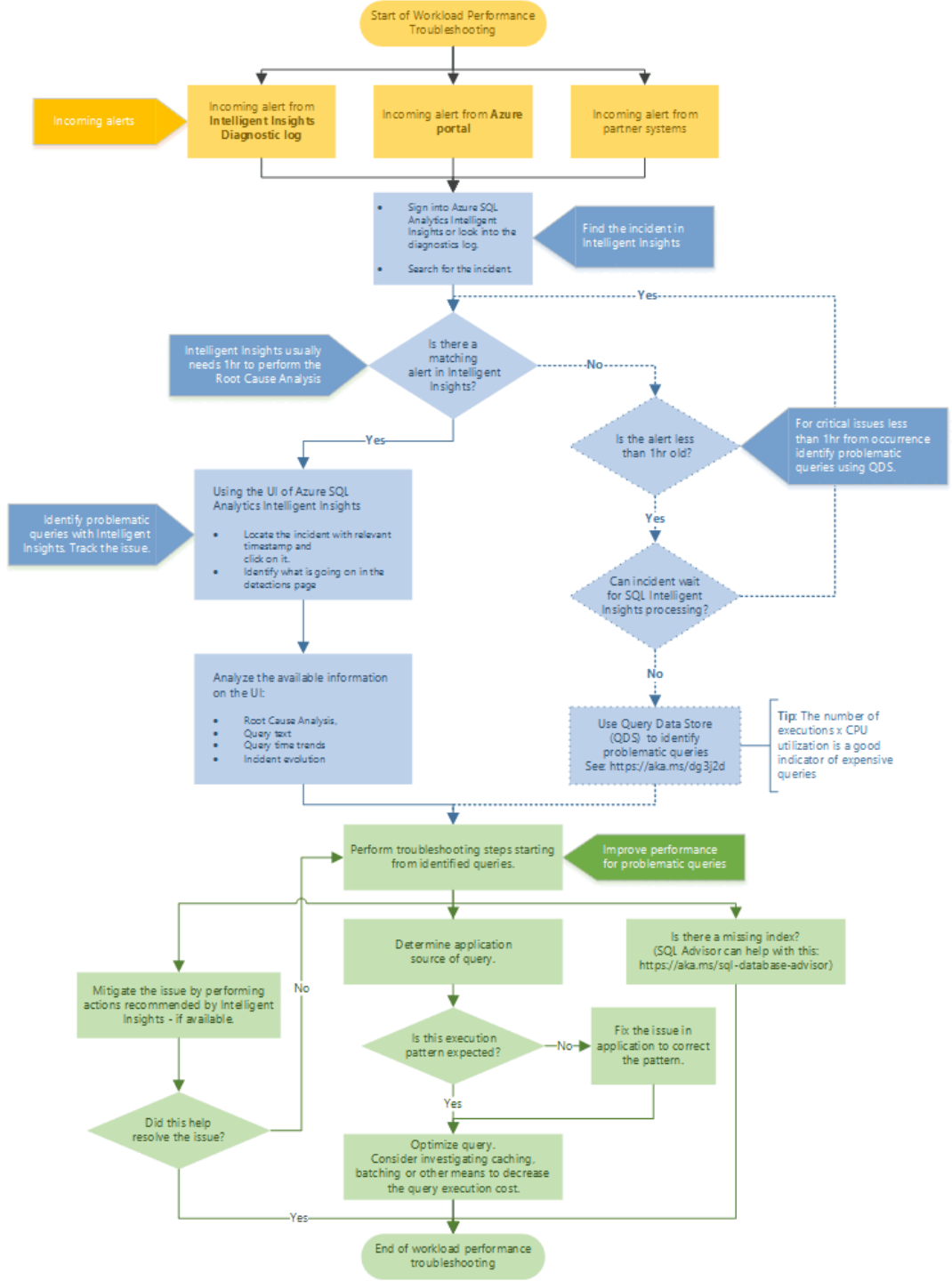
Feb 2026

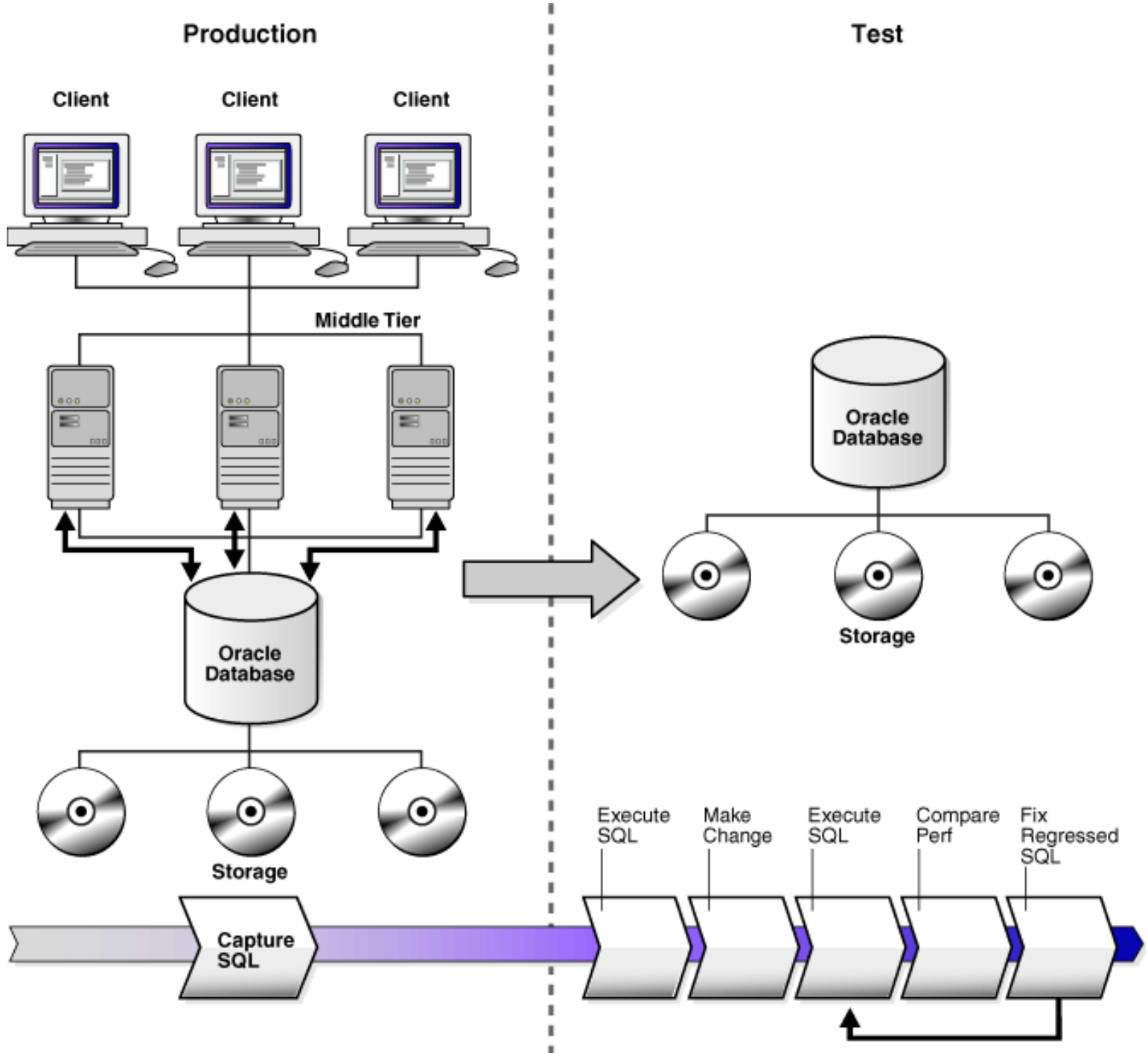
AGENDA GESTION INCIDENTS

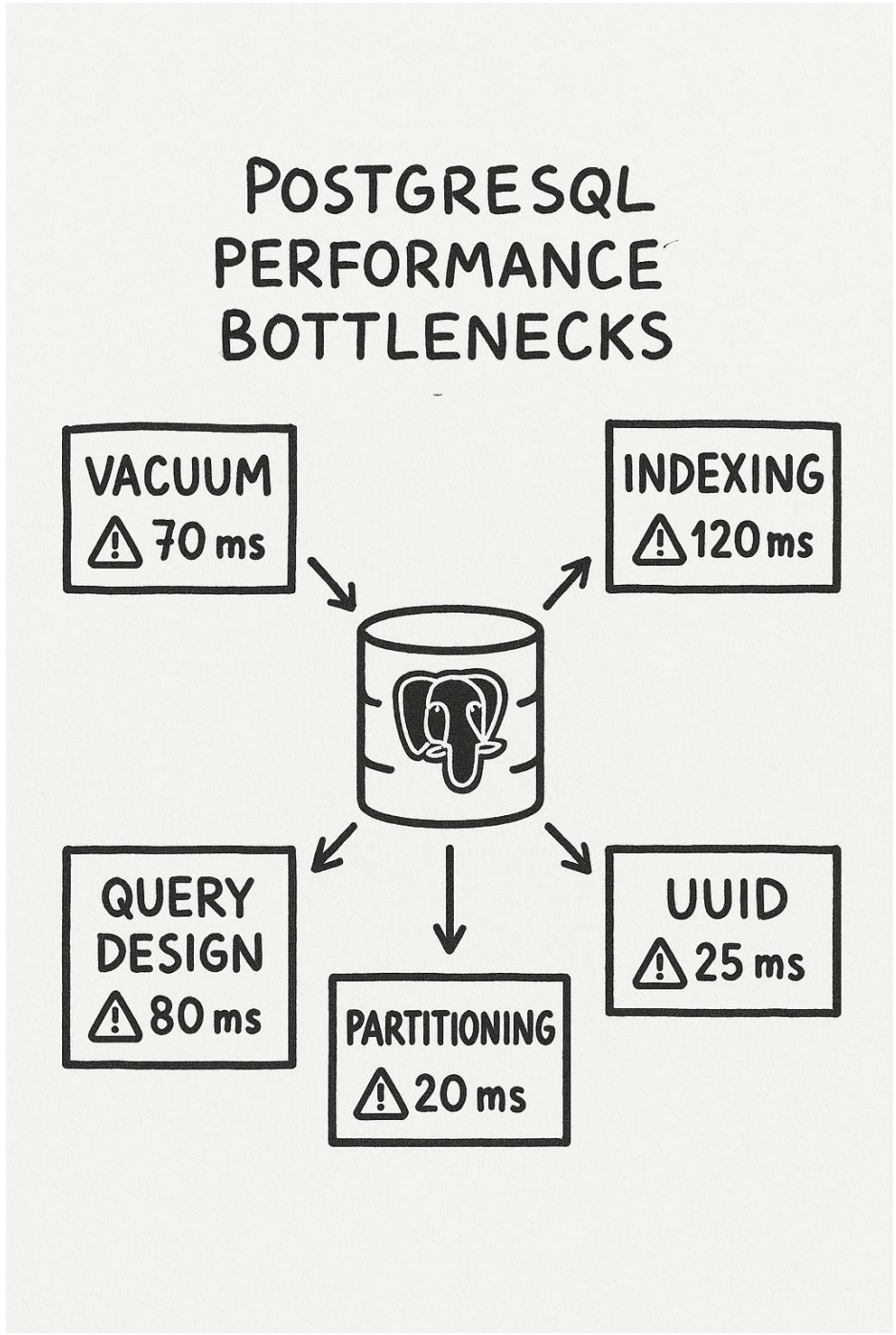
- LA RÈGLE D'OR (à dire presque mot pour mot)
- MÉTHODOLOGIE DBA – APPROCHE STRUCTURÉE (Oracle & PostgreSQL)
  - PHASE 0 — Sécurité & contexte (réflexe pro)
  - PHASE 1 — Qualification du problème (la plus importante)
  - PHASE 2 — Observation factuelle (pas d'intuition)
  - PHASE 3 — SQL d'abord, moteur ensuite
  - PHASE 4 — Concurrence & verrous (souvent oublié)
  - PHASE 5 — Configuration & architecture (en dernier)
  - PHASE 6 — Action graduée & mesurée
  - PHASE 7 — Capitalisation (niveau expert)

La règle d’Or

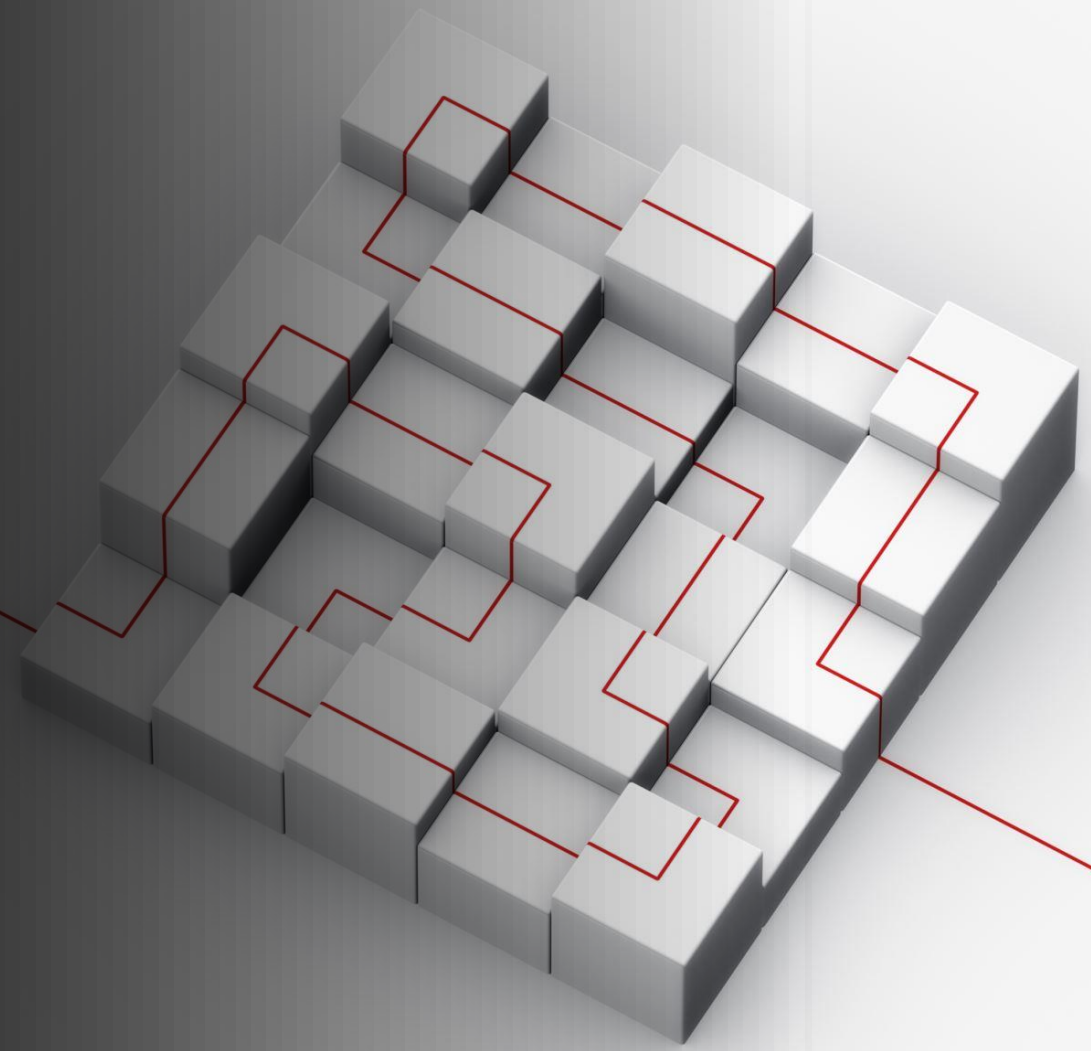
- ***« Face à un problème de performance, je commence toujours par qualifier le problème avant d’essayer de le résoudre. Une mauvaise qualification mène presque toujours à un mauvais diagnostic. »***


Start of Workload Performance Troubleshooting
Incoming alerts
Incoming alert from Intelligent Insights Diagnostic log
Incoming alert from Azure portal
Incoming alert from partner systems
Sign into Azure SQL Analytics Intelligent Insights or look into the diagnostics log.
Search for the incident.
Find the incident in Intelligent Insights
Yes
Intelligent Insights usually needs 1hr to perform the Root Cause Analysis
Is there a matching alert in Intelligent Insights?
No
Is the alert less than 1hr old?
For critical issues less than 1hr from occurrence identify problematic queries using QDS.
Yes
Identify problematic queries with Intelligent Insights. Track the issue.
Using the UI of Azure SQL Analytics Intelligent Insights
Locate the incident with relevant timestamp and click on it.
Identify what is going on in the detections page
Can incident wait for SQL Intelligent Insights processing?
Yes
No
Analyze the available information on the UI:
Root Cause Analysis,
Query text
Query time trends
Incident evolution
Use Query Data Store (QDS) to identify problematic queries
See https://aka.ms/dg3j2d
Tip: The number of executions x CPU utilization is a good indicator of expensive queries
Perform troubleshooting steps starting from identified queries.
Improve performance for problematic queries
Determine application source of query.
Is there a missing index? (SQL Advisor can help with this: https://aka.ms/sql-database-advisor)
Mitigate the issue by performing actions recommended by Intelligent Insights - if available.
No
Is this execution pattern expected?
No
Fix the issue in application to correct the pattern.
Yes
Did this help resolve the issue?
Optimize query. Consider investigating caching, batching or other means to decrease the query execution cost.
Yes
End of workload performance troubleshooting


Production
Test
Client
Client
Client
Middle Tier
Oracle Database
Storage
Oracle Database
Storage
Capture SQL
Execute SQL
Make Change
Execute SQL
Compare Perf
Fix Regressed SQL


POSTGRESQL
PERFORMANCE
BOTTLENECKS
VACUUM
70 ms
INDEXING
120 ms
QUERY
DESIGN
80 ms
PARTITIONING
20 ms
UUID
25 ms

MÉTHODOLOGIE DBA – APPROCHE STRUCTURÉE



IDEO·LAB

PHASE 0 — Sécurité & contexte (réflexe pro)

🎯 *Avant toute action technique*

- Suis-je **en production** ?
- Y a-t-il un **risque business immédiat** ?
- Ai-je le **droit d'agir** (redémarrage, kill session, paramètre) ?
- Le problème est-il :
  - **nouveau** ?
  - **récurrent** ?
  - **lié à un changement récent** ?

PHASE 1 — Qualification du problème

1. Nature du problème

2. Ressource impactée (le pivot du diagnostic)

PHASE 1 — Qualification du problème (la plus importante)

1 Nature du problème

tions simples :

Je commence toujours par répondre à ces questions simples :

- Qui se plaint ?
  - un utilisateur ?
  - une application ?
  - un batch ?
- Depuis quand ?

- C'est lent ou bloqué ?
- Depuis quand ?
- Global ou lou localisé ?

- Qui se plaint ?
- Depuis quand ?
- C'est lent ou bloqué ?

PHASE 1 — Qualification du problème

1 Nature du problème

Je commence toujours par répondre à ces questions simples :

- Qui se plaint ?
  - un utilisateur ?
  - une application ?
  - un batch ?
- Depuis quand ?
- C'est lent ou bloqué ?
- Global ou localisé ?
  - toute la base ?
  - un schéma ?
  - une requête ?

PHASE 1 — Qualification du problème

2 Ressource impactée (le pivot du diagnostic)

Je cherche **la ressource limitante dominante** :

Ressource	Symptômes typiques
CPU	load élevé, temps CPU dominant
I/O	latence disque, wait I/O
Mémoire	swap, cache inefficace
Lock / Concurrency	sessions bloquées
Réseau	lenteurs diffuses
SQL mal écrit	explosion logique

PHASE 2 — Observation factuelle (pas d'intuition)

🎯 *Je collecte des faits, pas des opinions.*

- Sessions actives
- Temps d'attente dominants
- Top requêtes consommatrices
- Évolution temporelle (pics, dérives)

Le temps, pas juste le CPU ou l'I/O.

« Je cherche toujours ce qui consomme du temps, pas ce qui consomme des ressources. »

PHASE 2 — Observation factuelle (pas d'intuition)

🎯 Je collecte des faits, pas des opinions.

PHASE 3 — SQL d'abord, moteur ensuite

À ce stade, je travaille requête par requête, pas globalement.

Je me pose systématiquement :

- Est-ce une **nouvelle requête** ?
- Un **plan d'exécution a-t-il changé** ?
- Une **cardinalité mal estimée** ?
- Index absents / inutiles / mal utilisés ?

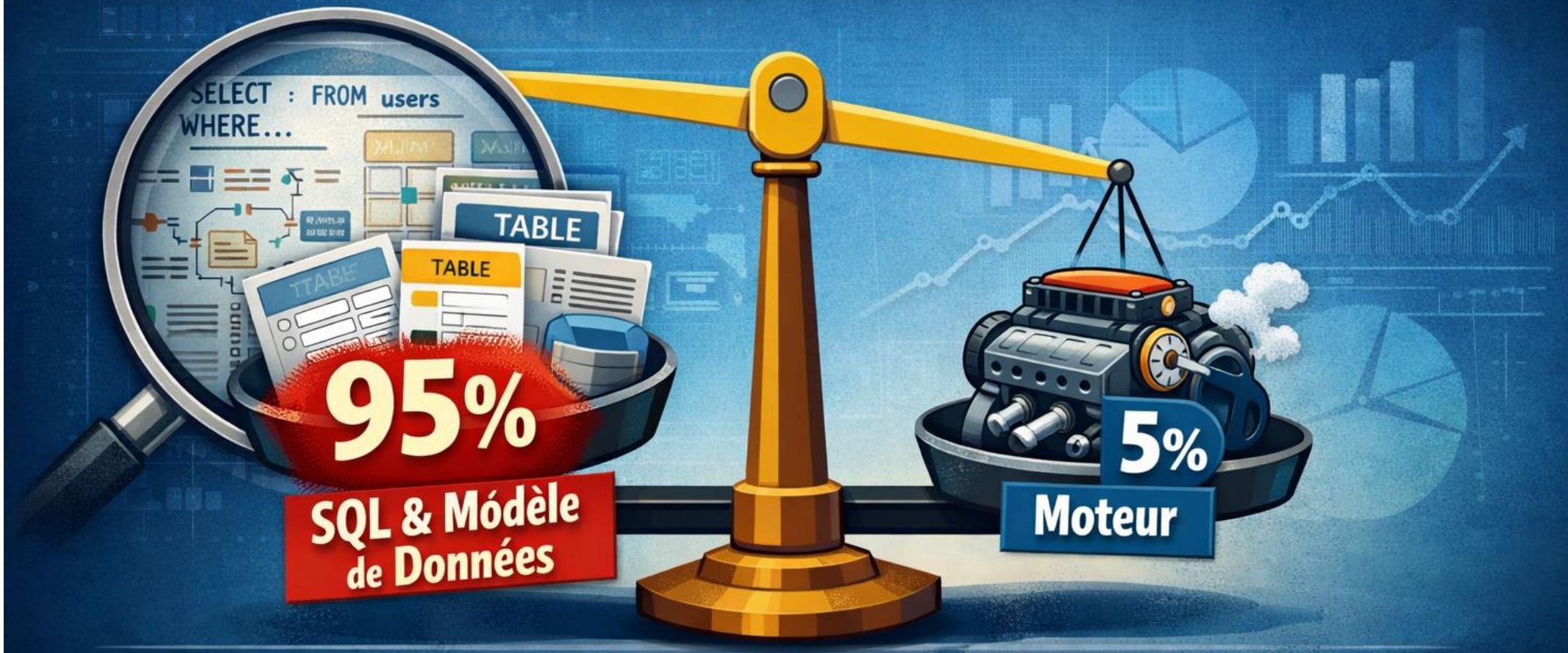
👉 À ce stade, je travaille **requête par requête**, pas globalement.

Principe fondamental

> "95% des problèmes de performance viennent du SQL ou du modèle de données, pas du moteur."

PHASE 3 – SQL d'abord, moteur ensuite

Principe fondomental



95% des problèmes de performance viennent du SQL ou du modèle de données, pas du moteur.

PHASE 4 — Concurrence & verrous

« Une base peut être lente sans être saturée. ».

Je vérifie ensuite :

- blocages
- verrous longs
- transactions ouvertes trop longtemps
- contention sur objets chauds

👉 Très vrai sur Oracle comme PostgreSQL.

Concurrence & verrous

Je cherche la **ressource limitante dominante :**

PHASE 5 — Configuration & architecture

Seulement **après** avoir écarté le SQL :

- Mémoire adaptée à la charge ?
- Paramètres cohérents ?
- I/O adaptés (latence, parallélisme) ?
- Architecture compatible avec l'usage ?

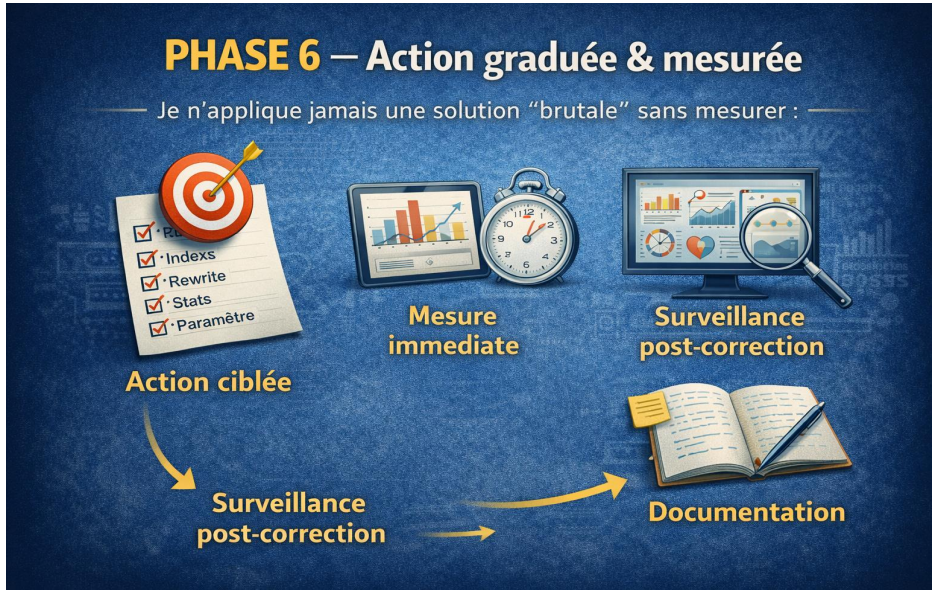
PHASE 5 – Configuration & architecture

PHASE 6 — Action graduée & mesurée

« Toute correction doit être mesurable et réversible «

Je n'applique jamais une solution "brutale" sans mesurer :

1. Action ciblée (index, rewrite, stats, paramètre)
2. Mesure immédiate
3. Surveillance post-correction
4. Documentation



PHASE 6 — Action graduée & mesurée

Je n'applique jamais une solution "brutale" sans mesurer :

Action ciblée

Mesure immediate

Surveillance post-correction

Documentation

Surveillance post-correction

PHASE 7 — Capitalisation (niveau expert)

Après l'incident :

- RCA (Root Cause Analysis)
- Détection précoce
- Recommandations dev
- Prévention (alertes, seuils)

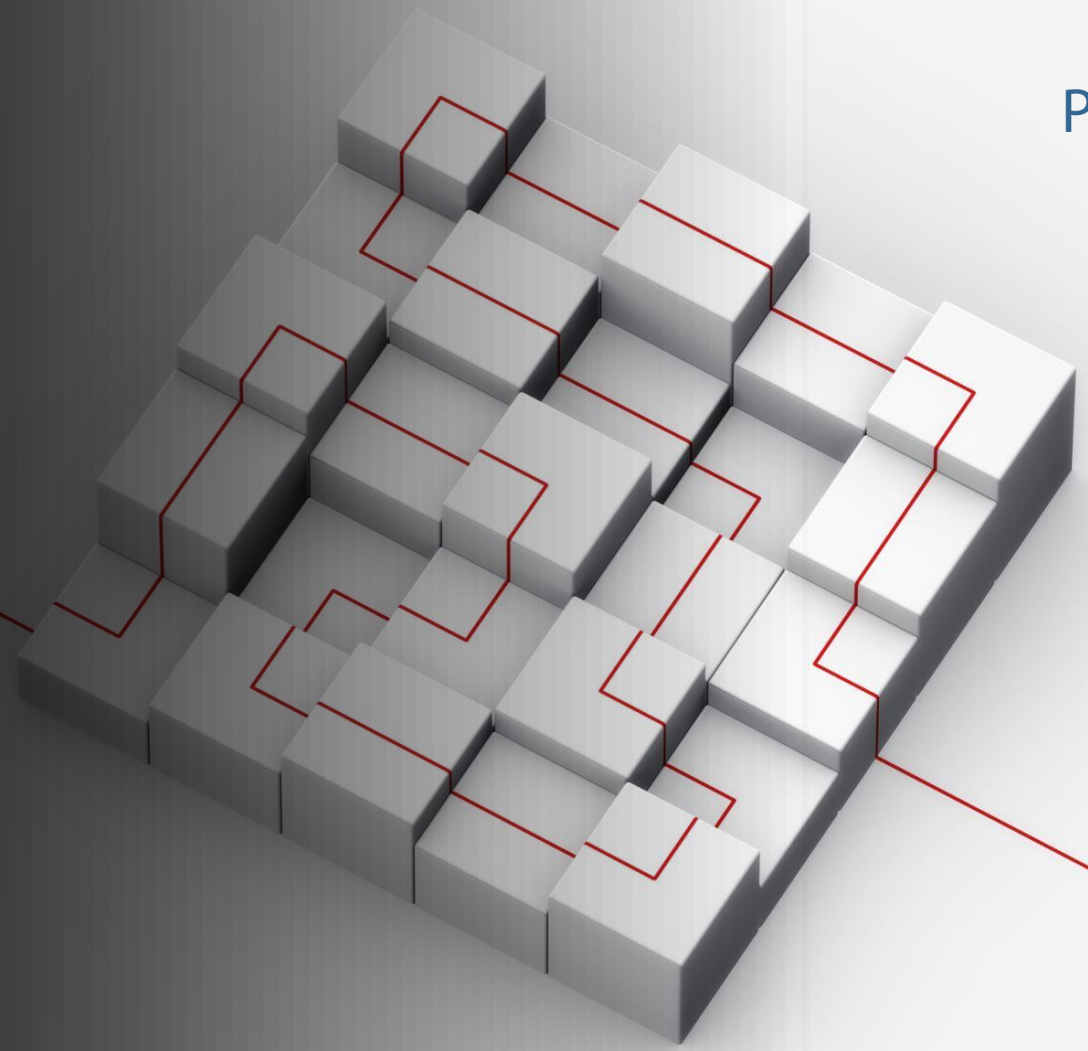
PHASE 7 — Capitalisation (niveau expert)

Après l'incident :

RCA
(Root Cause Analysis)
→ **Détection précoce**
→ **Recommandations dev**
→ **Prévention**
(alertes, seuils)

- ☑ **RCA** (Root Cause Analysis)
- ☑ **Détection précoce**
- ☑ **Recommandations dev**
- ☑ **Prévention** (alertes, seuils)

RÉSOLUTION D'INCIDENTS POSTGRESQL



PostgreSQL

IDEO·LAB

AGENDA GESTION INCIDENT - POSTGRESQL

PHASE 1

PAGE 1 — PHILOSOPHIE & CADRE DE GESTION D'INCIDENT

« En situation d'incident, je privilégie une solution temporaire sûre à une solution parfaite mais risquée. »

🧠 Principes fondamentaux

1. Stabiliser avant d'optimiser
2. Observer avant d'agir
3. Une action = une hypothèse
4. Toujours garder une porte de sortie

🔥 Typologie d'incidents PostgreSQL

Type	Exemple
Performance	lenteur généralisée
Blocage	transactions figées
Saturation	CPU / I/O / mémoire
Applicatif	requête défaillante
Systémique	OS / stockage
Logique	explosion cardinalité

🟥 PAGE 1 — PHILOSOPHIE & CADRE DE GESTION D'INCIDENT

🎯 Objectif

Un incident PostgreSQL ne se "résout" pas, il se gère.

👉 **En production,** le temps est plus critique que l'optimisation parfaite.

RCA
(Root Cause Analysis)

Stabiliser
avant d'optimiser

Détection précoce

Observer
avant d'agir

Une action =
une hypothèse

Prévention
(alertes, seuils)

Toujours garder
une porte de sortie

PAGE 2 — PHASE 1 : ANALYSE & QUALIFICATION

« Si je ne sais pas depuis quand c'est cassé, je ne peux pas comprendre pourquoi. »

1 Qualification immédiate

Je commence toujours par ces questions :

- Incident **actif** ou **passé** ?
- **Impact business** ?
- **Production / pré-prod** ?
- Incident **global** ou **localisé** ?
- Depuis **quand** ?

👉 Cela détermine **le niveau de risque acceptable.**

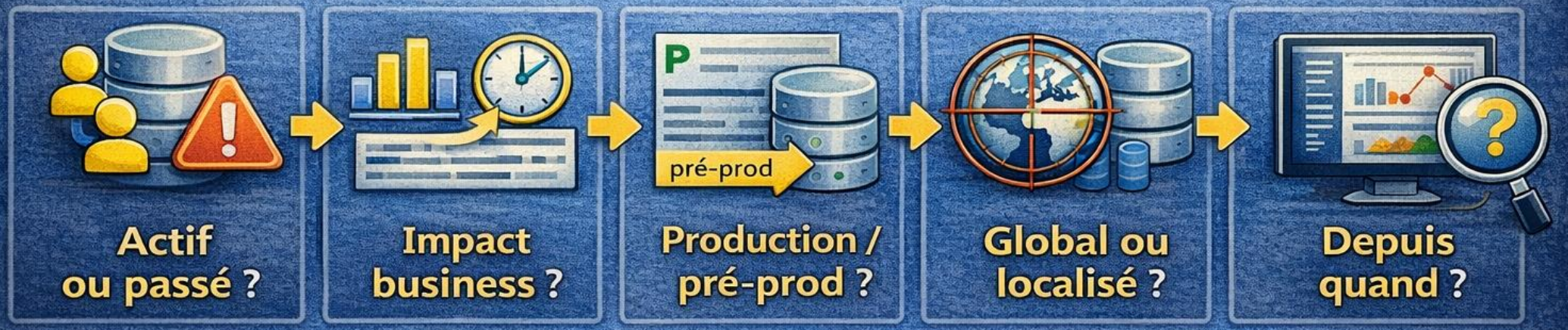
2 Indicateurs vitaux PostgreSQL

Je regarde **en priorité** :

- connexions actives
- sessions bloquées
- requêtes longues
- état des transactions
- évolution dans le temps

1 Qualification immédiate

Je commence toujours par ces questions :



👉 Cela détermine le niveau de risque acceptable.

2 Indicateurs vitaux PostgreSQL

Je regarde en priorité :

Connexions actives

Sessions bloquées

Requétes longues

État des transactions

Évolution dans le temps

- ✔ Connexions actives
- ✔ Sessions bloquées
- ✔ Requétes longues
- ✔ État des transactions
- ✔ Évolution dans le temps

PHASE 2

PAGE 3 — PHASE 2 : OUTILS D’OBSERVATION

« Je cherche ce qui consomme du temps, pas seulement ce qui consomme des ressources. »

A. Activité SQL

- requêtes lentes
- requêtes répétitives
- requêtes bloquantes
- explosions de résultats

B. Concurrence

- verrous
- transactions longues
- files d’attente implicites

C. Ressources

- CPU saturé
- I/O lent
- mémoire insuffisante
- swap

OUTILS PostgreSQL

Outil	Usage
statistiques SQL	top consommateurs
vues d'activité	sessions, états
vues de locks	blocages
logs	anomalies
monitoring	tendances

PHASE 3

C’est ici que le DBA fait la différence

PAGE 4 — PHASE 3 : AIDE À LA DÉCISION

« PostgreSQL ne tombe pas lentement par hasard. Il y a toujours une cause racine mesurable. »

🧭 Arbre de décision simplifié

🔴 **Beaucoup de sessions actives ?**

→ contention / SQL / pool

🔴 **Peu de sessions mais lenteur ?**

→ I/O / storage / locks invisibles

🔴 **CPU élevé ?**

→ SQL inefficace / plan dégradé

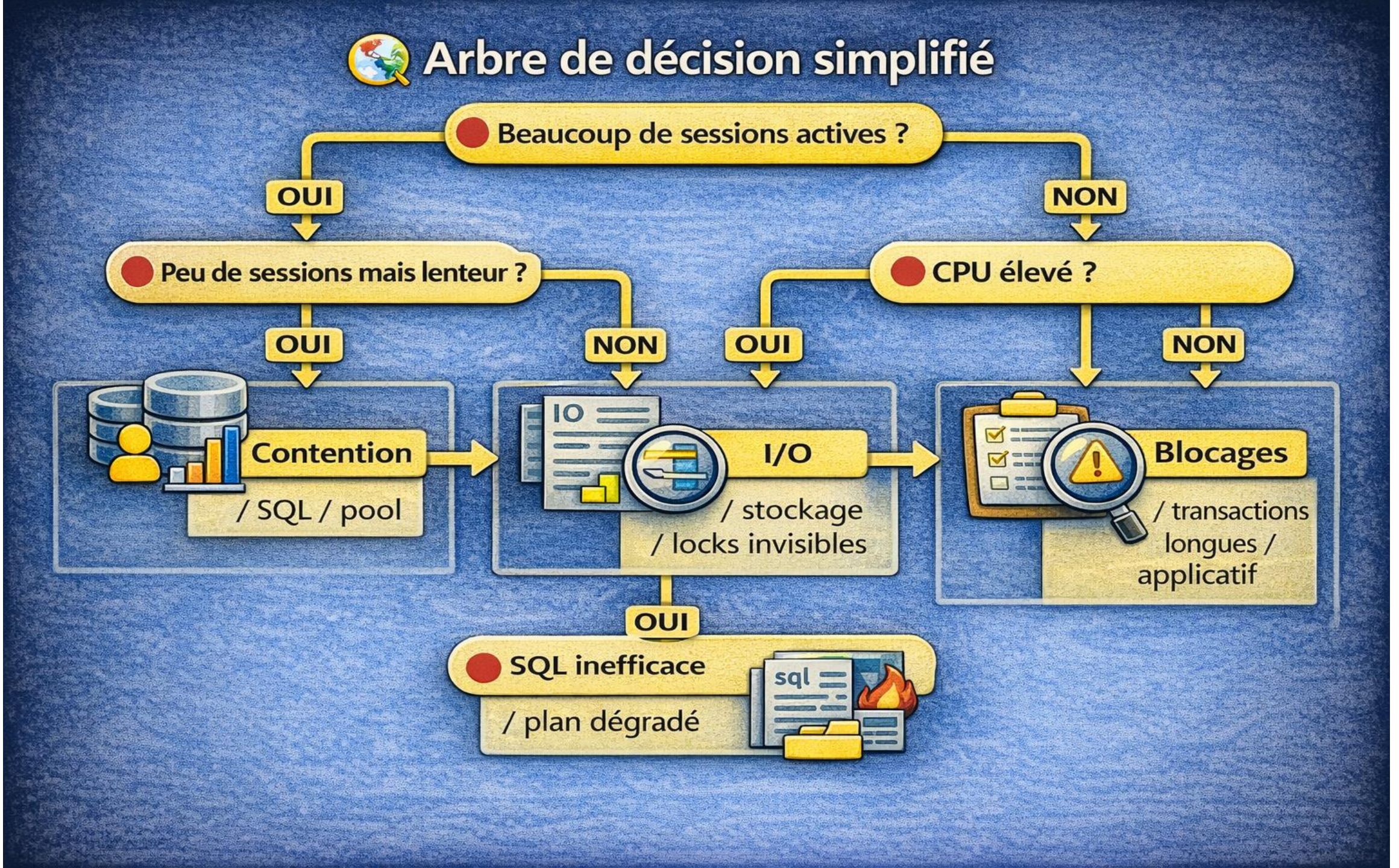
🔴 **Blocages ?**

→ transactions longues / applicatif

🧠 Règles d’or de diagnostic

- Un symptôme ≠ une cause
- Toujours une cause dominante
- Un incident peut masquer un autre

Arbre de décision simplifié



Intervenir sans aggraver

PAGE 5 — PHASE 4 : CORRECTIFS

« Toute action en production doit être justifiable, mesurable et réversible. »

🛠️ Typologie de correctifs

Correctifs immédiats (urgence)

- terminer une session bloquante
- limiter une requête
- soulager la charge
- rétablir le service

Correctifs intermédiaires

- ajout index ciblé
- correction SQL
- ajustement configuration léger
- nettoyage transactionnel

PHASE 4

Intervenir sans aggraver

PAGE 5 — PHASE 4 : CORRECTIFS

Correctifs long terme

- refonte requête
- modèle de données
- stratégie d'indexation
- gouvernance SQL

⚠️ Règles de sécurité

- jamais plusieurs changements à la fois
- toujours mesurer avant/après
- documenter chaque action
- anticiper rollback

Typologie de correctifs PostgreSQL

Correctifs immédiats (urgence)

- **Terminer** une session / SQL / pool
- **Soulager** la charge
- **Limiter** une requête
- **Rétablir** le service

Correctifs intermédiaires

- Ajout index ciblé
- Ajustement configuration léger
- Correction SQL
- Nettoyage transactionnel

Correctifs long terme

Refonte
requéte

Modèle
de données

Stratégie
d'indexation

Gouvernance
SQL

PHASE 5

Là où un DBA devient architecte

PAGE 6 — PHASE 5 : POST-INCIDENT & CAPITALISATION

« Un incident non documenté est un incident qui reviendra. »

🟦 Analyse post-mortem (RCA)

- chronologie de l'incident
- cause racine réelle
- facteurs aggravants
- actions efficaces
- actions inefficaces

🔐 Prévention

- alertes mieux calibrées
- seuils adaptés
- revues SQL régulières
- règles applicatives
- formation équipes dev

Analyse post-mortem (RCA)

Chronologie de l'incident

Facteurs aggravants

- **Facteurs** aggravants
- **Actions** inefficaces

Prévention

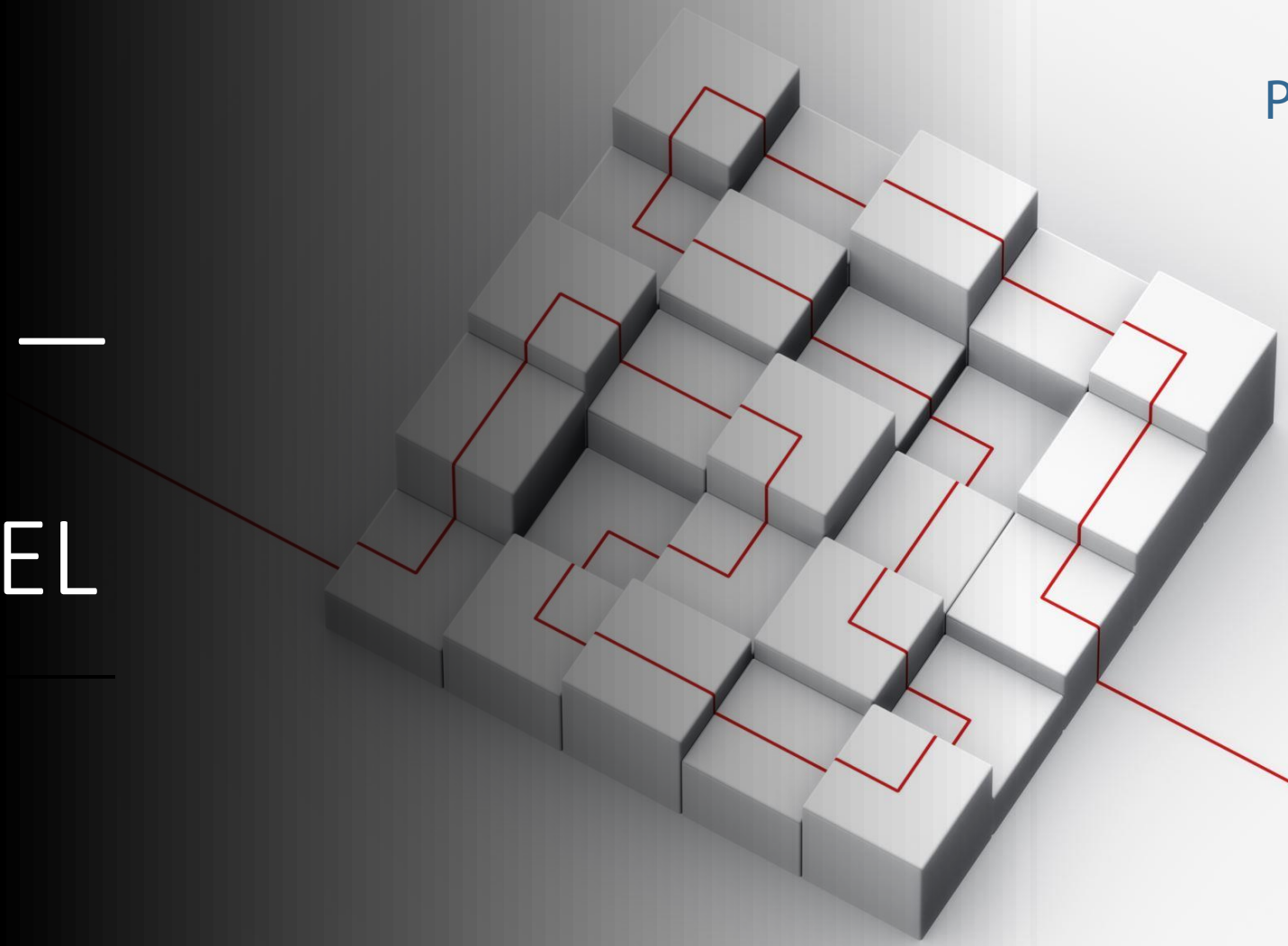
Alertes mieux calibrées

- Seuils adaptés

Revues SQL régulières

- Règles applicatives
- Formation équipes dev

INDICATEURS VITAUX POSTGRESQL — MODE OPÉRATIONNEL



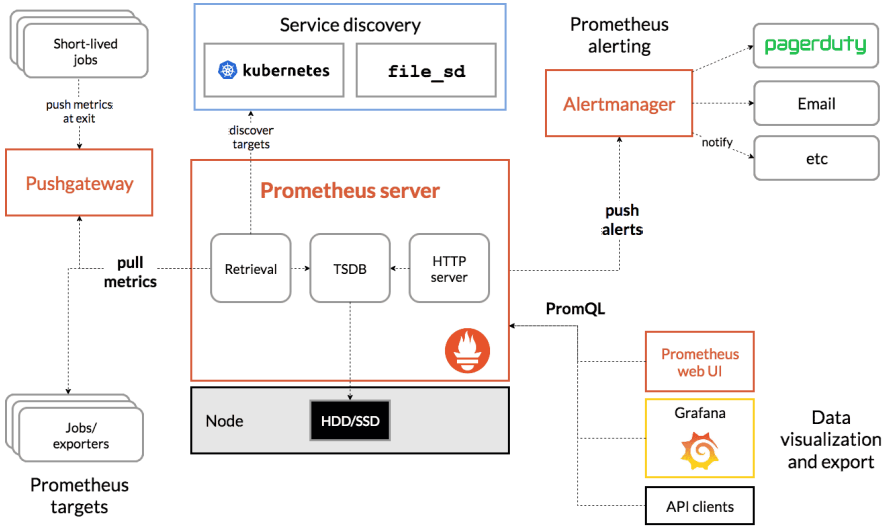
PostgreSQL

IDEO·LAB

AGENDA INDICATEURS VITAUX POSTGRESQL

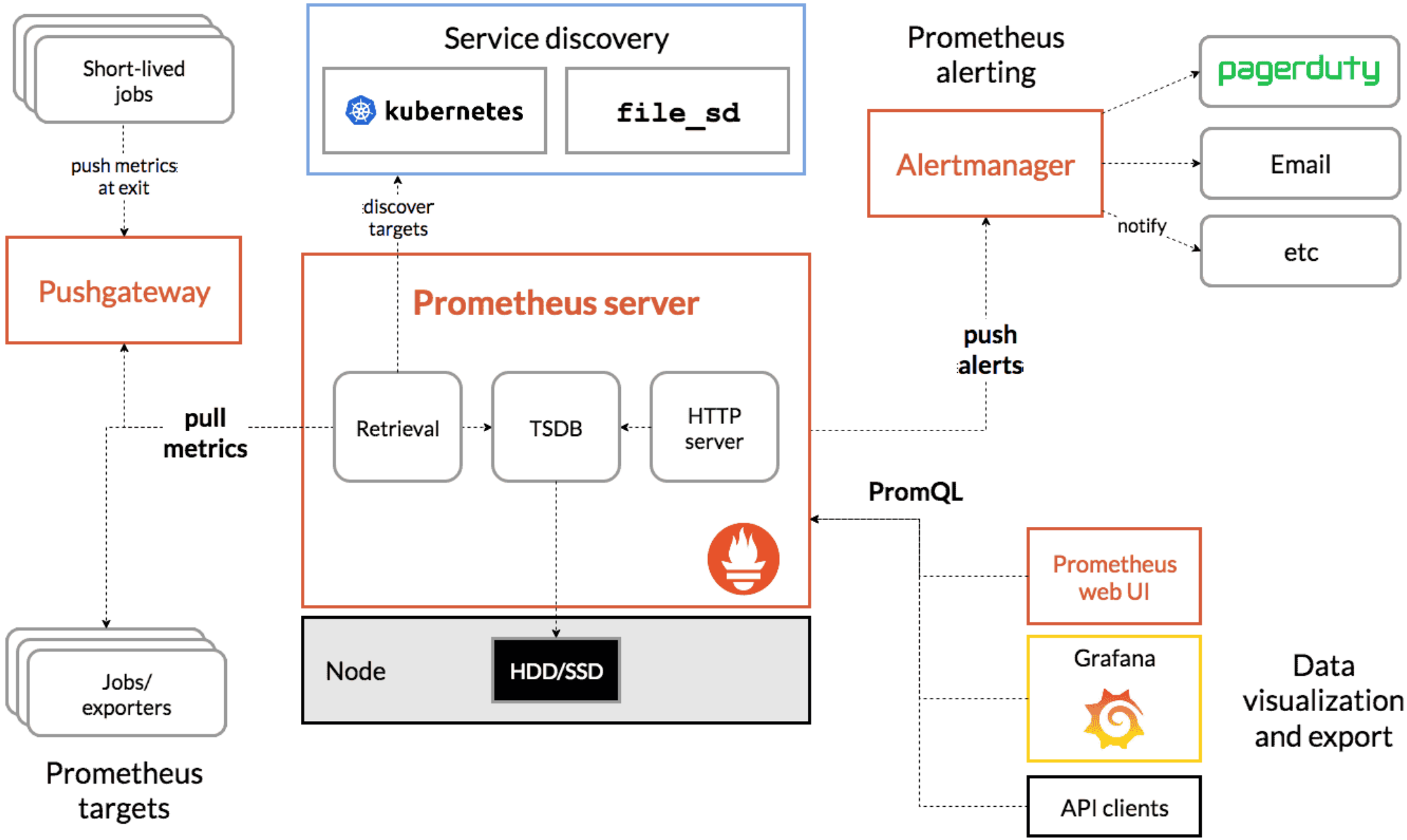
- Vue d’ensemble
- 1. CONNEXIONS & ACTIVITÉ GLOBALE
- 2. SESSIONS BLOQUÉES
- 3. REQUÊTES LONGUES
- 4. TRANSACTIONS PROBLÉMATIQUES
- ÉVOLUTION DANS LE TEMPS

Vue d’ensemble

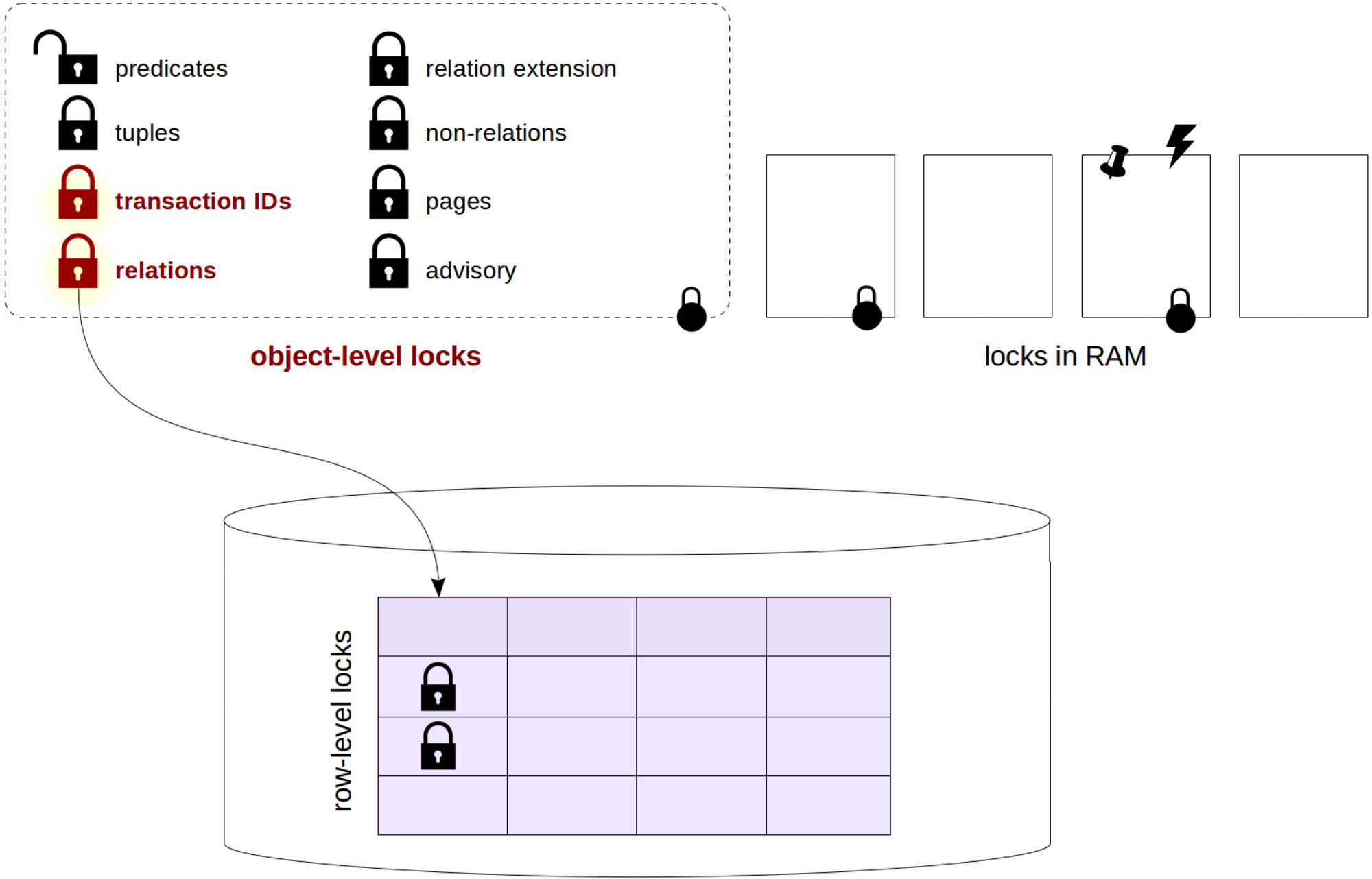


1. Activité globale
2. Blocages
3. Requêtes longues
4. Transactions anormales
5. Évolution dans le temps

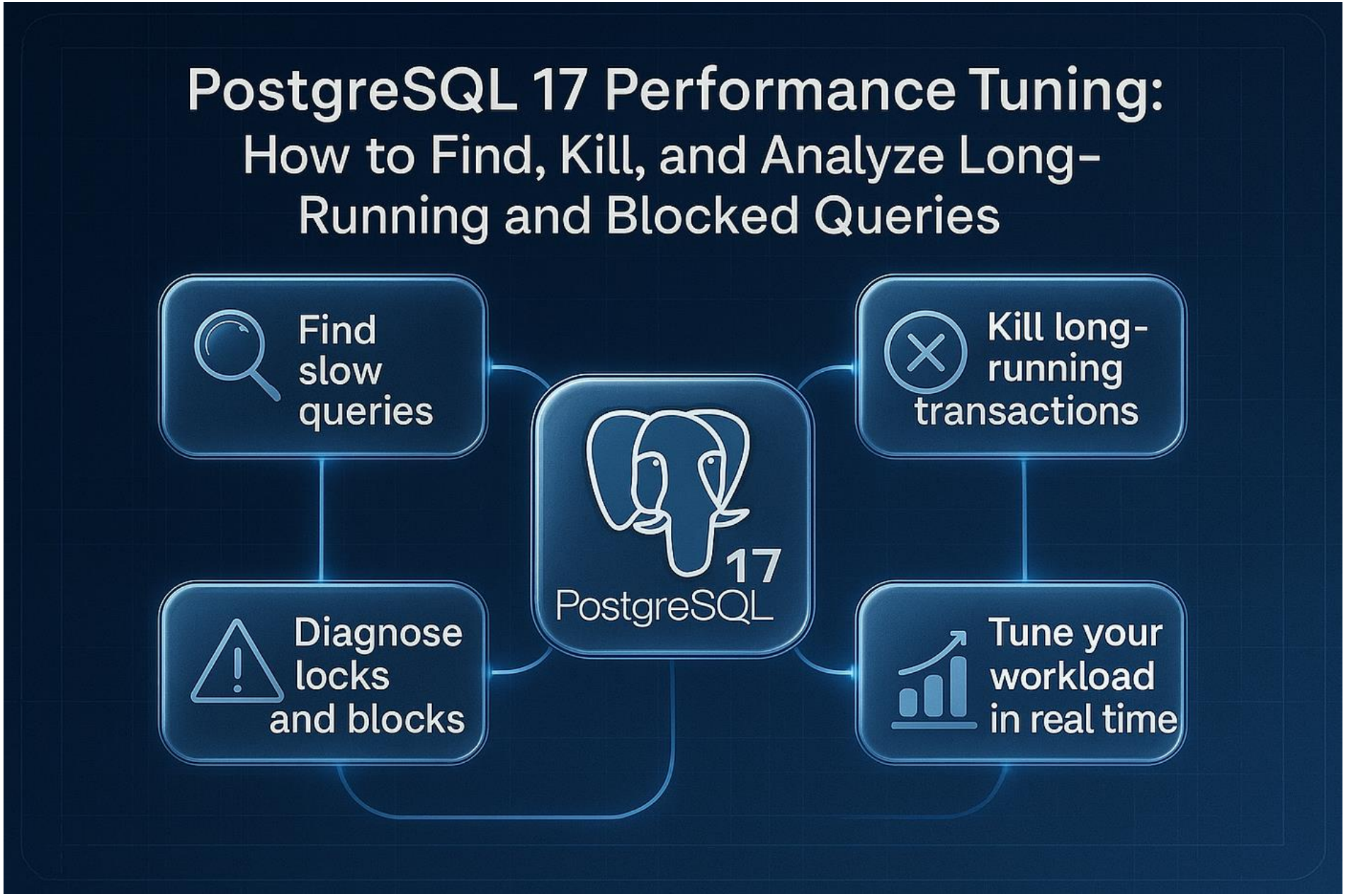
ACTIVITE GLOBALE



BLOCAGES & VERROUS



Requêtes longues



Est-ce que la base est “vivante” ou saturée ?

CONNEXIONS & ACTIVITÉ GLOBALE

🔧 Action immédiate (SQL)

```sql
SELECT
    state,
    COUNT(*) AS nb_sessions
FROM pg_stat_activity
GROUP BY state
ORDER BY nb_sessions DESC;
```

🧠 Ce que je regarde

- trop de `active` → charge SQL
- trop de `idle in transaction` → bug applicatif
- explosion du nombre total → pool mal réglé

Est-ce que la base est “vivante” ou saturée ?

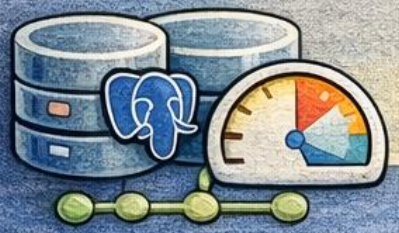
CONNEXIONS & ACTIVITÉ GLOBALE

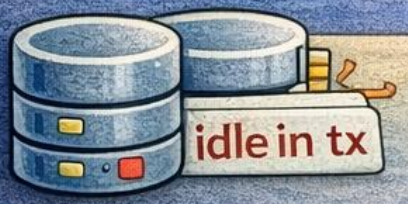
🚨 Décision rapide

Observation	Décision
Trop de connexions	limiter / pool
Peu d'actives mais lent	blocage ou I/O
Beaucoup d'idle in tx	incident applicatif

Décision rapide

Observation	Décision
Trop de connexions	**limiter / pool**
Peu d'actives mais lent	**blocage ou I/O**
Beaucoup d'idle in tx	**incident** applicatif
Beaucoup d'idle in tx	**incident applicatif**





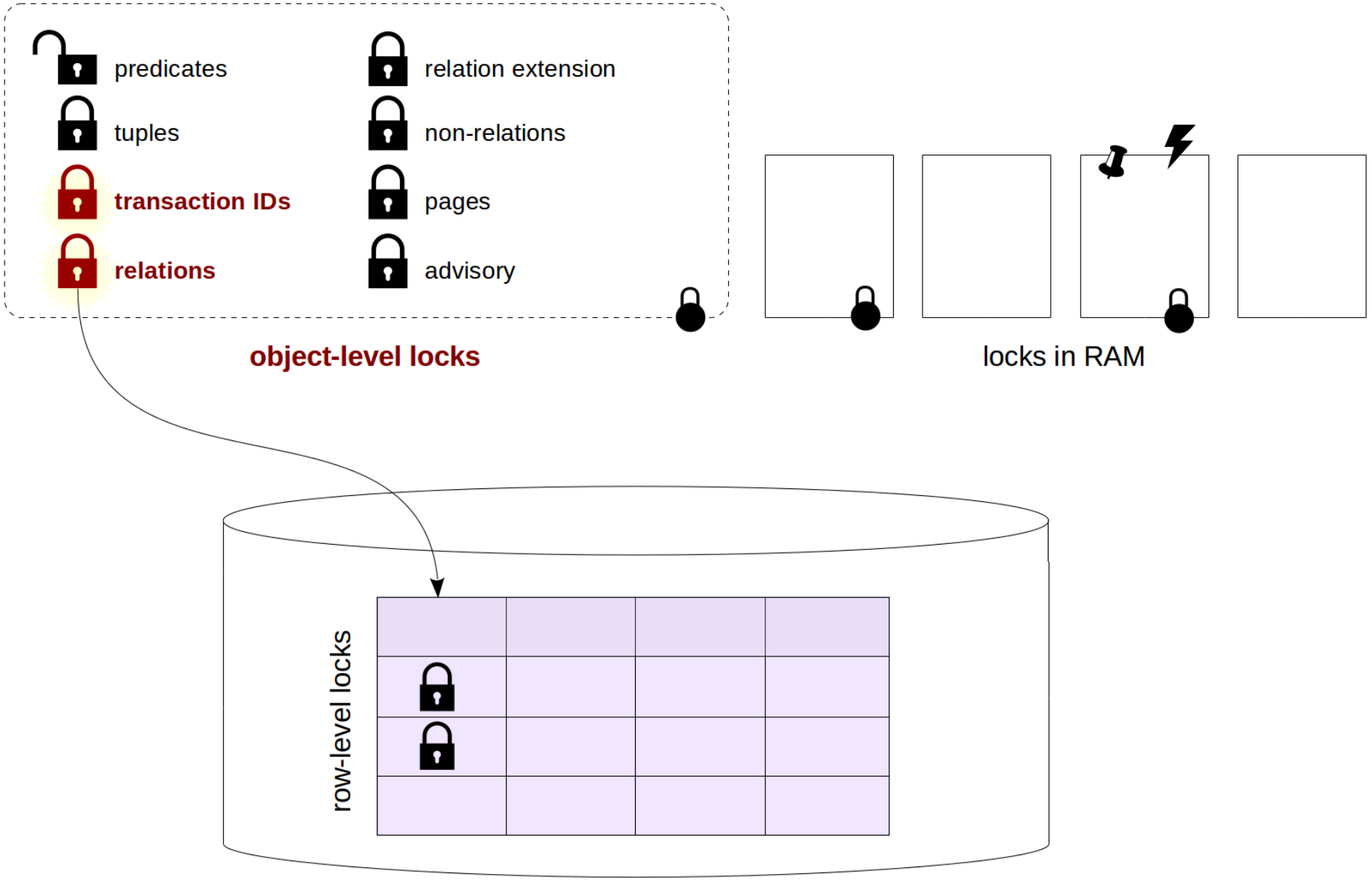
On ne tue jamais une session sans savoir qui elle bloque.

SESSIONS BLOQUÉES

- **1 session bloque 50 autres** → kill ciblé
- blocage long → transaction oubliée
- blocage DDL → attention effet domino

🔧 **Action DBA (la vraie)**

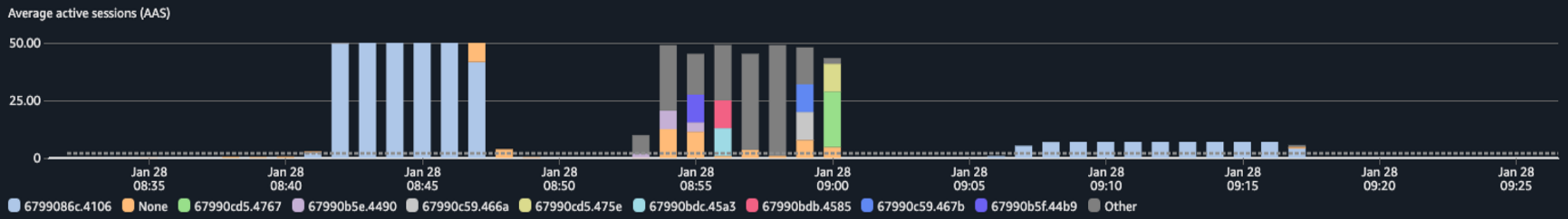
```
SELECT
    blocked.pid     AS blocked_pid,
    blocking.pid    AS blocking_pid,
    blocked.query   AS blocked_query,
    blocking.query  AS blocking_query,
    now() - blocking.query_start AS blocking_duration
FROM pg_stat_activity blocked
JOIN pg_locks bl ON blocked.pid = bl.pid
JOIN pg_locks bl2
  ON bl.locktype = bl2.locktype
 AND bl.database IS NOT DISTINCT FROM bl2.database
 AND bl.relation IS NOT DISTINCT FROM bl2.relation
 AND bl.page IS NOT DISTINCT FROM bl2.page
 AND bl.tuple IS NOT DISTINCT FROM bl2.tuple
 AND bl.transactionid IS NOT DISTINCT FROM bl2.transactionid
 AND bl.classid IS NOT DISTINCT FROM bl2.classid
 AND bl.objid IS NOT DISTINCT FROM bl2.objid
 AND bl.objsubid IS NOT DISTINCT FROM bl2.objsubid
 AND bl.pid <> bl2.pid
JOIN pg_stat_activity blocking ON blocking.pid = bl2.pid
WHERE NOT bl.granted;
```


predicates
tuples
transaction IDs
relations
relation extension
non-relations
pages
advisory
object-level locks
locks in RAM
row-level locks

Database load Info

Sliced by: Blocking Session | Bar | Line

Analyze performance



DB Load Analysis | Database Telemetry | Performance Analysis | Calling Services | Recommendations - *coming soon*

Lock Analysis | Top SQL | Top waits | Top hosts | Top users | Top session types | Top applications | Top databases

Top SQL (13) Info

Find SQL statements

1 2

Load by blocking_sessions (AAS)	SQL statements	ID	Calls/sec
9.98	update orders set shipped=$1 where order_id=$2	-3306409...	6.05
1.53	update orders set order_details=$1 where order_id=$2	-8677200...	0.02
1.33	update orders set shipped=now() where order_id=?	53807660...	0.00
0.03	update orders set shipped=$1 where order_id=$2	12877904...	43.56
0.01	COMMIT	30119699...	0.59

CONSOMMATION DE TEMPS

REQUÊTES LONGUES

🔧 Action SQL

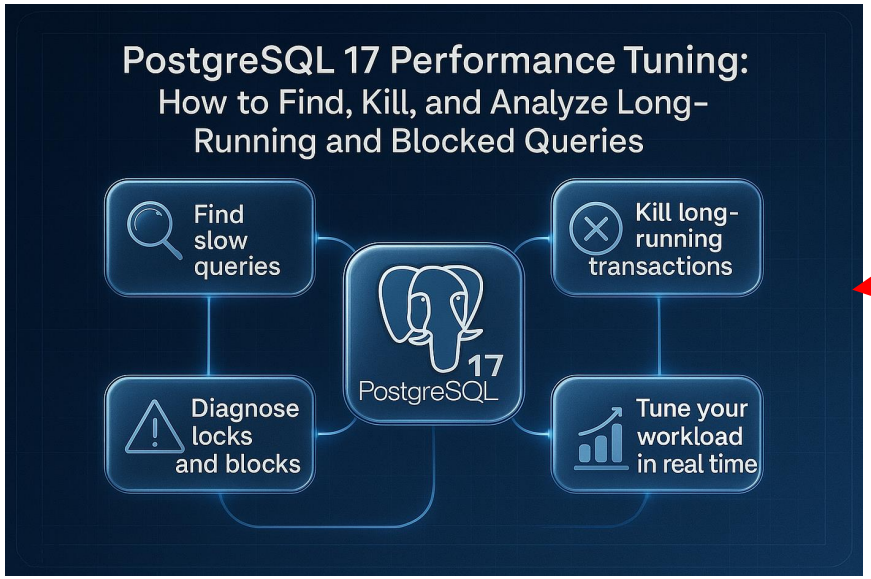
```sql
SELECT
    pid,
    usename,
    now() - query_start AS duration,
    state,
    query
FROM pg_stat_activity
WHERE state = 'active'
ORDER BY duration DESC
LIMIT 10;
```

🧠 Ce que j'analyse

- requête récente mais très longue → plan mauvais
- requête courte répétée → effet cumulatif
- requête active + locks → fausse piste

CONSOMMATION DE TEMPS

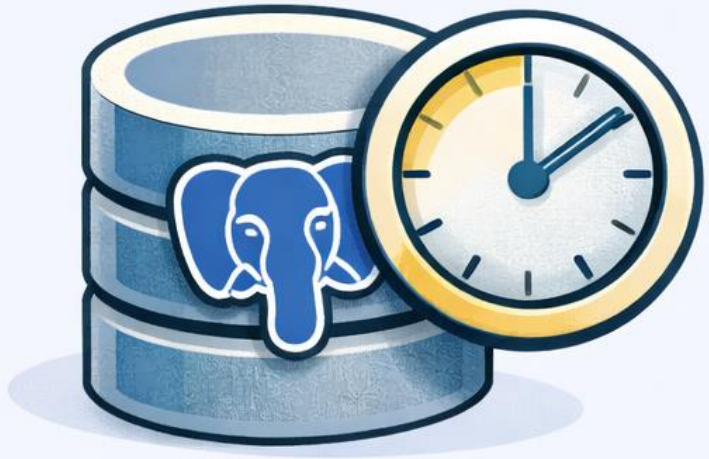
REQUÊTES LONGUES



Aide à la décision

Cas	Action
Requête unique très lente	analyse plan
Requête répétée	index / rewrite
Batch bloquant	fenêtre / isolation

Requêtes SQL longues



Requête unique très lente

Requête courte répétée

Requête active + verrous

🧭 Aide à la décision

 Requête unique très lente	→	Analyse du plan	
 Requête répétée	→	Index / Rewrite	
 Batch bloquant	→	Fenêtre / Isolation	

PostgreSQL 17 Performance Tuning:
How to Find, Kill, and Analyze Long-Running and Blocked Queries

Search or CTRL + SPACE
new
Clusters
Operational Reports
Email Notifications
Integrations
Key Management
User Management
Documentation
Give us Feedback
What's New
Support Forum
Switch Theme

Query Monitor
Monitoring Queries and Process List

- **Top Queries** — View Top Slow and Long-running Queries
- **Running Queries** — View Current Running Queries
- **Query Outliers** — Query Outliers
- **Query Statistics** — View Current Query Statistics

Top Queries
Query monitor is enabled

Off / On

Settings

Search Query Text | All hosts | Occurrences | Refresh rate: 30 secs | Show: 20 rows

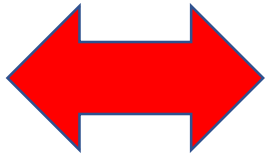
Purge Query Monitor

Query	DB	Count	Rows		Tmp tables		Exec time			Total exec time		Last seen
			Sent	Examined	RAM	On Disk	Max	Avg	Stdev	Absolute	Relative %	
BEGIN	pgbench	34431	0	0	0	0	20 ms	7 us	0 us	00:00:00	0.06	a few seconds ago
UPDATE pgbench_accounts SET abalance = abalance + $1 WHERE aid = $2	pgbench	34431	0	1	0	0	805 ms	9.1 ms	13 ms	00:05:13	74.56	a few seconds ago
SELECT abalance FROM pgbench_accounts WHERE aid = $1	pgbench	34427	0	1	0	0	38 ms	122 us	1 ms	00:00:04	1.01	a few seconds ago
UPDATE pgbench_tellers SET tbalance = tbalance + $1 WHERE tid = $2	pgbench	34427	0	1	0	0	212 ms	209 us	2 ms	00:00:07	1.71	a few seconds ago
UPDATE pgbench_branches SET bbalance = bbalance + $1 WHERE bid = $2	pgbench	34426	0	1	0	0	83 ms	213 us	1 ms	00:00:07	1.75	a few seconds ago
INSERT INTO pgbench_history (tid, bid, aid, delta, mtime) VALUES ($1, $2, $3, $4...	pgbench	34425	0	1	0	0	61 ms	76 us	0 us	00:00:02	0.62	a few seconds ago
END	pgbench	34424	0	0	0	0	16 ms	7 us	0 us	00:00:00	0.06	a few seconds ago
SELECT $1	postgres	7003	0	1	0	0	12 ms	10 us	0 us	00:00:00	0.02	a few seconds ago
SELECT datname, pid, username, application_name, COALESCE(client_hostname, host(...	postgres	3016	0	7	0	0	36 ms	176 us	0 us	00:00:00	0.13	a few seconds ago
SELECT archived_count, failed_count FROM pg_stat_archiver	postgres	3016	0	1	0	0	89 ms	121 us	1 ms	00:00:00	0.09	a few seconds ago
SELECT viewname FROM pg_views WHERE viewname = $1	postgres	3016	0	0	0	0	46 ms	40 us	0 us	00:00:00	0.03	a few seconds ago
SELECT sum(heap_blks_read), sum(heap_blks_hit), sum(idx_blks_read), sum(idx_blks...	postgres	3016	0	1	0	0	71 ms	188 us	1 ms	00:00:00	0.14	a few seconds ago
SELECT checkpoints_timed, checkpoints_req, checkpoint_write_time, checkpoint_syn...	postgres	3016	0	1	0	0	16 ms	102 us	0 us	00:00:00	0.07	a few seconds ago
SELECT sum(xact_commit), sum(xact_rollback), sum(tup_fetched), sum(tup_inserted)...	postgres	3016	0	1	0	0	117 ms	11.59 ms	3 ms	00:00:34	8.32	a few seconds ago
SELECT SUM(pg_database_size(datname)) FROM pg_database	postgres	3016	0	1	0	0	157 ms	2.08 ms	4 ms	00:00:06	1.49	a few seconds ago

TRANSACTIONS PROBLÉMATIQUES

🚨 Décision

- corriger l'application
- timeout transactionnel
- règles strictes côté dev



🔧 Action immédiate (SQL)

```sql
SELECT
    pid,
    usename,
    state,
    now() - xact_start AS xact_duration,
    query
FROM pg_stat_activity
WHERE xact_start IS NOT NULL
ORDER BY xact_duration DESC;
```

🧠 Lecture

- `idle in transaction` > quelques minutes = **anomalie**
- transactions longues = verrous + vacuum bloqué
- souvent invisible côté applicatif

Lecture

`idle in transaction` > quélques minutes = **anomalie**

- **transactions longues** = verrous + vacuum bloqué

- souvent invisible côté applicatif

Décision

corriger l'application

timeout transactionnel

règles strictes côté dev

Est-ce que c’est brutal ou progressif ?

ÉVOLUTION DANS LE TEMPS

Question clé

Interprétation

Évolution	Hypothèse
Brutale	déploiement / batch
Progressive	dérive stats / données
Cyclique	job planifié
Aléatoire	contention / pool

Question clé Est-ce que c'est brutal ou progressif ?

Brutale

Déploiement / batch

Évolution

Progressive

Dérive stats / données

Hypothése

Cyclique

Job planifié

Évolution

Aléatoire

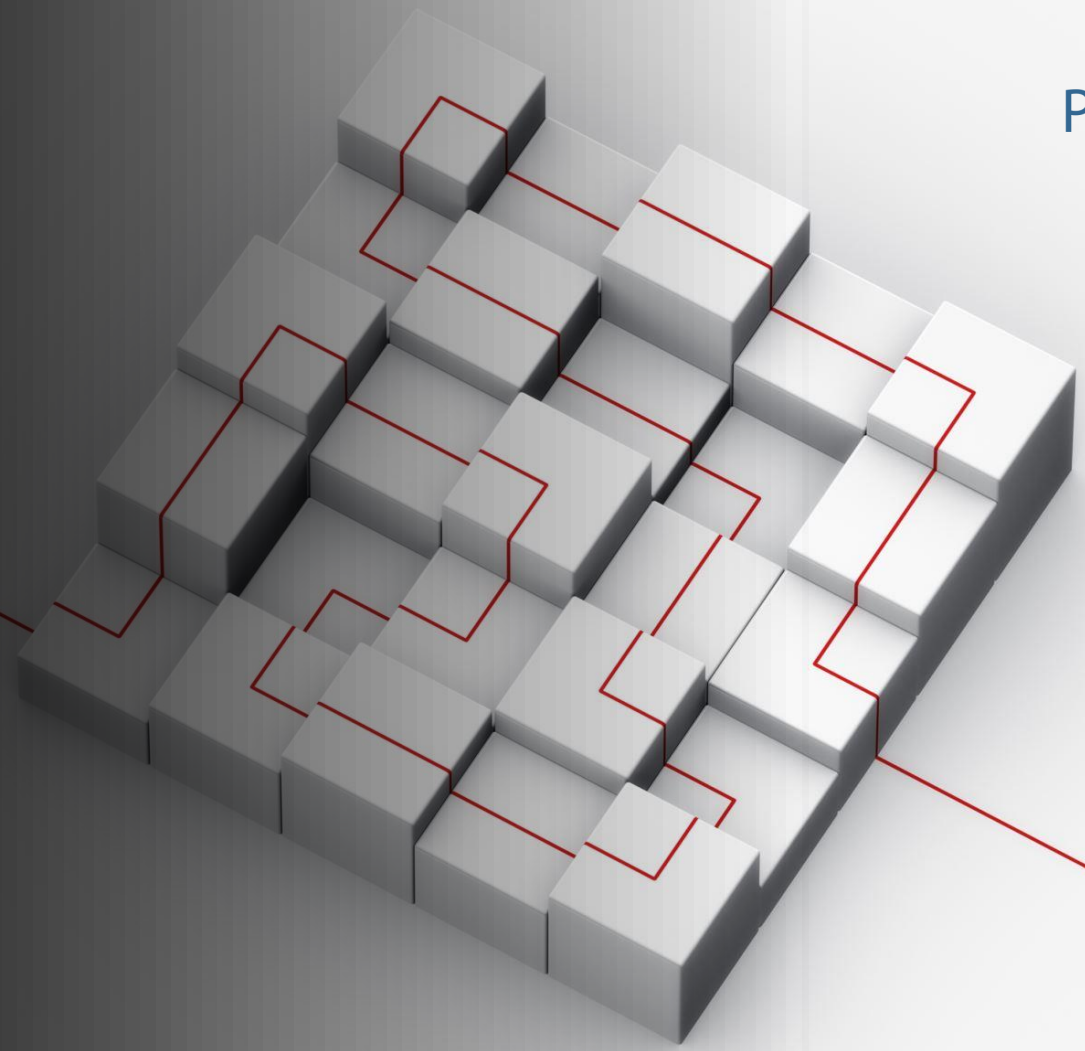
Contention / pool

Hypothèse

DIAGRAMME MENTAL DBA

```
Base lente
   |
   +-- Beaucoup de sessions ?
   |        |
   |        +-- Oui → SQL / Pool
   |        +-- Non → Locks / I/O
   |
   +-- Blocages ?
   |        |
   |        +-- Oui → Transaction longue
   |
   +-- Requêtes longues ?
            |
            +-- Oui → Plan / Index
```

INVESTIGATION DE NIVEAU CHIRURGICAL



PostgreSQL

IDEO·LAB

AGENDA INVESTIGATION « CHIRURGICALE »

- 1. PAGE — **Analyse approfondie SQL** (*EXPLAIN*)
  - Préparation “safe”
  - EXPLAIN réel (le trio gagnant)
  - Lecture “DBA”
  - Diagnostiquer vite (patterns)
  - Actions “DBA-safe” (sans casser prod)
  - Mini diagramme mental EXPLAIN

EXPLAIN D'UNE REQUETE SQL

Analyse approfondie SQL (EXPLAIN)

Le piège classique

- EXPLAIN seul = plan estimé (utile, mais parfois trompeur)
- EXPLAIN (ANALYZE, BUFFERS) = plan réel + timings + buffers → indispensable

```
Plan lent
  |
  +-- Estimations fausses ? (rows est vs rows act)
  |       -> stats / corrélation / histogrammes (extended stats)
  |
  +-- Spill temp ? (Sort/Hash -> Disk)
  |       -> work_mem session / rewrite
  |
  +-- Mauvais scan ? (SeqScan sur grosse table)
  |       -> index / predicate
  |
  +-- Join explosif ? (Nested Loop loops énorme)
          -> index join key / join order / rewrite
```

```
EXPLAIN
SELECT * FROM utilisateurs
WHERE ville = 'Paris';
```

```
EXPLAIN (ANALYZE, BUFFERS)
SELECT * FROM utilisateurs
WHERE ville = 'Paris';
```

L'EXPLAIN d'une requête SQL sous PostgreSQL

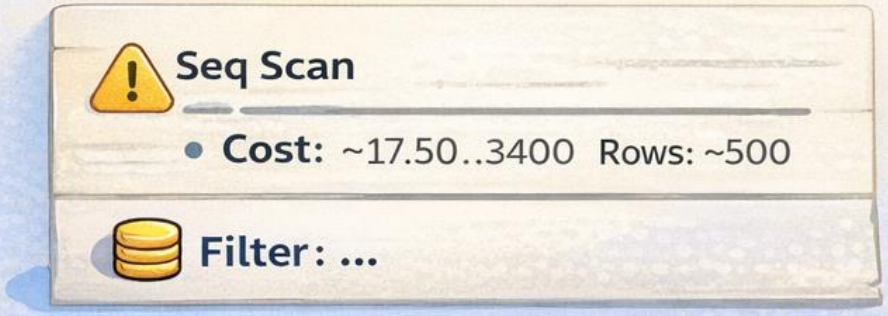
1 Analyse avec la commande EXPLAIN

Tapez la commande **EXPLAIN** devant votre requête SQL pour voir comment PostgreSQL va l'exécuter.

2 Plan d'exécution détaillé

Observez le plan généré par EXPLAIN : il détaille le chemin emprunté, les scans effectués, et les coûts estimés.

3 Soyez attentif à ces valeurs clés



4 Interprétez le plan pour optimiser

Ajoutez **un index**

Ajustez **un seuil**

Améliorez la requête

Règle DBA : je ne touche pas au SQL sans avoir vu le plan réel (quand c'est possible)

Préparation "safe" (en prod)

Identifier LA requête

```
SELECT pid, usename, now()-query_start AS dur, state, query
FROM pg_stat_activity
WHERE state='active'
ORDER BY dur DESC
LIMIT 5;
```

```
SELECT queryid, calls, total_exec_time, mean_exec_time, rows, query
FROM pg_stat_statements
ORDER BY total_exec_time DESC
LIMIT 10;
```

EXPLAIN réel

```
EXPLAIN (ANALYZE, BUFFERS, VERBOSE)
<ta_requête>;
```

```
EXPLAIN (ANALYZE, BUFFERS, VERBOSE, SETTINGS)
<ta_requête>;
```

```
EXPLAIN (ANALYZE, BUFFERS)
<ta_requête>;
```

sql Copier le code

```
EXPLAIN (ANALYZE, BUFFERS, VERBOSE)
<ta_requête>;
```

sql Copier le code

```
EXPLAIN (ANALYZE, BUFFERS, VERBOSE, SETTINGS)
<ta_requête>;
```

Si tu suspectes un tri / hash qui déborde disque : Copier le code

```
EXPLAIN (ANALYZE, BUFFERS)
<ta_requête>;
-- puis vérifie dans la sortie : Sort Method:
external merge / Disk:
```

Lecture “DBA”

E) Buffers (hyper parlant)

- `shared hit` élevé = cache OK
- `shared read` élevé = I/O disque
- `temp read/write` = spill (sort/hash) → work_mem ou stratégie

A) Temps total / temps par nœud

- Cherche le nœud avec le plus gros **Actual Time / Loops.**

B) Cardinalités : Estimated vs Actual

- Si `rows` estimées ≪ `rows` réelles → stats/estimation/corrélation
- Gros écart = **plan possiblement faux** (mauvais join order, mauvais choix d’index)

C) Type de scan

- `Seq Scan` : pas forcément mauvais (petite table), mais suspect si grosse table + filtre sélectif
- `Index Scan` / `Bitmap Index Scan` : bon signe si sélectif
- `Bitmap Heap Scan` : souvent OK mais peut exploser si beaucoup de hits

D) Mémoire / disque implicite

- `Sort Method: quicksort` (mémoire)
- `Sort Method: external merge` (disque) → **work_mem** ou requête à revoir
- `Hash` avec `Batch` > 1 → hash a “spill” sur disque

L'EXPLAIN d'une requête SQL sous PostgreSQL

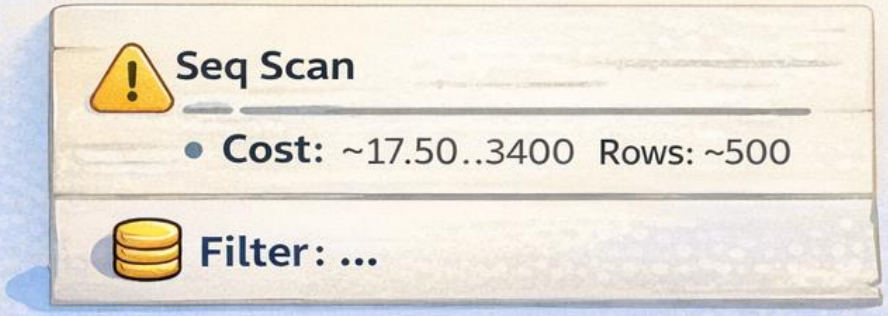
1 Analyse avec la commande EXPLAIN

Tapez la commande **EXPLAIN** devant votre requête SQL pour voir comment PostgreSQL va l'exécuter.

2 Plan d'exécution détaillé

Observez le plan généré par EXPLAIN : il détaille le chemin emprunté, les scans effectués, et les coûts estimés.

3 Soyez attentif à ces valeurs clés



4 Interprétez le plan pour optimiser

Ajoutez **un index**

Ajustez **un seuil**

Améliorez la requête

Diagnostiquer vite

Pattern 1 — “Nested Loop” qui explose

Symptôme : gros `Loops` + nœud interne cher

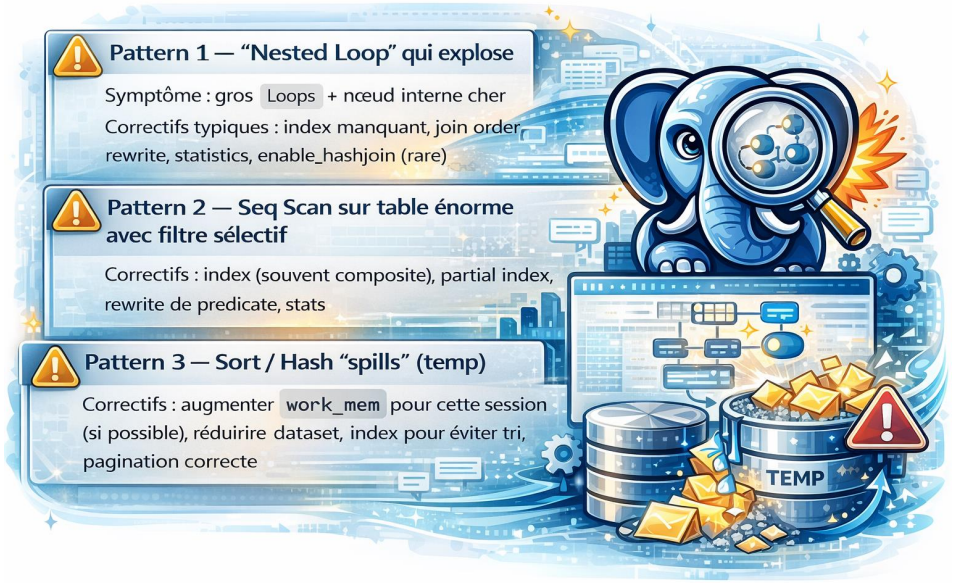
Correctifs typiques : index manquant, join order, rewrite, statistics, enable_hashjoin (rare)

Pattern 2 — Seq Scan sur table énorme avec filtre sélectif

Correctifs : index (souvent composite), partial index, rewrite de predicate, stats

Pattern 3 — Sort / Hash “spills” (temp)

Correctifs : augmenter `work_mem` **pour cette session** (si possible), réduire dataset, index pour éviter tri, pagination correcte



Pattern 1 — “Nested Loop” qui explose

Symptôme : gros `Loops` + nœud interne cher

Correctifs typiques : index manquant, join order rewrite, statistics, enable_hashjoin (rare)

Pattern 2 — Seq Scan sur table énorme avec filtre sélectif

Correctifs : index (souvent composite), partial index, rewrite de predicate, stats

Pattern 3 — Sort / Hash “spills” (temp)

Correctifs : augmenter `work_mem` pour cette session (si possible), réduirire dataset, index pour éviter tri, pagination correcte

Actions “DBA-safe”

a) Tester sans impacter

- reproduire en préprod si possible
- sinon limiter via `LIMIT` / sample (quand possible)

b) Améliorer sans changer le SQL (souvent le plus rapide)

- `ANALYZE` sur tables concernées (stats)
- vérifier bloat / vacuum si needed (selon cas)

c) Améliorer avec changement SQL / index

- index ciblé
- rewrite (JOIN/CTE/pagination)
- éviter fonctions sur colonnes filtrées (empêche index)

Actions “DBA-safe” (sans casser prod)

a) Tester sans impacter

- reproduire en **préprod** si possible
- sinon limiter via `LIMIT` / sample

Améliorer sans changer le SQL

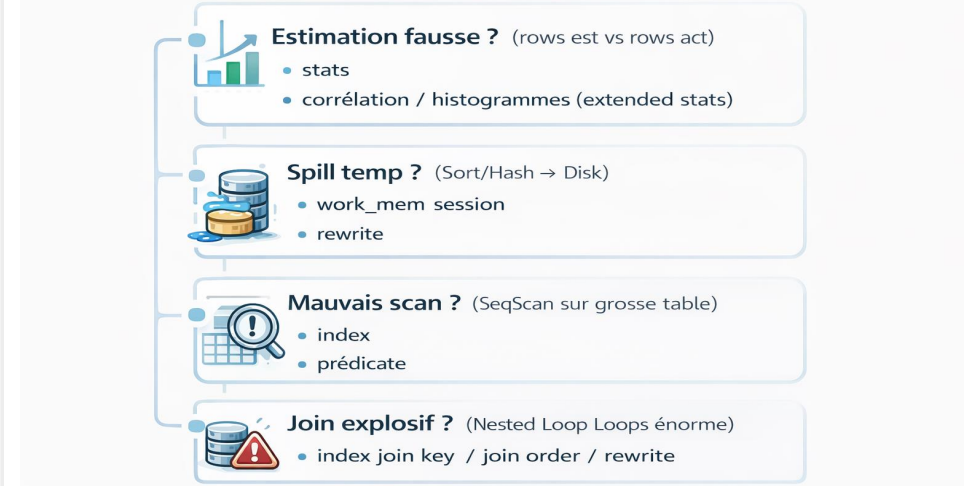
- `ANALYZE` sur tables concemées (stats)
- vérifier bloat / vacuum si needed

Améliorer avec changement SQL / index

- index ciblé
- rewrite (JOIN/CTE/pagination)
- éviter fonctions sur colonnes filtrées

Mini diagramme mental EXPLAIN

```
Plan lent
  |
  +-- Estimations fausses ? (rows est vs rows act)
  |       -> stats / corrélation / histogrammes (extended stats)
  |
  +-- Spill temp ? (Sort/Hash -> Disk)
  |       -> work_mem session / rewrite
  |
  +-- Mauvais scan ? (SeqScan sur grosse table)
  |       -> index / predicate
  |
  +-- Join explosif ? (Nested Loop loops énorme)
          -> index join key / join order / rewrite
```



Plan lent

Estimation fausse ? (rows est vs rows act)

- stats
- corrélation / histogrammes (extended stats)

Spill temp ? (Sort/Hash → Disk)

- work_mem session
- rewrite

Mauvais scan ? (SeqScan sur grosse table)

- index
- prédicate

Join explosif ? (Nested Loop Loops énorme)

- index join key / join order / rewrite

AGENDA INVESTIGATION « CHIRURGICALE »

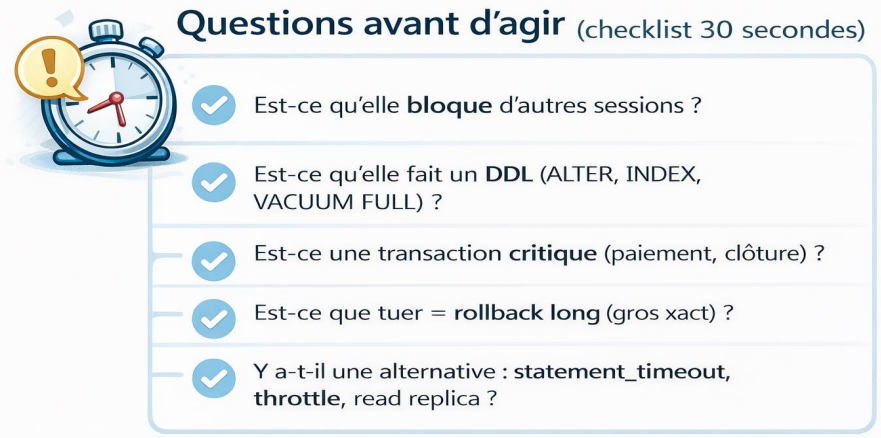
- 2. **Décision critique : tuer ou ne pas tuer une session**
  - Règle “senior”
  - Questions avant d’agir
  - Arbre de décision
  - Commandes “propres”
  - Voir le “blast radius”
  - Quand tuer est une mauvaise idée

Je ne tue pas une session “parce qu’elle est longue”.Je la tue si elle met en danger la prod ET si je peux justifier le risque.

Questions avant d’agir (*checklist 30 secondes*)

1) Questions avant d’agir (checklist 30 secondes)

1. Est-ce qu’elle **bloque** d’autres sessions ?
2. Est-ce qu’elle fait un **DDL** (ALTER, INDEX, VACUUM FULL) ?
3. Est-ce une transaction **critique** (paiement, clôture) ?
4. Est-ce que tuer = **rollback long** (gros xact) ?
5. Y a-t-il une alternative : **statement_timeout**, **throttle**, **read replica** ?

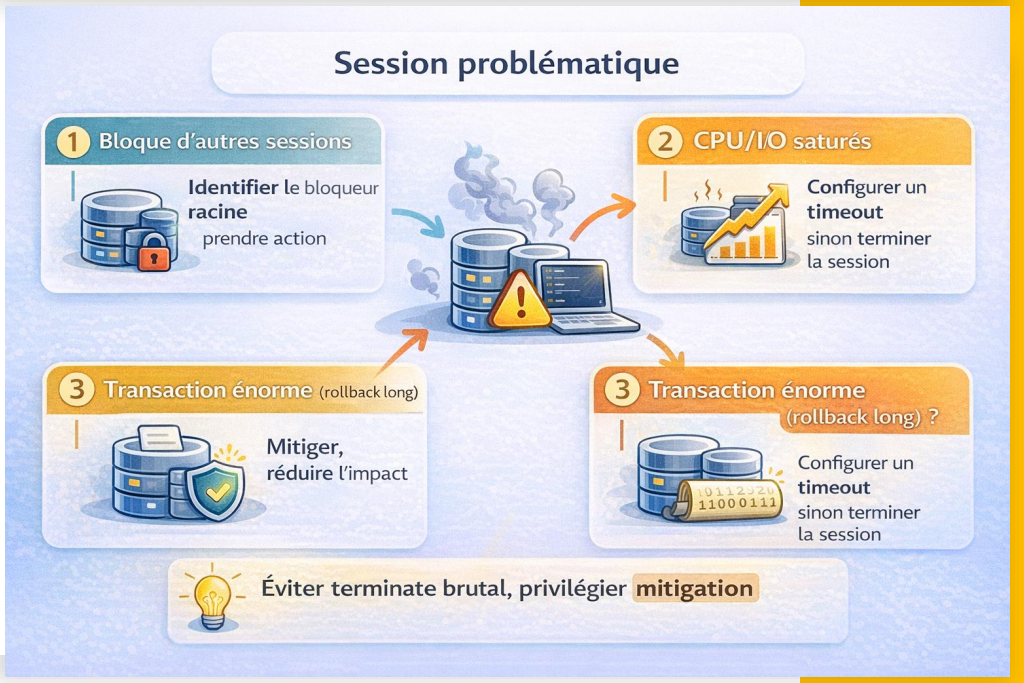


Questions avant d'agir (checklist 30 secondes)

- Est-ce qu'elle **bloque** d'autres sessions ?
- Est-ce qu'elle fait un **DDL** (ALTER, INDEX, VACUUM FULL) ?
- Est-ce une transaction **critique** (paiement, clôture) ?
- Est-ce que tuer = **rollback long** (gros xact) ?
- Y a-t-il une alternative : **statement_timeout, throttle**, read replica ?

Arbre de décision (pratique)

```
Session problématique
  |
  +-- Bloque d'autres sessions ?
  |      |
  |      +-- Oui -> Identifier le bloqueur racine -> action possible
  |
  +-- Consomme CPU/I/O au point de saturer ?
  |      |
  |      +-- Oui -> Peut-on la limiter (timeout) ? sinon terminate
  |
  +-- Transaction énorme (rollback long) ?
         |
         +-- Oui -> éviter terminate brutal, privilégier mitigation
```



Session problématique

1 Bloque d'autres sessions

Identifier le bloqueur **racine**

prendre action

2 CPU/I/O saturés

Configurer un **timeout**

sinon terminer la session

3 Transaction énorme (rollback long)

Mitiger,

réduire l'impact

3 Transaction énorme (rollback long) ?

Configurer un **timeout**

sinon terminer la session

Éviter terminate brutal, privilégier **mitigation**

Bon réflexe : toujours commencer par pg_cancel_backend, puis escalate.

Commandes “propres”

```
SELECT pg_cancel_backend(<pid>);
```

```
SELECT pg_terminate_backend(<pid>);
```

Commandes “propres” dans PostgreSQL

Annuler la requête (moins violent)

```
SELECT pg_cancel_backend(<pid>);
```

Terminer la session (plus violent)

```
SELECT pg_terminate_backend(<pid>);
```

Bon réflexe : toujours commencer par `pg_cancel_backend`, puis escalate.

C'est la commande "Gold Standard". Elle exécute la requête. Elle montre non seulement le temps réel, mais aussi comment PostgreSQL a interagi avec la mémoire (RAM) et le disque.

EXPLAIN (ANALYZE, BUFFERS)

```
EXPLAIN
SELECT * FROM utilisateurs
WHERE ville = 'Paris';
```

Ce qu'on y voit :

- Le coût estimé (arbitraire).
- Le nombre de lignes estimé (*rows*).
- La méthode d'accès (Scan séquentiel vs Index).

```
EXPLAIN (ANALYZE, BUFFERS)
SELECT * FROM utilisateurs
WHERE ville = 'Paris';
```

- **ANALYZE :** Affiche le `actual time`. Si l'estimation prévoyait 10 lignes mais qu'il y en a 1 000 000 en réalité, tes statistiques sont corrompues (un `ANALYZE` de la table s'impose).
- **BUFFERS :** Montre si les données viennent du cache (`shared hit`) ou du disque (`read`).
  - *Un scan rapide avec 0 "read" est bon.*
  - *Un scan lent avec beaucoup de "read" indique une pression sur les entrées/sorties (I/O).*

Voir le “blast radius” (qui est bloqué)

```
SELECT
  a.pid AS blocked_pid,
  a.query AS blocked_query,
  b.pid AS blocking_pid,
  b.query AS blocking_query,
  now() - b.query_start AS blocking_for
FROM pg_stat_activity a
JOIN pg_locks la ON la.pid = a.pid AND NOT la.granted
JOIN pg_locks lb ON lb.locktype = la.locktype
 AND lb.database IS NOT DISTINCT FROM la.database
 AND lb.relation IS NOT DISTINCT FROM la.relation
 AND lb.page IS NOT DISTINCT FROM la.page
 AND lb.tuple IS NOT DISTINCT FROM la.tuple
 AND lb.transactionid IS NOT DISTINCT FROM la.transactionid
 AND lb.classid IS NOT DISTINCT FROM la.classid
 AND lb.objid IS NOT DISTINCT FROM la.objid
 AND lb.objsubid IS NOT DISTINCT FROM la.objsubid
 AND lb.pid <> la.pid
JOIN pg_stat_activity b ON b.pid = lb.pid;
```

Analyser les requêtes bloquantes en PostgreSQL

Trouver les verrous et bloquants avec une requête SQL

```
SELECT blocked_pid, blocked_query,
blocking_pid, blocking_query, now() -
FROM pg_stat_activity a blocking_for
JOIN pg_stat_activity b ON b.pid = lb.pid;
```

1 Identifier le bloqué

pg_stat_activity.pid = pid bloqué

2 Identifier le bloqueur

pg_stat_activity.pid = pid bloquant

blocked_pid	blocked_query	blocking_pid

3 Diagnostiquer le probléme

Analysez la requête bloquante, testez des solutions

1 Identifier le bloqué

pg_stat_activity.pid = pid bloqué

2 Identifier le bloqueur

pg_stat_activity.pid = pid bloquant

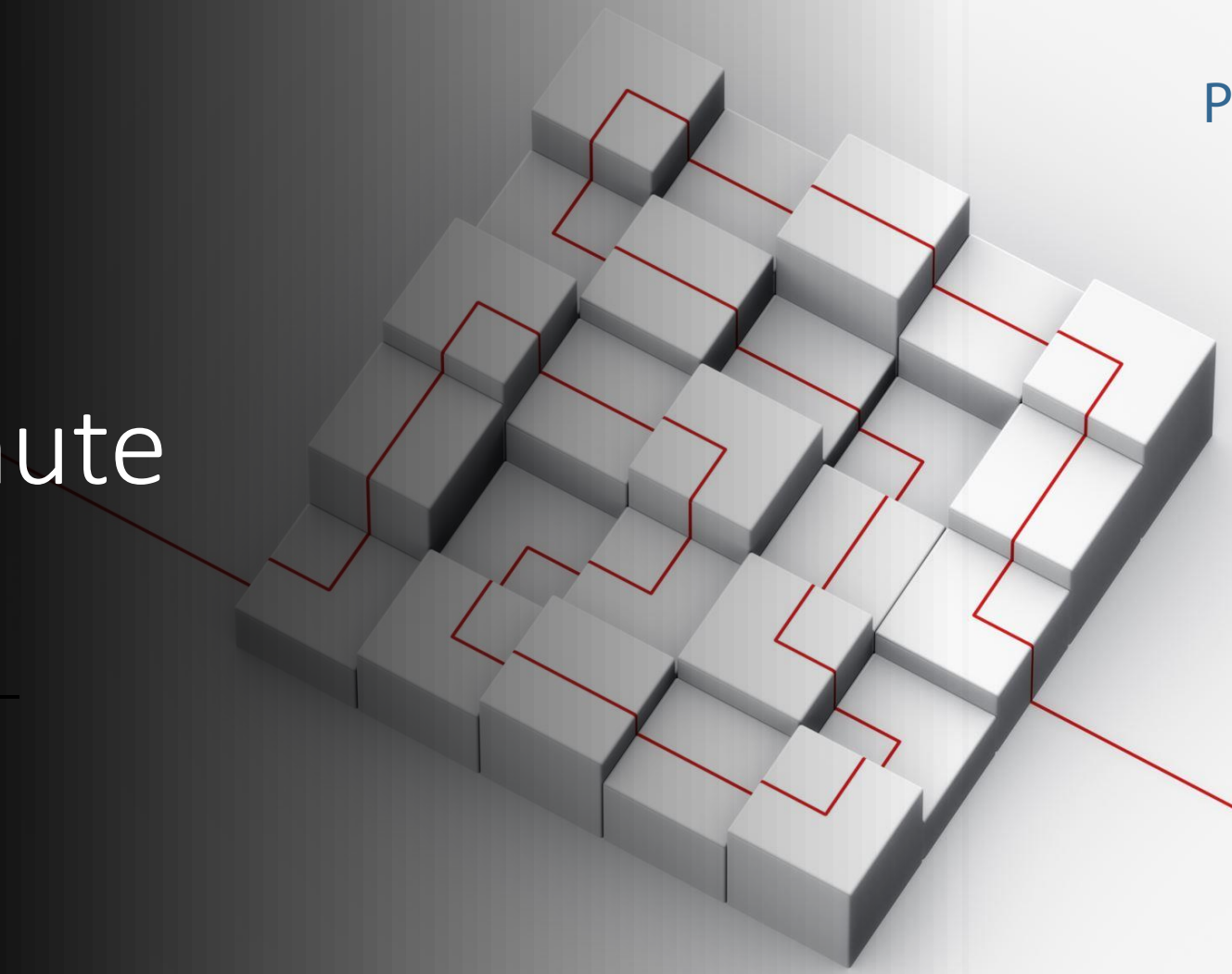
commentaires

Comparatif pour ton support de présentation

Commande	Exécution ?	Impact Prod	Utilité principale
EXPLAIN	Non	Aucun	Vérifier rapidement si un index est utilisé.
EXPLAIN ANALYZE	**Oui**	Dépend de la requête	Mesurer le temps réel et valider les stats.
BUFFERS	**Oui**	Dépend de la requête	Détecter les goulots d'étranglement disque/RAM.

Conseil de DBA : Attention avec `EXPLAIN ANALYZE` sur des requêtes de type `DELETE` ou `UPDATE` en production, car elles seront réellement exécutées ! Il faut toujours les wrapper dans un `ROLLBACK`.

CAS RÉEL — Incident PostgreSQL minute par minute



PostgreSQL

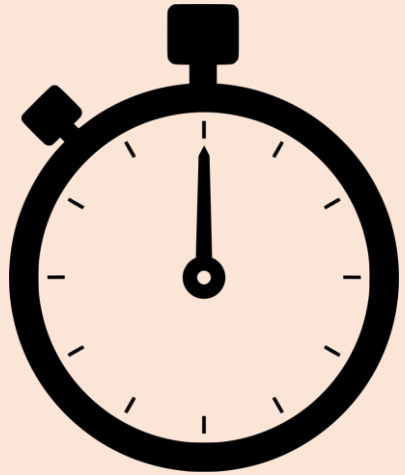
IDEO·LAB

Contexte

- App web + pool (PgBouncer ou driver)
- Déploiement lance `ALTER TABLE` + index **non concurrently**
- Résultat : **lock** + backlog + timeouts

Incident PostgreSQL minute par minute

- T+00 — Alerte
- T+02 — Identifier “waits”
- T+03 — Trouver le bloqueur racine
- T+04 — Évaluer le risque de tuer
- T+05 — Action graduée
- T+07 — Vérifier récupération
- T+20 — Correctif long terme (postmortem)



T+00 – T+02 — Alerte

Symptôme : latence HTTP x10, erreurs 500.

Action DBA (immédiate) T+00

```
SELECT now(), count(*) FROM pg_stat_activity;
SELECT state, count(*) FROM pg_stat_activity GROUP BY state;
```

- Si connexions explosent → pool/clients s'emballent
- Si beaucoup d'`active` → charge réelle, sinon locks/waits

Identifier “waits” T+02

```
SELECT pid, usename, now()-query_start AS dur,
       wait_event_type, wait_event, state, query
FROM pg_stat_activity
WHERE state <> 'idle'
ORDER BY dur DESC
LIMIT 20;
```

- Beaucoup de `wait_event_type = Lock` → priorité : chaîne de blocage

Alerte

Identifier une alerte par DBA

now()	count	state	state
2024-04-24 15:27:47	24	active	
2024-04-24 15:27:47	2500	idle	

Connexions pool saturées !

Charge forte probable

Nombre élevé d'état “active”

Charge faible, mais nombréux locks/waits ?

Analysez les "waits" en utilisant pg_stat_activity

T+03 – T+04 — Alerte

Lancer la requête blocked/blocking

Trouver le bloqueur racine

Lecture

- 1 pid “blocking” apparaît partout → **root blocker**
- souvent DDL, ou transaction oubliée

Évaluer le risque de tuer

- durée de transaction (xact_start)
- nature de la requête (DDL ?)
- “blast radius” (# bloqués)

```
SELECT pid, usename, state,
       now()-xact_start AS xact_dur,
       now()-query_start AS q_dur,
       query
FROM pg_stat_activity
WHERE pid = <blocking_pid>;
```

T+05 – T+07 — Alerte

Action graduée

Cancel d’abord

```
SELECT pg_cancel_backend(<blocking_pid>);
```

Attendre 5–15s, vérifier si la file se vide.

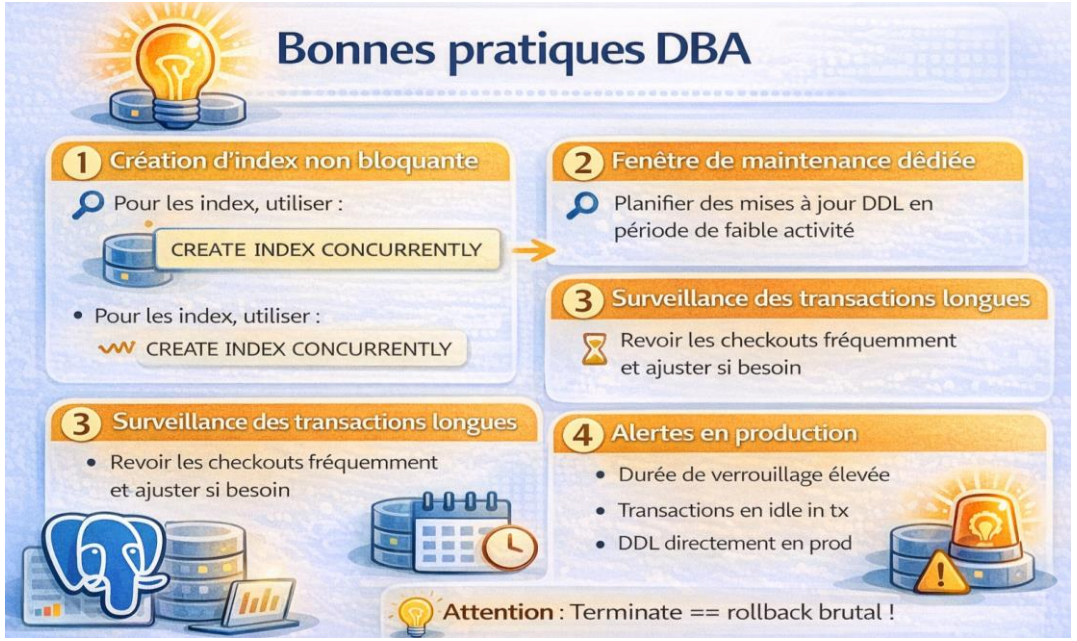
Si prod critique et aucun effet → terminate

```
SELECT pg_terminate_backend(<blocking_pid>);
```

Vérifier récupération

```
SELECT state, count(*) FROM pg_stat_activity GROUP BY state;
SELECT pid, now()-query_start AS dur, wait_event_type, wait_event
FROM pg_stat_activity
WHERE state='active'
ORDER BY dur DESC
LIMIT 10;
```

- baisse des waits lock
- baisse du backlog
- latence app qui redescend



si index : imposer CREATE INDEX CONCURREI

fenêtre de maintenance + gouvernance DDL

checkouts de transactions longues

alertes : lock duration, idle in tx, DDL en prod

T+20 — Correctif long terme (postmortem)