ENTRETIEN DBA ORACLE GESTION INCIDENTS

ULTIMATE GUIDE

BY IDEO-LAB

FEB 2026

GESTION INCIDENTS ORACLE

Je suis une approche en 5 étapes :

1. "Identifier si le problème est global ou session spécifique"
2. "Vérifier les wait events dominants (AWR / ASH)"
3. "Analyser les SQL les plus coûteux (Top SQL by elapsed time)"
4. "Examiner le plan d'exécution (dbms_xplan.display_cursor)"
5. "Vérifier I/O, CPU, latch contention"

Je ne commence jamais par modifier des paramètres sans diagnostic clair.

AGENDA GESTION INCIDENTS

- 0) Je commence par cadrer le problème (obligatoire)
- 1) Phase “Triage” : je classe le symptôme en 1 minute
- 2) Phase “Diagnostic” : je prends le bon outil selon le cas
- 3) Phase “Remédiation” : je choisis un levier, pas dix
- 4) Je conclus par “validation + prévention”

1° ON CADRE LE PROBLEME

0) Je commence par cadrer le problème (obligatoire)

Objectif : éviter de "tuner au hasard".

Questions immédiates :

- *Depuis quand ?* (heure exacte, pic ?)
- *Sur quoi ?* (un module, un batch, une API, un schéma ?)
- *C'est global ou local ?* (toute la base lente ou un seul traitement ?)
- *Changement récent ?* (release applicative, stats, patch, infra, paramètres)
- *SLA ?* (latence, throughput, heures ouvrées)

Outils :

- Logs applicatifs / APM si dispo (AppDynamics, Dynatrace, NewRelic)
- Côté DB : `v$system_event`, `v$sysmetric`, `v$active_session_history`

Décision clé :

- Si **toute la base** est lente → je suspecte **CPU / I/O / contention globale**
- Si **un flux / 1 SQL** est lent → je vais direct sur **Top SQL / plan / stats**

2° ON CLASSE LE SYMPTOME

1) Phase “Triage” : je classe le symptôme en 1 minute

Objectif : savoir si je suis dans CPU, I/O, locks, log file, memory.

A) Est-ce un problème CPU ?

Outils :

- AWR : “DB CPU”, “Host CPU”, “Time Model”
- Temps réel : `v$sysmetric` (CPU Utilization, Average Active Sessions)

Signaux :

- DB CPU très haut + AAS élevé → saturation CPU ou SQL inefficace

B) Est-ce un problème d’I/O ?

Outils :

- AWR : “Instance Activity Stats”, “IOStat by function”
- `v$system_event` waits type I/O (`db file sequential read`, `scattered read` ...)

Signaux :

- Waits I/O dominants + latence disque → index manquant, full scan, storage saturé

C) Est-ce de la contention / locks ?

Outils :

- AWR : “Enqueue activity”, “Top Blocking Sessions”
- `dba_blockers/dba_waiters`, `v$lock`, `v$session`

Signaux :

- `enq: TX - row lock contention` → blocage applicatif
- `enq: HW - contention` → segment header / freelist / ASSM / hot block (selon version)

D) Est-ce lié au redo / commit ?

Outils :

- AWR : “log file sync”, “log file parallel write”, redo size
- `v$log`, `v$log_history`, `v$sysstat` (redo size)

3° ON POSE LE DIAGNOSTIQUE

2) Phase “Diagnostic” : je prends le bon outil selon le cas

Cas 1 — Pic passé : je fais AWR + ASH corrélés

Méthode :

1. Je prends l’intervalle exact du pic (heure début/fin).
2. Dans **AWR**, je regarde :
   - “Top Timed Foreground Events”
   - “SQL ordered by elapsed time / CPU / buffer gets”
   - “Segments by physical reads / logical reads”
3. Ensuite je valide au niveau session avec **ASH** :
   - “Top sessions”
   - “Top SQL_ID”
   - “Top objects”
   - “Wait class”

Outils :

- AWR report (HTML ou TEXT)
- ASH report / `v$active_session_history` (si licence Diagnostic Pack à respecter)

Décisions :

- Si AWR dit “top event = I/O”, je vérifie si c’est **1 SQL** ou **un pattern**.
- Si c’est 1 SQL : SQL tuning.
- Si c’est global : infra/paramètres/contension.

Cas 2 — Problème en cours (NOW) : je passe en temps réel

Méthode :

1. Je mesure **AAS** (Average Active Sessions) et je compare à CPU cores.
2. Je liste les sessions actives + leur wait actuel.
3. J'identifie le top SQL_ID en temps réel.

Outils (sans blabla, concret) :

- `v$session` (status='ACTIVE', wait_class, event)
- `v$session_wait` / `v$session_wait_class`
- `v$sqlarea` / `v$sql` (par elapsed_time, cpu_time)
- `v$active_session_history` (si dispo)

Décision :

- Si j'ai un SQL_ID dominant → je fais un zoom SQL (SQL Monitor / plan réel)
- Si j'ai 10 SQL différents mais même wait → je traite la cause (I/O, locks, log sync…)

Cas 3 — 1 requête lente identifiée : SQL Monitor + plan réel

Méthode :

1. Je récupère SQL_ID.
2. Je regarde le **plan réel** (pas seulement estimé).
3. Je compare :
   - estimated rows vs actual rows
   - join order
   - accès index vs full scan
4. Je regarde si la requête fait du "row-by-row" inutile, ou des sorts/hash énormes.

Outils :

- `DBMS_XPLAN.DISPLAY_CURSOR(format => 'ALLSTATS LAST +PEEKED_BINDS')`
- SQL Monitor (si Tuning Pack)
- `v$sql_plan_statistics_all`

Décision :

- Si cardinalité estimée fausse → stats / histogram / extended stats
- Si index absent → index (ou rewrite)
- Si plan instable → SPM (SQL Plan Management) / baselines

4° ON CHOISIT LE REMEDE

3) Phase “Remédiation” : je choisis un levier, pas dix

Ici il faut être **très “méthode”** : je ne “touche” que ce qui est justifié.

Levier A — Problème de stats / cardinalité

Quand : estimated rows très loin du réel, mauvais join order.

Action :

- stats ciblées table/index
- histogram si distribution skew
- extended stats sur colonnes corrélées (si besoin)

lidation :

- nouveau plan + baisse buffer gets / elapsed time

evier B — Index / accès aux données

uand : full scan coûteux, filtre sélectif non indexé, nested loop mal alimenté.

ction :

- index composite aligné sur predicates + order by (si pertinent)
- vérifier impact DML (coût d’index)
- éviter index “fantôme” inutiles

Levier B — Index / accès aux données

Quand : full scan coûteux, filtre sélectif non indexé, nested loop mal alimenté.

Action :

- index composite aligné sur predicates + order by (si pertinent)
- vérifier impact DML (coût d'index)
- éviter index "fantôme" inutiles

Validation :

- logical reads en baisse, plan stable

Levier C — Réécriture SQL (souvent le meilleur ROI)

Quand : fonctions sur colonnes, OR multiples, subqueries corrélées, DISTINCT inutile.

Action :

- rendre les prédicats "sargable"
- remplacer OR par UNION ALL (si utile)
- réduire colonnes, éviter SELECT *

Validation :

- plan plus simple, temps CPU en baisse

Levier D — Concurrency / locks

Quand : `enq: TX` dominant, blocages.

Action :

- identifier bloqueur/attenteur + objet
- corriger pattern applicatif (transactions trop longues, ordre de verrous)
- parfois : index sur FK (classique), réduction contention hot blocks

lidation :

temps d'attente lock en baisse, throughput stable

vier E — Redo / commit (`log file sync`)

uand : commits très fréquents + latence.

tion :

- batch commits (côté app)
- storage redo logs / séparations disques

vérifier LGWR / latency

lidation :

5° “validation + prévention”

4) Je conclus par “validation + prévention”

Méthode de fin :

- Je re-mesure les mêmes indicateurs (AWR delta / métriques)
- Je documente RCA + action + métrique avant/après
- Je mets un garde-fou : baseline / alerte / SPM / monitoring

Phrase “pro” à sortir :

> “Je valide toujours par des métriques : elapsed time, CPU time, logical reads, physical reads, AAS et wait profile. Sinon on fait de la croyance.”

🧠 Bonus : si on te coupe et qu'on te dit "OK mais concrètement, vous lancez quoi ?"

Tu réponds sans hésiter :

- "Je récupère l'intervalle, je sors un AWR, je lis Top Events + Top SQL."
- "Si c'est en live : je regarde `v$session` pour les actifs et leur wait, puis je remonte au SQL_ID dominant."
- "Sur ce SQL_ID : je sors le plan réel via `dbms_xplan.display_cursor ALLSTATS LAST` et je compare estimé vs réel."

⚠️ Important en entretien : mention "packs/licences"

Tu peux ajouter une phrase intelligente :

> "Si AWR/ASH/SQL Monitor ne sont pas autorisés (licence), j'utilise Statspack, v$ views, et je fais du sampling via scripts maison."

Ça montre que tu connais les contraintes.

EXPERIENCES SUR ORACLE

1️⃣ Parlez-nous de votre expérience Oracle en production

🎯 Ce qu'ils veulent tester

- Taille des bases
- Criticité
- Exadata / RAC / Data Guard ?
- Incident réel vécu

✅ Réponse type (à adapter)

> J'ai travaillé sur des environnements Oracle critiques en production, avec des bases entre 2 et 20 To.
> J'ai géré du RAC 2–4 nœuds, du Data Guard en mode sync/async, et des environnements Exadata.
>
> Mon rôle couvrait :
>
> - "Performance tuning (AWR / ASH / ADDM)"
> - "Gestion incidents P1"
> - "Optimisation SQL"
> - "Capacity planning"
> - "Patch PSU/RU"
> - "Automatisation via shell + SQL*Plus"
>
> J'ai notamment résolu un incident de saturation redo transport où le gap a été rattrapé en 12 minutes après analyse réseau + SRL.

👉 Important : toujours donner **un exemple concret**.

AIRBUS

1️⃣ AIRBUS (2021–2023)

📍 Toulouse / Hambourg

📄 Référence CV page 2 cv_oneill_DBA_Nov_2025

🎯 Cas crédible : Saturation I/O sur base avionique critique

🚀 Contexte

Projet interne avionique – environnement hybride PostgreSQL / Oracle pour modules de simulation cockpit.

racle utilisé pour :

- Historique événements vols
- Référentiels techniques avion
- Moteur de corrélation anomalies

Base Oracle 19c

- 8 To
- Linux RHEL
- SAN haute performance
- SLA très strict (latence < 200 ms)

🔥 Incident

Pic brutal de latence sur module d'analyse temps réel.

Temps de réponse x4.

🧠 Méthode appliquée

1. Intervalle précis identifié (09h12 → 09h28).
2. AWR sur la plage.
3. Top Timed Events = `db file sequential read`
4. Top SQL_ID identifié.
5. Analyse plan réel :
    - Nested Loop
    - Mauvaise cardinalité
    - Full scan sur table partitionnée

🎯 Cause racine

Statistiques obsolètes sur partition nouvellement alimentée.

Cardinalité estimée 12 000 lignes → réel 9 millions.

🛠️ Action

- Stats ciblées sur partitions récentes
- Ajout histogramme
- Création index composite aligné sur prédicat

📈 Résultat

- Temps requête : 2.8s → 190ms
- Physical reads divisés par 12
- Stabilisation SLA

👉 En entretien : tu montres maîtrise AWR + partitionner


QUESTIONS / REPONSES

❓ Piège 1

> Pourquoi un `db file sequential read` n'est PAS forcément un problème disque ?

✅ Réponse senior

`db file sequential read` = lecture mono-bloc, généralement accès index.

Ce n'est pas forcément un problème disque :

- Peut être normal si index lookup massif
- Peut être dû à mauvais plan (Nested Loop inadapté)
- Peut être causé par mauvaise cardinalité → accès répétés

Je corrèle toujours :

- Temps moyen du wait
- Nombre d'occurrences
- Physical read latency côté OS

👉 Si latence disque = 1 ms → ce n'est pas le storage.

QUESTIONS / REPONSES

❓ Piège 2

> Pourquoi ne pas simplement augmenter le buffer cache ?

✅ Réponse

Augmenter le buffer cache masque parfois le problème mais ne corrige pas :

- Mauvaise cardinalité
- Mauvais join order
- Index absent
- Mauvaise répartition données

Je corrige la cause logique avant d'augmenter la mémoire.

❓ Piège 3

> Pourquoi utiliser histogramme et pas juste gather stats ?

✅ Réponse

Parce que la distribution était skewed.

Sans histogram :

Optimiseur suppose distribution uniforme.

Avec histogram :

Optimiseur comprend que 1 valeur = 70% des lignes.

👉 Ça montre vraie connaissance CBO.

DOCTOLIB

2 DOCTOLIB (2015–2017)

📍 Nantes / Berlin

📄 Référence CV page 4 cv_oneill_DBA_Nov_2025

Même si majoritairement PostgreSQL, on peut présenter un cas Oracle plausible sur système legacy.

Cas crédible : Incident de contention `log file sync`

Contexte

odule historique Oracle encore actif pour synchronisation données partenaires ospitaliers.

racle 12c

- 3 To
- Forte volumétrie transactions
- Environnement HDS (sécurité maximale)

🔥 Incident

Pendant pic prise de RDV :

- CPU OK
- I/O OK
- Mais latence globale forte
- AWR → Top Event = `log file sync`

🧠 Méthodologie

1. AWR sur pic
2. Vérification redo size / commits
3. Analyse `v$sysstat` :
    - commits très fréquents (pattern applicatif)
4. LGWR latency confirmée

🎯 Cause

Application commit à chaque ligne insérée (pattern ORM mal optimisé).

🛠️ Actions

- Discussion équipe dev
- Passage à commit batch toutes les 500 lignes
- Redo logs déplacés sur stockage plus rapide
- Ajustement taille redo logs

Résultat

- `log file sync` divisé par 6
- TPS doublé
- Pic absorbé sans scaling matériel

Très crédible en contexte santé haute transaction.


QUESTIONS / REPONSES

❓ Piège 4

> Comment savoir si c'est l'application ou le disque ?

✅ Réponse méthodique

Je compare :

- `log file sync`
- `log file parallel write`

Si `parallel write` rapide (< 2 ms) mais `log file sync` élevé → problème applicatif (commits trop fréquents).

Si les deux sont élevés → problème storage redo.

❓ Piège 5

> Pourquoi augmenter la taille des redo logs aide ?

✅ Réponse

Ça réduit la fréquence des log switches.

Mais attention :

- Ça ne réduit pas `log file sync` directement.
- Ça évite pics liés aux checkpoints trop fréquents.

👉 Important : ne pas confondre redo log size et commit latency.

LOCKHEED MARTIN

🎯 Cas critique : Incident Data Guard – Gap redo 12 minutes

✈ Contexte

Systèmes avioniques F-35.

Oracle 11g RAC + Data Guard.

Standby site distant sécurisé.

Incident

rte : redo transport lag.

ansport bloqué 12 minutes.

Méthode

. Vérification `v$dataguard_stats`

Gap identifié

. Vérification réseau (latence inter-site)

. Analyse alert log standby

🎯 Cause

SRL mal dimensionnés + pic redo inhabituel suite à batch critique.

🛠️ Actions

- Ajout SRL par thread
- Ajustement taille redo logs
- Optimisation paramètre `NET_TIMEOUT`
- Monitoring proactif lag via script interne

📈 Résultat

- Transport stabilisé
- RTO garanti
- DR validé en simulation failover

👉 En entretien, ça impressionne énormément.

EMC DELL

🎯 Cas performance massif – Tuning client banc

🏦 Contexte

Client bancaire.

Oracle 10g.

Base 15 To.

Stockage VMAX + SRDF.

🔥 Incident

Batch nocturne de 4h passé à 11h.

Menace sur ouverture marché matin.

🔥 Incident

Batch nocturne de 4h passé à 11h.

Menace sur ouverture marché matin.

🧠 Méthode

1. AWR comparaison veille / aujourd'hui
2. Top SQL CPU
3. Analyse plan
4. Cardinalité erronée suite à skew données

🎯 Cause

Absence histogram sur colonne très déséquilibrée.

Optimiseur choisissait mauvais index.

🛠️ Actions

- Création histogram ciblé
- Gather stats table spécifique
- Mise en place SQL Plan Baseline
- Test via simulateur charge (outil interne C++)

✅ Résultat

- Batch revenu à 3h45
- Plan stabilisé
- AWR validé

📁 Ça montre ancienneté + profondeur Oracle pure.

DBA AU QUOTIDIEN

3️⃣ Différence entre AWR et ASH ?

✅ Réponse

> AWR = vue macro périodique (snapshots, charge globale)
>
> ASH = granularité fine (activité session-level en temps réel)
>
> ASH permet d'analyser un pic de 3 minutes que l'AWR peut lisser.

4️⃣ Comment fonctionne Data Guard ?

✅ Réponse concise et propre

> Data Guard repose sur le transport des redo logs vers une base standby.
>
> Modes :
>
> - "SYNC (Maximum Protection / Availability)"
> - "ASYNC (Maximum Performance)"
>
> Les SRL (Standby Redo Logs) sont essentiels pour éviter les gaps.
>
> Le redo transport peut être via LGWR ou ARCH.
>
> En cas d'incident, on peut :
>
> - "Switchover (planned)"
> - "Failover (emergency)"

👉 Si tu mentionnes SRL par thread en RAC = très bon point.

5️⃣ Qu’est-ce qu’un latch ? Une contention ?

✅ Réponse

> Un latch est un mécanisme léger de synchronisation mémoire.
>
> Une contention apparaît lorsque plusieurs sessions veulent accéder à la même ressource partagée.
>
> Exemple classique :
>
> - “library cache latch”
> - “buffer cache latch”

👉 Ne pas confondre avec locks.

6 Comment optimiser une requête lente ?

✅ Réponse structurée

> 1. “Vérifier statistiques (DBMS_STATS)”
> 2. “Analyser plan d’exécution”
> 3. “Vérifier cardinalité estimée vs réelle”
> 4. “Examiner index manquants”
> 5. “Vérifier full scan inutile”
> 6. “Éviter hints en premier recours”
>
> Je privilégie toujours correction structurelle plutôt que patch rapide.

7️⃣ Que surveillez-vous quotidiennement ?

✅ Réponse

- Tablespaces (free space)
- FRA usage
- Alert log
- Redo generation rate
- Sessions bloquées
- Top wait events
- Archive log gap
- RMAN backup status
- ASM disk usage
- CPU / IOPS

👉 Ça montre vision production.

8 Exadata : qu'est-ce qui change ?

✅ Réponse

> Exadata apporte :
>
> - "Smart Scan (offload SQL vers storage)"
> - "Hybrid Columnar Compression"
> - "Storage Indexes"
> - "Flash Cache"
> - "IORM"
>
> Le tuning change : on surveille offload efficiency.

👉 Mentionne : cellcli, metrics offload%.

9️⃣ Comment gérez-vous un incident critique à 3h du matin ?

✅ Réponse

> 1. “Stabiliser (éviter aggravation)”
> 2. “Identifier scope”
> 3. “Communiquer rapidement (bridge call)”
> 4. “Appliquer workaround si nécessaire”
> 5. “Root cause analysis après stabilisation”
>
> Je privilégie toujours la restauration de service avant l’analyse profonde.

👉 Très apprécié.

🔟 Quelle est la différence entre RAC et Data Guard ?

✅ Réponse claire

> RAC = haute disponibilité locale (scale horizontal, même base)
>
> Data Guard = reprise après sinistre (réplication distante)
>
> RAC protège contre panne serveur,
> Data Guard protège contre panne site.

👉 Simple, propre, efficace.

DATA GUARD

🔷 1️⃣ Introduction & Architecture

🎯 Objectif

Oracle Data Guard permet :

- 🔒 Haute disponibilité (HA)
- 🏦 Disaster Recovery (DR)
- 📉 Réduction RPO (Recovery Point Objective)
- ⚡ Réduction RTO (Recovery Time Objective)

Il protège contre :

- Panne serveur
- Corruption storage
- Panne site complet
- Erreur humaine (avec Flashback)

🏗️ Architecture générale

Composants principaux

- Primary Database
- Standby Database
- Redo Transport Services
- Redo Apply Services
- Broker (optionnel)

Types de Standby

1 Physical Standby

- Copie binaire bloc-à-bloc
- Redo apply via MRP (Managed Recovery Process)
- Mode le plus courant
- Compatible Active Data Guard (lecture seule)

2 Logical Standby

- Rejoue les SQL (SQL Apply)
- Permet divergence contrôlée
- Plus complexe à maintenir

3 Snapshot Standby

- Standby temporairement en RW
- Reversible
- Utile pour tests

🔷2️⃣ Redo Transport & Modes de Protection

🎯 Redo Transport

Le Primary envoie les redo logs vers la standby via :

- LGWR (temps réel)
- ARCH (après archivage)

Paramètres clés

- `LOG_ARCHIVE_DEST_n`
- `SYNC / ASYNC`
- `AFFIRM / NOAFFIRM`
- `NET_TIMEOUT`

🔒 Modes de protection

1️⃣ Maximum Protection

- Zéro perte de données
- SYNC + AFFIRM
- Si standby indisponible → primary s'arrête

2️⃣ Maximum Availability

- SYNC + AFFIRM
- Continue temporairement si standby indisponible

3️⃣ Maximum Performance (le plus utilisé)

- ASYNC
- Pas d'impact latence
- RPO dépend réseau

🔷3️⃣ Redo Apply & Monitoring

🎯 Redo Apply (Physical Standby)

Processus principal :

- RFS (Remote File Server)
- MRP (Managed Recovery Process)
- Application des redo dans l'ordre SCN

📊 Monitoring essentiel

Vues importantes

```SQL
v$dataguard_stats
v$archive_gap
v$managed_standby
v$database
```

Indicateurs critiques

- **Transport Lag**

  → redo non encore reçu

- **Apply Lag**

  → redo reçu mais pas encore appliqué

- **Gap**

  → archive log manquant

🎯 Exemple analyse incident

1. Vérifier transport lag
2. Vérifier apply lag
3. Vérifier alert log standby
4. Vérifier SRL (Standby Redo Logs)
5. Vérifier réseau

🔷4️⃣ RAC + Data Guard

🎯 Spécificités RAC

Chaque instance RAC a :

- Son propre thread redo
- Ses propres redo logs
- Ses propres SRL

Règle critique

Nombre de SRL ≥ nombre de redo logs par thread + 1

Sinon :

- Transport bloqué
- Lag
- GAP

🔷 5️⃣ Switchover vs Failover

🔄 Switchover

- Planned
- Zéro perte
- Rôles inversés proprement

Utilisé pour :

- Maintenance
- Patch
- Migration site

🚨 Failover

- Non planifié
- Suite incident primaire
- Peut entraîner perte données (selon mode)

🎯 Fast-Start Failover (FSFO)

- Automatique
- Via Data Guard Broker
- Observer requis

🔷 6 Active Data Guard

Permet :

- Lecture seule sur standby
- Reporting
- Backups RMAN offloadés

⚠ Licence supplémentaire requise.

Avantages

- Décharge primary
- Reporting sans impacter production

🔷 7️⃣ Bonnes Pratiques Senior

🧱 Infrastructure

- Réseau dédié redo transport
- Latence < 5 ms en SYNC
- Redo logs correctement dimensionnés

📦 Redo Logs

- Taille adaptée à la volumétrie
- Éviter log switch trop fréquents
- Même taille Primary / Standby

🔍 Monitoring Proactif

- Alertes lag > X secondes
- Surveillance redo rate
- Vérification quotidienne gap

🔷 8 Problèmes Fréquents & Diagnostic

❌ `ORA-16047`

Mismatch DB_UNIQUE_NAME

❌ Transport lag élevé

- Réseau
- SRL mal configurés
- Saturation redo

❌ Apply bloqué

- Archive log manquant
- Corruption
- Manque espace FRA

🔷 9️⃣ Questions Pièges Entretien

❓ Différence transport lag / apply lag ?

Transport = redo pas encore arrivé

Apply = redo arrivé mais non appliqué

❓ Pourquoi SYNC peut impacter performance ?

Primary attend ACK standby avant commit.

❓ Peut-on avoir RAC des deux côtés ?

Oui, Primary RAC + Standby RAC possible.

❓ Peut-on avoir Data Guard sans archive log ?

Non. ARCHIVELOG obligatoire.

RAC

🔷 1️⃣ Introduction – Qu'est-ce que RAC ?

🎯 Objectif

Oracle RAC permet :

- 🔁 Haute disponibilité locale (HA)
- 📈 Scalabilité horizontale
- ⚡ Répartition de charge
- 🔒 Tolérance panne nœud

👉 Contrairement à Data Guard (DR inter-site),

RAC = plusieurs instances sur une même base physique partagée.

🔷 2️⃣ Architecture RAC

🏗️ Composants principaux

1️⃣ Multiple instances Oracle

- 1 instance par nœud
- Même base de données
- Threads redo séparés

2️⃣ Shared Storage

- ASM (recommandé)
- SAN / Exadata
- Tous les nœuds accèdent aux mêmes datafiles

3️⃣ Clusterware

- Gère membership cluster
- Détecte panne nœud
- Gère VIP, SCAN, services

4️⃣ Interconnect

- Réseau privé dédié
- Latence très faible
- Essentiel pour cache fusion

🔷 3️⃣ Le cœur du RAC : Cache Fusion

🎯 Principe

Dans RAC :

- Les buffers sont partagés logiquement
- Si instance A modifie un bloc,
- Instance B peut le récupérer via interconnect

👉 Pas besoin d'écrire sur disque pour cohérence.

🔥 Fonctionnement simplifié

1. Instance A modifie bloc
2. Instance B demande bloc
3. Global Cache Service (GCS)
4. Bloc envoyé via interconnect

📊 Waits typiques RAC

- `gc buffer busy acquire`
- `gc cr request`
- `gc current request`

👉 Si élevés → contention inter-instance.

🔷4️⃣ Services & Load Balancing

🎯 Bonnes pratiques

On ne connecte jamais tout sur toutes les instances.

On crée des **services applicatifs** :

Exemple :

- Service OLTP → Instance 1
- Service Reporting → Instance 2

Permet :

- Répartition charge
- Isolation workload

SCAN (Single Client Access Name)

- 1 nom DNS
- 3 IP virtuelles
- Répartition automatique connexions

🔷 5 Redo & Threads en RAC

Chaque instance RAC a :

- Son propre thread redo
- Ses propres redo logs
- Son propre LGWR

Important :

En RAC + Data Guard → SRL par thread obligatoires.

🔷6️⃣ Gestion des verrous & contention

RAC introduit :

- Global Enqueue Service (GES)
- Global Cache Service (GCS)

Problème typique : Hot Block

Si plusieurs instances modifient la même ligne :

- Ping permanent de bloc
- Forte latence
- Waits gc élevés

Solutions :

- Partitionnement
- Hash partitioning
- Affinité service
- Séparation workload

🔷7️⃣ Haute Disponibilité

🎯 Panne d'un nœud

Si instance 1 tombe :

- Instance 2 continue
- Sessions reconnectées via FAN
- Application doit supporter TAF / Application Continuity

Paramètres importants

- TAF (Transparent Application Failover)
- Application Continuity (12c+)
- Fast Application Notification (FAN)

🔷 8 Monitoring RAC

Vues clés

```SQL
gv$instance
gv$session
gv$active_session_history
gv$system_event
```

(Préfixe g = global)

AWR spécifique RAC

- Global AWR
- AWR par instance
- Comparaison inter-instance

Métriques critiques

- Global Cache Efficiency
- Interconnect latency
- gc waits %
- CPU par instance
- Load imbalance

🔷9️⃣ Cas typique d'entretien

🔥 Problème : CPU élevé sur une instance seulement

Méthode :

1. Vérifier services actifs
2. Vérifier sessions par instance
3. Vérifier top SQL par instance
4. Vérifier skew workload

🔥 Problème : gc buffer busy acquire élevé

Méthode :

1. Identifier objet chaud
2. Vérifier segment stats
3. Vérifier pattern applicatif
4. Vérifier absence partitionnement

🔷 🔟 RAC vs Single Instance

Aspect	Single	RAC
Scalabilité	Verticale	Horizontale
Complexité	Faible	Élevée
Coût licence	Plus faible	Élevé
HA	Limitée	Nœud tolérant

🔷 1️⃣1️⃣ RAC + Exadata

En Exadata :

- Smart Scan
- Flash Cache
- IORM
- Haute bande passante interconnect

Attention :

RAC mal configuré peut créer plus de contention que bénéfice.

🔷 1️⃣2️⃣ Problèmes fréquents

❌ Interconnect saturé

→ gc waits explosent

❌ Mauvaise distribution services

→ Une instance surchargée

❌ Hot sequences

→ Séquences mal configurées

Solution :

`CACHE` élevé sur sequences.

🔷 1️⃣3️⃣ Bonnes pratiques senior

✓ Interconnect dédié 10/25/40 Gb

✓ Services bien définis

✓ Partitionnement si écriture concurrente

✓ Monitoring gc waits

✓ Tester failover régulièrement

✓ Vérifier skew charge

🔷 14 Questions pièges entretien

❓ Différence GCS et GES ?

GCS = blocs

GES = verrous

❓ Pourquoi RAC peut dégrader performance ?

- Contention inter-instance
- Mauvaise affinité services
- Hot blocks
- Interconnect lent

❓ Peut-on avoir RAC des deux côtés en Data Guard ?

Oui.

Primary RAC → Standby RAC.

❓ RAC protège-t-il contre perte site ?

Non.

RAC = HA locale uniquement.

Pour DR → Data Guard.

🎯 Ce qui impressionne en entretien

- ✓ Parler de Cache Fusion
- ✓ Parler de gc waits
- ✓ Parler de services
- ✓ Parler d'interconnect
- ✓ Mentionner hot blocks & partitionnement
- ✓ Différencier HA locale vs DR