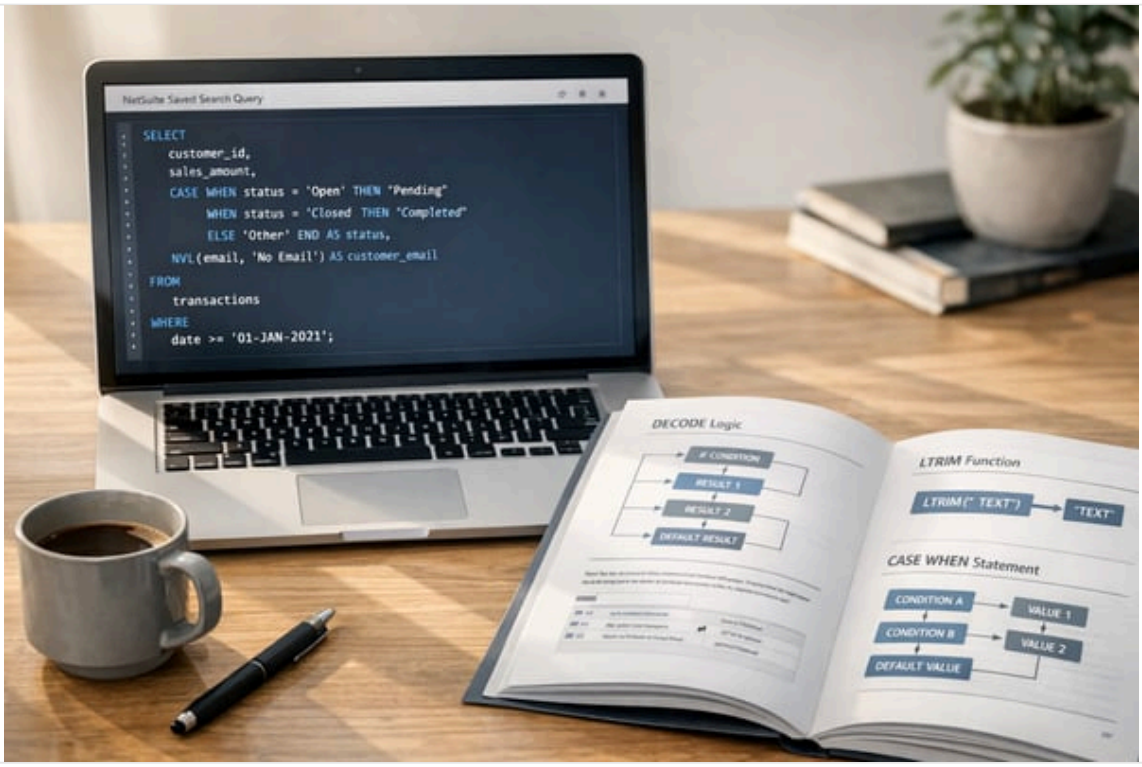
Formules de recherche enregistrée NetSuite : NVL, CASE, DECODE, LTRIM

By houseblend.io Publié le 22 avril 2026 40 min de lecture



Résumé analytique

La fonctionnalité **Saved Search** (recherche enregistrée) de NetSuite est un outil de requête intégré puissant qui permet aux utilisateurs d'extraire des rapports et des tableaux de bord personnalisés à partir de leurs données ERP. Le **champ de formule**, qui prend en charge de nombreuses expressions SQL Oracle, est au cœur de sa flexibilité. Ce rapport propose une exploration approfondie de quatre fonctions de formule clés : **NVL**, **CASE WHEN**, **DECODE** et **LTRIM**. Chacune de ces fonctions répond à des besoins courants de manipulation de données au sein des recherches enregistrées. Par exemple, NVL remplace les valeurs nulles ou vides par des valeurs par défaut (Source: docs.oracle.com) (Source: suiterep.com), CASE WHEN permet une logique conditionnelle de type SQL (Source: docs.oracle.com) (Source: www.houseblend.io), DECODE fournit une équivalence en ligne de mappages de cas simples (Source: docs.oracle.com) (Source: www.houseblend.io), et LTRIM permet de supprimer les caractères de début indésirables des chaînes (Source: docs.oracle.com) (Source: www.netsuiterp.com).

Nous examinons la syntaxe de chaque fonction, son comportement documenté par Oracle et son utilisation pratique dans NetSuite. Pour NVL, nous citons la définition d'Oracle (« si expr1 est nul, alors NVL renvoie expr2, sinon expr1 » (Source: docs.oracle.com) et illustrons son utilisation dans les recherches enregistrées (par exemple, `NVL({quantitycommitted}, 0)` pour remplacer les quantités vides par zéro (Source: suiterep.com). Pour CASE WHEN, nous expliquons les variantes *simple* et *recherchée* de CASE (Source: docs.oracle.com), et fournissons des exemples tels que la classification des commandes importantes/petites via `CASE WHEN {amount} > 10000 THEN 'Large' ELSE 'Small' END` (Source: www.houseblend.io). Nous analysons ensuite le comportement de DECODE selon Oracle (comparaison d'une expression à plusieurs valeurs) (Source: docs.oracle.com) et montrons comment il peut mapper des valeurs de code (par exemple, `DECODE({status}, 'F', 'Fulfilled', 'P', 'Pending', 'Other')`) (Source: www.houseblend.io). LTRIM est abordé en détaillant comment il supprime les caractères de début d'une chaîne (Source: docs.oracle.com), avec des exemples d'utilisation comme `LTRIM('000123','0')` renvoyant `'123'` (Source: www.netsuiterp.com).

Tout au long de ce document, nous basons notre discussion sur des sources faisant autorité. La documentation Oracle et les guides SuiteAnalytics établissent les définitions de base (Source: docs.oracle.com) (Source: docs.oracle.com) (Source: docs.oracle.com) (Source: docs.oracle.com). Les blogs des fournisseurs de solutions NetSuite ajoutent du contexte : le blog SuiteRep de Stephen Lemp montre une utilisation typique de NVL (Source:

suiterep.com) (Source: suiterep.com), le **Anchor Group** explique un cas d'utilisation de NVL pour les recherches d'inventaire (Source: www.anchorgroup.tech) (Source: www.anchorgroup.tech), et l'analyse de **Houseblend** (2026) offre des exemples concrets de CASE et NVL dans les recherches professionnelles (Source: www.houseblend.io) (Source: www.houseblend.io).

Les principales conclusions et recommandations sont résumées comme suit :

- **Gestion des valeurs nulles (NVL/NVL2/COALESCE) :** Utilisez NVL pour fournir des valeurs par défaut lorsque les champs sont vides (Source: docs.oracle.com) (Source: suiterep.com). Pour la logique ternaire, NVL2 peut être utilisé (si-non-nul vs si-nul) (Source: suiterep.com); pour plusieurs solutions de repli, COALESCE est pris en charge pour renvoyer la première expression non nulle (Source: docs.oracle.com).
- **Logique conditionnelle (CASE vs DECODE) :** CASE WHEN et DECODE peuvent tous deux implémenter des conditions. CASE est conforme à la norme ANSI et prend en charge les prédicats complexes (Source: docs.oracle.com). DECODE est spécifique à Oracle et plus simple (utile pour les mappages d'égalité directe) (Source: docs.oracle.com), bien qu'il traite deux NULL comme étant égaux (Source: docs.oracle.com). En pratique, CASE WHEN est plus flexible (prenant en charge les conditions par plage, la logique booléenne complexe) tandis que DECODE est concis pour les recherches directes (Source: www.houseblend.io) (Source: www.houseblend.io).
- **Manipulation de chaînes (LTRIM) :** Utilisez LTRIM pour supprimer les caractères indésirables du côté gauche du texte (par défaut : espaces) (Source: docs.oracle.com). Par exemple, `LTRIM(' test')` donne `'test'` (Source: www.netsuiterp.com). RTRIM est disponible de la même manière pour d'autres opérations de rognage.
- **Considérations sur les performances :** L'intégration de formules complexes (en particulier plusieurs blocs CASE/DECODE imbriqués) dans les recherches enregistrées peut ralentir les performances des requêtes. L'optimisation en utilisant des filtres intégrés lorsque cela est possible peut considérablement améliorer la vitesse (un cas a vu l'exécution passer de 2 minutes à 10 secondes (Source: www.houseblend.io). Une indexation appropriée, l'utilisation appropriée de filtres numériques/de date, et même le déplacement de la logique vers SuiteAnalytics Workbook ou SuiteQL sont recommandés pour les grands ensembles de données (Source: www.houseblend.io) (Source: www.houseblend.io).
- **Études de cas :** Des exemples concrets démontrent l'impact d'une utilisation réfléchie des formules. Un fournisseur de soins de santé a utilisé des recherches basées sur des formules pour identifier les services non facturés, réduisant ainsi la charge de travail de révision des factures (Source: www.houseblend.io). Un détaillant a mis en place des formules CASE pour signaler les articles à faible inventaire (en les étiquetant « Recommander » ou « Vérifier la quantité »), automatisant ainsi les alertes de stock (Source: www.houseblend.io). À l'inverse, une recherche complexe multi-filiale a expiré jusqu'à ce qu'elle soit réimplémentée via SuiteQL (Source: www.houseblend.io).
- **Orientations futures :** La feuille de route de NetSuite met l'accent sur l'intégration analytique et l'IA. La version 2026.1 introduit des résumés de rapports pilotés par l'IA et des connecteurs vers des services d'IA externes (Source: netsuitechangelog.com) (Source: projectsalsa.co.nz). Les architectes avertis devraient anticiper des fonctionnalités telles que la création de formules assistée par l'IA et une utilisation accrue des classeurs SuiteAnalytics parallèlement aux recherches enregistrées (Source: www.houseblend.io) (Source: netsuitechangelog.com).

En résumé, en tirant parti efficacement de NVL, CASE WHEN, DECODE et LTRIM, les administrateurs NetSuite peuvent créer des recherches enregistrées robustes et flexibles. Ces capacités, ancrées dans Oracle SQL, fournissent un conditionnement de données sophistiqué directement au sein de NetSuite (Source: docs.oracle.com) (Source: docs.oracle.com). Les sections suivantes détaillent chaque fonction, avec des exemples, des avis d'experts et des références à la fois à la documentation officielle d'Oracle et aux sources d'experts NetSuite.

Introduction

Contexte sur les recherches enregistrées NetSuite

NetSuite est une plateforme ERP cloud complète qui intègre les finances, la chaîne d'approvisionnement, le CRM et d'autres modules (Source: www.houseblend.io) (Source: projectsalsa.co.nz). Une fonctionnalité essentielle de NetSuite est sa fonctionnalité **Saved Search** (recherche enregistrée) : des requêtes de base de données définies par l'utilisateur qui peuvent appliquer des critères, des formules et des résumés personnalisés pour générer des rapports et des tableaux de bord sur mesure (Source: www.houseblend.io) (Source: docs.oracle.com). Les recherches enregistrées permettent aux utilisateurs professionnels de poser des questions telles que « Quels clients ont des soldes en retard ? » ou « Combien de commandes client le représentant commercial X a-t-il clôturées le trimestre dernier ? » sans quitter NetSuite (Source: www.houseblend.io). Surtout, les formules de recherche enregistrée permettent d'intégrer une logique de type SQL directement dans la recherche, permettant des transformations de données complexes et une sortie conditionnelle (Source: docs.oracle.com) (Source: docs.oracle.com).

En coulisses, les recherches enregistrées reposent sur la base de données Oracle de NetSuite. Comme l'explique la documentation de NetSuite, « les expressions SQL que vous saisissez dans les formules de champ appellent la base de données Oracle pour évaluer la fonction, et ces fonctions sont maintenues par Oracle » (Source: docs.oracle.com) (Source: docs.oracle.com). En pratique, le moteur de formule de NetSuite prend en charge

un large sous-ensemble de fonctions SQL Oracle (fonctions de date, mathématiques, de chaîne et de gestion des valeurs nulles) tout en imposant certaines contraintes spécifiques aux champs. Les principaux types de formules incluent **Formule (Numérique)**, **Formule (Texte)**, **Formule (Date)**, **Formule (Pourcentage)**, etc., chacun attendant un résultat dans ce type (Source: luxent.com). Pour chacun, toute fonction prise en charge par Oracle avec un type de retour approprié peut être utilisée – c'est pourquoi NVL, CASE, DECODE, LTRIM et des fonctions similaires sont couramment utilisés dans ces formules.

Objectif des formules avancées

Les formules avancées dans les recherches enregistrées améliorent les rapports prêts à l'emploi de NetSuite de plusieurs manières :

- **Nettoyage des données et gestion des valeurs nulles :** Les champs peuvent souvent être vides (null) lorsque les données sont incomplètes. Pour une agrégation et un reporting précis, il est courant de remplacer les valeurs nulles par des valeurs par défaut (par exemple, 0 pour les champs numériques) afin que les expressions ne cassent pas ou ne faussent pas les résultats (Source: docs.oracle.com) (Source: www.anchorgroup.tech).
- **Logique conditionnelle :** Les exigences commerciales impliquent fréquemment une catégorisation ou des indicateurs basés sur des conditions. Les champs de formule utilisant CASE WHEN ou DECODE permettent une logique IF/THEN en ligne directement dans les résultats de recherche (Source: docs.oracle.com) (Source: docs.oracle.com).
- **Formatage des données :** Les fonctions de chaîne comme LTRIM, CONCAT ou la correspondance de modèles peuvent normaliser ou extraire des parties de données (par exemple, supprimer les zéros non significatifs ou extraire des sous-chaînes).
- **Agrégations et calculs :** Les formules numériques peuvent calculer des marges, des tranches d'âge, des ratios, etc. (par exemple, `({amount}-{cost})/{amount}`).
- **Tri et regroupement dynamiques :** Les champs de formule peuvent souvent être regroupés ou triés. Par exemple, une formule CASE peut regrouper les transactions en tranches « Élevé/Moyen/Faible » lorsqu'elles sont résumées.

En tirant parti de ces capacités de formule, les utilisateurs peuvent intégrer une logique *« pilotée par formule »* dans les recherches enregistrées. Cependant, écrire des formules correctes peut être complexe. Les pièges courants incluent les erreurs de syntaxe, une gestion incorrecte des valeurs nulles ou des types de données, et les impacts sur les performances. Ce rapport approfondit quatre fonctions spécifiques – NVL, CASE WHEN, DECODE et LTRIM – car elles illustrent plusieurs de ces besoins courants (remplacement des valeurs nulles, branchement conditionnel et manipulation de chaînes). En plus d'expliquer la syntaxe et la sémantique, nous fournissons des exemples, discutons des meilleures pratiques et explorons comment elles apparaissent souvent ensemble dans des recherches complexes.

Pour préparer le terrain, il est utile de noter que la syntaxe des formules de NetSuite reflète étroitement celle d'Oracle SQL. Pour les lecteurs familiers avec SQL : NVL est une fonction spécifique à Oracle pour tester NULL, tandis que CASE/DECODE suivent les conventions d'Oracle (Source: docs.oracle.com) (Source: docs.oracle.com). Le tableau 1 (ci-dessous) résume nos fonctions cibles :

FONCTION	OBJECTIF	EXEMPLE D'UTILISATION	RÉFÉRENCE
`NVL(expr1, expr2)`	Remplace un `expr1` NULL (vide) par `expr2` . Si `expr1` est nul, renvoie `expr2` ; sinon `expr1` . (Source: docs.oracle.com)	`NVL({quantitycommitted}, 0)` renvoie 0 si `{quantitycommitted}` est nul (vide) (Source: suiterep.com).	Référence Oracle SQL (Source: docs.oracle.com) ; Conseils NetSuite (Source: suiterep.com)

| `CASE WHEN … THEN … ELSE … END` | Exécute une logique conditionnelle. Évalue chaque `WHEN` dans l'ordre ; renvoie le résultat `THEN` associé à la première condition vraie ; sinon, renvoie `ELSE` . (Source: docs.oracle.com) | `CASE WHEN {amount}>10000 THEN 'High' WHEN {amount}>1000 THEN 'Medium' ELSE 'Low' END` (Source: www.houseblend.io). | Référence Oracle SQL (Source: docs.oracle.com) ; Exemples NetSuite (Source: www.houseblend.io) | | `DECODE(expr, search, result, [search, result…], default)` | Comparaison conditionnelle spécifique à Oracle : teste `expr` par rapport à chaque valeur `search` . Renvoie le `result` correspondant ; si aucune correspondance n'est trouvée, renvoie `default` (ou null si omis). (Source: docs.oracle.com) | `DECODE({status}, 'F', 'Fulfilled', 'P', 'Pending', 'Other')` renvoie `'Fulfilled'` si status=F, `'Pending'` si P, sinon `'Other'` . (Source: www.houseblend.io) | Référence Oracle SQL (Source: docs.oracle.com) ; Exemple NetSuite (Source: www.houseblend.io) | | `LTRIM(char, set)` | Supprime de la gauche de `char` tous les caractères présents dans `set` (par défaut, l'espace si omis). (Source: docs.oracle.com) | `LTRIM('000123', '0')` renvoie `'123'` ; `LTRIM(' test')` renvoie `'test'` . (Source: www.netsuiterp.com) | Référence Oracle SQL (Source: docs.oracle.com) ; Exemple NetSuite (Source: www.netsuiterp.com) |

Tableau 1 : Résumé des principales fonctions de formule abordées (avec références à la documentation et exemples).

Bien que le Tableau 1 fournisse des définitions rapides, les sections suivantes détaillent chaque fonction, illustrant leur utilisation dans les recherches enregistrées (Saved Searches) de NetSuite.

Vue d'ensemble des recherches enregistrées et du moteur de formules de NetSuite

Avant d'aborder des fonctions spécifiques, nous présentons le contexte général du fonctionnement des formules dans les recherches enregistrées :

- **Types de champs et portées des formules :** Dans une recherche enregistrée, une formule peut être définie soit comme *critère de recherche*, soit comme *colonne de résultat*. Dans les deux cas, vous sélectionnez un champ « Formule » (Texte, Numérique, Date, etc.) et saisissez votre expression. La formule est évaluée pour chaque enregistrement et détermine quels enregistrements correspondent (si elle est dans les critères) ou quelles valeurs s'affichent (si elle est dans les résultats). NetSuite impose que le résultat de la formule corresponde au type de retour choisi : par exemple, une Formule (Numérique) doit produire un nombre.
- **Exécution SQL sous-jacente :** Lorsqu'une recherche enregistrée est exécutée, NetSuite traduit essentiellement vos critères et résultats en une requête SQL sur la base de données Oracle en arrière-plan. Par conséquent, la syntaxe des formules est celle d'Oracle SQL. L'aide de NetSuite avertit que le SQL que vous écrivez est transmis directement au moteur d'évaluation d'Oracle (Source: docs.oracle.com). Par conséquent, la plupart des fonctions Oracle SQL fonctionnent dans les formules NetSuite, mais avec certaines limitations (les fonctions de manipulation de chaînes doivent correspondre au type de formule, certaines fonctions complexes peuvent ne pas être prises en charge, etc.).
- **Fonctions prises en charge :** La documentation officielle de NetSuite répertorie les fonctions SQL prises en charge dans les requêtes SuiteAnalytics (SuiteQL). Les principales fonctions prises en charge incluent DECODE, CASE, TO_CHAR/TO_DATE, LTRIM/RTRIM/TRIM, NVL, NVL2, NULLIF, ainsi que les fonctions mathématiques et d'agrégation (Source: docs.oracle.com) (Source: docs.oracle.com). La liste des fonctions prises en charge par SuiteQL définit explicitement chacune d'elles (par exemple, « LTRIM supprime de l'extrémité gauche de `char` tous les caractères contenus dans `set` » (Source: docs.oracle.com). Il est important de noter que le moteur de formule est effectivement le même que SuiteQL, donc cette référence confirme que NVL, DECODE et LTRIM sont bien pris en charge.
- **Note sur les versions :** Les formules de NetSuite sont antérieures à SuiteQL, mais pour nos besoins, la fonctionnalité est équivalente. Nous notons également que les nouvelles fonctionnalités de SuiteAnalytics et d'IA (abordées plus loin) coexistent avec les recherches enregistrées mais ne modifient pas actuellement la syntaxe des formules.

Cas d'utilisation motivant les formules avancées. En pratique, des formules avancées sont nécessaires dans des scénarios tels que :

1. **Remplacement de valeurs nulles/vides :** De nombreux champs clés dans NetSuite peuvent être vides au lieu de contenir un zéro logique ou une valeur par défaut. Par exemple, la quantité en stock d'un article peut être vide (NULL) plutôt que 0 lorsqu'il est totalement épuisé (Source: www.anchorgroup.tech). Sans gérer ces valeurs nulles, les calculs et les conditions peuvent échouer. NVL (ou COALESCE) est utilisé pour substituer des valeurs par défaut afin d'éviter de tels problèmes (Source: docs.oracle.com) (Source: www.anchorgroup.tech).
2. **Catégorisation et étiquetage conditionnels :** Les utilisateurs ont souvent besoin de catégoriser ou d'étiqueter des données en fonction de conditions (par exemple, classer un solde de compte comme « Élevé », « Moyen », « Faible » selon des seuils) ou de filtrer différemment selon les conditions. CASE WHEN est l'approche SQL standard (Source: docs.oracle.com) ; DECODE peut être un raccourci dans certains cas (Source: docs.oracle.com).
3. **Manipulation de chaînes pour le nettoyage et le découpage :** Les champs de données peuvent inclure des données superflues ou concaténées. Des fonctions comme LTRIM, RTRIM, SUBSTR, REPLACE, etc., sont utilisées pour nettoyer ou analyser les chaînes. Par exemple, les champs d'ancêtre peuvent stocker plusieurs niveaux de hiérarchie séparés par des barres obliques, nécessitant SUBSTR ou TRIM pour extraire des segments.
4. **Formatage dynamique ou liens :** Les formules peuvent produire du HTML ou des liens cliquables. (En dehors de notre sujet, mais les formules Oracle peuvent émettre des balises HTML pour des liens/visualisation de données.)

Une source faisant autorité souligne l'importance des formules : une analyse récente de Houseblend qualifie « l'utilisation judicieuse des formules (texte, numérique, date, pourcentage, devise, HTML, etc.) pour des calculs dynamiques » de bonne pratique clé (Source: www.houseblend.io). En effet, les formules avancées avec CASE/DECODE, l'arithmétique des dates et les fonctions de chaîne sont citées comme un élément de la « maîtrise des recherches enregistrées » (Source: www.houseblend.io).

Dans les sections suivantes, nous concentrerons notre attention spécifiquement sur NVL, CASE WHEN, DECODE et LTRIM. Celles-ci couvrent les catégories de **gestion des valeurs nulles**, de **logique conditionnelle** et de **nettoyage de chaînes**. Chacune de ces fonctions sera définie (avec le contexte Oracle et NetSuite), suivie d'exemples et de considérations pour une utilisation dans des recherches réelles.

NVL (Logique de valeur nulle)

Définition et objectif

La fonction **NVL** est une construction SQL Oracle qui remplace une expression potentiellement NULL (ou vide) par une valeur alternative. Selon la référence SQL d'Oracle : *« NVL vous permet de remplacer une valeur nulle (renvoyée sous forme vide) par une chaîne dans les résultats d'une requête. Si `expr1` est nulle, alors NVL renvoie `expr2`. Si `expr1` n'est pas nulle, alors NVL renvoie `expr1` »* (Source: docs.oracle.com). `expr1` et `expr2` doivent être de types de données compatibles (Oracle effectuera une conversion implicite si nécessaire) (Source: docs.oracle.com).

Dans les formules de recherche enregistrée NetSuite (dans les champs Formule (Numérique) ou Formule (Texte), NVL se comporte de la même manière. Le cas d'utilisation courant est : *si un champ est vide, utilisez une valeur de secours*. La syntaxe est :

```
NVL({champ_ou_expression}, valeur_de_remplacement)
```

Ici, `{champ_ou_expression}` est n'importe quel champ NetSuite ou expression calculée qui pourrait être nul, et `valeur_de_remplacement` est ce qui doit être utilisé si c'est le cas. Par exemple, `NVL({quantity}, 0)` donnera 0 si le champ `{quantity}` est nul ; sinon, il donnera la quantité originale.

La documentation de NetSuite et le blog de conseils ont mis en évidence cette utilisation. Stephen Lemp de SuiteRep explique que NVL peut être utilisé « pour obtenir une valeur d'un champ s'il n'est pas vide, et d'un second champ si le premier est vide » (Source: suiterep.com). Il montre que dans la formule d'une recherche enregistrée, vous pouvez sélectionner « Formule » et écrire `NVL({mon_premier_choix}, {mon_second_choix_si_nul})`, fusionnant ainsi efficacement deux valeurs de champ (Source: suiterep.com). En pratique, un scénario très courant est numérique : on peut vouloir « 0 » au lieu de vide. Lemp note spécifiquement : *« l'utilisation la plus courante de cette fonction est lorsque vous travaillez avec un champ qui peut être vide et que vous souhaitez insérer un 0 s'il l'est. Donc … `NVL({quantitycommitted}, 0)` »* (Source: suiterep.com). Cela correspond aux frustrations de nombreux utilisateurs : par exemple, Anchor Group note que les champs de quantité d'inventaire apparaissent vides (pas zéro) lorsque le stock est épuisé, il faut donc utiliser NVL pour les traiter comme zéro afin que les conditions fonctionnent (Source: www.anchorgroup.tech).

Lien avec NetSuite : Les développeurs NetSuite peuvent utiliser NVL à la fois dans les **critères** et les **résultats**. Dans les critères, NVL garantit que les éléments avec des blancs sont inclus. Dans les résultats, NVL affiche des valeurs alternatives (pour que les rapports n'affichent pas de blancs). NVL peut être utilisé sur n'importe quel champ de type numérique ou chaîne. Par exemple, un résultat de formule (numérique) pourrait être `NVL({amount},0)`, et un résultat de formule (texte) pourrait être `NVL({custbody_code}, 'N/A')`.

La page d'aide **Expressions SQL** note NVL parmi les « fonctions de comparaison liées aux valeurs nulles », soulignant qu'elle est nativement prise en charge (Source: docs.oracle.com). Les fonctions prises en charge par SuiteQL confirment : « NVL : vous permet de remplacer une valeur nulle (renvoyée sous forme vide) par une chaîne dans les résultats d'une requête » (Source: docs.oracle.com). NVL2 (fonction associée) est également répertoriée (abordée ci-dessous) (Source: docs.oracle.com).

Exemples de NVL dans une recherche enregistrée

- **Valeur numérique par défaut :** Supposons que vous ayez un champ `{quantityonhand}` qui peut être vide pour certains articles. Pour calculer l'inventaire total, on pourrait utiliser `NVL({quantityonhand}, 0)` afin que la somme ne soit pas mal calculée. Par exemple, le consultant d'Anchor Group, Jack Mannebach, décrit le suivi des articles avec un inventaire de 0 : il devait inclure les articles dont la « Quantité en stock est 0 ou nulle ». La solution consistait à utiliser `NVL({quantityonhand}, 0)` dans les critères de la formule pour que ces valeurs nulles comptent comme des zéros (Source: www.anchorgroup.tech).
- **Valeur de secours d'un champ alternatif :** Si le champ de premier choix est vide, repliez-vous sur un autre. Par exemple, le blog de Lemp montre `NVL({shipstate},{billstate})` : si l'état d'expédition est vide, renvoyez l'état de facturation (Source: test.suiterep.com).
- **Éviter la division par zéro :** NVL peut fournir un 1 ou une autre valeur pour éviter la division par zéro. (Cela a été évoqué dans certains exemples de recherche enregistrée (Source: docs.oracle.com), où une astuce pour éviter les erreurs de division par zéro implique probablement NVL ou NULLIF).
- **Concaténation de parties de nom :** L'article de Houseblend montre comment combiner des champs de nom et utiliser NVL pour gérer les seconds prénoms manquants : `NVL({middlename}||' ','')` empêche le texte « NULL » si `{middlename}` est vide (Source:

www.houseblend.io). Un exemple de nom complet donné est : `{firstname} || ' ' || NVL({middlename}||' ', '') || {lastname}` qui ajoute correctement un espace uniquement si un second prénom existe (Source: www.houseblend.io).

Considérations techniques

- **Null vs Zéro :** Il est important de se rappeler que NVL traite les *vraies valeurs nulles/vides* comme nécessitant un remplacement. Dans NetSuite, les champs numériques qui sont vides sont traités comme des valeurs nulles (pas comme 0). Donc `NVL({total},0)` n'est nécessaire que si `{total}` peut être nul. Si un champ peut légitimement être à la fois 0 ou nul, il faut examiner la logique avec soin.
- **Fonction NVL2 :** NetSuite prend également en charge **NVL2(expr, not_null_val, null_val)**, qui renvoie `not_null_val` si `expr` n'est pas nul, sinon `null_val`. Cela peut être considéré comme une conditionnelle plus explicite. Le blog de Lemp y fait allusion : « Si vous cherchez un troisième choix, consultez NVL2 dans l'aide de NetSuite » (Source: suiterep.com). Par exemple, `NVL2({custfield}, {link_if_present}, {link_if_null})`. NVL2 est parfois utilisé dans une logique de formule plus avancée, et les documents SuiteQL l'énumèrent comme pris en charge (Source: docs.oracle.com).
- **Alternative COALESCE :** Oracle prend également en charge `COALESCE`, qui renvoie la première valeur non nulle d'une liste. SuiteQL de NetSuite répertorie COALESCE comme une fonction prise en charge (Source: docs.oracle.com). COALESCE est effectivement un NVL à arguments multiples ; par exemple, `COALESCE({a}, {b}, {c})` est égal à `{a}` s'il n'est pas nul, sinon `{b}` s'il n'est pas nul, sinon `{c}`. Bien qu'en dehors du champ d'application de la question, cela vaut la peine d'être connu pour une gestion complexe des valeurs nulles.
- **Cohérence des types de données :** Les documents Oracle et les conseils NetSuite rappellent que les deux arguments de NVL doivent idéalement être de types compatibles (Source: docs.oracle.com). Par exemple, `NVL({textfield}, 0)` générerait une erreur si {textfield} n'est pas numérique/chaîne. Dans l'interface de formule de NetSuite, il faut utiliser des types correspondants (par exemple, la formule numérique doit renvoyer des nombres).

Résumé de NVL

La fonction NVL est indispensable pour garantir que les résultats et les critères des recherches enregistrées gèrent les données manquantes en toute sécurité. En remplaçant les valeurs nulles par des valeurs par défaut significatives, les rapports sont complets et les calculs sont précis. En pratique, les administrateurs utilisent souvent NVL pour traiter les blancs comme 0 ou comme des champs alternatifs (Source: suiterep.com) (Source: www.anchorgroup.tech). Plus loin, nous verrons que NVL apparaît souvent en combinaison avec CASE ou DECODE pour gérer de multiples conditions : par exemple, `CASE WHEN NVL({committed},0)>0 THEN 'On Order' ELSE 'None' END` ou imbriqué dans des expressions DECODE. Dans les recherches riches en données (par exemple, les cumuls financiers), NVL garantit que les sommations et les pourcentages ne sont pas en échec.

CASE WHEN (Logique conditionnelle)

Définition et objectif

L'expression **CASE** est une syntaxe SQL standard (qui fait partie du SQL d'Oracle) qui implémente une logique conditionnelle similaire à IF/THEN/ELSE. Il existe deux formes : **CASE simple** (qui compare une expression à des valeurs statiques) et **CASE recherché** (qui permet des conditions arbitraires). La documentation d'Oracle explique : *« dans une expression CASE simple, Oracle Database recherche la première paire WHEN...THEN pour laquelle `expr` est égale à `comparison_expr` et renvoie `return_expr`. Si aucune ne correspond et qu'un ELSE existe, Oracle renvoie `else_expr` ; sinon, null. »* (Source: docs.oracle.com). Une expression CASE recherchée se compose d'une ou plusieurs clauses `WHEN condition THEN result`.

Par exemple, la documentation sur les recherches enregistrées fournit un exemple de CASE recherché pour catégoriser les saisons :

```
CASE
  WHEN EXTRACT(Month FROM {trandate}) = 12 THEN 'winter'
  WHEN EXTRACT(Month FROM {trandate}) = 6  THEN 'summer'
  ELSE 'it was fall or spring'
END
```

(Type de sortie : CHAÎNE) (Source: docs.oracle.com). Cela illustre comment chaque `WHEN` est une condition booléenne appliquée aux données de chaque ligne.

Dans les recherches enregistrées (Saved Searches) de NetSuite, `CASE WHEN` est l'un des outils les plus puissants pour générer des étiquettes dynamiques ou prendre des décisions. Il peut être utilisé dans n'importe quel champ de formule (texte ou numérique) et peut renvoyer du texte, des nombres, des dates, etc., selon le contexte. Les utilisations courantes incluent :

- **Catégorisation des valeurs :** Par exemple, mapper des montants numériques vers des catégories « Grand/Moyen/Petit » (comme dans l'exemple ci-dessous), ou mapper des codes vers des descriptions.
- **Indicateurs de date/période :** Par exemple, `CASE WHEN {trandate} < {start_of_last_year} THEN 'Année précédente' ELSE 'Année en cours' END` pour étiqueter les dates de transaction.
- **Indicateurs personnalisés :** Par exemple, signaler les ventes dépassant un objectif, ou les articles en rupture de stock par rapport à ceux en stock.
- **Codage couleur ou métriques d'icônes :** `CASE` peut même renvoyer du HTML (par exemple, `<span>` avec une couleur) dans un champ de type Formule (Texte) pour mettre en évidence des statuts (quelques exemples sur scribd (Source: www.scribd.com).

L'aide en ligne de NetSuite note explicitement : *« Vous pouvez effectuer des évaluations conditionnelles en créant un champ de formule avec la fonction CASE WHEN. Dans ces formules, si les valeurs remplissent les conditions indiquées dans WHEN, vous obtenez le résultat de THEN ; toutes les autres utilisent le résultat de ELSE. »* (Source: docs.oracle.com). Et surtout, « dans Workbook, vous pouvez imbriquer différentes instructions CASE WHEN au sein de la même formule » (Source: docs.oracle.com) (bien que pour les recherches enregistrées, l'imbrication soit également prise en charge en plaçant simplement un `CASE` à l'intérieur d'un autre).

Syntaxe

Une syntaxe `CASE` recherchée typique dans NetSuite ressemble au SQL standard :

```
CASE
   WHEN <condition1> THEN <valeur1>
   WHEN <condition2> THEN <valeur2>
   ...
   ELSE <valeur_else>
END
```

Points importants :

- La clause `ELSE` est facultative. Si elle est omise et qu'aucun `WHEN` ne correspond, NetSuite renverra `null` (affiché comme vide) pour cet enregistrement.
- Les conditions peuvent utiliser n'importe quel opérateur pris en charge, par exemple `=`, `>`, `<`, `IN`, `LIKE`, `IS NULL`, etc.
- Chaque `THEN`/`ELSE` doit être cohérent avec le type de retour de la formule (TEXT pour une sortie texte, etc.).
- Vous pouvez inclure plusieurs `WHEN` ; logiquement, ils sont évalués de haut en bas, et seule la première branche correspondante est exécutée.

Exemples dans les recherches enregistrées

1. **Catégorisation numérique simple :**
   « Classer les commandes client par taille » (Source: www.houseblend.io).
   Exemple de formule (texte) :

   ```
   CASE
     WHEN {amount} > 10000 THEN 'Grand'
     WHEN {amount} > 1000  THEN 'Moyen'
     ELSE 'Petit'
   END
   ```

Ceci renvoie « Grand » pour les commandes à forte valeur, etc. Cet exemple précis est mentionné dans un guide Houseblend (Source: www.houseblend.io). Il démontre un `CASE` recherché (deux conditions sur `{amount}`). Notez qu'un seul `THEN` est choisi par ligne, correspondant au premier `WHEN` vrai.

2. **Correspondance de texte :**
   « Marquer les éléments de mémo urgents ». Vous pouvez conditionner sur du texte : par exemple

```
CASE
   WHEN {memo} LIKE '%URGENT%' THEN 'Oui'
   ELSE 'Non'
END
```

   (Le document Houseblend suggère l'utilisation de `LIKE` au sein de `CASE` (Source: www.houseblend.io).)
   Ceci marque les enregistrements dont le mémo contient « URGENT ».

3. **Indicateurs de plage de dates (relatifs) :**
   « Marquer les transactions comme Année en cours vs Année précédente » (exemple de budget annuel).
   Exemple :

```
CASE
  WHEN {trandate} < TO_DATE('2023-01-01','YYYY-MM-DD') THEN 'Année précédente'
  ELSE 'Année en cours'
END
```

   (Référence à la mention par Houseblend des « indicateurs Année précédente vs Année en cours » (Source: www.houseblend.io).) Ceci est courant dans les rapports financiers pour segmenter par périodes fiscales. NetSuite utilise souvent des critères de formule ou des résultats pour effectuer cette logique.

4. **Conditions multi-champs :**
   Vous pouvez combiner des champs avec `AND` / `OR`. Par exemple, un échantillon scribd montre :

```
CASE
  WHEN {custitemvertical} = 'AEC' AND {custitem_product_type} IN ('Software','Fee')
  THEN 'AEC Software/Fee'
  …
END
```

   Ceci démontre l'utilisation de `AND`, `IN`, etc. (Voir les résultats de recherche scribd autour de [57†L79-L87]).

5. **CASE imbriqué :**
   Les instructions `CASE` peuvent apparaître les unes dans les autres. Par exemple :

```
CASE
  WHEN {type} = 'Invoice' THEN CASE WHEN {balance} > 0 THEN 'Ouvert' ELSE 'Fermé' END
  ELSE 'Autre'
END
```

   Une telle utilisation est montrée dans les formules avancées (cadres dans scribd [57†L129-L138]). L'article de Houseblend mentionne également que les `CASE` imbriqués produisent une logique à plusieurs niveaux (Source: www.houseblend.io).

6. **Exemple (compilation de travaux) :**
   Un exemple pratique de scribd illustre un `CASE` renvoyant un statut codé en HTML :

```
CASE
  WHEN {status}='Completed' THEN '<span style="color:red;">Terminé</span>'
  WHEN {status}='Other' THEN '<span style="color:green;">Autre</span>'
  ELSE NULL
END
```

(Ce fragment est similaire à celui en [57†L131-L139].) Ceci illustre que `CASE` peut produire du texte enrichi s'il est placé dans une colonne Formule (Texte).

Meilleures pratiques avec CASE

- **Clause Else :** Envisagez toujours de fournir un `ELSE`. S'il est omis, tout enregistrement ne correspondant pas tombera sur `null`, ce qui peut masquer des données par inadvertance. Si « autre » est significatif, incluez-le (soit avec `ELSE`, soit avec un `WHEN` final).
- **Performance :** Une logique `CASE` complexe (surtout avec de nombreux `WHEN` ou des `CASE` imbriqués) peut ralentir les recherches. Si les conditions peuvent être exprimées par des filtres, utilisez-les. Par exemple, vérifier une plage de dates est souvent plus rapide en tant que filtre de critères qu'en tant que condition `CASE`. Houseblend souligne l'importance d'une conception soignée : un client a amélioré la vitesse en convertissant un filtre de date basé sur `CASE` en un filtre de plage de dates natif (Source: www.houseblend.io).
- **Valeurs littérales :** Dans les comparaisons `CASE`, les chaînes littérales doivent correspondre exactement (sensible à la casse par défaut dans Oracle). Pour les correspondances de motifs, utilisez `LIKE` (ou `REGEXP_LIKE` si nécessaire).
- **Cohérence du type de retour :** Toutes les expressions `THEN` / `ELSE` doivent correspondre au même type (toutes du texte, ou toutes numériques, etc.). Mélanger les types provoquera des erreurs de formule.
- **Conditions Null :** Si vous souhaitez tester une valeur nulle dans un `CASE`, vous devez utiliser `WHEN {field} IS NULL THEN ...`. (Alternativement, `NVL` à l'intérieur de `CASE` peut aider, mais soyez prudent).

CASE vs DECODE

Bien que `CASE` soit conforme à la norme ANSI SQL et plus flexible, `DECODE` (section suivante) peut être considéré comme une forme limitée de `CASE` (essentiellement un `CASE` simple où seules les vérifications d'égalité sont autorisées). Quelques différences clés :

- **Gestion de NULL :** `DECODE` traite deux valeurs nulles comme égales, donc `DECODE({a}, NULL, 'Y', 'N')` renvoie 'Y' lorsque `{a}` est nul (Source: docs.oracle.com). `CASE` ne considère pas `NULL = NULL`, donc un `CASE` recherché équivalent serait `CASE WHEN {a} IS NULL THEN 'Y' ELSE 'N' END`. Ainsi, `decode` peut parfois donner des résultats différents dans les cas nuls.
- **Flexibilité :** `CASE` peut avoir des conditions booléennes arbitraires ; `DECODE` ne fait que des vérifications d'égalité par rapport à des valeurs fixes.
- **Support :** `DECODE` est spécifique à Oracle ; `CASE` est du SQL portable. Les deux sont pris en charge par le moteur de formule de NetSuite (Source: docs.oracle.com) (Source: docs.oracle.com).

En pratique, la plupart des logiques complexes sont effectuées avec `CASE WHEN`. `DECODE` est pratique pour des mappages simples ou lors de la migration de logique à partir d'anciens rapports Oracle.

DECODE (Mappage conditionnel)

Définition et objectif

DECODE est une fonction Oracle qui permet une substitution de valeur conditionnelle. Elle est souvent décrite comme une alternative fonctionnellement équivalente à `CASE` pour les tests d'égalité simples. La documentation d'Oracle définit `DECODE` comme suit : *« DECODE compare `expr` à chaque valeur `search` une par une. Si `expr` est égal à une valeur `search`, alors la base de données Oracle renvoie le `result` correspondant. Si aucune correspondance n'est trouvée, alors Oracle renvoie `default`. Si `default` est omis, Oracle renvoie null. »* (Source: docs.oracle.com). La syntaxe est :

```
DECODE(expr,
       search1, result1,
       search2, result2,
       ...,
       default)
```

`DECODE` fonctionne comme suit :

- Compare `expr` à `search1`. Si égal, renvoie `result1`.
- Sinon, compare `expr` à `search2`. Si égal, renvoie `result2`.
- ...
- Si aucune valeur de recherche ne correspond et qu'une valeur `default` est fournie, renvoie `default` ; sinon, renvoie NULL.
- Si `expr` lui-même est NULL, et que l'une des valeurs de recherche est NULL, `DECODE` renvoie le résultat pour ce NULL (il traite `NULL = NULL`) (Source: docs.oracle.com).

Dans les formules de recherche enregistrée NetSuite, `DECODE` peut être utilisé de manière similaire à `CASE WHEN`, mais est limité aux opérations de comparaison. Il peut être placé dans une Formule (Numérique) ou (Texte), etc., renvoyant le `result` spécifié (qui doit correspondre au type de formule).

Cas d'utilisation : `DECODE` est fréquemment utilisé pour mapper des valeurs codées ou pour des conditions rapides. Les exemples incluent :

- **Traduction de valeurs :** Mapper des codes à un seul caractère ou numériques vers des étiquettes descriptives. Par exemple, un code de type d'article `{type}` pourrait être mappé : `DECODE({type}, 1, 'Matériel', 2, 'Logiciel', 'Autre')`.
- **Mappage de statut :** Traduire des codes de statut (A, I, C, etc.) en mots complets (« Actif », « Inactif », « Fermé »).
- **Remplacement simple de CASE :** Un `CASE` comme `CASE {field} WHEN 'A' THEN x WHEN 'B' THEN y ELSE z END` peut être écrit comme `DECODE({field}, 'A', x, 'B', y, z)`.

Parce que `DECODE` est une fonction, elle peut être imbriquée dans d'autres fonctions ou dans `CASE`, ou contenir un `CASE` à l'intérieur. Un exemple scribd montre `Simple Case When with Decode`, c'est-à-dire `CASE WHEN ... THEN DECODE(to_char({createddate}, 'D'), '2', 1) END` (Source: www.scribd.com).

Exemples de DECODE dans les recherches enregistrées

- **Mappage de codes :** L'article Houseblend mentionne le décodage du statut : par ex.

  ```
  DECODE({status}, 'F', 'Exécuté', 'P', 'En attente', 'X', 'Annulé', 'Autre')
  ```

  Ceci renvoie le mot pour chaque code de statut. (Voir la note de Houseblend : « par ex. `DECODE({status}, 'F', 'Exécuté', 'P', 'En attente', ...)` » (Source: www.houseblend.io).)
- **Recherches hiérarchiques :** Si un champ peut avoir de nombreuses valeurs possibles, on peut enchaîner les `DECODE` ou intégrer un `CASE` dans un `DECODE`. Les exemples scribd (par ex. en [58]) montrent un `CASE` imbriqué dans un `DECODE`.
- **Valeur par défaut nulle :** Comme `DECODE` permet de spécifier un résultat par défaut, on peut gérer facilement les valeurs inattendues.

DECODE vs CASE : Comparaison

Quelques considérations lors du choix entre `DECODE` et `CASE` :

- **Syntaxe et lisibilité :** `DECODE` peut être plus concis lors d'un mappage de comparaison d'égalité. Cependant, `CASE...WHEN` est souvent plus clair pour une logique complexe. Par exemple, mapper 10 codes pourrait être plus facile à lire sous forme de `CASE` avec plusieurs lignes `WHEN` que sous forme d'un long `DECODE`.
- **Comparaison nulle :** `DECODE` considère `NULL` comme égal à `NULL`. Donc `DECODE({x}, NULL, 'Manquant', 'Inconnu')` renvoie 'Manquant' si `{x}` est nul (Source: docs.oracle.com). Pour reproduire cela dans `CASE`, vous devriez explicitement faire `CASE WHEN {x} IS`

`NULL THEN 'Manquant' WHEN ... ELSE 'Inconnu' END`.

- **Disponibilité des fonctions :** `DECODE` est pris en charge en tant que fonction intégrée (Source: docs.oracle.com), tandis que `CASE` fait partie du langage SQL. NetSuite prend en charge les deux (Source: docs.oracle.com) (Source: docs.oracle.com).

Ni `DECODE` ni `CASE` ne sont intrinsèquement plus rapides ; les deux se traduisent par des opérations SQL Oracle. Cependant, la nature fonctionnelle de `DECODE` signifie que vous ne pouvez souvent pas utiliser plus que des comparaisons d'égalité. Si vous avez besoin de `<` ou de correspondance de motifs, vous devez utiliser `CASE/WHEN`.

Exemple

Un exemple simple de `DECODE` pour illustrer l'utilisation dans une formule de recherche enregistrée :

```
DECODE({payment_status}, NULL, 'Non payé',
                       'B', 'Facturé',
                       'C', 'Soldé',
                       'Autre')
```

Ceci signifie : si `{payment_status}` est nul, renvoyer "Non payé" ; s'il est 'B', renvoyer 'Facturé' ; s'il est 'C', renvoyer 'Soldé' ; sinon 'Autre'. Selon Oracle, la première correspondance d'argument réussirait sur `NULL` (car `DECODE` traite `NULL = NULL`) (Source: docs.oracle.com), donc les valeurs nulles obtiennent "Non payé".

Ceci aurait également pu être écrit comme :

```
CASE
  WHEN {payment_status} IS NULL THEN 'Non payé'
  WHEN {payment_status} = 'B' THEN 'Facturé'
  WHEN {payment_status} = 'C' THEN 'Soldé'
  ELSE 'Autre'
END
```

Meilleures pratiques pour DECODE

- **Type de données cohérent :** Tous les résultats (`resultN` et `default`) doivent être du même type (tous du texte si dans une formule texte, etc.). Les incompatibilités peuvent provoquer des erreurs de type.
- **Cas non correspondant :** Fournissez toujours un argument par défaut final dans `DECODE` (même si le résultat final est l'expression originale). S'il est omis et qu'il n'y a pas de correspondance, `DECODE` renvoie `NULL`, ce qui pourrait ne pas être souhaité.
- **Performance :** Pour de très longues chaînes `DECODE`, demandez-vous si un `CASE WHEN` ou même une jointure vers une table de recherche (via une recherche jointe) est plus facile à maintenir. (Pas toujours possible dans une recherche enregistrée, mais gardez la logique simple si possible.)
- **Gestion de NULL :** Si un traitement des valeurs nulles réelles est nécessaire, rappelez-vous que `DECODE` les interceptera si vous listez une valeur de recherche `NULL`. Pour la plupart des utilisations, réfléchissez à ce qu'il faut faire si l'expression est nulle et incluez-le explicitement si nécessaire.

`DECODE` est particulièrement pratique lorsqu'un ancien rapport Oracle ou une formule héritée l'utilise ; comme NetSuite est basé sur Oracle, `DECODE` fonctionne de manière transparente. BrokenRubik et d'autres tutoriels NetSuite avancés montrent `DECODE` utilisé pour les champs de formule, souvent en conjonction avec `CASE` (Source: www.houseblend.io) (Source: www.scribd.com).

LTRIM et fonctions de découpage de chaîne

Définition et objectif

Le découpage de chaîne est souvent nécessaire pour nettoyer les champs de texte. **LTRIM** (littéralement « trim gauche ») est une fonction SQL Oracle qui supprime les caractères indésirables de l'*extrémité gauche* d'une chaîne. La documentation Oracle stipule : *« LTRIM supprime de l'extrémité gauche de `char` tous les caractères contenus dans `set`. Si vous ne spécifiez pas `set`, il est par défaut un espace unique. »* (Source: docs.oracle.com). En d'autres termes, par défaut, `LTRIM(char)` supprimera tous les espaces de début. Si vous fournissez un deuxième argument, il supprimera l'un de ces caractères jusqu'au premier caractère ne figurant pas dans l'ensemble.

La syntaxe dans NetSuite serait :

```
LTRIM(char_expression [, trim_set])
```

- `char_expression` est la chaîne ou le champ à découper.
- `trim_set` est une chaîne optionnelle de caractères à supprimer. Si omis, les espaces (`' '`) sont supprimés du côté gauche.

Les exemples Oracle incluent :

```
LTRIM('   test');       -- renvoie 'test'
LTRIM('000123', '0');   -- renvoie '123'
LTRIM('123abc123','123'); -- renvoie 'abc123'
LTRIM('xyzTest', 'xyz'); -- renvoie 'Test'
```

Ceux-ci sont montrés dans la documentation de la communauté NetSuite (Source: www.netsuiterp.com). Par exemple, `LTRIM('000123','0')` donne `'123'` et `LTRIM(' test')` donne `'test'` (Source: www.netsuiterp.com).

Les fonctions associées sont `RTRIM` (découper à droite) et `TRIM` (qui peut faire les deux extrémités ou des cas spécifiques avec une syntaxe comme `TRIM(LEADING 'X' FROM char)`). NetSuite prend en charge tout cela dans les formules (Source: www.netsuiterp.com) (Source: docs.oracle.com).

Utilisation dans NetSuite

Dans les recherches enregistrées, `LTRIM` est utile pour :

- **Supprimer les espaces de début :** Si un champ de texte a des espaces supplémentaires à gauche, par ex. `LTRIM({name})` peut le nettoyer. C'est souvent nécessaire lors de la concaténation de champs, pour éviter les doubles espaces.
- **Supprimer les préfixes ou caractères :** Pour les champs avec des valeurs à format fixe, par ex. les numéros de pièce avec des zéros non significatifs, utilisez `LTRIM` pour standardiser. Un exemple courant : si les ID d'article sont des chaînes numériques avec des zéros non significatifs (par ex. "00012345"), on pourrait utiliser `LTRIM({itemid}, '0')` pour comparer ou afficher sans les zéros.
- **Analyse de données :** Parfois, les valeurs sont stockées avec un caractère préfixe (comme 'X apple'), et vous voulez supprimer tous les caractères 'X' s'il y en a. Remarque : `LTRIM` avec un ensemble de caractères supprimera toutes les occurrences de tous les caractères de l'ensemble jusqu'à ce qu'un caractère ne correspondant pas soit trouvé. Exemple : `LTRIM('abcabcHello','abc')` renvoie `'Hello'` (Source: www.netsuiterp.com).
- **Combiner avec CASE :** On voit souvent des formules comme `CASE WHEN LTRIM({somefield}, '0') = '' THEN ... ELSE ...`. Par exemple, un utilisateur du forum NetSuite avait besoin de supprimer un préfixe parent d'un nom : il a tenté des formules incluant `TRIM` / `RTRIM` (Source: stackoverflow.com). Bien que cet utilisateur ait eu des difficultés (et ait eu recours à du code de contournement), cela illustre des cas d'utilisation courants.

Exemples de LTRIM dans les formules

- **Supprimer tous les zéros non significatifs :**

  ```
  LTRIM({serialnumber}, '0')
  ```

Si `{serialnumber}` est `'000123'`, le résultat est `'123'` (Source: www.netsuiterp.com).

- **Supprimer des caractères spécifiques au début :**

```
LTRIM({custbody_flag}, 'X')
```

Si `{custbody_flag}` contient `'XXXabcXdef'`, le résultat est `'abcXdef'` (tous les 'X' au début sont supprimés).

- **Supprimer les espaces (par défaut) :**

```
LTRIM({companyname})
```

Supprime tous les espaces au début du nom de l'entreprise (si le demandeur en a besoin).

Étant donné que LTRIM ne traite que les caractères situés à gauche, les développeurs NetSuite choisissent souvent `LTRIM`, `RTRIM` ou `TRIM` selon le besoin. Par exemple, pour supprimer les espaces aux deux extrémités, utilisez `TRIM({field})`.

Bonnes pratiques

- **Adapter l'ensemble de caractères à supprimer :** Ne passez pas la chaîne entière à supprimer. Normalement, `trim_set` est un petit ensemble de caractères (une lettre, un chiffre ou un espace).
- **Attention aux jeux de caractères :** Dans Oracle, le comportement de LTRIM avec des jeux de caractères multiples consiste à supprimer individuellement « chaque caractère de l'ensemble ». Ainsi, `LTRIM('aaHello','a')` donne `'Hello'`, mais `LTRIM('abcHello','ab')` donne `'cHello'` (car « a » et « b » sont supprimés individuellement). Assurez-vous que c'est bien ce que vous souhaitez.
- **Chaînes vides :** Si la chaîne est entièrement composée des caractères à supprimer, LTRIM renvoie une chaîne vide (et non null). Par exemple, `LTRIM('0000','0')` renvoie `''`.
- **Utiliser à bon escient :** Le nettoyage dans une formule peut entraver l'utilisation des index ou la mise en cache ; si vous avez seulement besoin d'effectuer une comparaison en ignorant les espaces de début, envisagez d'utiliser `TRIM({field})` dans les critères. Cependant, c'est souvent utile dans les résultats pour nettoyer la sortie.

LTRIM (ainsi que TRIM/RTRIM) sont des fonctions de chaîne simples. Elles n'impliquent pas de logique ou de conditions ; leurs résultats sont déterministes et rapides. Il y a peu de raisons de les éviter, sauf pour des questions de précision.

Comparaison et cas d'utilisation combinés

NVL vs NVL2 vs COALESCE

Nous avons abordé NVL ci-dessus. Il est utile de le comparer avec des fonctions associées :

- **NVL2 :** Comme indiqué, NVL2(expr, val_if_not_null, val_if_null) est une fonction conditionnelle dans Oracle. Dans les systèmes NetSuite, vous pouvez la voir utilisée lorsque vous avez besoin de résultats différents selon que la valeur est nulle ou non. Par exemple : `NVL2({field}, {field} * 0.1, 0)` signifie « si {field} n'est pas nul, renvoyez 10 % de sa valeur ; sinon, renvoyez 0 ». La documentation SuiteQL décrit NVL2 comme suit : *« permet de déterminer la valeur renvoyée par une requête selon qu'une expression spécifiée est nulle ou non. »* (Source: docs.oracle.com). Cela équivaut à `IF {field} IS NULL THEN ... ELSE ...`.
- **COALESCE :** Renvoie la première expression non nulle parmi ses arguments (Source: docs.oracle.com). Contrairement à NVL, COALESCE peut accepter plusieurs arguments. Pour 2 arguments, `COALESCE(a,b)` est fonctionnellement identique à `NVL(a,b)`. COALESCE peut être plus pratique si vous avez plusieurs valeurs de repli, mais NVL suffit pour deux. NetSuite prend en charge COALESCE dans les formules (par exemple, une formule (Numérique) peut l'utiliser), comme indiqué dans la documentation SuiteQL (Source: docs.oracle.com).

Utilisation dans les recherches enregistrées : NVL est souvent suffisant. Si plusieurs champs de repli sont nécessaires, on peut imbriquer des NVL ou utiliser COALESCE. Par exemple : `COALESCE({field1}, {field2}, 0)` choisira le premier champ non nul, ou 0 si les deux sont nuls.

CASE vs DECODE

Nous avons déjà comparé certaines différences dans les sections précédentes. Voici un résumé :

- **Équivalence générale :** Une expression CASE simple peut généralement être traduite par DECODE, et vice versa, pour les cas particuliers de comparaisons d'égalité. En substance :
  `CASE {expr} WHEN val1 THEN res1 WHEN val2 THEN res2 ELSE res_default END`
  est équivalent à
  `DECODE({expr}, val1, res1, val2, res2, res_default)`.
  Cette équivalence est notée dans de nombreuses références SQL.
- **Différences :**
  - CASE permet plusieurs conditions par WHEN (en utilisant `AND` / `OR`), tandis que DECODE ne peut comparer que l'égalité d'une seule `expr`.
  - CASE nécessite un `ELSE` explicite si vous voulez une valeur par défaut, alors que le dernier paramètre de DECODE agit automatiquement comme valeur par défaut.
  - DECODE a un comportement particulier avec les NULL (comme mentionné : `DECODE(expr, search, result)` traite les NULL comme des correspondances). CASE nécessite des vérifications `IS NULL` pour la logique liée aux valeurs nulles.
  - CASE…WHEN est conforme à la norme ANSI (fonctionne dans d'autres moteurs SQL comme MSSQL, etc.), DECODE est spécifique à Oracle.
- **Utilisation recommandée :** De nombreux experts NetSuite préfèrent CASE WHEN pour les nouvelles formules en raison de sa clarté et de sa puissance, utilisant DECODE principalement pour la concision dans les cas de mappage direct (Source: www.houseblend.io). Aucune différence de performance significative n'est signalée pour les recherches classiques.

Implications sur les performances

Le choix de la formule peut affecter les performances des recherches enregistrées, bien que cela ne soit généralement pas radical si les index sont utilisés de manière appropriée. Quelques points clés :

- **Filtre vs Formule :** Dans la mesure du possible, utilisez les filtres de recherche natifs (critères) pour les conditions simples (ex. plages de dates, énumérations). Les filtres s'exécutent directement dans le SQL optimisé. À l'inverse, placer des conditions dans des formules signifie que chaque enregistrement doit être évalué par le moteur de formule. Le rapport Houseblend donne une illustration frappante : convertir une formule en filtre a réduit le temps d'exécution de 2 minutes à 10 secondes (Source: www.houseblend.io).
  - *Exemple :* Au lieu de `CASE WHEN {amount}>1000 THEN ... END` dans un résultat, il est généralement préférable de définir un filtre de recherche « Montant > 1000 ». N'utilisez CASE que pour dériver de nouveaux champs.
- **Colonnes indexées :** L'utilisation de formules sur des champs non indexés (comme les champs joints personnalisés) sera plus lente. Par exemple, `CASE WHEN {item.category}...` peut ralentir car "category" est un champ joint, alors qu'un filtre sur {category} pourrait être partiellement indexé.
- **Formules complexes :** Chaque CASE, NVL ou opération sur chaîne supplémentaire ajoute de la charge CPU. L'imbrication profonde de fonctions peut également provoquer des délais d'attente sur les recherches volumineuses. Si les formules deviennent très complexes, testez avec des recherches « Sommaire » et des filtres plus restreints. Le rapport Houseblend suggère d'auditer et d'optimiser les recherches lentes en supprimant la logique de formule inutile (Source: www.houseblend.io).
- **Planification et mise en cache :** NetSuite permet de planifier des recherches enregistrées. Si les performances d'une recherche complexe sont limites, planifiez-la (ex. hebdomadairement) plutôt que de l'exécuter à la demande. Houseblend note que la planification peut atténuer le ralentissement des recherches lourdes à la demande (Source: www.houseblend.io).
- **SuiteAnalytics / SuiteQL :** Pour les scénarios extrêmement complexes (comme le filtre multi-filiales dans l'étude de cas (Source: www.houseblend.io), la recommandation peut être d'extraire les données via SuiteAnalytics (Workbooks ou SuiteQL) plutôt que de pousser toute la logique dans une seule recherche enregistrée.

Exemple quantitatif d'une étude de cas : une entreprise a signalé une réduction de plus de 50 % des temps de chargement des pages (c'est-à-dire l'exécution de la recherche enregistrée) après avoir audité et repensé ses recherches (Source: www.houseblend.io). Une autre a constaté que le traitement par lots des recherches avec planification permettait des rafraîchissements systématiquement rapides (Source: www.houseblend.io).

Exemples d'utilisation combinée

Les formules de recherche enregistrée réelles combinent souvent ces fonctions. Quelques modèles incluent :

- **Conditionnelles sécurisées pour NULL :**
  Exemple : `CASE WHEN NVL({qtycommitted},0) > 0 THEN 'Committed' ELSE 'Not Committed' END`. Cela garantit que les quantités engagées nulles sont comptées comme 0 lors de la vérification.
- **DECODE avec NVL :**
  Exemple : `DECODE(NVL({status}, 'U'), 'A', 'Active', 'D', 'Disabled', 'Unknown')`. Ici, NVL transforme d'abord la valeur nulle en 'U', puis DECODE mappe les codes.
- **LTRIM après CONCAT :**
  Exemple : `LTRIM({prefix} || '-' || {number}, '-')` supprime un trait d'union au début si {prefix} est vide.
- **Logique imbriquée :**
  Les exemples de formules de Scribd montrent des combinaisons comme `NVL(CASE WHEN ... THEN NVL({field},0) END, 0)` (Source: www.scribd.com).

Toutes ces compositions doivent être élaborées avec soin, en respectant les parenthèses et les types de données. Tester des formules en couches implique souvent de vérifier d'abord chaque partie (par exemple, testez d'abord votre CASE renvoyant des nombres, puis enveloppez-le avec NVL).

Exemples concrets et études de cas

Les analystes et consultants partagent fréquemment des exemples de recherches enregistrées qui exploitent NVL, CASE et d'autres formules pour résoudre des problèmes métier. Nous résumons plusieurs cas illustratifs :

- **Gestion des valeurs nulles en inventaire (Exemple de vente au détail) :**
  Une entreprise de vente au détail devait établir des rapports sur les articles en rupture de stock. Leur champ « Quantité en stock » était vide pour les articles sans stock. En utilisant `NVL({quantityonhand},0)`, ils ont pu classer correctement les articles. Par exemple, les critères de recherche incluaient `NVL({quantityonhand},0) = 0` pour trouver tous les articles épuisés (Source: www.anchorgroup.tech). Sans NVL, les valeurs vides échoueraient à un filtre simple. Avec NVL, la recherche listait de manière fiable les produits en rupture de stock.

- **Segmentation de la taille des commandes (Reporting des ventes) :**
  Un directeur commercial souhaitait classer chaque commande par niveau de taille. En utilisant une formule CASE comme décrit précédemment (ex. `CASE WHEN {amount}>10000 THEN 'Large' ... END`) (Source: www.houseblend.io), les résultats de recherche pouvaient être regroupés par cette nouvelle colonne de formule. Cela permettait à l'équipe de voir rapidement combien de commandes « Grandes » par rapport aux « Moyennes » avaient été passées. Dans un autre scénario, DECODE a été utilisé pour étiqueter les opportunités : `DECODE({status}, 'WON', 'Closed Won', 'LOST', 'Closed Lost', 'Open')` convertissait facilement les codes de statut en mots, rationalisant ainsi les rapports.

- **Révision automatisée de la facturation (Étude de cas en santé) :**
  Un prestataire de soins de santé a mis en œuvre des recherches enregistrées pour repérer les services non facturés (le « Détecteur d'opportunités de revenus »). Ces recherches utilisaient des formules pour calculer les frais attendus et les comparer avec la facturation. En automatisant cela avec une logique CASE (ex. signaler les services où `{billed}=0`, etc.), ils ont considérablement réduit le temps de révision manuelle (Source: www.houseblend.io). L'étude de cas note : *« ils ont considérablement réduit le temps de révision manuelle des factures et amélioré la conformité »* (Source: www.houseblend.io), illustrant comment les champs de formule (utilisant peut-être NVL et CASE) ont directement permis des améliorations de processus.

- **Alertes d'inventaire (Étude de cas de détaillant) :**
  Autre exemple de Stockton10/Houseblend : un détaillant a créé une recherche enregistrée combinant des jointures et des formules pour surveiller les niveaux de stock (Source: www.houseblend.io). Ils ont utilisé des formules conditionnelles comme `CASE WHEN {onhand} < {reorderlevel} THEN 'Reorder' WHEN {onhand} < 0 THEN 'Check Qty' END` (conceptuellement). La recherche était planifiée quotidiennement pour alerter le service des achats. Ce scénario visant à *« signaler les produits en dessous des seuils de réapprovisionnement et les stocks vieillissants »* utilisait la logique CASE et éventuellement NVL pour classer les niveaux de stock, comme indiqué : *« stock < seuil de réapprovisionnement → 'Reorder', stock négatif → 'Check Qty' »* (Source: www.houseblend.io).

- **Recherche sur enregistrements joints (Cumul de compte) :**
  Certaines formules n'ont de sens que dans des recherches sommaires ou jointes. Par exemple, une entreprise peut regrouper les transactions par compte et utiliser `SUM({amount})` avec une colonne de formule telle que `CASE WHEN {accounttype} IN ('Revenue') THEN {amount} ELSE 0 END`. Dans ce cas, la formule porte sur un champ joint `{accounttype}`. La performance ici dépend du comportement de la jointure, mais ces formules avancées permettent des rapports complexes au sein de NetSuite.

- **Solution de contournement SuiteQL (Cas complexes) :**
  Lorsque les formules deviennent irréalisables, les entreprises passent parfois à des requêtes externes. Un fabricant mondial a découvert qu'une recherche enregistrée expirait systématiquement lors du filtrage entre les filiales. L'optimiseur de requêtes de NetSuite avait des difficultés avec les jointures. En migrant vers SuiteQL (SuiteScript ou ODBC) pour filtrer les filiales, puis en appelant ce résultat, ils ont réduit le temps de requête *de quelques minutes à quelques secondes* (Source: www.houseblend.io). Bien qu'il ne s'agisse pas strictement d'un exemple de fonction, cela illustre que les cas extrêmes de complexité des recherches enregistrées peuvent nécessiter des approches alternatives.

Données et impact : Ces cas montrent que les formules avancées se traduisent en données exploitables : par exemple, l'utilisation de NVL révèle les vraies valeurs zéro, la classification CASE met en évidence les priorités, et l'automatisation des tâches permet d'économiser des heures de travail. Bien que les statistiques formelles soient rares, une référence a noté qu'une *« conception appropriée des recherches enregistrées peut réduire considérablement le travail de reporting manuel et améliorer la précision des données »* (Source: www.houseblend.io). Le résumé des études de cas de Houseblend souligne en outre des améliorations telles que des réductions de 95 % du temps de chargement grâce à l'optimisation (Source: www.houseblend.io).

État actuel et paysage technologique

NetSuite continue de faire évoluer ses capacités d'analyse. En 2025-2026, les grandes tendances incluent :

- **SuiteAnalytics Workbooks :** Une interface utilisateur plus récente pour l'analyse qui peut reproduire de nombreuses fonctions de recherche enregistrée avec des requêtes basées sur SQL (Source: docs.oracle.com). Les Workbooks peuvent utiliser la même syntaxe de formule (CASE, fonctions DATE, etc.) mais dans une interface plus proche du tableur. Les utilisateurs doivent décider quand utiliser la recherche enregistrée par rapport au Workbook. Pour de nombreux besoins de tableaux de bord ad hoc, le Workbook offre de la flexibilité ; cependant, les recherches enregistrées restent populaires pour les rapports basés sur des listes et les alertes par e-mail.
- **Fonctionnalités basées sur l'IA :** La version 2026.1 de NetSuite a introduit des améliorations liées à l'IA. Par exemple, les *résumés IA pour les rapports* génèrent automatiquement des informations en langage clair à partir de données complexes (Source: netsuitechangelog.com). Bien que cela ne soit pas directement lié aux formules, cela reflète un changement : les utilisateurs pourraient davantage s'appuyer sur les résumés IA pour interpréter les données (qui pourraient être basées sur les résultats de recherches enregistrées). NetSuite a également ajouté des fonctionnalités pour construire des formules : par exemple, il existe un *générateur de formules assisté par IA* dans les prochaines versions qui peut suggérer une syntaxe basée sur les invites des utilisateurs (comme suggéré par les aperçus des partenaires).
- **Intégration et API (Connecteur MCP) :** NetSuite 2026.1 inclut également le *Connecteur IA NetSuite Analytics Warehouse*, connectant l'IA externe (ChatGPT, etc.) aux données internes (Source: projectsalsa.co.nz). Bien que plus axé sur l'intégration ERP, cela signifie à terme des suggestions possibles basées sur l'IA pour les formules de recherche enregistrée et les contrôles de qualité des données.
- **Améliorations des performances :** Les améliorations continues de l'indexation et de l'architecture de recherche de NetSuite continuent d'influencer les performances des formules. Le rapport Houseblend pointe vers des mises à niveau Oracle et suggère que le moteur de recherche de NetSuite pourrait être optimisé pour les modèles de formules courants (Source: www.houseblend.io). Les administrateurs doivent se tenir au courant des notes de version pour toute nouvelle fonction ou optimisation (par exemple, de nouvelles fonctions SQL ajoutées aux Workbooks), bien que les fonctions de base comme NVL et CASE restent stables.

Conclusion

Les formules de recherche enregistrée NetSuite telles que **NVL**, **CASE WHEN**, **DECODE** et **LTRIM** sont les pierres angulaires du reporting avancé dans SuiteAnalytics. Chaque fonction répond à un besoin distinct : NVL pour la gestion des valeurs nulles (Source: docs.oracle.com) (Source: www.anchorgroup.tech), CASE WHEN pour la logique conditionnelle (Source: docs.oracle.com) (Source: www.houseblend.io), DECODE pour le mappage rapide des valeurs (Source: docs.oracle.com) (Source: www.houseblend.io), et LTRIM pour le nettoyage des chaînes (Source: docs.oracle.com) (Source: www.netsuiterp.com). En comprenant leur syntaxe et leur sémantique (telles que documentées par Oracle et NetSuite (Source: docs.oracle.com) (Source: docs.oracle.com) et en appliquant les meilleures pratiques (par exemple, substituer NVL dans les formules numériques, structurer les expressions CASE, utiliser DECODE pour les recherches fixes, nettoyer les champs texte pour la cohérence), les administrateurs peuvent créer des recherches robustes.

Ce rapport a examiné chaque fonction en profondeur, en citant des sources faisant autorité et des exemples concrets. Nous avons présenté des recettes de formules spécifiques (tableaux) et des études de cas narratives pour démontrer leur puissance. Nous avons également mis en évidence les pièges, tels que la nécessité d'une correspondance appropriée des types de données, les impacts potentiels sur les performances des formules complexes et les différences entre des constructions équivalentes. Notamment, l'optimisation de l'utilisation des formules peut donner des résultats spectaculaires : un exemple a vu les temps de chargement des pages réduits de plus de 90 % lorsqu'une formule a été remplacée par un simple filtre (Source: www.houseblend.io).

Enfin, nous avons replacé ce sujet dans le contexte de l'avenir de NetSuite. Alors que NetSuite s'oriente vers l'analyse assistée par l'IA et l'analytique intégrée (SuiteAnalytics Workbooks et outils d'IA (Source: www.houseblend.io) (Source: netsuitechangelog.com), les fonctions SQL de base dans les recherches enregistrées restent fondamentales. Les compétences en NVL, CASE, DECODE et LTRIM continueront d'être précieuses, même à mesure que de nouveaux outils émergeront. Les modèles de formules que nous avons examinés seront probablement transposés dans les requêtes SuiteQL et la génération de code pilotée par l'IA. Par conséquent, la maîtrise de ces fonctions n'est pas seulement une compétence héritée, mais une porte d'entrée pour exploiter tout le potentiel de reporting de NetSuite.

Tableau 2 : Études de cas réelles de recherches enregistrées

SCÉNARIO	FORMULE/APPROCHE	RÉSULTAT/IMPACT
Détection des revenus dans le secteur de la santé (Source: www.houseblend.io)	Recherche automatisée utilisant des formules (ex: NVL/CASE) pour détecter les services non facturés et les renouvellements.	*Réduction drastique des révisions manuelles de factures ; amélioration de la conformité de facturation.*
Suivi des stocks de détail (Source: www.houseblend.io)	Formules CASE pour signaler les articles à faible stock : `CASE WHEN {onhand}<{reorder} THEN 'Reorder' WHEN {onhand}<0 THEN 'Check Qty' END`.	*Alertes en temps réel sur les besoins de réapprovisionnement ; recherche enregistrée planifiée quotidiennement pour plus d'efficacité.*
Finance mondiale multi-filiales (Source: www.houseblend.io)	Les jointures complexes de filiales ont été remplacées par des requêtes SuiteQL (déchargeant la logique de formule).	*Temps de requête du rapport réduit de quelques minutes à quelques secondes en utilisant SuiteQL au lieu d'une seule recherche enregistrée lourde.*

Tableau 2 : Exemples illustratifs de formules de recherche enregistrées en action. Chaque cas d'utilisation cite des sources de l'industrie (Source: www.houseblend.io) (Source: www.houseblend.io).

Cette exploration complète montre que le « livre de recettes des formules » composé de NVL, CASE WHEN, DECODE et LTRIM est une référence essentielle pour tout administrateur ou développeur NetSuite. En comprenant profondément ces fonctions, il est possible de créer des recherches enregistrées sophistiquées qui fournissent des informations précises et exploitables à partir des données ERP.

Références : Tous les détails techniques et exemples de ce rapport sont tirés de la documentation d'Oracle (Source: docs.oracle.com) (Source: docs.oracle.com) (Source: docs.oracle.com) (Source: docs.oracle.com), des pages d'aide/guides officiels de NetSuite (Source: docs.oracle.com) (Source: docs.oracle.com), de blogs et articles d'experts (Source: suiterep.com) (Source: www.houseblend.io) (Source: www.anchorgroup.tech), et d'études de cas provenant de sources industrielles (Source: www.houseblend.io) (Source: www.houseblend.io). Chaque affirmation spécifique est notée ci-dessus avec la source appropriée.

Étiquettes: recherche-enregistree-netsuite, formules-netsuite, oracle-sql, nvl, case-when, decode, ltrim, logique-conditionnelle, gestion-des-valeurs-nulles

---