NetSuite SuiteQL CustomField : Étiquette vs Nom et types de champs

Publié le 23 avril 2026 32 min de lecture

Résumé analytique

SuiteQL est le langage de requête basé sur SQL de NetSuite qui libère toute la puissance du modèle de données de NetSuite pour une analyse avancée. La personnalisation de NetSuite (« champs personnalisés ») ajoute des milliers de champs définis par l'utilisateur à travers les enregistrements, et comprendre comment interroger ces champs via SuiteQL nécessite une connaissance des tables de métadonnées de NetSuite. En particulier, la table de métadonnées **CustomField** contient des informations essentielles sur chaque champ personnalisé : son étiquette d'affichage, son identifiant interne, son type de données et d'autres propriétés. Une source majeure de confusion est la différence entre l' *étiquette* (label) d'un champ (le texte affiché dans l'interface utilisateur) et son *nom* ou *Script ID* (l'identifiant unique en arrière-plan utilisé dans les scripts et les requêtes). Ce rapport approfondit la structure et l'utilisation des métadonnées des champs personnalisés de NetSuite dans SuiteQL : nous expliquons comment les étiquettes et les Script IDs sont représentés, comment le type du champ est encodé (y compris le placement « Corps vs Colonne » et le type de données), et comment des indicateurs comme **IsMandatory** ou **IsInactive** sont (ou ne sont pas) exposés. Nous examinons le contenu de la table `CustomField` et des tables associées (telles que `CustomRecordType`, `CustomList` et `CustomSegment`), et citons des requêtes documentées pour illustrer ces concepts.

Nous fournissons des exemples détaillés et des études de cas montrant comment les analystes et les intégrateurs utilisent SuiteQL pour découvrir des champs personnalisés, les joindre aux données et créer des rapports. Par exemple, nous analysons une requête SuiteQL publiée qui inventorie tous les champs personnalisés à des fins d'audit (Source: support.cloudextend.io), et nous discutons de la manière dont les organisations intègrent les données SuiteQL avec des outils de BI (par exemple, Tableau) pour obtenir des informations (Source: coefficient.io). Tout au long du document, des commentaires d'experts et la documentation officielle guident notre analyse. Nous concluons en discutant des implications en termes de performance et de gouvernance liées à une utilisation intensive de SuiteQL, ainsi que des orientations futures (telles que les API de métadonnées améliorées et les outils de requête pilotés par l'IA qui pourraient davantage faciliter l'analyse dans NetSuite. Toutes les affirmations et descriptions sont fondées sur des sources crédibles allant des guides de développement NetSuite aux experts communautaires expérimentés.

Introduction et contexte

NetSuite est une plateforme ERP/CRM basée sur le cloud de premier plan qui unifie les données financières, d'inventaire, de vente et client (Source: www.houseblend.io). Pour répondre à des besoins métier complexes, NetSuite permet à pratiquement chaque enregistrement de posséder de nombreux **champs personnalisés**. Ces champs peuvent être de divers types de données (texte, liste, case à cocher, etc.) et peuvent être ajoutés aux transactions, aux enregistrements clients, aux articles, aux enregistrements personnalisés, et plus encore. Les métadonnées de ces champs (noms, types et associations) résident dans le schéma de données de NetSuite. Historiquement, l'extraction et l'analyse des valeurs des champs personnalisés étaient difficiles : les utilisateurs devaient s'appuyer sur des Recherches enregistrées ou sur le SuiteAnalytics Workbook, qui présentent des limites (par exemple, un seul niveau de jointures, agrégation limitée) (Source: www.houseblend.io). SuiteQL, introduit vers 2020, fournit une interface de requête entièrement conforme à SQL-92 vers la **source de données analytiques** de NetSuite (souvent appelée NetSuite2.com) (Source: www.houseblend.io) (Source: www.houseblend.io). Selon la documentation d'Oracle, « SuiteQL est un langage de requête puissant... permettant des requêtes avancées au-delà des capacités des recherches enregistrées et des rapports » (Source: www.houseblend.io). En pratique, SuiteQL peut accéder aux mêmes données que celles visibles dans les workbooks ou les recherches enregistrées de NetSuite, mais avec toute la flexibilité du SQL (jointures, sous-requêtes, unions, etc.) (Source: www.houseblend.io) (Source: www.houseblend.io). Comme le note un expert, « SuiteQL alimente la source de données SuiteAnalytics, garantissant que toute donnée visible dans un Workbook ou une Recherche enregistrée NetSuite peut également être interrogée via SuiteQL... SuiteQL permet des jointures multi-tables complexes, des sous-requêtes et des agrégations, ouvrant la voie à des analyses plus approfondies qui pourraient être fastidieuses ou impossibles avec les seules recherches enregistrées » (Source: www.houseblend.io).

Comprendre comment les *champs personnalisés* apparaissent dans SuiteQL nécessite d'examiner deux parties du schéma de NetSuite : (1) les **tables de métadonnées** qui décrivent les définitions personnalisées (noms des champs, types, etc.), et (2) les **tables analytiques** réelles où résident les valeurs des données. SuiteQL fournit un accès en lecture seule à ces tables. Les tables de métadonnées incluent **CustomRecordType** (listant tous les types d'enregistrements personnalisés) et **CustomField** (listant tous les champs personnalisés à travers les enregistrements). Par exemple, interroger `CustomRecordType` renvoie des colonnes comme *Name, ScriptID, InternalID, Description* (Source: timdietrich.me), tandis qu'interroger `CustomField` renvoie des colonnes incluant *Name, ScriptID, Description, FieldType, FieldValueType, FieldValueTypeRecord, IsMandatory, IsStored, IsShowInList* et des informations sur le propriétaire (Source: www.houseblend.io) (Source: timdietrich.me). Il est crucial de noter que dans ces tables, la colonne `Name` contient en réalité l' *étiquette visible par l'utilisateur* du champ, tandis que `ScriptID` contient le nom programmatique unique (par exemple `custentity_myfield`) (Source: www.houseblend.io) (Source: blogs.oracle.com). Cette distinction — *étiquette vs nom* — est un point central de ce rapport.

Dans les sections qui suivent, nous détaillons d'abord le contenu et la signification de la table `CustomField` et des tables associées. Nous analysons ensuite comment les requêtes SuiteQL font référence aux champs personnalisés (en mettant l'accent sur la référence par étiquette vs scriptID, et sur l'interprétation des indicateurs de type et d'activité). Nous incluons des exemples SuiteQL illustratifs provenant d'experts et de la documentation Oracle, et nous complétons avec des données provenant de NetSuite et d'études de cas d'intégration. Enfin, nous discutons des implications pour la performance, la sécurité/gouvernance et les fonctionnalités à venir dans l'écosystème SuiteQL.

Métadonnées SuiteQL pour les champs personnalisés

La table `CustomRecordType`

Chaque type d'enregistrement personnalisé dans NetSuite (créé via *Personnalisation > Listes, Enregistrements et Champs*) est catalogué dans la table `CustomRecordType` de SuiteQL. Chaque ligne représente une définition d'enregistrement personnalisé. Les colonnes clés incluent `Name` (l'étiquette du type d'enregistrement), `ScriptID` (par exemple `customrecord_myrecord`), `InternalID` (l'identifiant interne numérique de NetSuite) et des indicateurs comme `AllowQuickSearch` ou `AllowAttachments` (Source: timdietrich.me). Par exemple, une requête SuiteQL `SELECT Name, ScriptID, InternalID, Description FROM CustomRecordType ORDER BY Name;` listera tous les types d'enregistrements personnalisés dans le compte (Source: timdietrich.me). Si un analyste souhaite interroger les champs personnalisés pour un enregistrement personnalisé donné, il trouverait d'abord l'ID interne de cet enregistrement ici, puis l'utiliserait pour filtrer la table `CustomField` (via `WHERE RecordType = <ID>`).

L'importance de `CustomRecordType` réside dans le fait qu'il fournit le lien interne : la colonne `InternalID` de `CustomRecordType` est utilisée dans `CustomField.RecordType` pour indiquer à quel enregistrement ce champ appartient (Source: timdietrich.me). Ainsi, on peut joindre ou filtrer, par exemple, `CustomField` sur `RecordType = CustomRecordType.InternalID` pour isoler les champs d'un enregistrement personnalisé particulier. Ce lien est particulièrement important car SuiteQL ne permet pas directement d'interroger les métadonnées des enregistrements *standard* ; il n'expose que les métadonnées des éléments définis par l'utilisateur (personnalisés). (Les enregistrements et champs standard doivent être obtenus via l'API Records Catalog ou le point de terminaison de métadonnées REST.)

La table `CustomField`

Le référentiel central des métadonnées des champs personnalisés est la table `CustomField`. Chaque ligne de cette table est une définition de champ. Les colonnes de `CustomField` capturent toutes les propriétés clés d'un champ personnalisé. Selon les sources d'Oracle et de la communauté SuiteQL, les colonnes principales sont :

- **Name** (STRING) – l'étiquette d'affichage du champ personnalisé, c'est-à-dire ce que l'utilisateur voit sur les formulaires. Par exemple, un champ pourrait avoir le nom « Âge du client ».
- **ScriptID** (STRING) – l'ID programmable interne, tel que `custentity_age`. Il est unique par champ et utilisé dans les scripts et les requêtes.
- **Description** (STRING) – tout texte descriptif saisi par l'administrateur pour le champ.
- **RecordType** (INTEGER) – l'ID interne du type d'enregistrement (`CustomRecordType.InternalID`) auquel ce champ appartient. (Pour les champs d'enregistrement personnalisé, cela pointe vers l'ID de l'enregistrement personnalisé. Pour les champs de transaction ou d'entité, cela pointe vers l'ID de l'enregistrement standard.)
- **FieldType** (STRING) – indique le placement du champ dans l'enregistrement (par exemple `'BODY'` ou `'COLUMN'`). En d'autres termes, **FieldType** catégorise si le champ personnalisé est un champ de corps (dans le corps principal du formulaire) ou un champ de colonne (partie d'une sous-liste). Par exemple, les requêtes dans l'exemple Oracle filtrent `CustomField.fieldType IN ('BODY','COLUMN')` pour obtenir les champs sur l'enregistrement de facture fournisseur (Source: blogs.oracle.com).
- **FieldValueType** (STRING) – le type de données réel du champ. Cela reflète le type NetSuite (Case à cocher, Date, Texte, Liste/Enregistrement, etc.). Par exemple, **FieldValueType** pourrait être `'CheckBox'`, `'Date'`, `'LongText'` ou `'List/Record'`. (Il existe une certaine confusion dans les sources communautaires sur la terminologie, mais les conseils faisant autorité indiquent que **FieldValueType** contient le type de données du champ. La requête de Tim Dietrich montre `FieldValueType` et `FieldValueTypeRecord` côte à côte, impliquant que **FieldValueType** est le type du champ (Source: timdietrich.me).)
- **FieldValueTypeRecord** (INTEGER) – si **FieldValueType** est `'List/Record'` ou une sélection multiple, cette colonne stocke l'ID interne du type de liste ou d'enregistrement utilisé. Par exemple, si un champ personnalisé est une liste déroulante de *Départements*, cette colonne pourrait contenir l'ID du type d'enregistrement Département. Les exemples de blog Oracle joignent `CustomField.fieldValueTypeRecord = ScriptRecordType.internalId` (après avoir filtré les types de champs appropriés) pour identifier la liste/l'enregistrement réel référencé (Source: blogs.oracle.com).
- **IsMandatory** (BOOLEAN) – vrai (`'T'`) si le champ est marqué comme « obligatoire », sinon faux.
- **IsStored** (BOOLEAN) – vrai si la valeur du champ est stockée dans la base de données (par opposition à un champ de formule sans stockage).
- **IsShowInList** (BOOLEAN) – vrai si le champ est configuré pour apparaître dans les vues de liste.
- **Owner** (REFERENCE) – ID interne de l'utilisateur qui possède le champ (généralement l'administrateur qui l'a créé). (Les requêtes SuiteQL enveloppent souvent `BUILTIN.DF(Owner)` pour obtenir le nom du propriétaire.)
- **Indicateurs supplémentaires** – d'autres indicateurs booléens comme `IsFormulaCheckbox`, `IsFormulaNumeric`, etc., dépendent de la catégorie du champ (s'il s'agit d'une formule). Ceux-ci existent dans la définition mais ne sont généralement pas utilisés dans les rapports de base.
- **LastModifiedDate** (TIMESTAMP) – date et heure auxquelles les métadonnées du champ ont été modifiées pour la dernière fois (visibles via SuiteAnalytics Connect/ODBC, ou via SuiteScript*).
- (*Note : Dans SuiteQL directement, la colonne peut être nommée `lastModifiedDate` ou similaire, comme on le voit dans la requête de CloudExtend.)

Ces colonnes peuvent être résumées comme suit :

COLONNE	DESCRIPTION
Name	Étiquette du champ (chaîne vue dans l'interface utilisateur) (Source: www.houseblend.io).
ScriptID	ID de champ interne unique (par exemple `custentity_age`) (Source: www.houseblend.io).
Description	Description du champ saisie par l'administrateur.
RecordType	ID interne du type d'enregistrement (table) auquel appartient le champ (joindre à CustomRecordType/InternalID).
FieldType	Placement du champ dans l'enregistrement ('BODY' vs 'COLUMN', c'est-à-dire champ de corps ou colonne de sous-liste) (Source: blogs.oracle.com).
FieldValueType	Type de données du champ (Case à cocher, Texte, Date, Liste/Enregistrement, etc.) (Source: timdietrich.me).
FieldValueTypeRecord	Si `FieldValueType='List/Record'`, l'ID interne du type d'enregistrement ou de liste référencé (Source: blogs.oracle.com) ; sinon NULL.
IsMandatory	*T* si le champ est requis, sinon *F*.
IsStored	*T* si la valeur est stockée (comme dans la plupart des champs) ; un champ de schéma pour les formules.
IsShowInList	*T* si affiché dans la vue de liste.
Owner	ID interne du propriétaire du champ (BUILTIN.DF(Owner) renvoie le nom de l'utilisateur).
LastModifiedDate	Horodatage de la dernière modification du champ (là où exposé).

Il est important de noter, comme cité par Houseblend, que la colonne **Name** dans `CustomField` est bien *l'étiquette du champ* (Source: www.houseblend.io). Les utilisateurs de SuiteQL l'aliasent souvent en « label » dans les requêtes, par exemple `CustomField.name AS label` (Source: blogs.oracle.com). La colonne **ScriptID** est le nom technique du champ. Par exemple, la requête du blog Oracle montre :

```
CustomField.scriptId AS id,
CustomField.name AS label,
...
```

soulignant cette distinction (Source: blogs.oracle.com). Dietrich et Oracle sont d'accord : utilisez `ScriptID` pour faire référence au champ en interne, et `Name` pour le texte d'affichage.

Un autre champ d'intérêt est **FieldType** vs **FieldValueType**. La confusion survient parfois parce que différentes sources semblent interchanger ces termes. Dans `CustomField` de SuiteQL, **FieldType** contient des valeurs comme `'BODY'` ou `'COLUMN'` (l'emplacement du champ sur le formulaire) (Source: blogs.oracle.com), tandis que **FieldValueType** contient le type de données réel (par exemple Case à cocher, Date, Liste/Enregistrement) comme le suggère la requête de Dietrich (Source: timdietrich.me). (La description de Houseblend semblait appeler `FieldType` le type d'interface utilisateur, mais les requêtes faisant autorité indiquent que le type d'interface utilisateur du champ se trouve en fait dans `FieldValueType`.) Nous ferons référence à `FieldType` comme corps/colonne et à `FieldValueType` comme type de données dans notre analyse.

Enfin, notez qu'il n'y a **pas de colonne `IsInactive` sur la table `CustomField` elle-même**. Dans NetSuite, les champs personnalisés peuvent être « inactifs » (cachés de l'utilisation), mais cet indicateur de métadonnées n'est pas exposé dans la table `CustomField` de SuiteQL. En revanche, **d'autres définitions personnalisées** ont un indicateur `IsInactive` : par exemple, la table `CustomList` a `IsInactive` pour marquer si une liste personnalisée est désactivée (Source: timdietrich.me), et la table `CustomSegment` en a un aussi (Source: blogs.oracle.com). Mais le fait que `IsInactive` soit absent des requêtes publiées sur `CustomField` signifie que SuiteQL listera tous les champs (actifs ou non). Si l'on doit simuler les

inactifs, l'étiquette de l'interface utilisateur ou d'autres indices doivent être utilisés (mais cela n'est pas documenté). En pratique, presque toutes les requêtes de métadonnées supposent simplement que les champs sont actifs ; les administrateurs nettoient généralement les champs inutilisés à l'aide d'outils de gouvernance distincts.

Exemple : Interroger `CustomField`

Une utilisation typique de la table `CustomField` consiste à **lister tous les champs pour un type d'enregistrement donné**. Par exemple, Tim Dietrich démontre l'interrogation des champs personnalisés pour une « commande client personnalisée » spécifique (ID interne 297 dans son compte) comme suit (Source: timdietrich.me) (Source: timdietrich.me) :

```
SELECT
    Name,
    ScriptID,
    Description,
    FieldType,
    FieldValueType,
    FieldValueTypeRecord,
    BUILTIN.DF(FieldValueTypeRecord) AS FieldValueTypeRecordName,
```

IsMandatory, IsStored, IsShowInList, BUILTIN.DF(Owner) AS Owner FROM CustomField WHERE RecordType = 297;

```


Cela renvoie chaque champ personnalisé sur ce bon de commande, en affichant son étiquette (`Name`), son ID interne (`Scri

Le guide de Houseblend note de la même manière que la table `CustomField` « stocke les définitions des champs personnalis

## Références de valeur de champ (`FieldValueTypeRecord`) et jointures

Pour les champs de type *Liste/Enregistrement* ou à sélection multiple, `CustomField.FieldValueTypeRecord` indique de que

```sql
COALESCE(
    CustomField.fieldValueType,
    CASE
      WHEN CustomField.fieldValueType = 'List/Record' THEN CustomField.fieldValueType
      ELSE ScriptRecordType.name
    END
) AS type,
CASE WHEN CustomField.fieldValueType <> 'List/Record' THEN ''
     ELSE ScriptRecordType.skey
END AS listRecord
```

avec un `CROSS JOIN ScriptRecordType` filtrant sur une correspondance entre `CustomField.fieldValueTypeRecord` et `ScriptRecordType.name` (Source: blogs.oracle.com) (Source: blogs.oracle.com). (Cette requête est quelque peu avancée ; l'essentiel est qu'il est souvent nécessaire de consulter à la fois la définition du champ personnalisé et le catalogue standard des enregistrements/tables pour résoudre complètement « de quelle liste ce champ tire-t-il ses données ».) Pour la plupart des besoins en reporting, il suffit de noter : si un champ personnalisé est défini comme une liste déroulante, `FieldValueTypeRecord` vous indique *quelle* liste déroulante (quelle table ou liste) est utilisée. SuiteQL peut joindre cela à la table associée (par exemple, la table de recherche « États », ou un enregistrement personnalisé).
```

Notamment, le **Catalogue des enregistrements** (accessible via le centre d'aide de NetSuite ou les métadonnées REST) est l'endroit officiel pour parcourir tous les types d'enregistrements et les noms de champs. La FAQ de la documentation SuiteQL dirige explicitement les utilisateurs vers le catalogue des enregistrements pour obtenir une liste des enregistrements et des champs (Source: blogs.oracle.com).

Les tables `CustomList` et `CustomSegment` (Contexte)

Bien que nous nous concentrions sur les champs personnalisés, il est utile de comparer avec deux tables connexes :

- **CustomList** : Les listes de valeurs (options de liste déroulante) sont elles-mêmes créées en tant que « Listes personnalisées ». La table `CustomList` dans SuiteQL inclut des colonnes telles que `Name`, `ScriptID`, `Description`, `Owner`, `IsOrdered` et, surtout, `IsInactive`. Dans le blog de Dietrich sur SuiteQL et les listes personnalisées, il interroge `CustomList` pour `Name, Description, ScriptID, Owner, IsOrdered` **et ajoute** `WHERE (IsInactive = 'F')` pour exclure les listes désactivées (Source: timdietrich.me). Cela montre comment SuiteQL peut accéder aux métadonnées de liste et filtrer celles qui sont inactives. Ensuite, pour obtenir les valeurs d'une liste spécifique, il utilise le ScriptID de la liste comme nom de table (par exemple, la table `CUSTOMLIST_BED_SIZE`) et filtre à nouveau `WHERE IsInactive = 'F'` sur ces valeurs (Source: timdietrich.me).

- **CustomSegment** : Les « Segments personnalisés » (classifications) peuvent également être attachés aux transactions. L'exemple du blog SuiteQL pour les factures fournisseurs récupère `CustomSegment.scriptId` et `name`, en filtrant `WHERE CustomSegment.IsInactive <> 'T'` (Source: blogs.oracle.com) pour ignorer les segments inactifs. Notez que la colonne `name` contient à nouveau l'étiquette d'affichage du segment, tandis que `scriptId` est l'identifiant programmatique.

Ces tables illustrent comment `IsInactive` est géré ailleurs. `CustomList` et `CustomSegment` *incluent* bien un champ `IsInactive`, et les requêtes utiles le filtrent explicitement (par exemple `IsInactive = 'F'` ou `<> 'T'`) (Source: timdietrich.me) (Source: blogs.oracle.com). En revanche, aucun filtre de ce type n'est montré pour `CustomField` ; **cela semble être dû au fait que** `CustomField` **ne possède pas de colonne** `IsInactive` (d'où l'absence de `WHERE IsInactive` dans les exemples de requêtes sur CustomField (Source: www.houseblend.io) (Source: blogs.oracle.com). En pratique, si un champ personnalisé est inactif, SuiteQL le renvoie simplement quand même. Les administrateurs qui souhaitent ignorer les champs inactifs doivent le faire par d'autres moyens (par exemple, en vérifiant la colonne `Status` via SuiteScript, ou en excluant les champs contenant « (Inactive) » dans leur nom, bien qu'il s'agisse d'heuristiques non documentées).

Résumé des métadonnées de `CustomField`

Le point clé pour les utilisateurs de SuiteQL est d'utiliser les bonnes colonnes :

- **Étiquette vs Nom** : Utilisez `CustomField.name` pour obtenir l'étiquette lisible par l'humain, et `CustomField.scriptId` pour l'identifiant interne (Source: www.houseblend.io) (Source: blogs.oracle.com). Une requête peut leur donner des alias pour plus de clarté, par exemple `CustomField.name AS "Label", CustomField.scriptId AS "ID"`. Rappelez-vous toujours que `name` est un texte d'*affichage*, non adapté aux jointures avec d'autres tables.

- **Type** : Utilisez `FieldValueType` pour voir quel type de données ce champ contient (et éventuellement `FieldValueTypeRecord` pour connaître le type de liste pour les listes déroulantes) (Source: timdietrich.me) (Source: blogs.oracle.com). La colonne `FieldType` (parfois mal interprétée) concerne en réalité les éléments de corps (body) ou de colonne et est principalement utilisée pour filtrer la partie de l'enregistrement que vous ciblez (Source: blogs.oracle.com). Par exemple, filtrer `WHERE FieldType='BODY'` donnera les champs de **corps** personnalisés, par opposition à `FieldType='COLUMN'` pour les colonnes de sous-liste personnalisées.

- **Activité** : Contrairement aux listes et aux segments, `CustomField` n'a **pas** d'indicateur `IsInactive` exposé. Tous les champs définis apparaîtront dans les requêtes. Lors d'audits ou de nettoyages, cela signifie qu'il faut inclure une logique en dehors de SuiteQL pour omettre les champs inactifs.

Le tableau suivant consolide les principales colonnes de la table `CustomField`, basées sur les sources NetSuite :

COLONNE (SUITEQL)	SIGNIFICATION / NOTES
Name	Étiquette du champ (nom d'affichage). Ex. « Customer Age » (Source: www.houseblend.io).
ScriptID	ID interne du champ. Ex. `custentity_age` (Source: www.houseblend.io).
Description	Description du champ (mémo optionnel).
RecordType	ID interne de l'enregistrement parent (joindre à `CustomRecordType`).
FieldType	Emplacement : généralement `'BODY'` ou `'COLUMN'` (Source: blogs.oracle.com).
FieldValueType	Type de données (Case à cocher, Date, Texte long, Liste/Enregistrement, etc.) (Source: timdietrich.me).
FieldValueTypeRecord	Si le type de données est Liste/Enregistrement, l'ID interne de ce type de liste/enregistrement (Source: blogs.oracle.com). Sinon NULL.
IsMandatory	Indicateur (vrai/faux) pour les champs obligatoires.
IsStored	Vrai si les valeurs sont stockées (c.-à-d. pas seulement une mise en page de formule).
IsShowInList	Vrai s'il s'affiche dans les vues de liste.
Owner	ID utilisateur interne (admin) qui possède le champ.
LastModifiedDate	Horodatage de la dernière modification (tel qu'exposé dans l'analytique).

(*Sources : Formation SuiteQL de NetSuite par Tim Dietrich et analyse de Houseblend (Source: www.houseblend.io) (Source: timdietrich.me). Dans ces sources, `Name` est explicitement l'étiquette et `ScriptID` l'identifiant unique.*)

Étiquette de champ vs Nom de champ

Un thème central dans l'interrogation des champs personnalisés est la différence entre l' *étiquette* d'un champ et son *nom interne/ID de script*. Dans le jargon NetSuite, **étiquette** fait référence au texte affiché aux utilisateurs finaux sur les formulaires, tandis que **nom** (ou « ID » ou « ID de script ») est l'identifiant unique utilisé en coulisses. La table `CustomField` de SuiteQL aide à clarifier cela :

- La colonne `Name` (parfois affichée sous le nom `CustomField.name`) contient l'**étiquette** du champ. Il s'agit d'un champ de texte libre défini lors de la création du champ personnalisé. Par exemple, si une entreprise crée un champ personnalisé appelé « Customer Rating », cette chaîne « Customer Rating » est le `Name`.
- La colonne `ScriptID` est l'**ID de script** ou « nom de champ » du champ. Il a généralement un préfixe comme `custentity` (pour les champs d'entité) ou `custcol` (pour les champs de colonne) et est généré automatiquement en fonction de l'étiquette (bien qu'il puisse être modifié dans une certaine mesure). Par exemple, le ScriptID du champ précédent pourrait être `custentity_customerrating`. Les IDs de script sont uniques et ne contiennent pas d'espaces.
- Comme le note Houseblend, `CustomField.Name` est littéralement l'« étiquette du champ », tandis que `CustomField.ScriptID` est l'« identifiant de chaîne unique » (Source: www.houseblend.io). Dans les requêtes, on leur donne souvent des alias (par exemple `CustomField.name AS label, CustomField.scriptId AS id`) pour éviter toute confusion, comme le fait la documentation d'Oracle (Source: blogs.oracle.com).

Il est important de **ne pas les confondre**. De nombreux outils de métadonnées afficheront le ScriptID lors de la liste des champs, ce qui peut induire en erreur les utilisateurs non familiers avec les conventions de nommage de NetSuite. À l'inverse, utiliser l'étiquette à la place du ScriptID dans les jointures ou les filtres échouera. Par exemple, si vous souhaitez sélectionner des données de la transaction qui utilise le champ personnalisé, vous devez y faire référence par son ScriptID dans le `SELECT` SuiteQL (par exemple `entity.custentity_customerrating`), et non par son étiquette. L'étiquette n'est pas reconnue dans les requêtes SuiteQL.

La différence a également des implications pour les enregistrements renommés. La FAQ SuiteQL d'Oracle avertit que la fonction intégrée `BUILTIN.DF()` renvoie l'étiquette d'affichage, et non le nom original (Source: blogs.oracle.com). Par exemple, si l'enregistrement « Department » a été renommé « Cost Center » dans une interface utilisateur, alors `BUILTIN.DF(CustomField.fieldValueTypeRecord)` pourrait renvoyer « Cost Center », mais le scriptID réel reste « department ». Si l'on utilise par erreur ce nom d'affichage comme ID dans une requête ultérieure, celle-ci échouera. En bref, SuiteQL repose toujours sur les IDs de script ou les IDs internes pour les jointures, et `Name` (étiquette) est purement lisible par l'humain.

Types de données et références de liste/enregistrement

La colonne `FieldValueType` dans `CustomField` encode le type de données que chaque champ contient. Les valeurs courantes incluent (mais ne sont pas limitées à) :

- *Case à cocher* (indicateur booléen)
- *Texte long* (texte multiligne)
- *Texte libre* (chaîne sur une seule ligne)
- *Date* / *Date/Heure*
- *Nombre décimal*, *Pourcentage*, etc.
- *Liste/Enregistrement* (une référence à un autre type d'enregistrement)
- *Sélection multiple* (permettant plusieurs valeurs de liste)

Lorsque `FieldValueType = 'List/Record'` ou un type à sélection multiple, la colonne `FieldValueTypeRecord` indique de quelle liste d'enregistrements il provient. Pour interpréter cela, les utilisateurs de SuiteQL joignent souvent la table **ScriptRecordType** (ou **CustomRecordType** s'il s'agit d'une liste personnalisée) pour obtenir le nom de cet enregistrement référencé. Par exemple, si les choix de liste déroulante d'un champ personnalisé proviennent de la liste « Department » (un enregistrement standard), alors `FieldValueTypeRecord` pourrait être l'ID interne de « department ». Les fonctions intégrées peuvent traduire cela en ID de script ou en étiquette : on pourrait utiliser `BUILTIN.DF(CustomField.fieldValueTypeRecord)` pour obtenir « Department » (Source: blogs.oracle.com), bien qu'une certaine prudence soit nécessaire avec les étiquettes renommées (Source: blogs.oracle.com).

Les exemples SuiteQL utilisent fréquemment `BUILTIN.DF` pour obtenir des noms d'affichage pour les IDs référencés. Par exemple, la requête de Tim inclut `BUILTIN.DF(FieldValueTypeRecord) AS FieldValueTypeRecordName` (Source: timdietrich.me). De même, dans l'exemple d'Oracle, une jointure croisée avec `ScriptRecordType` est effectuée afin de pouvoir récupérer `ScriptRecordType.name` (l'ID de script de l'enregistrement référencé) au lieu de l'étiquette d'affichage (Source: blogs.oracle.com) (Source: blogs.oracle.com). Cette distinction est importante pour interpréter les résultats : vous pourriez voir un département nommé « Cost Center », mais en interne, il s'agit toujours de `"department"` dans le contexte SuiteQL.

Il est important de noter que si le champ n'est pas basé sur une liste (par exemple, une case à cocher ou un champ texte), alors `FieldValueTypeRecord` sera nul/vide. Les requêtes SuiteQL prévoient souvent des protections en conséquence (par exemple, les instructions CASE dans l'exemple d'Oracle vérifient si le champ est une liste/enregistrement ou non) (Source: blogs.oracle.com). Dans la plupart des requêtes pratiques, on ignore `FieldValueTypeRecord` pour les non-listes ou on l'utilise uniquement comme clé de jointure pour les types de liste.

Champs inactifs et filtrage

Contrairement à `CustomList` ou `CustomSegment`, **la table `CustomField` ne possède pas d'indicateur « IsInactive »** que les utilisateurs de Studio peuvent interroger. Par conséquent, SuiteQL renverra les définitions des champs personnalisés actifs et inactifs sans distinction. Si un utilisateur souhaite exclure les champs inactifs, il doit le faire par un filtrage a posteriori sur des modèles connus (par exemple, certains champs inactifs ont « (Inactive) » ajouté à leur étiquette dans l'interface utilisateur) ou en comparant `lastModifiedDate` / `Owner` (s'ils savent qu'un champ inactif n'a pas été touché récemment). Cependant, aucune colonne de métadonnées explicite n'indique le statut actif des champs personnalisés.

En revanche, SuiteQL fournit des colonnes `IsInactive` bien définies pour plusieurs autres types d'enregistrements, et les requêtes les filtrent généralement. Par exemple :

- **CustomList.IsInactive** – La requête de liste personnalisée de Dietrich utilise explicitement `WHERE (IsInactive = 'F')` pour omettre les listes inactives (Source: timdietrich.me).
- **CustomSegment.IsInactive** – L'exemple d'extraction de métadonnées filtre `WHERE CustomSegment.IsInactive <> 'T'` pour ignorer les segments inactifs (Source: blogs.oracle.com).

- **Subsidiary.IsInactive**, **Department.IsInactive**, etc. – Les enregistrements standard comme les filiales ont également une colonne `isInactive` (où `'T'` signifie que l'entité est désactivée). Les utilisateurs de SuiteQL incluent souvent `WHERE isInactive = 'F'` lors de l'interrogation de listes standard d'entités.

En résumé, **pour exclure les champs personnalisés inactifs dans SuiteQL, il faut être vigilant en dehors de la table `CustomField` elle-même**. Une approche consiste à utiliser le **Catalogue des enregistrements** via le service de métadonnées REST pour obtenir la liste actuelle des champs (qui respecte le statut actif) puis à filtrer les résultats SuiteQL par rapport à celle-ci. Actuellement, SuiteQL ne dispose pas de filtre intégré pour les éléments inactifs sur `CustomField`.

Exemples de requêtes et études de cas

Les requêtes SuiteQL sur les champs personnalisés répondent à de nombreux besoins pratiques. Nous passons ci-dessous en revue quelques scénarios courants, en nous appuyant sur des exemples publiés.

- **Inventaire des champs personnalisés (Audit/Nettoyage)** : Les administrateurs ont souvent besoin d'une liste complète des champs personnalisés pour effectuer des audits et du nettoyage. Le centre d'aide de CloudExtend fournit un exemple de requête SuiteQL pour un « Inventaire des champs personnalisés » qui sélectionne le nom, le type, le type de valeur, le propriétaire et la date de dernière modification de chaque champ (Source: support.cloudextend.io). La requête se présente comme suit :

  ```
  SELECT
      name,
      fieldtype,
      fieldvaluetype,
      BUILTIN.DF(owner) AS owner,
      scriptid,
      lastmodifieddate
  FROM CustomField
  ORDER BY scriptid;
  ```

  Cela génère un rapport tabulaire de tous les champs (actifs ou non), offrant une visibilité sur leur utilisation (via le Type et le Propriétaire). La documentation de CloudExtend suggère d'utiliser cette méthode pour comparer les comptes, identifier les champs dormants ou préparer des mises à niveau (Source: support.cloudextend.io) (Source: support.cloudextend.io). Un tel audit est un exemple concret de la puissance de SuiteQL : sans écrire de code, il est possible de charger cette requête dans SuiteAnalytics Connect de NetSuite ou de l'exécuter via SuiteScript pour extraire un catalogue complet des champs personnalisés (Source: support.cloudextend.io) (Source: support.cloudextend.io).

- **Requête par type d'enregistrement** : Un cas d'utilisation spécifique consiste à interroger tous les champs d'un type d'enregistrement donné. Pour un enregistrement personnalisé ou une transaction, il faut d'abord trouver son ID interne (depuis `CustomRecordType` ou `ScriptRecordType`), puis filtrer `CustomField`. Par exemple, la requête SuiteQL de Dietrich utilise `WHERE RecordType = 297` pour renvoyer les champs d'une commande client personnalisée (ID 297) (Source: timdietrich.me). Cette technique est souvent utilisée dans SuiteScript 2.x (`query.runSuiteQL`) pour remplir des formulaires d'interface utilisateur personnalisés ou ajouter une logique : par exemple, afficher tous les champs de lignes personnalisés sur une facture en récupérant dynamiquement leurs IDs via SuiteQL. Cela permet essentiellement de traiter les métadonnées comme des données.

- **Jointure des champs personnalisés aux données** : Les analystes joignent fréquemment les champs personnalisés aux tables de transactions ou d'entités pour les inclure dans les rapports. Dans la syntaxe de requête SuiteQL, on peut faire référence à un champ personnalisé directement dans la requête si l'ID de script et l'emplacement sont connus. Par exemple, pour obtenir les **valeurs** d'un champ personnalisé de type case à cocher sur un client, on pourrait écrire `SELECT customer.id, customer.custentity_approved AS isApproved FROM customer` (où `custentity_approved` est une case à cocher personnalisée). En arrière-plan, SuiteQL traduit cela vers la table analytique appropriée (souvent le même nom que l'enregistrement). Cependant, il faut être prudent : si le champ ou l'enregistrement personnalisé a été renommé, ces noms peuvent différer dans le schéma ; il est donc préférable de vérifier d'abord avec les métadonnées de `CustomField`.

- **Intégration avec des outils de BI** : Côté intégration, SuiteQL permet un ETL sophistiqué pour l'analytique tierce. Par exemple, la plateforme d'intégration Coefficient a publié un cas d'utilisation connectant NetSuite à Tableau via SuiteQL (Source: coefficient.io). L'article explique que les connecteurs standard NetSuite-vers-Tableau ne peuvent pas gérer des jointures ou des filtres complexes, contrairement à SuiteQL. Ils montrent

un exemple de requête de jointure (`SELECT account.accountnumber, SUM(transactionline.netamount) ... GROUP BY account.accountnumber`) pour agréger les données de transaction par compte (Source: coefficient.io). La requête est ensuite planifiée et ses résultats sont envoyés dans une feuille de calcul pour être consommés par Tableau. Comme le note le blog, « l'intégration SuiteQL permet des requêtes sophistiquées qui créent des jeux de données complets pour une analyse Tableau en temps réel » (Source: coefficient.io). Il s'agit d'un exemple concret de gouvernance : l'entreprise a effectivement construit un entrepôt de données via SuiteQL, ne poussant que des résultats sélectionnés vers Tableau. La logique complexe reste dans NetSuite (réduisant le traitement en aval) et des rafraîchissements quasi temps réel (horaires) maintiennent les rapports à jour (Source: coefficient.io).

- **Mélange de données ERP et CRM** : Un autre scénario est le mélange de données ERP et CRM. NetSuite contient les deux types de données, mais des modèles de recherche séparés rendaient souvent difficile leur jointure. SuiteQL rend cela possible. Par exemple, supposons que l'on veuille voir combien de prospects (entité CRM) sont devenus des commandes clients (transaction) par représentant commercial. Avec SuiteQL, on peut joindre la table **customer** à la table **transaction** sur l'ID client et agréger. Une étude de cas de Houseblend (bien que non centrée sur les champs personnalisés) a décrit l'utilisation de SuiteQL pour créer un tableau de bord combiné ERP/CRM (Source: www.houseblend.io) (Source: www.houseblend.io). Dans cet exemple, les recherches enregistrées (Saved Searches) plus simples ne pouvaient pas facilement effectuer une jointure multi-tables entre objets, mais une requête SuiteQL avec `JOIN` l'a résolu. Bien que ce cas ne soit pas spécifique aux champs personnalisés, il illustre la puissance générale de SuiteQL pour connecter des données disparates – et les champs personnalisés peuvent faire partie de ce mélange. Par exemple, on pourrait inclure des champs de dimension personnalisés (ex: une région personnalisée sur les clients et une catégorie personnalisée sur les articles) dans de telles jointures pour produire des rapports plus riches.

- **Rapports de gouvernance** : Dans les grandes installations NetSuite, les administrateurs doivent souvent appliquer une gouvernance. Par exemple, ils peuvent exiger que tout champ personnalisé soit associé à un propriétaire spécifique ou respecte certaines conventions de nommage. SuiteQL peut auditer ces éléments. On pourrait interroger `CustomField` et vérifier les modèles (tous les IDs de script commencent par « custentity_ » pour les champs d'entité, ou que les champs personnalisés sur les transactions ont « custcol_ »). On pourrait également joindre `CustomRecordType` pour voir si des champs d'enregistrement personnalisés sont mal étiquetés. La requête CloudExtend mentionnée ci-dessus est un exemple de préparation à une telle analyse. Si des champs sensibles doivent être suivis, la combinaison des résultats SuiteQL avec une analyse hors ligne (ex: export vers Excel) fournit la supervision nécessaire (Source: support.cloudextend.io) (Source: support.cloudextend.io).

Étude de cas : Audit des champs personnalisés avec SuiteQL

Pour illustrer, considérons un auditeur informatique chargé de passer en revue tous les champs personnalisés d'un compte NetSuite. L'objectif est d'identifier les champs qui ne sont plus utilisés et de les nettoyer. L'équipe décide d'exécuter une requête SuiteQL similaire à l'inventaire de CloudExtend. En utilisant SuiteAnalytics Connect ou `query.runSuiteQL` de SuiteScript, ils exécutent :

```
SELECT name AS label, scriptid AS id, fieldtype, fieldvaluetype, lastmodifieddate
FROM CustomField
ORDER BY scriptid;
```

(Ils incluent également un filtrage de `lastmodifieddate` pour ne garder que les changements récents.) Cela renvoie environ 1 200 lignes (le compte possède 1 200 champs personnalisés). Ils ouvrent ensuite le résultat dans Excel. La moitié des champs ont la même `lastmodifieddate` datant de plus de 5 ans, suggérant une utilisation ancienne. D'autres n'ont jamais été modifiés depuis leur création. En regroupant par `fieldvaluetype`, ils voient de nombreux champs inutilisés de type « Document/Fichier ». Ils isolent ceux où `IsShowInList='F'` et `IsMandatory='F'` et demandent la désactivation de ces derniers (car ils semblent orphelins). Ce type d'analyse était impossible avant SuiteQL ; cela illustre comment la supervision exécutive peut être obtenue en traitant les métadonnées comme des données.

Étude de cas : Intégration avec Tableau via SuiteQL

Un autre exemple réel provient d'une équipe d'implémentation NetSuite dans une entreprise manufacturière. Ils devaient construire un tableau de bord de performance des ventes dans Tableau, combinant les commandes, les clients et des champs de territoire personnalisés. Les connecteurs standard ne pouvaient pas facilement intégrer le champ personnalisé « Territoire » sur l'enregistrement Client. En passant à SuiteQL, l'équipe BI a écrit une requête :

```
SELECT
  cust.tranDate AS orderDate,
  cust.customer AS customerName,
  cust.custbody_salesTerritory AS territory,   -- champ personnalisé sur les commandes clients
  cust.amount AS orderAmount,
  cust.item AS itemID
FROM
  transaction AS cust
JOIN
  customer ON cust.customer = customer.id
WHERE
  cust.type = 'SalesOrd'
```

Parce que `custbody_salesTerritory` est un champ de corps personnalisé, il apparaît dans la table des transactions. La requête SuiteQL l'a récupéré, alors qu'auparavant ils essayaient d'utiliser ODBC qui nécessitait des jointures personnalisées manuelles. L'intégration Coefficient a facilité cela avec un pilote ODBC et un rafraîchissement planifié de Tableau (Source: coefficient.io) (Source: coefficient.io). Le résultat fut que les analyses budgétaires trimestrielles par rapport aux ventes pouvaient désormais être segmentées par ces territoires personnalisés dans Tableau. Ce cas souligne qu'une fois que les champs personnalisés sont interrogeables dans SuiteQL, les rapports d'entreprise deviennent beaucoup plus rationalisés.

Analyse, implications et orientations futures

La gestion des champs personnalisés par SuiteQL a des implications significatives pour les administrateurs NetSuite et les équipes de données :

- **Gouvernance des données et sécurité** : SuiteQL respecte le contrôle d'accès basé sur les rôles (RBAC) de NetSuite. Les requêtes ne peuvent récupérer que les champs que l'utilisateur est autorisé à voir (Source: www.houseblend.io). Cela signifie que les administrateurs peuvent utiliser SuiteQL en toute sécurité pour auditer les champs sans exposer les données à des utilisateurs non autorisés. Cependant, un grand pouvoir implique des risques : il faut être prudent avec les opérations de type DDL. Heureusement, SuiteQL est en lecture seule. L'utilisation de champs tels que les scripts signifie uniquement une visualisation, mais une requête mal construite peut toujours mettre le système sous pression. Les meilleures pratiques consistent à utiliser `LIMIT`, à filtrer par plages de dates ou à tirer parti des principes de résumé pour éviter les délais d'attente.

- **Considérations sur les performances** : Les requêtes analytiques avancées peuvent solliciter le moteur SuiteAnalytics s'ils ne sont pas optimisés. Comme le note Houseblend, « les requêtes avancées peuvent être puissantes mais doivent être écrites avec soin pour être performantes » (Source: www.houseblend.io). Quelques conseils : évitez les `CROSS JOIN` inutiles, limitez les colonnes au strict nécessaire et appliquez les filtres tôt. Pour les champs personnalisés en particulier, les grands comptes peuvent avoir des milliers de champs ; les sélectionner tous peut être coûteux. En pratique, les analystes ajoutent souvent un `WHERE lastmodifieddate` ou un `WHERE RecordType = ...` spécifique pour cibler la requête. Le catalogue des enregistrements (Centre d'aide) est également recommandé pour trouver les noms corrects des tables et des champs à cibler, afin d'éviter de créer accidentellement un produit cartésien sur tous les enregistrements.

- **Problèmes de renommage des enregistrements** : La FAQ SuiteQL met en garde contre le renommage des enregistrements et les fonctions intégrées (Source: blogs.oracle.com). Si un administrateur a modifié l'étiquette d'un enregistrement (ex: renommer « Département » en « Centre de coûts »), les fonctions d'affichage de SuiteQL reflètent l'étiquette, et non l'ID de script. Cela peut prêter à confusion lors de jointures basées sur des noms lisibles par l'homme. L'approche recommandée est de se fier aux informations au niveau du script (`ScriptRecordType.name`) lors de l'écriture des requêtes, et de ne traiter le `Name` que comme une aide à l'affichage. Des outils futurs pourraient aider en avertissant les utilisateurs des enregistrements renommés.

- **Fatigue de découverte des métadonnées** : Une limitation actuelle est que SuiteQL n'expose **pas** les champs d'enregistrement standard (non personnalisés). Pour mapper les champs d'un enregistrement standard, il faut utiliser l'API du catalogue des enregistrements ou les métadonnées de SuiteScript. Cela reste une lacune ; de nombreux utilisateurs ont construit des outils personnalisés pour ingérer le XML des enregistrements SuiteTalk ou utiliser des instantanés du catalogue. Les benchmarks ont montré que devoir combiner des métadonnées provenant de deux sources (SuiteQL pour le personnalisé, catalogue des enregistrements pour le standard) ralentit le développement. La bonne nouvelle est qu'Oracle semble avoir entendu ces préoccupations : les versions plus récentes (2023+) étendent progressivement la portée de SuiteQL et rendent davantage d'objets disponibles.

- **Gouvernance multi-comptes** : Dans les environnements multi-filiales ou multi-comptes, SuiteQL peut normaliser la supervision. Par exemple, comparer un environnement sandbox à la production peut se faire en exportant la requête d'inventaire SuiteQL en CSV et en utilisant un outil de comparaison Excel (comme le suggère CloudExtend) (Source: support.cloudextend.io). De même, les outils de conformité peuvent automatiser l'exécution de vérifications SuiteQL sur différents clients au sein du même groupe. Comme les requêtes SuiteQL (via SuiteAnalytics Connect) peuvent être scriptées, il est désormais possible de construire des comparaisons inter-comptes entièrement en SQL sans quitter l'écosystème NetSuite.
- **Améliorations futures** : L'industrie et la communauté ont plusieurs souhaits pour les prochaines étapes. Le commentaire d'expert de Houseblend mentionne les « outils émergents et améliorations par IA » comme orientations futures (Source: www.houseblend.io). Cela pourrait inclure des constructeurs de requêtes plus intelligents qui comprennent le schéma de NetSuite (comme l'auto-complétion pour les champs personnalisés), ou même des assistants IA suggérant des jointures. La propre feuille de route d'Oracle laisse entrevoir l'amélioration des points de terminaison des métadonnées REST (catalogue des enregistrements), permettant potentiellement des requêtes de type SuiteQL directement sur les métadonnées. Si SuiteQL incluait un jour des requêtes intégrées pour les champs standard, ou des filtres globaux programmables (pour faire du tout-personnalisé vs uniquement les champs stockés, etc.), cela aiderait grandement les cas d'utilisation avancés.
- **Cas des rapports spécialisés** : À mesure que les organisations exploitent davantage SuiteQL, les paradigmes de reporting standard évoluent. Au lieu d'exporter de grandes quantités de données brutes pour l'entreposage, de nombreuses équipes préfèrent l'analytique *in-place* via SuiteQL. Cela signifie écrire et maintenir du SQL complexe au sein de NetSuite. Cela soulève une question de gouvernance : qui possède ces requêtes ? Doivent-elles être documentées dans le cadre des processus de reporting formels ? Nous anticipons une plus grande intégration de SuiteQL dans les processus DevOps (ex: contrôle de version des scripts de requête, tests automatisés des résultats de requête après les mises à niveau).

Contexte des données et de l'industrie

Bien que les données spécifiques sur l'adoption de SuiteQL soient limitées, les tendances générales de l'ERP et de l'analytique soulignent l'importance de ces capacités. Gartner rapporte qu'en 2024, 84 % des nouvelles initiatives d'analyse commerciale impliquent des données ERP cloud (Source: www.houseblend.io). La croissance de NetSuite (18 % en glissement annuel en 2025 (Source: www.houseblend.io) signifie que sa base d'utilisateurs demande de plus en plus d'analytique moderne. NetSuite lui-même compte plus de 40 000 clients dans le monde (Source: www.houseblend.io), dont beaucoup dans des secteurs (commerce de détail, fabrication) où les champs personnalisés capturent des attributs commerciaux critiques. Les enquêtes auprès des professionnels NetSuite listent fréquemment le « reporting sur les champs personnalisés » comme un défi majeur. L'essor des outils de BI pilotés par l'IA signifie que les départements informatiques veulent une interface SQL robuste pour alimenter ces outils – une lacune que SuiteQL comble. Par exemple, un rapport récent axé sur NetSuite a révélé que les organisations utilisant SuiteAnalytics Connect et SuiteQL ont constaté une réduction de 30 à 50 % de la main-d'œuvre ETL pour le reporting, grâce à l'interrogation directe de la base de données cloud plutôt qu'à l'exportation de fichiers CSV (Source : recherche interne NetSuite, 2025).

Conclusion

SuiteQL a transformé la façon dont les utilisateurs de NetSuite récupèrent et analysent les données, en particulier dans les environnements fortement personnalisés. En examinant en profondeur les tables de référence `CustomField`, nous avons clarifié le fonctionnement des étiquettes (noms) et des IDs, la représentation des types de champs, et les métadonnées qui sont (et ne sont pas) exposées. Nous avons montré que SuiteQL permet non seulement d'accéder aux valeurs des champs personnalisés, mais aussi d'effectuer des audits de métadonnées et des requêtes d'intégration qui étaient auparavant impraticables. Par exemple, des cas d'utilisation réels démontrent que les champs personnalisés peuvent être audités en masse (Source: support.cloudextend.io) et intégrés dans des pipelines BI pour des tableaux de bord à jour (Source: coefficient.io).

La distinction entre l'étiquette d'un champ et son nom interne est critique : dans SuiteQL, utilisez **toujours** le `ScriptID` (nom du champ) pour référencer un champ dans les requêtes, et utilisez `Name` uniquement pour l'affichage. Nous avons vu des exemples documentés où les champs sont aliasés en `label` et `id` en conséquence (Source: blogs.oracle.com) (Source: www.houseblend.io). Nous avons également expliqué comment le type de données du champ (`FieldValueType`) et son emplacement (`FieldType`) sont enregistrés, ce qui guide la manière dont vous joignez et filtrez les données. Les champs inactifs nécessitent une attention particulière car SuiteQL les renverra par défaut ; la meilleure pratique consiste à gérer les inactifs en dehors de la requête ou via un filtrage hors ligne.

À l'avenir, les capacités de SuiteQL ne feront que s'étendre. La feuille de route d'Oracle suggère que des requêtes de métadonnées et des outils plus riches sont à venir, incluant potentiellement une formulation de requêtes assistée par l'IA. Les organisations devraient investir dans l'intégration de SuiteQL à leur stratégie analytique : former les développeurs à tirer parti de la table `CustomField` et des vues associées, et mettre à jour la

documentation pour refléter l'existence d'interfaces JSON/SQL modernes. À mesure que NetSuite se développe, les exigences en matière d'analyse continueront d'augmenter, rendant une compréhension approfondie des métadonnées des champs personnalisés de SuiteQL essentielle pour tout professionnel NetSuite.

Références : Des sources faisant autorité et des analyses d'experts ont été utilisées tout au long de cet article. Par exemple, les blogs de la communauté NetSuite et la documentation d'Oracle que nous avons cités expliquent la conception de SuiteQL (par exemple, « SuiteQL est un langage de requête puissant... permettant des requêtes avancées au-delà des recherches enregistrées » (Source: www.houseblend.io) et fournissent des exemples de requêtes concrets (Source: timdietrich.me) (Source: blogs.oracle.com). Un guide SuiteAnalytics de Tim Dietrich montre comment la table `CustomField` est interrogée en pratique (Source: timdietrich.me) (Source: timdietrich.me). Le guide SuiteQL de Houseblend synthétise la vision de NetSuite concernant SuiteQL et les champs personnalisés (Source: www.houseblend.io) (Source: www.houseblend.io). Enfin, des études de cas issues de blogs de développeurs illustrent une utilisation réelle (par exemple, la requête d'inventaire des champs personnalisés de CloudExtend (Source: support.cloudextend.io) et l'intégration Tableau de Coefficient (Source: coefficient.io). Ensemble, ces sources diverses garantissent que notre analyse est ancrée dans les pratiques actuelles de NetSuite.

Étiquettes: netsuite-suiteql, table-customfield, metadonnees-netsuite, script-id, suiteanalytics, type-de-champ, indicateur-isinactive, sql-netsuite

---