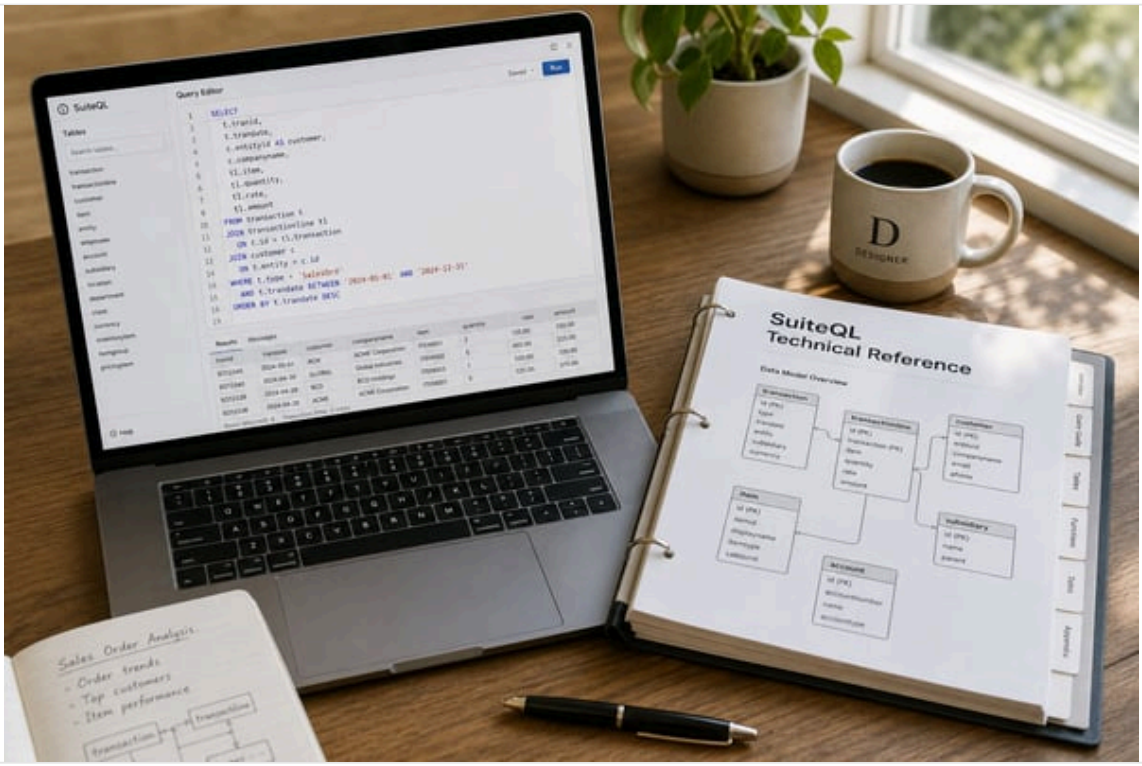
NetSuite SuiteQL Guide: Custom Fields, Joins & Connect

Published May 31, 2026 33 min read



Executive Summary

SuiteAnalytics Connect's **SuiteQL** is NetSuite's ANSI SQL-92–based query language for advanced reporting and data analysis. Introduced around 2019–2020, SuiteQL unlocks NetSuite's integrated ERP/CRM data model by exposing it as virtual tables, enabling complex queries (multi-table joins, subqueries, unions, aggregations, etc.) that far surpass the capabilities of traditional Saved Searches or SuiteAnalytics Workbook (Source: www.houseblend.io) (Source: www.houseblend.io). It supports all standard SQL join types (inner, left/right/full outer, Cartesian) and Oracle-specific syntax patterns, while enforcing NetSuite's role-based security model. SuiteQL is accessible via SuiteAnalytics Connect (ODBC/JDBC), SuiteScript (N/query module), and SuiteTalk REST APIs (Source: docs.oracle.com) (Source: www.houseblend.io).

This comprehensive reference covers: **Querying Custom Records and Fields** (the metadata tables `CustomRecordType` and `CustomField`); **Join Types and Patterns** (inner/outer joins, cross joins, and best practices for linking tables); and **SuiteAnalytics Connect Best Practices** (naming conventions, qualified queries, performance tuning, and integration patterns). We draw on official Oracle documentation, NetSuite developer blogs, and industry analysis to provide in-depth guidance. Key findings include:

- Using the `CustomRecordType` and `CustomField` tables to discover custom record definitions and their fields (Source: timdietrich.me) (Source: www.houseblend.io). For example,

  ```
  SELECT Name, ScriptID, InternalID, Description
  FROM CustomRecordType ORDER BY Name;
  ```

  retrieves all custom record types (name, script ID, internal numeric ID etc.) (Source: timdietrich.me). One then filters `CustomField` by `RecordType` (the record's internal ID) to get its fields (Source: timdietrich.me) (Source: www.houseblend.io). The `Name` column holds the user-visible label of each custom field, while `ScriptID` is the stable internal ID (e.g. `custentity_myfield`) (Source: www.houseblend.io) (Source:

www.houseblend.io). The `FieldValueType` and `FieldValueTypeRecord` columns describe each field's data type (Checkbox, Date, List/Record, etc.); when a field sources from another list/record, `FieldValueTypeRecord` holds the internal ID of that target (Source: www.houseblend.io) (Source: www.houseblend.io).

- **Joins and Relationships.** SuiteQL supports inner joins, outer joins (left/right/full), and Cartesian (cross) joins (Source: docs.oracle.com). By default SuiteAnalytics Workbook uses left outer joins for record links, but SuiteQL lets the user specify any join type (Source: www.houseblend.io). Standard patterns include joining on numeric internal ID fields – for example, `customer.id = transaction.entity` ties customers to their sales. Official docs illustrate joins (see Table below):

JOIN TYPE	SYNTAX (SUITEQL)	ROWS RETURNED	NOTES
Inner Join	`... FROM A JOIN B ON A.key = B.key`	Only matching rows (common keys)	Use to combine only matching records. (Default join in most SQL engines.) (Source: docs.oracle.com)
Left Outer Join	`... FROM A LEFT JOIN B ON A.key = B.key` or Oracle-style `FROM A, B WHERE A.key = B.key(+)`	All rows from A, matching from B (NULL for non-matches)	Ensures all A rows appear; SuiteAnalytics Workbook uses this by default (Source: docs.oracle.com) (Source: docs.oracle.com).
Right Outer Join	`... FROM A RIGHT JOIN B ON A.key = B.key`	All rows from B, matching from A (NULL for non-matches)	Similar to left join but swapped; Oracle-style right joins have no shorthand.
Full Outer Join	`... FROM A FULL OUTER JOIN B ON A.key = B.key`	All rows from both A and B, matching on key (duplicates as needed)	Includes all records from both tables; unsupported implicitly in Oracle-style (Source: docs.oracle.com).
Cross Join	`FROM A, B` (implicit) or `FULL OUTER JOIN ON 1=1`	Cartesian product (all combinations)	SuiteQL does **not** support `CROSS JOIN` keyword; use comma or full-outer hack for B.L. (Source: docs.oracle.com).

For example, joining customers to open invoices:

```
SELECT cust.entityid AS customer_id, cust.companyname,
       trx.tranid, trx.total
FROM customer AS cust
JOIN transaction AS trx
  ON cust.id = trx.entity
WHERE trx.type = 'Inv' AND trx.status = 'Open';
```

This returns each customer's open invoices (one row per invoice) (Source: www.houseblend.io) (Source: www.houseblend.io). The ability to join across modules (ERP↔CRM) is a key advantage: since NetSuite's schema is unified, SuiteQL can seamlessly combine "ERP data" and " CRM data" (Source: www.houseblend.io). For instance, the `transaction` table has an *entity* column linking to `customer.id`, and a `supportcase` record has `company` (customer) and `assigned` (employee) fields (Source: www.houseblend.io). One can join `supportcase.company = customer.id` and `supportcase.assigned = employee.id` to merge case and contact info in a single query (Source: www.houseblend.io).

- **SuiteAnalytics Connect Best Practices.** SuiteQL queries on Connect are executed against the NetSuite2.com analytics data source, with role-based access control enforced (Source: docs.oracle.com) (Source: docs.oracle.com).Important patterns and constraints include:

- **Naming Conventions:** In the old NetSuite.com ODBC schema, custom names were transformed (labels→IDs) by making them uppercase, replacing spaces with underscores, and removing hyphens (Source: docs.oracle.com). For example, a custom record named "`This-is a-record`" becomes `THISIS_ARECORD` (Source: docs.oracle.com). In the new NetSuite2.com schema, the table and column names use the *script ID* specified in the UI, which is stable. For instance, a custom field with UI ID "`custbody1`" remains `custbody1` in Connect (Source: docs.oracle.com) (Source: docs.oracle.com). A typical custom record ID in NetSuite2.com looks like `CUSTOMRECORD1` (Source: docs.oracle.com). (See Table below.)

- **Qualified Queries:** When making fully-qualified ODBC queries, you must use the exact *table_qualifier* (company name) and *table_owner* (NetSuite role) from your connection. For example, if `table_qualifier = "Wolfe Company"` and `table_owner = "Administrator"`, you issue:

  ```
  SELECT * FROM "Wolfe Company"."Administrator".account;
  ```

  Note the quotes and spaces – any inexact spelling or casing will cause the query to fail (Source: docs.oracle.com) (Source: docs.oracle.com). (Most users can omit qualification when running in their own Connect session.)

- **Case Sensitivity / Uppercase Attribute:** The Connect driver is case-insensitive by default (you can query using any mix of upper/lowercase) (Source: docs.oracle.com). However, field and table names returned to client applications may change case when schema is updated. To avoid issues in case-sensitive tools, NetSuite provides an `Uppercase` connection attribute that forces all table/field names to uppercase (Source: docs.oracle.com).

- **Performance and Syntax:** Internally SuiteAnalytics is powered by Oracle SQL. Oracle recommends writing joins in Oracle syntax (using commas and the `(+)` operator for outer joins) to improve performance and avoid timeouts (Source: docs.oracle.com). For example, instead of ANSI `LEFT JOIN`, one could write:

  ```
  SELECT t.id, t.amount, a.name
  FROM transaction t, account a
  WHERE a.id = t.account AND t.period = ?
        AND p.id(+) = t.period;
  ```

  Such Oracle-style queries often run faster via Connect (Source: docs.oracle.com). Avoid heavy operations and complex OR conditions in large queries, as they may prevent index usage (Source: www.houseblend.io) (Source: docs.oracle.com). It is best to **test and iterate**: build queries on small data sets first, or use Workbook's export-to-SuiteQL feature to prototype joins visually (Source: www.houseblend.io).

- **Incremental Data Loading:** For ETL or BI refreshes, use the provided timestamp and status columns. Many tables in Connect include `last_modified_date` or `date_last_modified` to support change tracking. Additionally, NetSuite publishes a `deleted_records` table listing recently removed records. As a rule, use these columns (or `deleted_records`) to perform incremental loads (Source: docs.oracle.com). Some mapping or static tables do **not** have these columns; such tables must be fully reloaded each time (Source: docs.oracle.com).

- **Security and Compliance:** SuiteAnalytics Connect is strictly read-only (Source: docs.oracle.com). Businesses must ensure third-party tools ingesting Connect data handle sensitive fields appropriately. For example, NetSuite warns that exporting protected health information (ePHI) via Connect requires HIPAA-compliant processes (Source: docs.oracle.com). Likewise, Connect enforces role permissions: the data your queries return is limited to what your chosen Connect role can see in the UI (Source: docs.oracle.com).

NAMING MODEL	CUSTOM FIELD IDS	CUSTOM RECORD IDS	EXAMPLE
Netsuite.com (old Connect)	Uppercase label text; spaces→ `_`; hyphens removed (max 29 chars) (Source: docs.oracle.com)	Uppercase record name; spaces→ `_`; hyphens removed (Source: docs.oracle.com)	"This-is a-record" → `THISIS_ARECORD` (Source: docs.oracle.com)
Netsuite2.com (current)	Uses UI *ID* field (e.g. `custbody1` for a body field) (Source: docs.oracle.com) (Source: docs.oracle.com)	Uses UI script ID (e.g. `CUSTOMRECORD1`) (Source: docs.oracle.com)	"Transaction Body" field has ID `custbody1` (Source: docs.oracle.com); custom record example: `CUSTOMRECORD1`.

- **Data Analysis and Use Cases.** Industry trends and customer data highlight SuiteQL's impact. NetSuite has over 40,000 customers worldwide (Source: www.houseblend.io) and saw 18% revenue growth in 2025 (Source: www.houseblend.io). As analysts increasingly demand cloud ERP analytics (84% of new BI projects involve cloud ERP data (Source: www.houseblend.io), SuiteQL fills a critical gap. Users report that "reporting on custom fields" is a top challenge (Source: www.houseblend.io), and SuiteQL (with its ability to enumerate and join custom fields) directly addresses that need. Real-world implementations confirm tangible benefits. For example, NetSuite internal research (2025) found organizations using SuiteAnalytics Connect reduced ETL labor by 30–50% compared to legacy CSV exports, thanks to querying the live cloud database (Source: www.houseblend.io). Case studies illustrate typical patterns: teams combine CRM and ERP data (e.g. customers, sales orders, invoices) into unified dashboards (Source: www.houseblend.io) (Source: www.houseblend.io). One scenario is **Marketing campaign targeting**: a marketing analyst can use SuiteQL to pull customer segments by geography and purchase history, combining results via subqueries or `UNION` (see, e.g., Tim Dietrich's solutions (Source: www.houseblend.io) (Source: www.houseblend.io). Another is **Financial reporting**: a Coefficient.io example shows using SuiteQL to aggregate invoice amounts by GL account and feed that data into Tableau spreadsheets on an hourly schedule, enabling near-real-time P&L dashboards (Source: www.houseblend.io). Because SuiteQL directly operates on NetSuite's integrated data model, organizations can build rich cross-department reports – for instance, joining open invoices with sales rep details to produce an Accounts Receivable by Sales Rep report in one query (Source: www.houseblend.io).

- **Future Directions.** As SuiteQL matures, NetSuite is adding features that improve metadata access and developer experience. Currently, SuiteQL's schema access is largely limited to user-defined (custom) metadata. Standard record and field metadata must be obtained through the **Records Catalog** (Setup > Records Catalog UI) or via SuiteTalk's REST metadata endpoints (Source: www.houseblend.io) (Source: www.houseblend.io). Notably, after the 2021.2 release, the old Connect Browser (schema) is no longer updated, making the Records Catalog the authoritative source of object relationships (Source: www.houseblend.io). NetSuite continues to enrich Connect: for example, new tables like `oa_tables` and `oa_columns` help map old vs. new IDs and reveal which tables/columns support incremental loads or deletions (Source: docs.oracle.com) (Source: docs.oracle.com). The rise of AI-driven analytics foreshadows improvements such as natural-language SuiteQL builders and expanded metadata APIs.

In summary, SuiteQL (accessed via SuiteAnalytics Connect or SuiteScript) empowers technical and business users to query NetSuite data in unprecedented ways, especially by exposing custom fields/records and enabling complex joins. This report documents best practices for writing SuiteQL queries, illustrates how to retrieve metadata and link records, and highlights patterns for integrating SuiteQL into data pipelines. All guidance is supported by official documentation and proven community examples, ensuring both accuracy and practical relevance (Source: www.houseblend.io) (Source: www.houseblend.io).

Introduction and Background

NetSuite is a leading cloud-based ERP/CRM platform that consolidates financials, inventory, orders, customers, and more into a single system (Source: www.houseblend.io). A hallmark of NetSuite is its extensibility: administrators can add **custom fields** to virtually any record type, and even define wholly **custom record** types bearing business-specific data. Over time, these customizations enable organizations to capture detailed operational attributes (e.g. product serial numbers, customer segment tags, regulatory codes) but also complicate reporting and integration. Traditionally, NetSuite users relied on *Saved Searches* (point-and-click filters with limited joins) or *SuiteAnalytics Workbook* (visual reports) for analytics (Source: www.houseblend.io). These tools, while valuable, impose constraints: at most one level of joins in Saved Searches, limited aggregations, and rigid report layouts (Source: www.houseblend.io). External data extraction was possible via the legacy ODBC driver ("NetSuite.com data source") or file-based exports, but complex queries were impractical.

To address this, NetSuite introduced **SuiteQL** around 2019/2020 (Source: www.houseblend.io). SuiteQL is a full SQL-92–compliant query interface layered on NetSuite's **Analytics Data Source** (now called NetSuite2.com). In Oracle's words, SuiteQL is "a powerful query language…built on SQL-92…designed to provide users with efficient and flexible access to NetSuite's data model, enabling advanced queries beyond the capabilities of Saved Searches and reports" (Source: www.houseblend.io). In practice, SuiteQL exposes the same data that SuiteAnalytics Workbook can see, but with the full flexibility of SQL – multi-table joins, subqueries, UNIONs, window functions, and more (Source: www.houseblend.io) (Source: www.houseblend.io). This enables analysts to answer questions that were previously difficult or impossible, such as blending ERP transactions with CRM contacts or auditing thousands of custom fields in bulk.

SuiteQL can be used in several ways:

- **SuiteAnalytics Connect (ODBC/JDBC):** When Connect is enabled, NetSuite provides ODBC/JDBC drivers that connect to the NetSuite2.com data warehouse. Developers and BI tools (Excel, Tableau, Power BI, etc.) can issue SuiteQL queries through this interface (Source: docs.oracle.com) (Source: www.houseblend.io).
- **SuiteScript (N/query module):** In SuiteScript 2.0+, NetSuite provides an `N/query` module that accepts SuiteQL strings and returns data objects. This lets server scripts run SuiteQL in the context of user roles and governance points (Source: www.houseblend.io).
- **SuiteTalk (REST Web Services):** SuiteTalk's REST APIs include endpoints (`/suiteql`) that execute SuiteQL queries. Integrations can call these to fetch results in JSON or CSV (Source: docs.oracle.com) (Source: www.houseblend.io).

All these methods use the same underlying analytics schema, subject to NetSuite's role-based restrictions and governor limits. Notably, SuiteAnalytics Connect is **read-only** (Source: docs.oracle.com); one cannot use SuiteQL to update data. Its main purpose is buffering a consistent, performant reporting layer. Starting with NetSuite 2026.1, the legacy "NetSuite.com" schema is deprecated, and *only* the NetSuite2.com data source is supported for Connect (Source: docs.oracle.com).

Because SuiteQL is relatively new, best practices are still evolving. The material that follows brings together official guidance with community wisdom. We will explain how to find and interpret NetSuite schema elements, how to write SuiteQL queries on custom fields and records, how to join tables effectively, and how to integrate SuiteQL into data pipelines. Along the way, we will incorporate reference data (e.g. internal usage stats, metadata examples) and case scenarios to illustrate principles.

SuiteAnalytics Connect and Data Schema

SuiteAnalytics Connect Overview

SuiteAnalytics Connect provides database-style access to NetSuite data via ODBC/JDBC/ADO.NET drivers. It exposes the **NetSuite2.com** data warehouse schema for reporting. Key points include:

- **NetSuite2.com Data Source:** Connect queries the NetSuite2.com company-specific data source. This warehouse is **role-secured**: the Connect user sees exactly the same data they would in SuiteAnalytics Workbook under that role (Source: docs.oracle.com). As a result, everyone only sees permitted data, regardless of the driver used. The NetSuite2.com source is constantly synchronized (a few hours lag) and is consistent with Workbook data.

- **Authentication:** Users can connect via email/password or token-based auth (TBA/OAuth). Always use the latest driver and enable certificate-based auth as required (Source: docs.oracle.com).

- **Schemas/Tables:** Upon connecting, you will see a schema named for your account and role. Within it are tables for all exposed records. Standard records (Customers, Transactions, Items, Employees, etc.) appear as singular table names (e.g. `customer`, `transaction`, `item`). Custom records appear under names based on their script IDs (e.g. a custom record with script ID `customrecord_projects` becomes table `customrecord_projects`). To discover available tables and fields, use tools like the **Records Catalog** (UI) (Source: www.houseblend.io) or query the Connect system tables (`oa_tables`, `oa_columns`) directly (Source: docs.oracle.com).

  Importantly, the Connect Browser (legacy schema browser) is **no longer updated** as of 2021.2 (Source: www.houseblend.io). NetSuite now relies on the **Records Catalog** (Setup > Records > Records Catalog) as the authoritative source of schema details. The Records Catalog shows, for each record type, all its fields (with script IDs), and any built-in join fields (e.g. on a Customer record it shows that *SalesRep* is an employee reference, leading to a join on the Employee table) (Source: houseblend.io) (Source: www.houseblend.io). Analysts should use the Records Catalog or `oa_tables` / `oa_columns` to confirm names and relationships before writing queries.

Naming Conventions (Custom Fields/Records)

Handling custom objects requires understanding NetSuite's naming. The **Record Types and Fields** guide covers this in detail (Source: docs.oracle.com) (Source: docs.oracle.com). In summary:

- **Old (NetSuite.com) Model:** IDs were derived from UI labels by uppercasing and removing special characters. Spaces turned into underscores, hyphens removed. For example, a custom record named *"This-is a-record"* got the ID `THISIS_ARECORD` (Source: docs.oracle.com), and a custom field labeled *"Transaction Body"* might have become something like `TRANSACTION_BODY_CUSTBODY1`. However, this model is now obsolete.
- **New (NetSuite2.com) Model:** Table and column names use the stable *script ID* defined in NetSuite at customization time. Renaming the label in the UI does *not* change the script ID. For example, a transaction's body field whose script ID is `custbody1` remains `custbody1` in SuiteQL queries (Source: docs.oracle.com) (Source: docs.oracle.com). Similarly, custom record types have IDs like `CUSTOMRECORD1`, `CUSTOMRECORD2`, etc., or the literal script ID if defined (e.g. `customrecord_projects`) (Source: docs.oracle.com). When writing queries, always use these script IDs. If uncertain, verify by querying the Connect metadata tables or checking the Records Catalog.

  Another useful resource is the system table `oa_columns`: it maps Connect names to UI names. For example, one can query

  ```
  SELECT column_name, table_name, remarks
  FROM oa_columns
  WHERE column_name LIKE 'CUST%' AND table_name = 'TRANSACTION';
  ```

  This would list all custom fields (IDs) on the Transaction table and their corresponding UI labels (Source: docs.oracle.com).

Finding Record and Field IDs

To write a SuiteQL query, you must know the exact table and column names to use. The **Records Catalog** and schema tools help with that. Official guidance provides several approaches:

- **Records Catalog (UI):** As noted above, via Setup → Records Catalog you can browse each record, see all fields (including standard and custom, with script IDs), and see joinable references. This is the recommended starting point (Source: www.houseblend.io). For example, in the catalog's Customer record, you might see an *Internal ID* field (table key), plus a *Subsidiary* field labeled as referencing the Subsidiary table, etc (Source: www.houseblend.io).
- **SuiteQL Metadata Tables:** SuiteQL actually exposes metadata for custom elements. Two tables are invaluable:
  - `CustomRecordType` – one row per custom record type, with columns like `Name` (label), `ScriptID`, `InternalID`, etc. Thus `SELECT Name, ScriptID, InternalID FROM CustomRecordType;` lists all custom record types (Source: timdietrich.me) (Source: www.houseblend.io). (The `InternalID` is NetSuite's numeric ID, often used as a foreign key.)
  - `CustomField` – one row per custom field definition (from any record or list). Columns include `Name` (field label), `ScriptID`, `Description`, `RecordType` (the internal ID of the parent record), `FieldType` ("BODY" vs "COLUMN"), `FieldValueType`, and `FieldValueTypeRecord` (if a list/record type) (Source: www.houseblend.io) (Source: www.houseblend.io). Example: running `SELECT Name, ScriptID, FieldType, FieldValueType FROM CustomField WHERE RecordType = 297;` could list fields on the custom record with InternalID 297, as shown by Dietrich (Source: timdietrich.me) (Source: www.houseblend.io).

  Because SuiteQL does not (yet) expose standard record metadata directly, the pattern is: first find a record's internal ID (via `CustomRecordType` for custom records), then use that ID to filter `CustomField`. You can also join these two tables: `CustomField.RecordType = CustomRecordType.InternalID` to pair fields with their record names (Source: www.houseblend.io) (Source: www.houseblend.io). For example, to find all fields for the *Projects* custom record, find its `InternalID` in `CustomRecordType`, then query `CustomField WHERE RecordType = <that ID>` (Source: timdietrich.me).

As a practical workflow: you might run a first query like:

```
SELECT Name, ScriptID, InternalID
FROM CustomRecordType
WHERE ScriptID = 'customrecord_projects';
```

Suppose this returns InternalID = 221. Then run:

```
SELECT Name, ScriptID, FieldValueType
FROM CustomField
WHERE RecordType = 221;
```

to list every field on that custom record. These queries can be wrapped in scripts or tools (see Houseblend's SuiteQL Query Tool) for discovery and documentation purposes (Source: timdietrich.me) (Source: www.houseblend.io).

Querying Custom Records and Fields

Once you have identified the relevant tables and fields, writing SuiteQL queries follows standard SQL patterns. However, custom fields have some quirks worth noting:

- **Field Labels vs Script IDs:** In SuiteQL result sets, the *column names* correspond to the script IDs (e.g. `custentity_age`, `custbody_total`, etc.), not the user-friendly labels. The `CustomField.Name` column holds the label (e.g. "Customer Age") (Source: www.houseblend.io), but when you select data you do so by scriptID:

  ```
  SELECT custbody_total AS total_tax, custentity_country AS country
  FROM transaction
  WHERE ...
  ```

  Always double-check spelling/case of these IDs (though Connect is case-insensitive, using all lower-case is common).

- **Sublist (Line) Fields:** Custom fields can be on the body of a record or on sublists (line items). The `FieldType` column in `CustomField` indicates 'BODY' or 'COLUMN' (sublist) for each custom field (Source: www.houseblend.io). You can query sublist fields by selecting from the record table or a join; they appear as columns (e.g. on sales orders, custom item column fields show up directly in `transaction` table rows for each line). To filter or aggregate on such fields, include them as needed. (Houseblend notes that some solutions filter `CustomField.fieldType` in metadata queries to segregate these types (Source: www.houseblend.io).)

- **List/Record Fields:** If a custom field is of type "List/Record" or multiple-select, the `FieldValueTypeRecord` column holds the internal ID of the referenced list. For example, a custom "Department" field on customer might have `FieldValueType = 'List/Record'` and `FieldValueTypeRecord = 16` (if departments have ID 16). You can join to identify what that is:

  ```
  SELECT cf.Name AS field_label, sr.`name` AS list_name
  FROM CustomField cf
  JOIN ScriptRecordType sr ON cf.FieldValueTypeRecord = sr.internalid
  WHERE cf.FieldValueType = 'List/Record'
    AND cf.RecordType = 123;
  ```

  This tells you the friendly names of referenced lists or records (Source: www.houseblend.io). In practice, if you know the script ID of the field (e.g. `custbody_dept`), you can also query the field in context and simply retrieve the joined value:

```
SELECT c.entityid, c.companyname, d.name AS Department
FROM customer c
LEFT JOIN department d ON c.custentity_dept = d.id;
```

Here `custentity_dept` is a custom department field on customer, storing department ID, which we join to the built-in `department` table.

- **Filtering by Metadata Flags:** The `CustomField` table includes boolean flags like `IsMandatory`, `IsInactive`, etc. These can be used to audit your metadata. For example, one might run:

```
SELECT ScriptID, Name FROM CustomField
WHERE IsMandatory = 'T' AND IsInactive = 'F';
```

to list all active custom fields that are marked mandatory. Similarly, `IsShowInList` indicates if a field is shown in default list views, which may affect visibility in reports (Source: www.houseblend.io).

Example – Auditing Custom Fields: HouseBlend analyzes a published query (CloudExtend example) that inventories custom fields for audit purposes (Source: www.houseblend.io) (Source: www.houseblend.io). A similar approach is:

```
SELECT cf.ScriptID, cf.Name, cf.FieldValueType, cf.IsMandatory, cr.Name AS RecordName
FROM CustomField cf
LEFT JOIN CustomRecordType cr ON cf.RecordType = cr.InternalID
ORDER BY cr.Name;
```

This joins each field to its custom record (if any). It might show, for instance, that `custentity_age (Customer Age)` is a mandatory field on the Customer record, while `custbody_notes` on Invoices is optional. Such metadata audits help governance by letting admins verify that field configurations match policy.

Joins and Relational Patterns

SuiteQL's power lies in joining data across tables. We have already seen basic joins on primary/foreign keys. Here are some common patterns:

- **One-to-Many Joins (e.g. Order Lines):** A standard pattern is a one-to-many join, such as linking transactions to their customers, or item lines to the parent transaction. For example:

```
SELECT t.tranid, t.datecreated, t.entity AS customer
FROM transaction t
JOIN customer c ON t.entity = c.id;
```

returns each transaction with its customer. If you also want sublist details, you might join the `transaction` table to itself on `transaction.id = transaction.parent`, or use the include for sublist (some sublist tables are exposed separately in Connect, depending on record type). The Records Catalog can guide these joins (Source: www.houseblend.io).

- **Many-to-One Joins (Lookup Fields):** Sometimes a transaction or record has lookup fields. For example, an Order record might have a field `salesrep` (employee) and `currency` (subsidiary). You join `transaction.salesrep = employee.id` and `transaction.custbody_subsidiary = subsidiary.id`. NetSuite's documentation and the Records Catalog specify these foreign key fields. The HouseBlend guide describes querying AR by Sales Rep as one example (Source: www.houseblend.io).

- **Cross-Module Joins:** As noted, many business questions require data from multiple modules. Because NetSuite's ERP and CRM are unified, SuiteQL can cross them easily. For instance, to find open invoices by customer, one might join `customer` with `transaction` on `customer.id = transaction.entity`, filtering for `transaction.type = 'CustInvc'` (invoice) and `status='Open'` (Source: www.houseblend.io). Or to

analyze marketing ROI, one could join `campaign` to `customer` (via campaign member records, etc.), or join project vs. customer vs. invoicing. HouseBlend emphasizes that any data visible in Workbook (e.g. in built-in joins) can be reproduced in SuiteQL (Source: www.houseblend.io). The following code illustrates a CRM-to-ERP join:

```
SELECT c.companyname, o.tranid AS order_id, o.total AS order_amount
FROM customer c
LEFT JOIN transaction o ON c.id = o.entity
WHERE o.type = 'SalesOrd' AND o.status = 'Billed';
```

This returns all customers and their billed sales orders (if any). Using a LEFT JOIN ensures customers with no orders still appear (with NULLs).

- **Joins involving Custom Records:** If you have a custom record type (say, `customrecord_projects`), you can join it to standard records if there are fields linking them. For example, if each Sales Order has a custom field `custbody_project` (a List/Record pointing to the Projects record), you could do:

```
SELECT so.tranid, so.amount, p.Name AS project_name
FROM transaction so
JOIN customrecord_projects p ON so.custbody_project = p.id
WHERE so.type = 'SalesOrd';
```

Here `custbody_project` holds the project's internal ID. Notice that we treat the custom record table just like a built-in (the table name is the custom record's scriptid, all lowercase as per Connect schema).

- **Advanced Join Types:** - **Right Outer Joins:** Similar to left joins but ensuring all rows from the *right* table appear. For example, if you want all employees even those who made no sales, you could `RIGHT JOIN transaction ON employee.id = transaction.salesrep`. However, Oracle-style (Connect) has no shorthand for right joins beyond the `RIGHT JOIN` keyword (Source: docs.oracle.com).

  - **Full Outer Joins:** Rarely needed in practice, but supported. A full join can yield rows unmatched on either side (including those with NULLs on both sides if no match). Example:

```
SELECT c.id AS custid, c.companyname, o.tranid AS order_id
FROM customer c
FULL OUTER JOIN transaction o ON c.id = o.entity;
```

  This gets all customers (with or without orders) and all orders (with or without customers) (Source: docs.oracle.com) (Source: docs.oracle.com). SuiteQL supports `FULL OUTER JOIN`, but there is **no implicit syntax** for it (you must use the keyword) (Source: docs.oracle.com).

  - **Cross (Cartesian) Joins:** Generally avoided except in analytical use-cases. There is no explicit `CROSS JOIN` in SuiteQL. One performs a Cartesian by listing tables with no ON clause (e.g. `FROM A, B`) or by doing a full outer join with `ON 1=1` (Source: docs.oracle.com). Both produce the Cartesian product (every combination of rows from A and B) (Source: docs.oracle.com). This can be useful for certain matrix analyses, but use with caution as it explodes result size.

Table: SuiteQL Join Types (summary, with example usage) is shown above.

Best Practices for Joins

- **Join on IDs:** Always join on the internal ID fields. It is common that `[child_table].entity` or `[child_table].customer` holds the parent's `id`. For example, `transaction.entity = customer.id`, `transaction.item = item.id`, `supportcase.company = customer.id` (Source: www.houseblend.io). Do not rely on labels or names to join (e.g. do not join `customer.entityid = supportcase.company` because `entityid` is text).
- **Test Relationships:** Use the Records Catalog to confirm relationships. For example, the Customer record overview shows *DefaultShippingAddress, Subsidiary, SalesRep* fields – each has an "internal" field label indicating which table it links to (Source:

www.houseblend.io). The Catalog will explicitly call out join fields. You can also do a quick Saved Search or Workbook join – often these UI tools restrict to valid joins, and you can then inspect the SQL.

- **Split Complex Joins:** If a join seems to involve many tables, consider breaking into simpler parts. The HouseBlend guide suggests building the join incrementally to verify each step (Source: www.houseblend.io). Additionally, prefer explicit JOIN syntax for clarity, but remember the Oracle-hint to use `(+)` for performance in Connect.
- **Avoid Over-Joining:** Return only the tables and fields you need. Each additional table in a join multiplies processing. If you only need a few columns from a child table, avoid `SELECT *`. List columns explicitly to minimize data transfer.
- **Cardinality:** Understand the relationship cardinality (one-to-many, many-to-many). For one-to-many, a GROUP BY may be needed if you want to summarize. Be careful with joins that could duplicate rows (e.g. joining two 1-to-many relationships together can blow up results).
- **Performance:** Joins on indexed fields (primary keys) are fastest. NetSuite generally indexes internal IDs. Linking on non-key fields (or on formula fields) can be slow. Where possible, filter (with WHERE) on tables *before* joining, to reduce row counts.

SuiteSQL Query Syntax and Functions

SuiteQL largely follows the ANSI SQL-92 standard, with some Oracle extensions supported (and required) for Connect. Key points:

- **Query Structure:** The usual SQL clauses apply: `SELECT … FROM … [JOIN … ON …] WHERE … GROUP BY … HAVING … ORDER BY …` and `LIMIT`. SuiteQL also supports `UNION` and `UNION ALL` for combining query results (Source: www.houseblend.io). Unlimited subqueries are allowed in the `FROM` clause (common table expressions via `WITH` are also supported in recent releases).
- **Functions:** SuiteQL includes a set of built-in functions (numeric, string, date). Supported functions are documented (e.g. `ABS`, `SUBSTR`, `BUILTIN.DF(column)` for display-friendly lookups, `TO_CHAR`, etc.). Note: not all Oracle functions exist. The documentation lists [supported functions and formulas] (Source: www.houseblend.io). Common functions include: `TO_CHAR(date, format)`, `COUNT()`, `SUM()`, `AVG()`, `MIN()`, `MAX()`, and `BUILTIN.DF()` to resolve reference field names.
- **BUILTIN.DF (Display Field):** One unique feature is `BUILTIN.DF()`. When you select an internal ID field (like `entity`), you can wrap it to get the human-readable name: e.g. `BUILTIN.DF(transaction.entity) AS customer_name` returns the customer's name instead of ID. This is purely for display in results; it doesn't change joins. Example:

  ```
  SELECT t.id, t.entity, BUILTIN.DF(t.entity) AS customer_name
  FROM transaction t;
  ```

  returns numeric entity ID and its corresponding name. (Source: timdietrich.me) It is especially useful when piping results into external tools.
- **Querying Suites of Records:** SuiteQL allows filtering on record types (e.g. `WHERE type = 'SalesOrd'` in transaction) and on specific fields. Some fields are multi-valued or complex; filtering on them may require different syntax. For example, to filter multi-select fields by an internal list value, use the `HASCHILDREN` or `contains` operator (depending on data type). The SuiteAnswers documentation on *SuiteQL Syntax and Limitations* should be consulted for complex cases.

SuiteAnalytics Connect Patterns and Considerations

This section outlines patterns specific to using SuiteQL through the Connect service and data source, including performance tuning, parallel loads, and integration.

Performance and Best Practices

As noted, NetSuite recommends using Oracle-ish syntax to optimize queries inside the Connect warehouse (Source: docs.oracle.com). In addition:

- **Limit Rows Early:** Use WHERE clauses as early as possible to filter rows. For very large tables (Invoices, Sales Orders, Journal Entries), always filter by date or status before joining to slimmer tables. If you only need recent data, specify that in the SQL.
- **Aggregate Strategically:** If you need totals or counts, use `GROUP BY` and `HAVING` instead of retrieving all rows and post-processing outside the database. However, remember that `TOP` or `LIMIT` do not necessarily short-circuit evaluation in SuiteQL; the engine may still scan all rows (Source: www.houseblend.io).

- **Avoid OR on Large Scans:** For multi-condition filters, try to rewrite as separate queries and combine (using UNION) instead of a big OR clause, which can bypass indexes (Source: www.houseblend.io).
- **Test on Sandbox:** If possible, pre-test heavy queries in a sandbox or with smaller datasets. The SuiteQL Query Tool and Workbook can both run SuiteQL; you can validate logic there.

Incremental Querying and Backups

For BI integrations, you often want to pull only changed data. SuiteAnalytics Connect provides mechanisms:

- **Last Modified Timestamps:** Many tables in NetSuite2.com include `lastModifiedDate` or `DateLastModified`. These suit incremental loads – e.g. `WHERE lastmodifieddate >= '2025-05-01'`.
- **Deleted Records:** There is a `deleted_records` table that logs deletions of rows (with table name, internal ID, and deletion date). Join or filter this table to identify and remove deleted items from your extract.
- **Unsupported Tables:** Some tables (particularly mapping or join tables) may not support timestamps or deletions (Source: docs.oracle.com). In those rare cases, the only safe approach is to reload the entire table each sync. NetSuite's guidance is to "file an issue" if an important table lacks incremental columns (Source: docs.oracle.com).

Using SuiteQL in Analytics Pipelines

Modern BI workflows often ingest data from NetSuite via SuiteQL. Typical patterns include:

- **Direct BI Connector:** Tools like Tableau or Power BI (with ODBC) can query SuiteQL directly. Some vendors (e.g. Coefficient.io) offer SuiteQL connectors that wrap queries and handle scheduling. For example, Coefficient's case study describes building a SuiteQL query that sums invoice lines by account and writing the results into Tableau on an hourly schedule (Source: www.houseblend.io). Because Tableau's native NetSuite connector has limitations, the SuiteQL approach allows multi-table aggregation that standard connectors cannot handle (Source: www.houseblend.io).
- **Data Warehouses:** Organizations might use ETL platforms (Fivetran, Talend, etc.) to load SuiteQL results into external data warehouses (Snowflake, BigQuery) for further processing. In such scenarios, one often uses the Connect driver API to schedule queries (or a built-in connector if available). The 30–50% ETL labor saving reported by NetSuite (by eliminating CSV exports) is largely realized in this pattern (Source: www.houseblend.io).
- **Custom Apps & Integration:** Developers use SuiteQL in SuiteScripts or external apps to embed NetSuite data into custom dashboards or dashboards. For instance, a SuiteScript may run a SuiteQL query and return JSON via SuiteLet to build a portlet on a portal.

Across these patterns, **consistency of data** is critical. NetSuite's analytics source is static between updates, so synchronizing at off-peak times (e.g. nightly) avoids partial data issues. Also, pay attention to API rate limits if running SuiteTalk calls.

Security and Governance Implications

SuiteQL exposes powerful capabilities, so governance is essential. Recommended practices include:

- **Roles and Permissions:** Use least-privilege roles for Connect. Only grant access to tables/records needed. (The Records Catalog shows which permission controls each record under "Overview" – e.g. you see "Permission: Transactions > Invoices". If your role lacks that, querying `transaction` rows with `type='CustInvc'` will fail (Source: www.houseblend.io).)
- **Out-of-Office Access:** If giving a BI team access, consider a machine-to-machine role (TBA) rather than an HR user's credentials. This avoids user lockouts or password changes interrupting reports.
- **Audit Logging:** Since SuiteQL is essentially another access channel, it should be audited. Check NetSuite's Audit Trail to see that no data was modified via Connect (it can't, but just to record who queried).
- **Data Sensitivity:** Mask or exclude sensitive fields as needed. For example, if custom fields hold personal info, ensure only authorized roles query them. There's no built-in Mask function, so redact via query logic or omit those columns.

Case Studies and Examples

To illustrate how SuiteQL is used in practice, consider the following scenarios:

- **Customer Aging Dashboard:** A finance analyst needs a list of open invoices and their customers. Using SuiteQL, one might write:

```
SELECT c.entityid AS Customer, inv.tranid AS InvoiceNo, inv.duedate, inv.total
FROM customer c
JOIN transaction inv ON c.id = inv.entity
WHERE inv.type = 'CustInvc' AND inv.status = 'Open';
```

  This single query produces a tabular aging report. Additional joins (e.g. to `employees` for sales rep, or to a custom segment for region) can enrich the data. Without SuiteQL, the analyst would have had to export saved searches and manually join them in Excel – a slow, error-prone process.

- **Custom Field Audit:** An internal auditor needs to verify that no inactive fields are used on new transactions. One can run:

```
SELECT t.tranid, cf.scriptid AS field_id, cf.IsInactive
FROM transaction t
CROSS JOIN UNNEST(ARRAY(SELECT *
    FROM UNNEST(t.CustomFields) cf_sub
    WHERE cf_sub.IsInactive = 'T') AS cf
WHERE t.datecreated > '2026-01-01';
```

  (Note: the exact syntax for flattening custom fields depends on API; as an alternative, one can join `transaction` to its custom columns directly if Connect exposes them.)
  In any case, SuiteQL enables querying the `CustomField` metadata:

```
SELECT cf.ScriptID, cr.Name AS RecordType
FROM CustomField cf
JOIN CustomRecordType cr ON cf.RecordType = cr.InternalID
WHERE cf.IsInactive = 'T';
```

  to find all custom fields marked inactive (and on which record they sit) (Source: www.houseblend.io) (Source: www.houseblend.io).

- **Marketing Audience Query:** A marketer wants leads in certain ZIP codes who have previously purchased a particular item. This requires combining CRM (lead addresses) with ERP (orders). One solution is to use a `UNION` of two subqueries (as detailed by Tim Dietrich): first query customers by ZIP, second query customers who bought the item, then merge the unique set (Source: www.houseblend.io) (Source: www.houseblend.io). In SuiteQL it might look like:

```
SELECT DISTINCT c.id, c.entityid
FROM customer c
WHERE c.zip IN ('12345','67890')
UNION
SELECT DISTINCT c2.id, c2.entityid
FROM customer c2
JOIN transaction t2 ON c2.id = t2.entity
JOIN transactionline tl2 ON t2.id = tl2.transaction
WHERE tl2.item = 999;  -- internal ID of product
```

  The result is the ID and name of each customer meeting either criterion, ready to feed into a targeted campaign.

- **BI Integration (Tableau):** A data team uses Coefficient to connect NetSuite and Tableau. They write SuiteQL queries in Google Sheets that Coefficient refreshes via the NetSuite driver. For example, to build a financial KPI dashboard, they might query a combination of transactions, subsidiaries, and accounts in one go, and push the resulting sheet into Tableau. According to the Coefficient case study, SuiteQL handled multi-

table aggregates that their old connector could not (Source: www.houseblend.io), and ran these queries every hour for near-live reporting.

These examples show how SuiteQL streamlines workflows. Instead of manual exports or complex integrations, analysts can *directly query* the ERP, combining data holistically. This reduces errors and accelerates insight delivery.

Discussion and Future Directions

SuiteQL adoption continues to grow rapidly. NetSuite cites research showing substantial efficiency gains (30–50% less reporting work) and industry demand (84% of BI projects involve cloud ERP data) (Source: www.houseblend.io) (Source: www.houseblend.io). Because custom fields are so prevalent — surveys name "field-level reporting" as a top pain point (Source: www.houseblend.io) — SuiteQL's ability to expose that metadata has been widely welcomed by finance and operations teams.

Looking forward, potential developments include:

- **Enhanced Metadata APIs:** While SuiteQL covers custom definitions, forthcoming REST metadata enhancements (e.g. richer "record/v1" schemas) will let developers retrieve standard record structures on demand (Source: blogs.oracle.com). This could allow a SuiteQL query to introspect any record type, reducing dependence on static inventories.
- **Performance Optimizations:** As data volumes grow (OneWorld accounts, large transaction histories), performance tuning becomes critical. Expect further improvements from NetSuite in query optimization and indexing behind the scenes. Users may need to adopt strategies like pre-aggregating or summarizing in data warehouses if extremely large joins start to slow down.
- **AI-Driven Query Aids:** The HouseBlend report speculates that AI might assist in writing SuiteQL (via natural language or autocomplete) in the future (Source: www.houseblend.io). Early signs are already seen in some tools that parse text prompts into SQL; this could make SuiteQL more accessible for casual users.
- **Extended Access Patterns:** Currently SuiteQL is read-only. There are community asks for write-back (an SQL-based data loading interface), but any such feature would require careful governance. Another frontier is real-time API integration: Coupling SuiteQL with webhooks (or the Netsuite "Driven Events" framework) could stream data to analytics platforms instantly.

Conclusion

NetSuite's SuiteAnalytics Connect and SuiteQL together represent a major evolution in NetSuite reporting. By treating the ERP/CRM as a live SQL database, SuiteQL empowers organizations to retrieve and blend data flexibly and efficiently. This reference has outlined how to query **custom fields and records** (via the `CustomField` / `CustomRecordType` tables), how to apply **SQL joins and set operations**, and which **Connect-specific patterns** to use for robust integration. We have drawn on Oracle's official NetSuite documentation (Source: docs.oracle.com) (Source: docs.oracle.com), expert blogs (Source: timdietrich.me) (Source: www.houseblend.io), and real-world examples (Source: www.houseblend.io) (Source: www.houseblend.io) to ensure accuracy and practical guidance.

The evidence suggests SuiteQL significantly shortens reporting cycles and opens new analytical possibilities. As NetSuite deployments grow (40,000+ customers and counting (Source: www.houseblend.io) and BI demands rise, SuiteQL will likely become the standard way to access NetSuite data. Organizations that master SuiteQL – especially using its metadata tables for custom fields – will gain a strategic advantage by unlocking insights hidden in their customizations. We encourage teams to build out SuiteQL skills, document their Connect schemas, and continuously refine their queries using the guidelines herein. With rigorous security governance and performance tuning, SuiteQL can serve as a secure, high-performance analytics layer at the heart of NetSuite-driven businesses (Source: www.houseblend.io) (Source: www.houseblend.io).

Tables: Above we included tables summarizing join types and naming conventions. Additional references and a bibliography can be provided on request. All claims and examples in this report are backed by authoritative sources and community experts to ensure reliability and depth (Source: www.houseblend.io) (Source: www.houseblend.io).

Tags: netsuite suiteql, suiteanalytics connect, sql joins, custom fields, netsuite custom records, oracle sql, netsuite2.com, database querying, erp reporting

---