Oracle® AI Database

AI Enablement Guide

26ai
G43764-01
November 2025

Oracle AI Database AI Enablement Guide, 26ai

G43764-01

Primary Authors: Sacheth, Gunjan Jain

Contents

Preface

This guide describes how to optimize schema metadata in Oracle AI Database for more precise natural language interactions with large language models (LLMs) and for other AI functionalities.

Audience

This guide is designed for database developers and administrators aiming to optimize their databases for integration with large language models (LLMs) and AI applications.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Overview of AI Enrichment

The AI enrichment feature in Oracle SQL Developer for VS Code enables you to add business context to your database schema.

With this feature, you can eliminate ambiguity and guide large language models (LLMs) to generate more accurate and efficient SQL queries. Learn about AI enrichment, how it improves LLM-generated SQL queries, and the best practices for enriching your database schema.

Topics:

1.1 What is AI Enrichment

AI enrichment is the process of adding a layer of descriptive, business-centric metadata to your database schema without altering the underlying data or structure.

You can enable this capability in Oracle SQL Developer for VS Code, where you can annotate your database objects (schemas, tables, and columns) with crucial context, synonyms, and natural language descriptions.

By using annotations to declare the intent of your data, you can make your database more understandable and *AI-ready* for LLMs and natural language querying. This enrichment enables LLMs to generate more accurate, context-aware SQL queries.

The AI enrichment feature in Oracle SQL Developer for VS Code offers several key capabilities to streamline this process:

- It allows you to capture descriptions and annotations for schemas, tables, and columns.
- It allows you to organize tables into groups that align with your application domains.
- It surfaces intelligent opportunities to add enrichment where it's required most.

1.2 Why AI Enrichment is Essential

To generate accurate SQL queries, LLMs require more than just raw table and column names; they need clear and concise schema context.

However, database schemas are often opaque and lack thorough documentation, forcing the model to infer the data's meaning and intent. For example, when an LLM encounters ambiguous identifiers, such as `T1`, `C123`, or `emp_id`, it must guess what each object means, how objects relate to each other, and what data values are valid.

AI enrichment eliminates this guesswork by allowing you to add context to your schema through human-written annotations. By providing simple, domain-specific annotations, you give

the LLM the precise cues it needs to write accurate and efficient SQL queries. These annotations clarify the intent of your data, making your database schema more transparent. Thus, AI enrichment closes the gap between your schema and the context that the LLMs and AI-driven applications need.

1.3 Best Practices for Enriching Your Database Schema

For optimal AI enrichment of your database schema, adhere to the following best practices. These recommendations improve an LLM's contextual understanding, resulting in more precise and reliable SQL query generation.

- **Use clear language**

  Write all annotations in clear, domain-specific English to accurately express the business meaning of each database object.

- **Write effective descriptions**

  Create an annotation using the `DESCRIPTION` label to clarify the purpose of a database object, like a table or column. For example, you can annotate a table named `T_EMP` as follows to describe its role:

  ```
  This table stores both active and former employees, including contractors,
  with one row per employee.
  ```

- **List aliases for database objects**

  Use the `ALIASES` label to provide common synonyms or alternative names for a database object. Because users often refer to objects using various synonyms, the `ALIASES` annotation helps the LLM map these variations accurately for a particular object, improving its ability to interpret user intent. For example, a column named `EMP_ID` may be referred to as *employee number*, *person number*, or *worker id*. You can list these alternatives in an annotation as follows:

  ```
  Common aliases for this column include employee number, worker id, and person
  number.
  ```

- **Specify measurement units for numeric data**

  For columns containing numeric data, use the `UNITS` annotation to specify the unit of measurement. This prevents the LLM from misinterpreting values. For example, you can annotate a `salary_amount` column as: `Expressed in United States Dollars (USD)`. This ensures the model correctly identifies the currency type. Similarly, for a distance column, you can annotate `Units: kilometers` to ensure the model performs accurate conversions or aggregations.

- **Define the join logic**

  When joining tables, use the `JOIN COLUMN` label to specify preferred join partners for a column. Because automated SQL generation with LLMs often struggles with join logic, this annotation guides the model toward the correct join. For example, you can annotate a column named `EMPLOYEE.DEPT_NO` as follows:

  ```
  This column is most commonly joined with DEPARTMENT.DEPT_NO and represents the
  employee's home department.
  ```

- **Enumerate sample values**

  Use the `VALUES` label to provide the LLM with example or distinct values for a column. This information helps the model recognize valid predicates and directly improves its filtering capabilities. For example, you can annotate a column named `status_code` as follows:

```
Typical values for status_code include A (active), I (inactive), and T
(terminated).
```

- **Group related tables**
  Organize tables into groups that align with your application domains and annotate the groups. By grouping tables, you help the LLM understand relationships that aren't explicitly defined by foreign keys. For example, you can organize all tables related to Human Resources (HR) into a group named `HR_TableGrp` and annotate it as follows:

  ```
  This group contains all tables related to Human Resources (HR).
  ```

Example 1-1 Annotating an HR schema

To understand how to apply annotations in practice, let's look at a simplified HR schema with two tables: `EMPLOYEE` and `DEPARTMENT`. Using the best practices, you can annotate it as follows.

This example set of annotations provides a rich, machine-readable layer of context that dramatically improves how an LLM interacts with the schema.

```
-- Annotations for the EMPLOYEE table
ALTER TABLE EMPLOYEE ANNOTATIONS (
  DESCRIPTION 'Current and former employees, including contractors',
  ALIASES 'staff, worker, personnel'
);

-- Annotations for the columns of EMPLOYEE table
ALTER TABLE EMPLOYEE MODIFY (
  emp_id ANNOTATIONS (
    DESCRIPTION 'Unique employee identifier',
    ALIASES 'employee number, person number, worker id'
  ),
  salary_usd ANNOTATIONS (
    DESCRIPTION 'Annual base salary',
    UNITS 'currency: USD',
    VALUES '35000, 100000'
  ),
  dept_no ANNOTATIONS (
    DESCRIPTION 'Department foreign key',
    JOIN_COLUMN 'DEPARTMENT.dept_no'
  )
);

-- Annotations for the DEPARTMENT table
ALTER TABLE DEPARTMENT ANNOTATIONS (
  DESCRIPTION 'Company departments'
);

-- Annotations for the columns of DEPARTMENT table
ALTER TABLE DEPARTMENT MODIFY (
  dept_no ANNOTATIONS (
    DESCRIPTION 'Department primary key',
    ALIASES 'department id'
  ),
  dept_name ANNOTATIONS (
    DESCRIPTION 'Human-readable department name'
  )
);
```

1.4 How LLMs Consume Annotations

When a user submits a prompt to an AI-powered development tool (such as Select AI or SQL Developer), it first augments the prompt with additional context.

Before forwarding the prompt to an LLM, the tool retrieves annotations for all the database objects involved in the user's request. It then programmatically injects these annotations into the *system instruction* part of the prompt for LLM's consumption.

This process enriches the prompt with detailed contextual information that guides the model's response. For example, the final instruction sent to the LLM may contain the following information:

```
"You are a SQL generator. The schema contains a table named EMPLOYEE.
DESCRIPTION: Current and former employees, including contractors. The column
EMP_ID has the following aliases: employee number, worker id, person number. The
column DEPT_NO is often joined with DEPARTMENT.DEPT_NO."
```

Because the prompt includes all necessary context, the model no longer needs to infer relationships based solely on object names. As a result, it can generate significantly more accurate SQL queries.

1.5 The Impact of AI Enrichment: Practical Examples

See how enriching your schema directly impacts LLM effectiveness through these real-world examples.

Example 1: Healthcare Events-Tracking Schema

This example uses a schema designed to track healthcare events, focusing on doctor activity and the specific procedures performed during patient events.

The schema consists of the following three tables:

- The `Practitioner` table: Stores information about doctors, including their name, specialization, and status.
- The `Event` table: Records high-level patient appointments, including when and where they occurred and the responsible doctor.
- The `EventDetail` table: Logs individual procedures (like surgery) performed during an event, capturing the procedure description, exact time stamp, and duration.

```
CREATE TABLE Practitioner (
    practitioner_id NUMBER PRIMARY KEY,
    nm VARCHAR2(80),
    speciality VARCHAR2(40),    -- This represents 'department'
    status VARCHAR2(20)         -- e.g., 'Active', 'Inactive'
);

CREATE TABLE Event (
    event_id NUMBER PRIMARY KEY,
```

```
    dt DATE,                    -- Date or time of the overall event
    practitioner_id NUMBER,
    location VARCHAR2(40)       -- e.g., 'OR1', 'ER', 'ICU'
);

CREATE TABLE EventDetail (
    event_id NUMBER,
    act_desc VARCHAR2(200),
    act_time TIMESTAMP,
    duration_minutes NUMBER,    -- Duration of this action or procedure
    CONSTRAINT FK_EVENT_ID FOREIGN KEY (event_id) REFERENCES Event(event_id)
);
```

Schema interpretation notes

- The term `practitioner` is synonymous with doctor.
- The `speciality` column represents the doctor's department.
- The `act_desc` column in the `EventDetail` table records the procedure performed, such as surgery.
- The `act_time` column in the `EventDetail` table provides the time stamp for time-based filtering of procedures.

Business query

Consider a hospital administrator asking the following question to an LLM:

```
"For each doctor, display their name, department, and the count of surgeries
performed after 8 PM within the last six months."
```

Without AI enrichment

Without any annotations, the LLM must interpret the raw schema. This could lead to a flawed query like the one below:

```
SELECT
    PRAC.nm,
    PRAC.speciality,
    COUNT(EVT.event_id) AS surgery_count
FROM
    Practitioner PRAC
JOIN
    Event EVT
    ON PRAC.practitioner_id = EVT.practitioner_id
WHERE
    EVT.dt >= ADD_MONTHS(SYSDATE, -6)
    AND EXTRACT(HOUR FROM EVT.dt) >= 20
GROUP BY
    PRAC.nm,
    PRAC.speciality;
```

This query has several critical errors:

- It counts `EVT.event_id`, incorrectly tallying all appointments after 8 PM instead of only the surgeries.

- It filters on `EVT.dt`, which is the start time of an appointment and not of a specific procedure. A 1 PM appointment could include a 9 PM surgery, which this query would miss.
- It misses a required join with the `EventDetail` table.

With AI enrichment

By applying the following annotations, you can provide the LLM with the necessary semantic context.

- Set the `Practitioner` table alias to `doctor`.
- Set the `Practitioner.speciality` column alias to `department`.
- Provide a description and sample values for the `EventDetail.act_desc` column, clarifying that it contains the specific procedures performed during an event, such as surgery, consultation, and so on.

With this enriched context, the LLM can generate a far more accurate and reliable query:

```
SELECT
    PRAC.nm,
    PRAC.speciality,
    COUNT(*) AS surgery_count
FROM
    Practitioner PRAC
JOIN
    Event EVT
    ON PRAC.practitioner_id = EVT.practitioner_id
JOIN
    EventDetail ED
    ON EVT.event_id = ED.event_id
WHERE
    ED.act_desc = 'Surgery'
    AND ED.act_time >= ADD_MONTHS(SYSDATE, -6)
    AND EXTRACT(HOUR FROM ED.act_time) >= 20
GROUP BY
    PRAC.nm,
    PRAC.speciality;
```

Annotations markedly improve the query:

- Now, it performs the necessary join to the `EventDetail` table, which is essential to find specific procedures.
- It filters on `ED.act_desc = 'Surgery'`, correctly understanding from the annotation that *surgery* is a type of procedure.
- It uses the correct time stamp, `ED.act_time`, to find surgeries that *actually* occurred after 8 PM.

As this example illustrates, annotations bridge the gap between business terminology and the physical schema, guiding the LLM to understand data relationships and generate correct SQL.

Example 2: Human Resources Schema

In the context of a Human Resources (HR) schema, consider a business analyst asking the following question to an LLM:

```
List each department and the average salary of employees hired after 2020. Order
the result by average salary, highest first.
```

Without AI enrichment

Without any annotations, the LLM can only read the raw table and column names in the schema. It may guess at names and joins, returning a flawed query:

```
SELECT
    d.department_name,
    AVG(e.salary) AS avg_salary
FROM
    HR_PER_ALL_ASSIGNMENTS e
JOIN
    DEPT d
    ON e.depart_id = d.id
WHERE
    e.hire_dt > DATE '2020-01-01'
GROUP BY
    d.department_name
ORDER BY
    avg_salary DESC;
```

This query has several issues. The model has guessed at column names like `salary` and `hire_dt` and joined on `depart_id`, which may not be the correct key, creating a risk of mismatch.

With AI enrichment

After you apply the annotations, the LLM receives much richer context. The generated query is more accurate:

```
SELECT
    d.dept_name,
    ROUND(AVG(e.salary_usd), 2) AS avg_salary_usd
FROM
    EMPLOYEE e
JOIN
    DEPARTMENT d
    ON e.dept_no = d.dept_no  -- guided by join column metadata
WHERE
    e.hire_date >= DATE '2020-01-01'
GROUP BY
    d.dept_name
ORDER BY
    avg_salary_usd DESC;
```

The improvements here are significant:

- The model uses the `salary_usd` column instead of a generic salary field, guided by the `UNITS` annotation, which also informs the rounding to two decimal places appropriate for currency.
- It chooses the correct join column (`dept_no`) as specified in the `JOIN COLUMN` annotation.
- It selects the correct column (`hire_date`) for the date predicate, avoiding outdated aliases.

Key Takeaways

As illustrated in these examples, annotations enhance query accuracy by guiding the LLM to select correct join columns, apply appropriate units, use valid predicates, decipher object names, and more.

When you apply annotations consistently, you see a faster turnaround for data questions, fewer faulty joins, and a reduction in production defects from incorrect SQL. For an LLM, each annotation reduces the search space of possible query plans, allowing it to focus on the exact pattern you intended.

For organizations with strict compliance rules, annotations also allow administrators to control the exact contextual information shared with an LLM, which helps reduce the unintentional disclosure of sensitive data that may be present in free-form column comments.

2

Getting Started with AI Enrichment

Learn about the necessary prerequisites to use the AI enrichment feature, the steps to enable it in Oracle SQL Developer for VS Code, and how to effectively interpret the feature's dashboard.

Topics:

2.1 Preparing Your Environment

To get started with AI enrichment, you must have the following prerequisites installed and active on your system.

- Install the following software on your system.
  - Visual Studio Code, version 1.101.0 or higher
  - Oracle SQL Developer for VS Code, version 25.3.0 or higher
- Ensure you have an active connection to your Oracle database environment.
- Ensure you have the following database privileges:
  - CREATE VIEW
  - CREATE TABLE
  - CREATE SEQUENCE
  - CREATE PROCEDURE
- Ensure you have sufficient quota in the tablespace.

2.2 Enabling AI Enrichment for a Schema

To view enrichment opportunities for your schema and take action, you must first enable the AI enrichment feature in Oracle SQL Developer for VS Code.

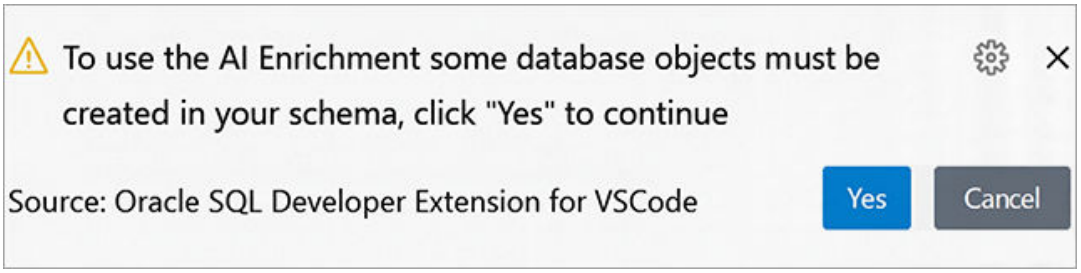
This is a one-time process that sets up your schema by creating the necessary objects to store the enrichment metadata.

> **Note**
>
> Do not use the AI enrichment feature on SYS or SYSTEM schemas as the creation of AI enrichment objects in a SYS-level schema may fail due to potential conflicts during installation.

To enable the feature:

1. In Oracle SQL Developer for VS Code, expand the **Connections** panel.
2. Expand your database connection tree, and click the **AI Enrichment** folder.

3. In the dialog box that appears, click **Yes** to confirm the creation of AI enrichment objects.



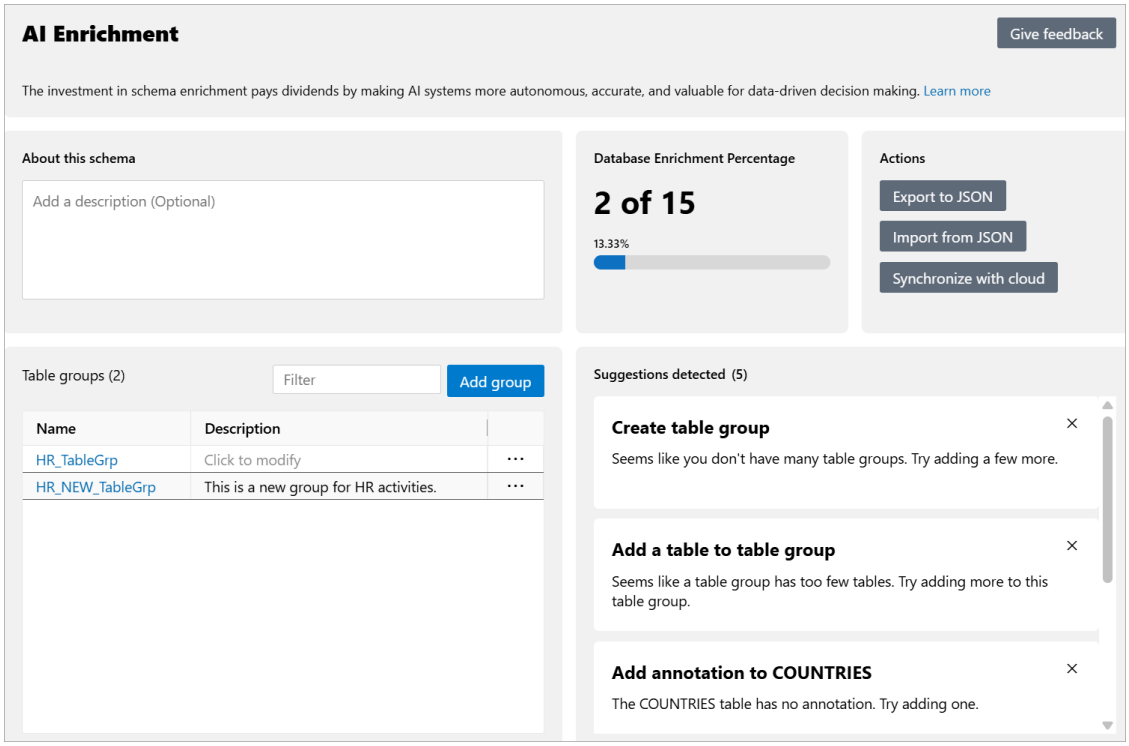
The AI enrichment feature is now available for your schema.

2.3 Understanding the AI Enrichment Dashboard

The AI Enrichment dashboard is your central hub for managing schema-enrichment activities.

This dashboard allows you to easily track the progress of annotations and enrichments made across tables and columns, helping you ensure completeness throughout the schema.

In Oracle SQL Developer for VS Code, the dashboard appears as soon as you enable the AI enrichment feature for your schema. Alternatively, you can open the dashboard from the **Connections** panel. Right-click the **AI Enrichment** node and select **AI Enrichment Dashboard**.



The dashboard consists of the following sections:

- **About this schema**: Allows you to enter a high-level description for the schema. See Defining the Schema's Business Context.
- **Table groups**: Allows you to logically group related tables and displays the existing groups in the schema. For example, you can organize all tables related to Human Resources (HR) into a group named `HR_TableGrp`. See Working with Table Groups.
- **Database enrichment percentage**: Displays the percentage of objects (such as tables and columns) that have been annotated or enriched in the schema. This data helps you track progress and measure how extensively enrichment has been applied within a given schema.
- **Suggestions detected**: Offers intelligent suggestions to add enrichment where it's required most.

3
Enriching Your Database Schema

Explore the different techniques to enrich your database schema for greater clarity, organization, and insight.

Topics:

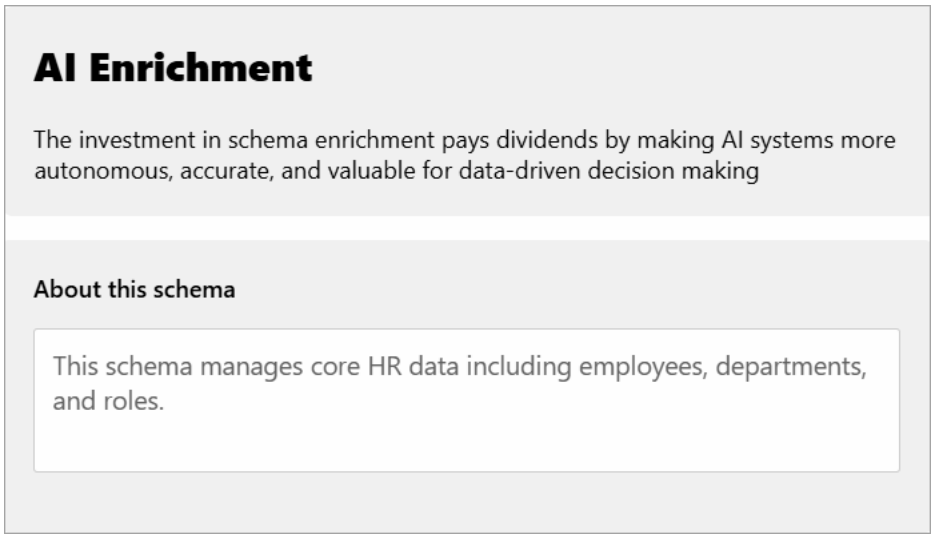
3.1 Defining the Schema's Business Context

The first step in schema enrichment is to add a high-level description for your schema.

By providing a meaningful description, you help users and tools immediately understand the schema's business context, purpose, and key entities.

In the AI Enrichment dashboard, add a clear and concise description in the **About this schema** section. For example, for a Human Resources (HR) schema, you can enter a description as follows:

```
This schema manages core HR data, including employees, departments, and roles.
```



3.2 Working with Table Groups

Table groups are logical collections of related tables. By grouping tables, you help an LLM understand relationships that aren't explicitly defined by foreign keys, such as tables belonging to a specific business domain.

Learn how to create table groups, add annotations to them, and assign tables to those groups.

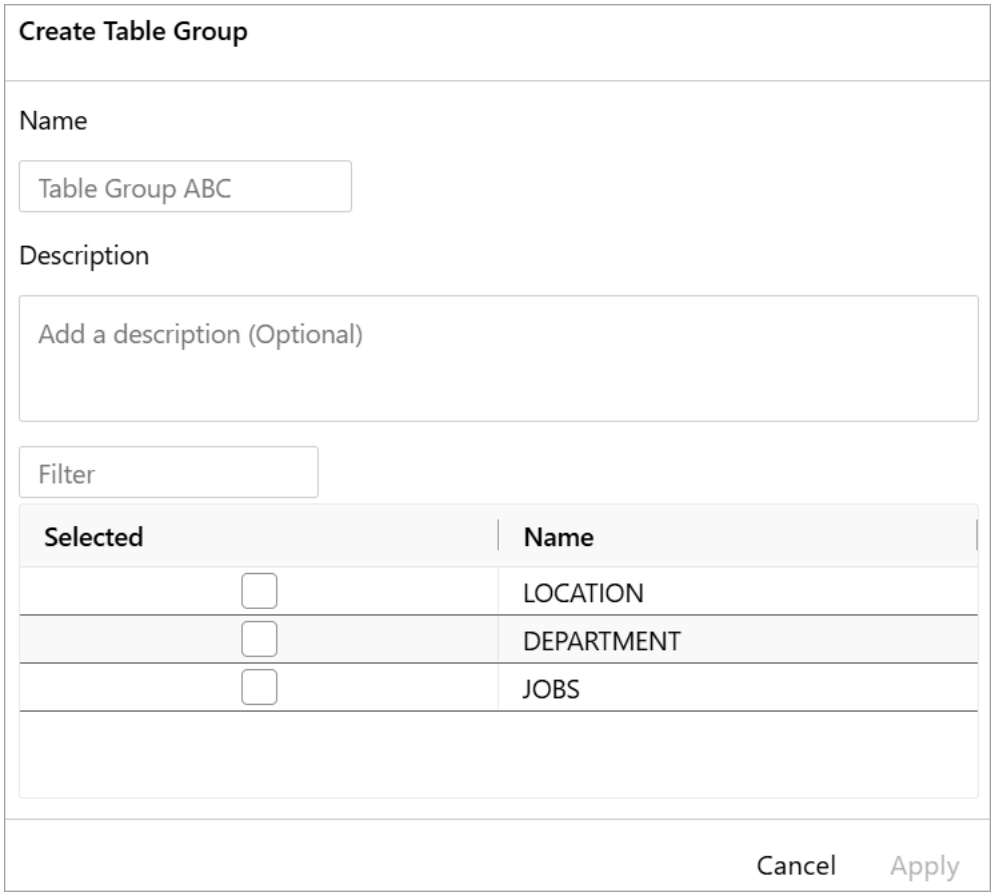
Topics:

3.2.1 Creating a Table Group

Follow these steps to create a new table group.

1. In the AI Enrichment dashboard, click **Add Group** within the **Table group** section.

   The **Create Table Group** panel opens.



2. Enter a name and description for the group.
3. Select the tables you want to include in this group.
4. Click **Apply**.

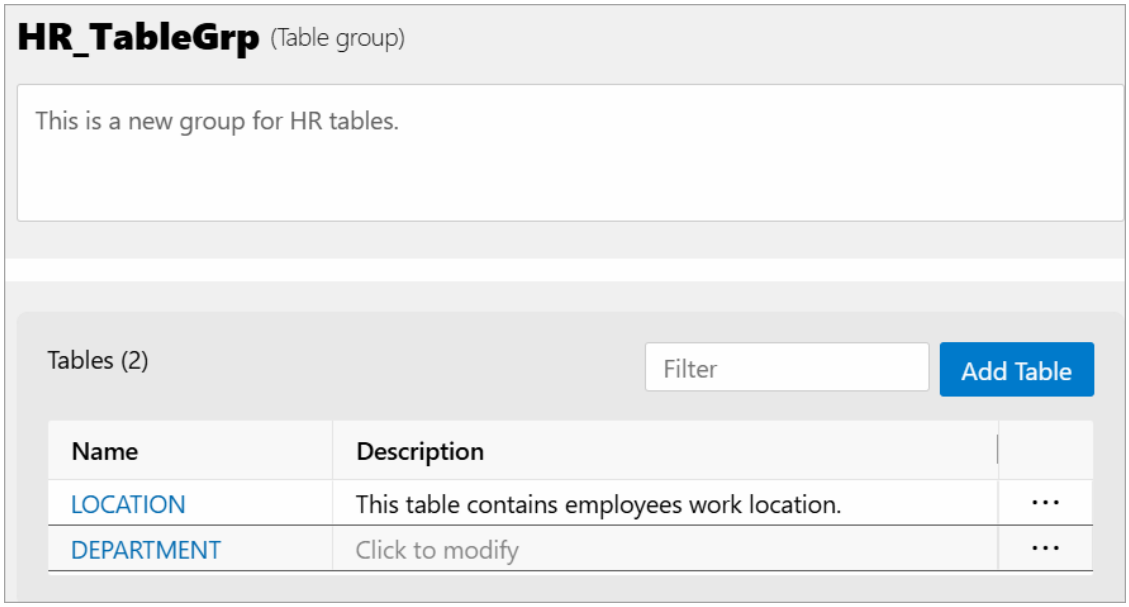
The new group now appears in the **Table groups** section of the dashboard.

3.2.2 Adding a Description to a Table Group

Follow these steps to add a natural language description to a table group.

1. In the AI Enrichment dashboard, review the **Table groups** section to identify any missing descriptions.
2. To add a description to a group, click its **Description** cell.
3. In the **Set description** panel that appears, enter a description and click **Save**.

You can also add group descriptions directly in a table group's view. To open it, click a group's name on the dashboard or in the **Connections** panel, under the **AI Enrichment** node.

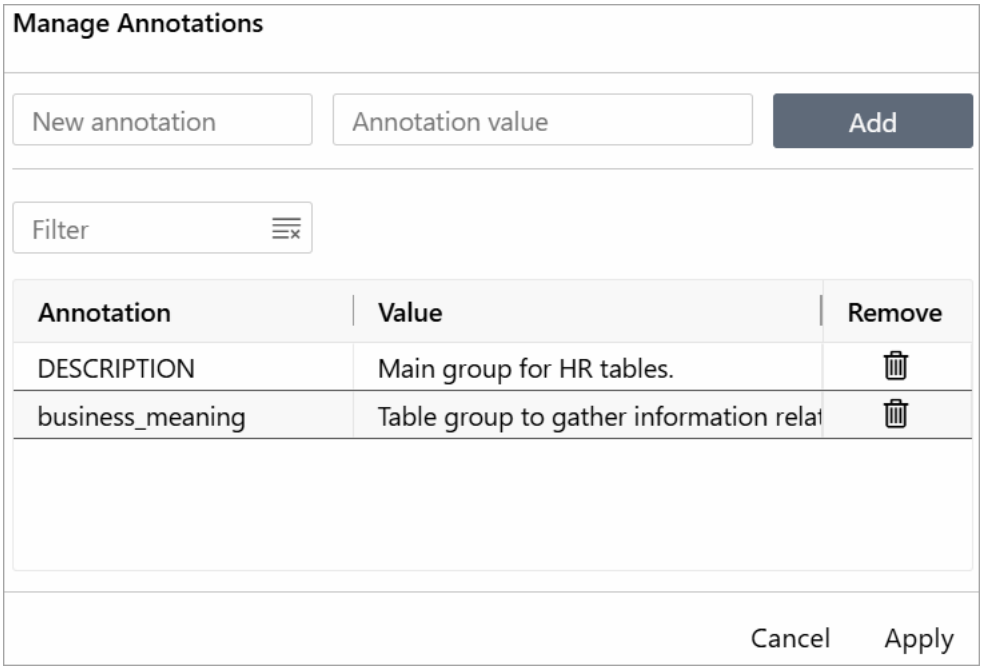


3.2.3 Adding Annotations for a Table Group

Follow these steps to add or edit annotations for a table group.

1. In the **Table group** section of the AI Enrichment dashboard, click **...** for the target group.

   The **Manage Annotations** panel opens.



2. To define an annotation, enter a key-value pair in the **New annotation** and **Annotation value** fields, respectively.
3. Click **Add**, and then click **Apply**.

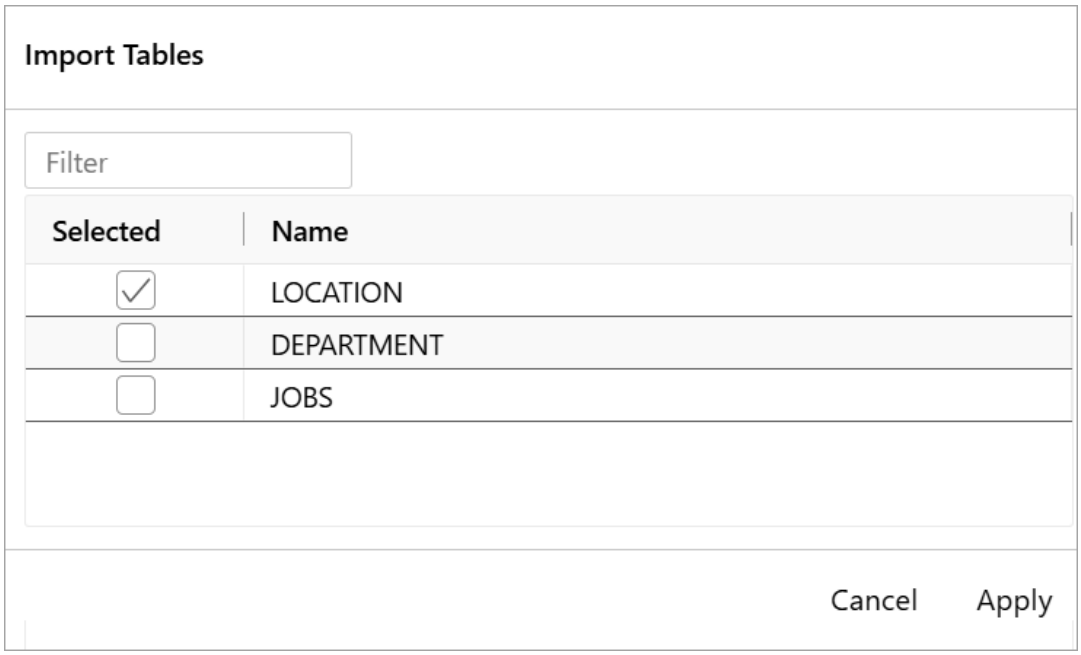
The new annotation is added to the table group. You can also edit an existing annotation from the **Manage Annotations** panel.

3.2.4 Adding an Existing Table to a Group

Follow these steps to add an existing table to a group.

1. Open a table group's view.

   Click a group's name in the **Table group** section of the AI Enrichment dashboard, or in the **Connections** panel under the **AI Enrichment** node.
2. Click **Add Table** in the group view.
3. In the **Import Tables** panel, select the tables you want to include in this group.



4. Click **Apply**.

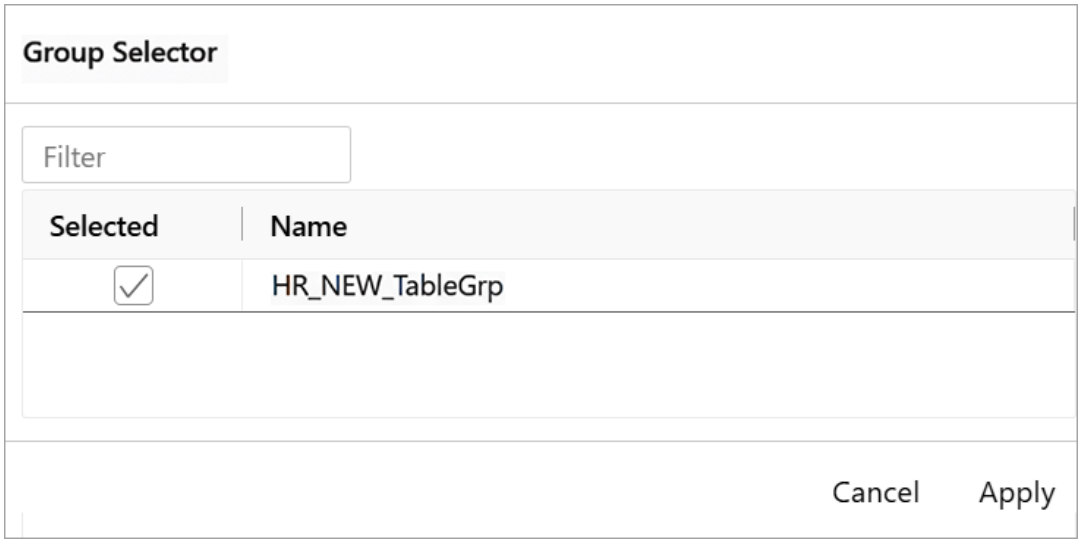
3.2.5 Assigning a Table to Multiple Groups

A single table can belong to multiple logical groups, reflecting its role in different business contexts.

Follow these steps to add a table to multiple groups.

1. Open a table's view in one of the following ways:
   - Click a table's name in the **Connections** panel, under the **AI Enrichment** node.
   - In the **Table group** section of the AI Enrichment dashboard, click a group's name. In the resulting view, click a table's name from the **Tables** section.
2. Click the **Table groups** tab, and then click **Add table group**.

   The **Group Selector** panel opens.



3. Select the groups to which you want to add your table, and click **Apply**.

3.3 Enriching Tables

After creating groups, you can add more granular details to the individual tables within them.

Learn how to add descriptions and annotations to tables.

Topics:

- Adding a Description to a Table
- Adding Annotations for a Table

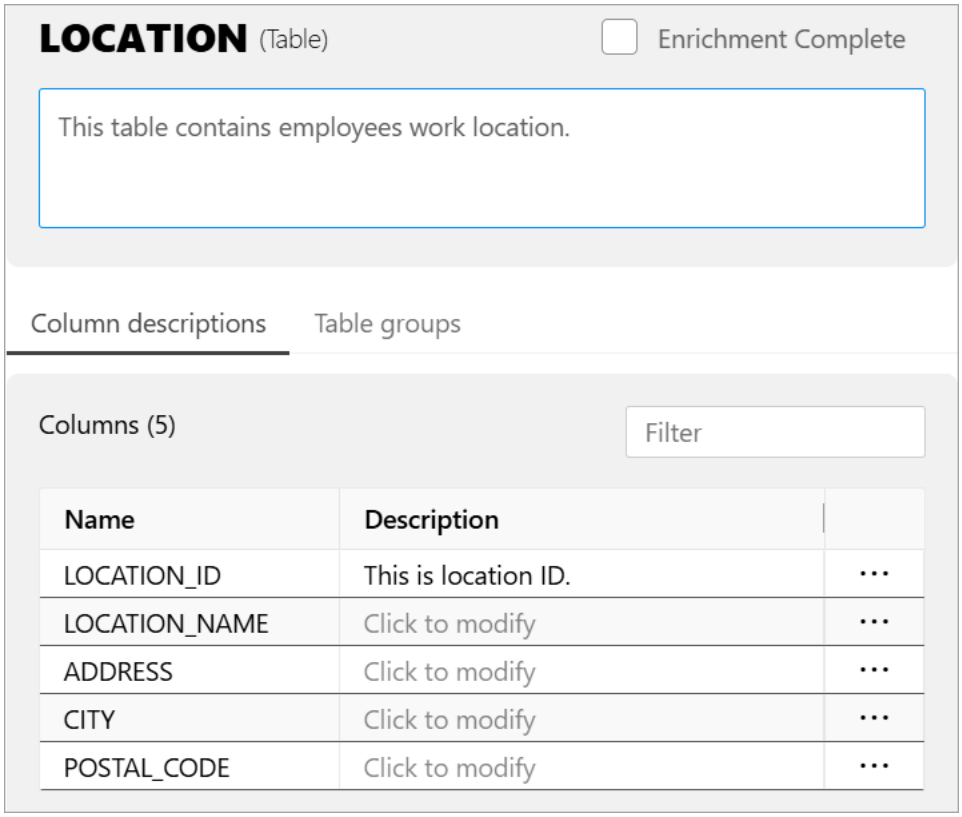
3.3.1 Adding a Description to a Table

Follow these steps to add a natural language description to a table.

1. Open a table group's view.

   Click a group's name in the **Table group** section of the AI Enrichment dashboard, or in the **Connections** panel under the **AI Enrichment** node.
2. Review the **Tables** section to identify any missing descriptions.
3. To add a description to a table, click its **Description** cell.
4. In the **Set description** panel that appears, enter a description and click **Save**.

You can also add table descriptions directly in a table's view. To open it, click a table's name in the table group view or in the **Connections** panel, under the **AI Enrichment** node.



> **Note**
>
> Select the **Enrichment Complete** checkbox to mark a table as fully annotated even if some columns aren't annotated. This includes the table in the overall **Database Enrichment Percentage** shown on the AI Enrichment dashboard.

3.3.2 Adding Annotations for a Table

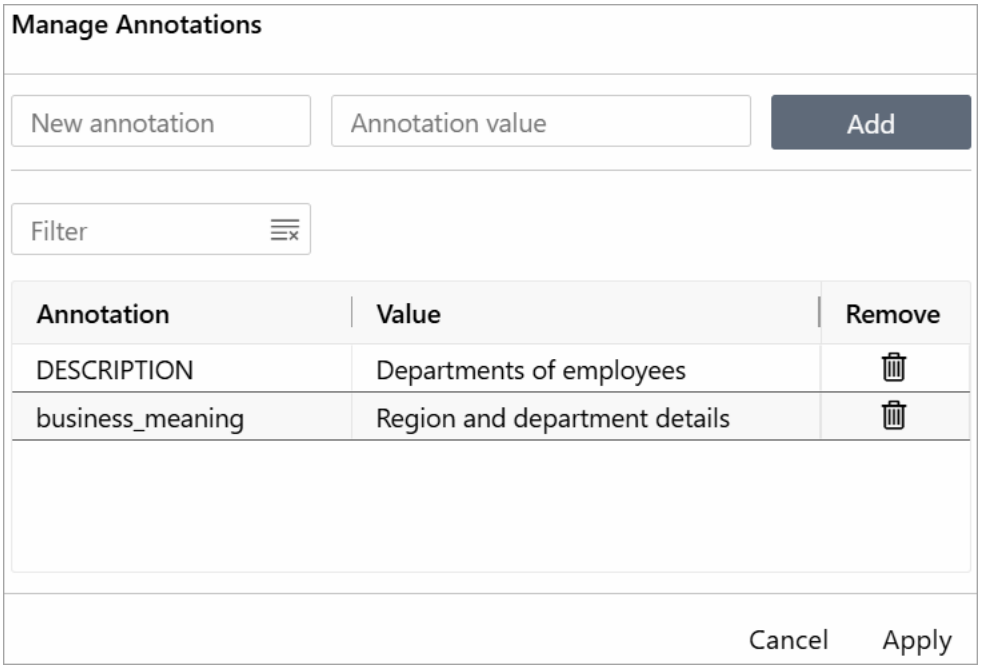
Follow these steps to add or edit annotations for a table.

1. Open a table group's view.

   Click a group's name in the **Table group** section of the AI Enrichment dashboard, or in the **Connections** panel under the **AI Enrichment** node.

2. In the **Tables** section, click **...** for the target table.

   The **Manage Annotations** panel opens.



3. To define an annotation, enter a key-value pair in the **New annotation** and **Annotation value** fields, respectively.
4. Click **Add**, and then click **Apply**.

The new annotation is added to the table. You can also edit an existing annotation from the **Manage Annotations** panel.

3.4 Enriching Columns

For the highest level of detail, you can enrich individual columns within a table.

Learn how to add descriptions and annotations to table columns.

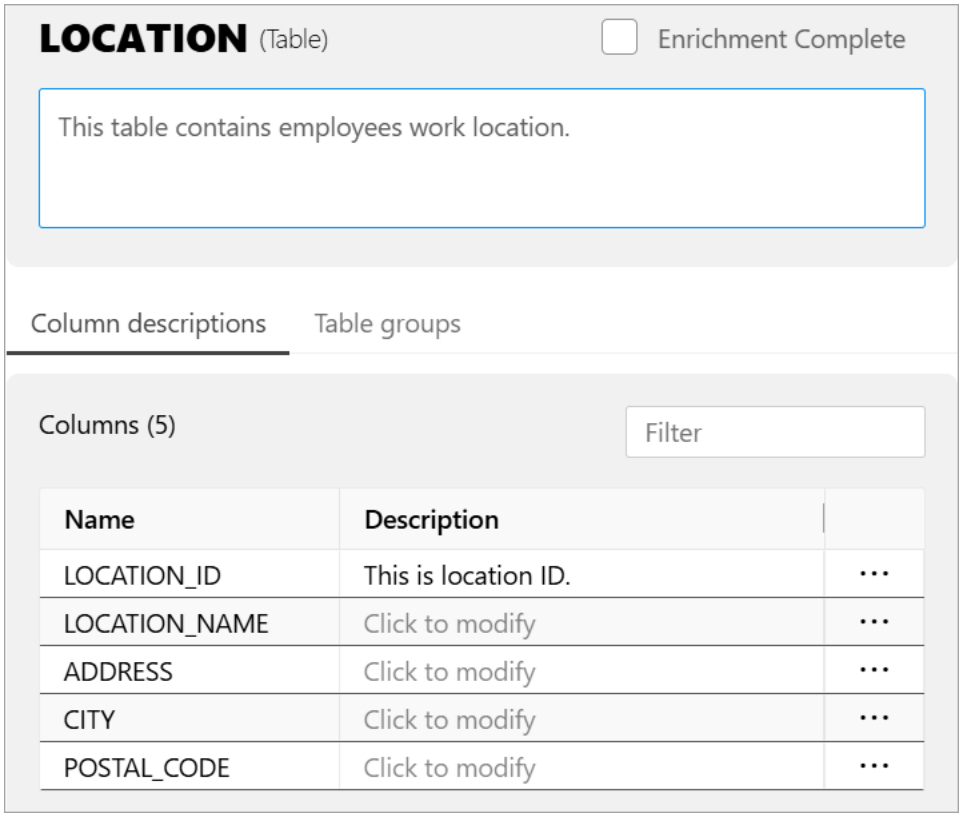
Topics:

- Adding a Column Description
- Adding a Column Annotation

3.4.1 Adding a Column Description

Follow these steps to add a natural language description to a table column.

1. Open a table's view in one of the following ways:
   - Click a table's name in the **Connections** panel, under the **AI Enrichment** node.
   - In the **Table group** section of the AI Enrichment dashboard, click a group's name. In the resulting view, click a table's name from the **Tables** section.
2. Review the **Columns** section to identify any missing descriptions.



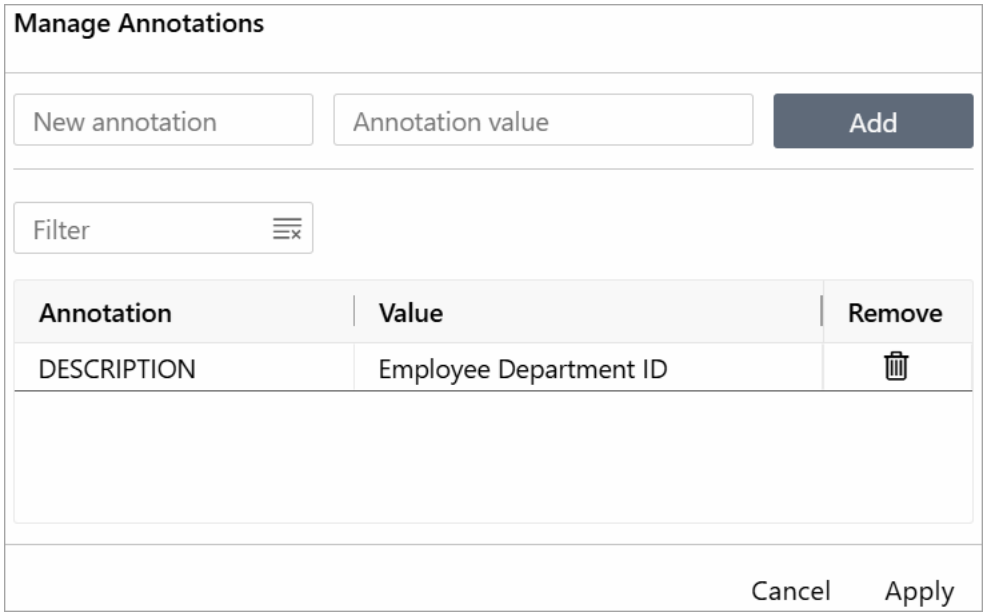
3. To add a description to a column, click its **Description** cell.
4. In the **Set description** panel that appears, enter a description and click **Save**.

3.4.2 Adding a Column Annotation

Follow these steps to add or edit annotations to a table column.

1. Open a table's view in one of the following ways:
   - Click a table's name in the **Connections** panel, under the **AI Enrichment** node.
   - In the **Table group** section of the AI Enrichment dashboard, click a group's name. In the resulting view, click a table's name from the **Tables** section.
2. In the **Columns** section, click **...** for the target column.

   The **Manage Annotations** panel opens.



3. To define an annotation, enter a key-value pair in the **New annotation** and **Annotation value** fields, respectively.
4. Click **Add**, and then click **Apply**.

The new annotation is added to the column. You can also edit an existing annotation from the **Manage Annotations** panel.