Oracle® Cloud

SQL Dialogs in Digital Assistant

F51479-02
September 2022

Oracle Cloud SQL Dialogs in Digital Assistant,

F51479-02

Primary Author: Chris Kutler

Welcome to *SQL Dialogs in Digital Assistant*.

Contents

Abstract

This document describes how to build SQL Dialogs, which are skills that can translate a user's natural language utterances into SQL queries and return a response to the user.

Preface

Welcome to *SQL Dialogs in Digital Assistant*.

Audience

SQL Dialogs in Digital Assistant is intended for developers who want to develop digital assistants with skills that can translate a user's natural language utterances into SQL queries, send the queries to a backend data source, and display the response.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

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.

Preface

Welcome to *SQL Dialogs in Digital Assistant*.

Audience

SQL Dialogs in Digital Assistant is intended for developers who want to develop digital assistants with skills that can translate a user's natural language utterances into SQL queries, send the queries to a backend data source, and display the response.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

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

Basic Concepts

SQL Dialogs are skills that can translate a user's natural language utterances into SQL queries, send the queries to a backend data source, and display the response.

When writing skills that provide database information to end users, developers typically need to define the use cases, write custom components to retrieve the data, create intents for the use cases, map user utterances to intents, and write the dialog flow to handle every intent. With SQL skills, you don't need to do these steps. Instead, you map the users' mental models of the data to the physical data source, and the skill uses the map to dynamically generate SQL from natural language utterances.

For example, users might know that employees belong to departments that are in various locations, and they have employee IDs, job titles, hire dates, and sometimes commissions. Given their mental model, they can retrieve the data by asking the skill "What is James Smith's job?", "When was James Smith hired?", "How many employees are in New York?", and "Who has the highest commission?

This version of SQL Dialogs supports integration with Oracle database services, such as Oracle Enterprise Database Service.

> **Note:**
>
> The current version doesn't support more than one language per skill.

How SQL Dialogs Work

To implement a SQL Dialogs, you create a visual dialog skill and import information about the physical model (database schema) from the data service. Oracle Digital Assistant uses this information to create a query entity for each table you selected to import (the logical model). The query entities contain attributes that model the table columns.

If a table in the physical model has a foreign key, then the query entity as an attribute that links to the related query entity. For example, if an `Emp` table has a foreign key to the `Dept`table, then the `Emp` query entity has a `dept` attribute, which links to the `Dept` entity.

As soon as you create the query entities and identify their primary keys, you can train the skill and it's ready to use in a rudimentary way. That is, you can use free form utterances, but, for now, the query must use the exact entity and attribute primary names, which are initially derived from the physical model's names (the canonical names). This will change as you enhance the logical model to more closely reflect natural language.

To enable the end users to inquire about the data by using natural language, you map end-user terminology to the physical model by changing the primary names and adding synonyms for both the query entities and their attributes. For example, you might change the primary name for the `Emp` table to "employee" and add the "staff member" synonym. Adding primary names and synonyms is one of the ways that you *train* the natural language parser (NLP) to resolve utterances into *Oracle meaning representation query language* (OMRQL) queries.

OMRQL queries are like SQL queries but are based on the canonical names of the object models (the query entities). For example, if you change the primary name for `empno` to "employee number" then "what is Joe Smith's employee number" resolves to SELECT empno FROM emp WHERE ename = 'Joe Smith'. To even further improve the natural language processor (NLP) resolution, you also can associate the attributes with value lists, which are automatically populated from the data service upon creation.

For example, let's say that you import `Emp` and `Dept` tables from a data service, which results in `Emp` and `Dept` query entities. Immediately after you import the tables and train the skill, you can query the query entities using utterances like following:

```
Show all Emp in dept 10
```

After you change the primary names for the entities and attributes to more natural language terms, such as `Employees` for the entity and `department` for the attribute, you can use utterances like this one:

```
Show all the employees in department 10
```

You can further add synonyms to model all the ways people typically refer to the entity or attribute. For example, you might add the synonyms `district` and `territory` for `department` so that the NLP recognizes this utterance:

```
Show all employees in district 10
```

With more complex utterances, you can teach the skill how to resolve the queries to OMRQL by adding custom data that associates the utterances with specific OMRQL statements.

When the skill outputs a query result, it lets the user give a thumbs up or thumbs down to indicate whether the result is correct. The Insights page shows the thumbs up (correct queries) and thumbs down (incorrect queries) counts so you can see how well the skill is doing.

Supported Queries

Users can ask questions relating to the database and receive the relevant information. Using the example of an employees/departments database, this table shows the kinds of queries that a user can make.

Category	Behavior	Examples
Request an entity	Returns a table with all the rows in the table that correspond to the request. When the attributes aren't named, the columns are the default attributes defined in the logical model.	• show me all the employees • show details about each employee

Category	Behavior	Examples
Request an attribute	Returns a table with the requested attribute as well as the minimum attributes of the entity that the attribute belongs to.	• salary of employees • what is the salary of all employees?
Request distinct attribute	Returns a table with only the unique values of the given attribute.	• Show the **different** jobs of employees: • what are the **unique** jobs that an employee can have? • return **distinct** jobs
Request an aggregation	Users can choose to request a summary of the information in the database.	• **how many** employees • what is the **total** salary of all employees • what is the **average** salary of employees • what is the **lowest** salary of employees • what is the **maximum** salary of employees
Filter by attribute equal to a value	This can be used for all attributes.	• show me employees whose hire date is 10 Dec 2020 • who are the employees with hire date 10 Dec 2020
Filter by implicit value reference	Can be used for values that are present in an associated value list (including synonyms).	• salary of every**clerk** • what's the salary of all employees **who are clerks**
Filter by attribute containing a value	Can be used for both text and numeric attributes	• Employees with name **containing** Jo • employees where the ID **includes** 0153
Filter by value comparison	Compares a numerical attribute to a value.	• Employees whose salary is **less than** 1000 • employees who earn **over** 100000 • employees who earn **at most** 50000 • employees who earn 1000 **or more** in salary • employees who earn **between** 1000 and 1500 in salary
Filter by aggregate	Returns only the attributes that satisfy criteria based on an aggregation functions.	• Show jobs where average salary is more than 70000

Category	Behavior	Examples
Combined filters	Users can combine filters with either • AND: both conditions are true • OR: one of the conditions is true You can't have more than 2 filters. For example, you can't ask "Employees whose first name is John and belongs to the Sales dept and their salary is more than 70000"	• employees whose job is clerk **and** salary is above 1000 • employees who earn a salary above 100000 **or** commission above 300
Filter or group across entities	Users can filter or group by attributes from linked entities.	• salary of employees located in New York City • average salary per department name • Department names with average salary above 70000
Order by an attribute	Returns the rows sorted by the specified attribute.	• show employees sorted by their names • employees ordered by name from Z to A
Limit the number of rows	Users can order and restrict the number of rows returned . If you don't specify the attribute to use for comparison, then the entity's **Measure By** value determines which attribute to sort on. See Define Query Rules. However the NLU doesn't always predict ORDER BY in the OMRQL for these types of queries.	• what are the 10 highest salaries of all employees • employee with the lowest salary
Group by attributes	Users can group the results based on attributes, and then request summary statistics for each group.	• how many employees per job
Queries with dates	The queries can contain absolute and relative dates and times as well as date intervals. They also can include natural language terms that are associated with date operators. For example, you can use these natural terms to refer to the relative future and past: • **Past (<):** before, earlier than • **Future (>):** after,later than	• what invoices are due after 11/10/2022 • what invoices are due in the next 10 days • what invoices were paid last year • what installments are due after 5th August 3 pm • which vulnerabilities were discovered yesterday? • which vulnerabilities were discovered last Saturday?

Here are examples of queries that aren't currently supported:

- User pronouns. For example: what is my salary?
- Implicit date and time, date, or time values. For example: what is the next event?
- Date durations which need to be coerced to dates. For example: due in 2 days.
- Date intervals with both date and time values. "show meetings between 9 am July 5 and 5 pm July 6".
- A date interval where the start or end date does not contain all the information. For example "show all invoices due between July and December 2022" (the start date is missing the year).
- A date utterance where the natural language phrase for a date operator can't be resolved correctly. For example, for "what installments have due date more than 40 days ago", "more than" implies ">" but "more than 40 days ago" implies "<".
- Resolving ambiguous dates based on the context. For example: "invoices that were due on Wednesday" (ambiguious) versus "invoices that were due last Wednesday" (not ambiguous). All ambiguous dates are resolved based on the **Resolve Date & Time as** setting (Default, Past, Future, Nearest) for the DATE_TIME entity. If **Resolve Date & Time as** is set to Future, then "invoices that were due on Wednesday" resolves to the upcoming Wednesday.
- Limited performance for domain specific entity/attribute synonyms. When two words are synonyms in the context of the domain and only in the context of the domain, then the model may not accurately map the synonym to the correct attribute. For example, in the mobile phone domain, IP address and device are synonymous, but that's not true outside the domain.
- Implicit distinct. Show the cities of all employees (currently shows multiple rows with repeating cities).
- Implicit aggregations. For example: how much do we pay to all employees in Accounting dept?
- Implying an attribute by referring only to the entity. For example: Show invoices containing 1234.
- Group by attributes across multiple entities with aggregations. For example: Show number of employees for every department location.
- Group by attributes across multiple entities with aggregations. For example: Show number of employees for every department location.
- Negation. For example: Which employees are not in the accounting department?.
- Disambiguation. For example: Show amount for all invoices (not clear if invoice amount or gross amount).
- Implicit attribute reference for values not in the value list entity. For example: Show salary of all employees in HM (where HM is not in the value list for department names, nor is it a synonym for one of the values in the list).
- Group by attributes across multiple entities with aggregations. For example: Show number of employees for every department location.
- Queries in any language other than English.
- Limited resolution of word names for numbers. You can use the following word names for numbers. All other word names are not supported.
  - one, two, three, four, five, six, seven, eight, nine
  - ten

  - twenty
  - fifty
- Typographical errors. For example: Employees in acounting department (accounting is misspelled).
- Fuzzy matching for value list entities. For example: Show invoices for Amazon (where the value list contains "Amazon LLC").
- Yes/no question. For example: Do we have any employees who don't belong to any department?
- Query containing "starts with" or "ends with". For example: Employees whose name ends with Smith.
- Subqueries. For example: Show employees whose salaries are more than the highest salary of the Sales dept.
- Set operations. There is no support for queries which require the use of INTERSECT, UNION, EXCEPT, or NONE.
- Arithmetic operations. For example: How much money is left for project VMON.
- The EXISTS and NOT EXISTS keywords.
- Order-by superlatives. NLU does not consistently predict the order by clause in the OMRQL for superlative queries.. For example: Show top 10 employees.

2
Get Started

You build a SQL Dialogs skill differently than regular skills. To enable the skill to understand and respond to natural language utterances, you create a logical model and you teach that model by using natural language terms to describe describing the physical model.

SQL Dialogs Workflow

How you build a SQL Dialogs skill differs from regular skills. Here are the major steps to build a SQL Dialogs skill and train it so that people can use natural language to query the data services.

The participants in the following steps are the skill developer, service administrator, database expert, and AI trainer.

- The **skill developer** gathers the skill requirements (user personas, use cases, and tasks) and training corpus (sample user utterances), and creates the skill. The developer also helps define how the results are displayed. This person is sometimes referred to as the conversation designer.
- The **service administrator** adds a connection to the data service.
- The **database expert** analyzes the skill requirements and training corpus to identify the tables and attributes that provide the answers. The expert then creates the base logical model by importing information from the physical model into the skill. The expert also assists the skill developer and AI trainer with tasks such as adding SQL-expression based attributes, associating attributes with value lists uploaded from tables, and performing custom training.
- The **AI trainer** adds primary names and synonyms teach the natural language parser how to understand the natural language utterances. For utterances that the skill can't translate to OMRL, the AI trainer adds custom training to teach the natural language parser how to understand these utterances. The trainer continually monitors and tests the skill to increase the accuracy of translating natural language into database queries.

To help illustrate the workflow, we'll use an example accounts payable data service with the following tables. For brevity, we just show the columns mentioned in this topic.

Table	Columns
invoices	• invoice_num • invoice_date • pmt_status_flag • invoice_amount • vendor
payment_schedules	• invoice_num • due_date • amount_remaining
suppliers	• vendor_num • vendor_name

1. **Define the Requirements:** The skill developer gathers the use cases and tasks that the SQL Dialogs skill is expected to support. For example, an accounts payable department might have this use case:
   - **Use Case:** *Pay* all invoices with outstanding balances that are due within 30 days so that we can avoid penalties.
     - **Task:** Find all *unapproved* invoices that are due within 30 days so that we can *approve* them in time.
     - **Task:** Find all *outstanding* approved invoices due within 30 days so that we can *schedule to pay* them in time.

   As part of this requirements phase, the skill developer compiles a representative list of the different ways people ask for this information. This list serves as the set of example utterances that the AI trainer uses for the *training corpus*.
2. **Set Up the Skill:** The service administrator, skill developer, and database expert work together to set up the basic skill.
   a. **Integrate with the Service:** The **service administrator** creates a connection from Oracle Digital Assistant to the data service. See Connect to the Data Service.
   b. **Create the SQL Dialogs Skill:** The **skill developer** creates the SQL Dialogs skill, ensuring that the dialog mode is set to **Visual** in the **Create Skill** dialog. See Create the SQL Dialog Skill.
   c. **Import the Schema:** The **database expert** identifies the tables and fields that are necessary to support the use cases and then, from the skill's **Entities** page, imports them from the data service as described in Create Query Entities to Model the Data Service. This creates a base logical model that contains a *query entity* for each imported table.

      In our example, the database expert imports the `invoices`, `payment_schedules`, and `vendors`, tables.

      At this point, the skill is ready for use with limited functionality. For the base logical model, the entity and attribute names are derived from the physical model's table and field names. For example, if the table name is `payment_schedules`, then the primary name is `payment schedules`. The AI trainer can **test queries** from the **Entities** page or use the conversation tester (**Preview**) to try out the SQL functionality.

      In our example data service, they can use test utterances such as "show invoices with pmt status flag N", "show invoice num 17445", or "show payment schedules with due date before 2022-08-30".
3. **Train:** Add training data through primary names, synonyms, value lists, and natural language queries mapped to OMRQL.
   a. **Add Natural Language Terminology:** To help associate natural language phrases with the underlying data structure, the **AI trainer** teaches the skill the different ways that end users refer to the entities and attributes. That is, it teaches the names that people will use in their natural language utterances. The trainer starts by analyzing the phrases that the skill developer gathered to identify the utterances that the skill should handle (the training corpus). Additionally, they can consult a thesaurus for synonyms and crowd-source for similar phrasing. Then the AI trainer records the equivalent terms by changing the primary names and adding synonyms. See Provide Training Data Through Names and Synonyms.

In our example, one of the utterances gathered during the requirements phase is "Give me list of invoices with an outstanding balance greater than zero." The attribute that contains the balance is `amount remaining`, so the AI trainer adds the synonym `outstanding balance` to that attribute.

b. **Associate with Value Lists:** To improve accuracy, the **AI trainer** can, where appropriate, create value lists that contain sample values from the data service. The skill automatically associates the lists with their respective attributes, which helps the natural language parser understand the kinds of values those attributes can hold. See Provide Training Data Through Value Lists.

   In our example, they associate a `vendor_name` attribute with a value list retrieved from the data service. If the value list includes "Seven Corporation" and a user asks "show summary flag for Seven Corporation", the NLP will deduce that Seven Corporation is a vendor name.

c. **Map Complex Queries:** In cases where the skill isn't able to translate a valid utterance into OMRQL, the **AI trainer** adds that utterance to the query entity dataset and maps it to OMRQL as described in Provide Training Data Through Utterances. For example, you can map "show unpaid invoices" to `SELECT * payment_schedules WHERE payment_status_flag = 'Y'`.

d. **Train the NLP Model:** To incorporate training data into the NLP model, the **AI trainer** clicks the **Train** icon and clicks **Submit**.

4. **Configure How Information is Displayed:** The **database expert** and **skill developer** work together to fine tune how each entity's results are displayed, as described in Configure How Entity Rows and Attributes are Displayed. For example, they can set an entity's default sort order, set the minimum attributes to include in the output, and add attributes that display derived or calculated data.

   In our example, they might set both the invoice entity's default sort order and minimum attributes to `invoice_num`, and set the default attributes to `invoice_num`, `invoice_date`, `pmt_status_flag`, and `invoice_amount`. They might also add an `age` attribute that is calculated using the difference between today's date and the invoice date.

5. **Configure Query Rules:** The **database expert** and **AI trainer** work together to set the query rules, such as when to use partial matching and what attribute to use for measuring when someone asks to compare rows without specifying an attribute to compare with. See Define Query Rules.

   In our example, they anticipate end users asking for the 10 most payments to make, so they'll configure the `payment schedules` entity to use `due_date` for comparisons, and they'll invert comparisons for that attribute so that earlier dates rank higher than later dates.

6. **Test and Repair:** The **AI trainer** uses the query tester from the **Entities** page to verify that the test utterances resolve to the desired OMRQL, and that the skill can translate the OMRQL to executable SQL. When the query tester can't translate the OMRQL to SQL, it requests training data. In many cases, you can resolve this by adding the utterance to the *query entities dataset* and associating it with an OMRQL statement. See Test and Repair.

7. **Monitor and Improve:** After the skill enters the beta testing phase and beyond, the **AI trainer**, **skill developer**, project manager, and stakeholders can continually monitor Insights data to see how well the skill is performing and to identify areas for improvement. See Monitor and Improve.

Connect to the Data Service

Before you can access a data service from any SQL Dialogs skill, you need to add a *data service integration* that enables Oracle Digital Assistant to access the data service. You only need one integration per data service.

For this release, integrations have been tested with Oracle Database Cloud Service Enterprise Edition 12c and 19c Oracle Autonomous Transaction Processing.

> **Note:**
>
> After you create the service, you can't change it. Should the password change, you'll need to delete and recreate the data service integration.

1. In Digital Assistant, click ☰ to open the side menu, click **Settings**, click **Additional Services**, and click the **Data** tab.
2. Click **+ Add Service**.
3. In the **New Data Service** dialog, provide this basic information:

Field Name	Description
Name	A unique name for the service.
Data Service Description	An optional description of the data service integration such as a description of the database or the purpose.
Authentication Type	Your database administrator will tell you whether to select **Default**, **Kerberos**, or **OS**.
Role	Currently, there is one choice; **Default**, which is the user's default role.
User Name	Ask your database administrator for the user name and password that gives access to the tables that the skill developers need to create the composite entities for their SQL Dialogs skill as described in Create Query Entities to Model the Data Service.
Password	The user's password. Note that for Oracle Digital Assistant integration, a password must be at least 14 characters and no more than 30 characters, and it must contain at least one upper case character, one lowercase character and one number. It also can't start with a digit.

4. Click **Continue** to configure end-user authentication.
5. If your data service is configured for role-based access, then select **End-User Authentication is required**, select the authentication service that you configured in **Settings > Authentication Services**, and then select the end-user identifier. If you choose **Custom** then provide the custom expression.
6. Click **Continue** to add the connection details.

7. On the **Connection Details** page, select **Basic** or **Cloud Wallet Connection** for the connection type.
8. If the connection type is **Basic**, enter these values, which you can get from the database administrator:

Field Name	Description
Host Name	Enter the host for the data service. Leave out the `https://` prefix. For example: `example.com`.
Port	The port that allows client connections to the database.
Service Identifier	Do one of the following: • Select **SID** and enter the Oracle system identifier of the database *instance*. • Select **Service Name** and enter the service name for the *database*.

9. If the connection type is **Cloud Wallet Connection**, enter these values, which you can get from the database administrator:

Field Name	Description
Wallet File	Find and select the Cloud Wallet file that contains the client credentials or drag and drop it into the field.
Wallet Password	Enter the password that was provided when the wallet file was downloaded. Note that for Oracle Digital Assistant integration, a wallet password must be at least 15 characters and no more than 30 characters, and it must contain at least one upper case character, one lowercase character, one special character, and one number. It also can't start with a digit.
Service	Select the name of the database service.

10. Click **Add Service**.

    You now can import the database schema into a skill to create query entities, which enable users to query the database using natural language.

Create the SQL Dialog Skill

To create a SQL Dialog skill, you simply create a skill with the **Dialog mode** set to **Visual**.

> **Note:**
>
> This version of Oracle Digital Assistant does not support adding intents to the skill, nor does it support adding SQL Dialog skills to a digital assistant.

Create Query Entities to Model the Data Service

To enable data service queries in a SQL Dialogs skill, you import information about a data service's physical model (the tables and columns) to create a base *logical model*. During the import, the skill adds *query entities* to the logical model, where each query entity represents a physical table.

When you train your skill, it uses the information from the query entities to build a model for the natural language parser, which enables the skill to translate user utterances into *OMRQL*. OMRQL is a query language that's similar to SQL but is based on object models, which, in this case, are the query entities.

Before you begin, you need the following:

- A skill that was created using **Visual** mode.

  > **Note:**
  >
  > In this version of Oracle Digital Assistant, the skill can't have intents.

- A data service integration for connecting to the data service as described in Connect to the Data Service.

To create query entities for the desired tables in your data service:

1. From the **Entities** page, click **More**, and then select **Import from Data Service**.

   The **Import Query Entities** dialog appears.

2. Select the data service, and then select the tables and attributes that you want to use in the skill.

3. Click **Import**.

   The skill adds query entities for the selected tables. It sets the entity and attribute primary names based on the canonical names. For example, if the canonical name is "invoice_num", the primary name will be "invoice num".

4. For each query entity that was added, select the entity, click the **Configuration** tab, and verify that the primary key is set.

At this point, you can test the queries using the primary names for the entities and attributes, such as "show invoices where invoice num is 12345". But first, you must click 

, and then, after it completes, you can click **Test Queries** to try out utterances, or click **Preview** to test in the conversation tester.

Because you are working with a *minimal* SQL dialog skill, which doesn't have custom training data, you can train with either Trainer Ht or Trainer Tm. However, after you add custom training data, Trainer Tm produces more accurate results.

Your next step is to teach the skill how the end users refer to the entities and attributes. See Train the Skill to Convert Natural Language Utterances into SQL.

Train the Skill to Convert Natural Language Utterances into SQL

As an AI trainer, your job is to enable the natural language parser to translate natural language utterances such as "how many invoices have a due date before 12/15/22" into an OMRQL query for retrieving the answer from the underlying data source (the physical model). You do this by building an intuitive logical model of the data that closely reflects natural language.

After the logical model is created by importing from the data source, you use primary names, synonyms, value lists, and utterances to help the skill's natural language parser associate natural language phrases with the physical model's tables and columns.

- To teach the skill about the different ways that people refer to the objects, you add primary names and synonyms as described in Provide Training Data Through Names and Synonyms. For example, you might want to teach the skill that people use "invoice number" to refer to the `invoice_num` column, and you might also want to add "invoice no" and "ref number" as synonyms.
- To help the skill identify attribute values in an utterance, you create sample value lists and associate them with attributes as described in Provide Training Data Through Value Lists. For example, you might create a value list that contains actual payment statuses and associate the list with the invoice's `payment status` attribute.
- When the skill isn't able to correctly translate an utterance into OMRQL, you can add an utterance-to-OMRQL mapping to the query entity dataset as described in Provide Training Data Through Utterances and Test and Repair. You can also add utterances to help the skill know when to route an utterance to the flow that processes it as an SQL execution (that is, translates to OMRQL and then sends an SQL query to the data source).

Provide Training Data Through Names and Synonyms

To help a SQL Dialogs skill associate natural language phrases with the underlying data structure (physical model), start by taking the identified utterances that the skill should handle (the training corpus), and analyzing them to discover the different ways that end users refer to the entities and attributes.

For example, suppose that you have these utterances in your training corpus:

- Show me the invoices with outstanding balances greater than zero.
- What's the amount due for reference 12656?

Here, you see that people use "outstanding balance" and "amount due" to refer to the `amount_remaining` column. You also see that "reference" is one way people refer to `invoice_num`.

In addition to the training corpus, you also might want to crowd-source utterances from your target users to get more phrases and analyze those as well.

After you compile your list of the ways people refer to the entities and attributes, pick which term you want to use for the primary name of each entity and attribute. They should be names that are closest to the most common usages. When you choose the name, consider that the out-of-the-box NLP model won't likely understand domain-specific relationships. For example, it won't automatically understand that *invoice number* and *reference* refer to the same thing. Because *invoice number* is commonly used and also is closest to other

commonly used terms such as *invoice no* and *bill number*, you would make it the primary name.

Treat the rest of the terms as synonyms. In the above example, you would add *reference*, *invoice no*, and *bill number* to the synonym list.

Note that the primary name is what the skill uses for the result column headers and labels.

Using your list, you create the training data in the **Entities** page.

- To set the entity's **Primary Name** and **Synonyms**, open the entity's **Configuration** tab.
- To set the attribute's **Primary Name** and **Synonyms**, open the attribute's **Natural Language** tab.

> **Note:**
>
> When processing utterances, the natural language parser doesn't consider the physical model's canonical names, that is, it doesn't look at table and column names. It only uses the natural language mappings that you define using names and synonyms (the logical model).

Provide Training Data Through Value Lists

You can improve the natural language parser's accuracy by associating attributes with value lists or dynamic entities. This helps the parser identify an attribute based on its known values. You use **Referenced Entity** on the attribute's **General Information** tab to associate the attribute with the reference entity's values. For value list entities, you can automatically create the entity, import the data service's values, and associate it as a referenced entity all in one step.

When deciding whether to use a value list or a dynamic entity to store the values, consider whether the entity is *open* or *closed*.

- An *open* list is one that is infinite or dynamic (or both). For open lists, consider creating and maintaining a dynamic entity instead of a value list. If you choose to use a value list, then you should curate the list to make sure that it at least contains the most commonly used values. For example, for a vendors list that most likely grows over time, you'll want the list to include your most frequently used vendors. This is because queries about a vendor without using the word "vendor", such as "show the summary flag for Seven Corporation", won't match if that value isn't in the value list. You thus increase the frequency of correct resolutions by at least including the most frequently used values.
- A *closed* list is a static finite list. These are ideal for value list entities.

For both value lists and dynamic entities, add both plural and singular versions of synonyms for each entity value to indicate the ways that end users will refer to the value. This is especially important when the list contains values that end users don't typically use. Take, for example, this list of valid payment statuses. Users will be much more likely to use words like paid, unpaid, and partially paid than to use Y, N, and P. Adding these words as synonyms helps to insure that the NLP recognizes that the users are referring to the payment status attribute.

Payment Status Values	Synonyms
Y	paid
N	unpaid, not paid
P	partial, partially paid, unpaid

When a term describes more than one entity value, then add that term as a synonym for each one. For example, both `N` and `P` indicate that the invoice is unpaid. If you add "unpaid" as a synonym for both statuses, then "show unpaid invoices" will retrieve invoices with a payment_status value of `N` or `P`.

For dynamic entities, you create the entity and then use **Referenced Entity** on the attribute's **General Information** tab to associate the attribute with the list.

For value lists, you can create a value list from the data service and associate with an entity by following these steps:

1. On the **Entities** page, edit the attribute and go to the **General Information** tab.
2. Select **Entity** from the **Type** drop-down list.
3. Click **If the desired entity doesn't exist, you can generate a value-list entity based on the background mapping by clicking here**. The value list is created and populated from the data service, and the **Referenced Entity** points to the new value list.

   Note that it won't extract more than 100 values but you can add more values manually. Another option is to upload the values into a value list entity as described in "Import Value List Entities from a CSV File" in *Oracle Cloud Using Oracle Digital Assistant* and then associate the attribute with that value list entity.
4. Open the entity and add plural and singular versions of synonyms for the values, if there are any.
5. Click **Apply** to save your changes.

> **Note:**
>
> If any value in the data service's physical table ends with a period, a question mark, or spaces, then those characters are not included in the value list because they are not allowed for canonical names. The value list's **Fuzzy Matches** switch is set to On automatically. In this version, when this happens, you should switch it to Off.

Provide Training Data Through Utterances

As an AI trainer, you'll encounter natural languages utterances that the skill can't translate to OMRQL. For example, the model may not be able to handle domain-specific synonyms that don't seem to be closely related to the primary name. Another example is when the model is not able to distinguish between two similar entities. When this happens, you can use the query entities dataset to teach the skill how to correctly parse the utterance into OMRQL.

Adding to the dataset is often referred to as *custom training*. You use custom training to teach the model to associate words and phrases with attributes, entities, and OMRQL keywords in the context of a full utterance by mapping the utterance to OMRQL.

For each scenario that you are fixing, start with 20 utterances and add more as needed. Because too many examples might cause the model to over predict attributes and operators, you should focus on a smaller set of diverse utterances rather than a large set of similar, lesser quality ones.

All the values in the OMRQL statement must exactly match the database value and format. Take, for example, the utterance "who is the employee whose name is Jones", If the database values for the name attribute are all capital letters, then the name value must also be all capital letters. That is "SELECT * FROM Emp WHERE name = 'JONES'".

When the utterance that you are mapping uses a synonym for the actual database value, then that synonym must be defined for the value in a value list, and the OMRQL must use the actual database value. For example, if the utterance is "show the department whose location is the big apple", then "big apple" must be defined in the dept_loc value list as a synonym for the value "NEW YORK", and the OMRQL must be "SELECT * FROM Dept WHERE loc = 'NEW YORK'".

You can add utterances that contain absolute dates, such as "invoices due on 5 Jan 2022", but don't use utterances with relative dates or dates without the year. For example, if the utterance is "invoices due today", then today's date would be hard-coded into the OMRQL as SELECT * FROM invoices WHERE due_date = '2022-01-01'.

Here are some best practices for custom training utterances:

- **Balance the number of utterances:** Some of the more complex scenarios may need more utterances than the simple ones, but try to balance the number of utterances per scenario.
- **Balance the training of similar attributes and entities:** If you have two similar attributes, and you need to provide custom training data for one of them, then you also should provide the same amount of training data for the other. When the training data concentrates only on one of the similar attributes, then the model might over predict that attribute over its counterpart. The same is true for similar entities. For example, payment currency and invoice currency are similar attributes. If payment currency is over-represented in the training data, the model might predict payment currency even when the utterance asks for invoice currency.

  When you need to teach the model how to distinguish between two similar or closely-related attributes, balance the weighting of importance by providing half the utterances for one attribute and half the utterances for the other.

  Vary the utterances that refer to these similar attributes. For example, here are contrasting pairs of utterances to help the model distinguish between `amount_remaining` and `amount_paid`:

  - tell me the **amount remaining** for approved invoices
  - show us the **amount paid** for approved invoices
  - view total **amount due to be paid** to vendor AAD
  - calculate the total **amount that was paid** to vendor AAD
  - what is the **amount due** on invoices to vendor AAD
  - list the **paid amount** on invoices to vendor AAD
- **Balance the training of values that match primary names or synonyms:** Say, for example, that your model has a `manager` attribute and "manager" is also a

value for the employee `job` attribute. If you want to add "How many managers" to the query entities dataset, then you should balance this training data with utterances that use the `manager` attribute, such as "Who is the manager of employee Adam Smith", as well as utterances that use the manager `job`, such as "Show all managers". That way, the model can learn to differentiate between the two usages. If you don't include examples for both types of usage, then the skill might over predict one usage over the other.

- **Diversify phrases:** The best practices for diverse phrasing for custom data are similar to those for intent utterances:
  - Use full sentences.
  - Use different verbs. For example: view, list, show, tell, and see.
  - Use various synonyms and paraphrases in addition to the entity or attribute name.
  - Use different pronouns. For example: show me, can we see, tell us, I want.
  - Vary the sentence structure. For example, put the attribute value near the beginning, middle, and end of the sentences.
  - If you have utterances with an aggregation, such as `AVG`, then also add utterances with other operators as well.
  - If possible, use different clauses, such as group by and where clauses with AND and OR conditions.
- **Diversify Values:** When you use more than one value in your scenario's utterances, the model is better able to recognize different values. Include values with different word lengths. Include some values with special characters such as '/' and "-". Include a few values with special keywords such as 'and'.
- **Include a mix of known and unknown values.** For value-list attributes, use a representative set of attribute values (but not all) to train that value-list matches are important signals. Also, for value lists that aren't *closed* lists, include values that aren't in the value list to teach it to also associate particular phrasings with the attribute.

To add a mapped utterance to the query entities dataset (custom training data):

1. If the **Train** button has a red badge, click 

.

   Note that when the skill doesn't have custom training data, you can train with either Trainer Ht or Trainer Tm. However, after you add custom training data, Trainer Tm produces more accurate results.
2. In the **Entities** page, go to the **Dataset** tab and click **Query Entities**.
3. Click **Add Utterance**.

   The **Add Utterance to Dataset** dialog displays.
4. Enter the utterance and click **Next**.

   The dialog displays the OMRQL query for the utterance. If it can't translate the utterance into the query, the query will be blank.

   Note that if the skill hasn't been trained, it can't translate the utterance into an OMRQL query.
5. Review the query and correct it if it's wrong.

   For OMRQL keywords and examples, see OMRQL Reference.
6. Click **Add** to add the mapped utterance to the dataset.

Configure How Entity Rows and Attributes are Displayed

Here are the things you can do to control when and how the entity rows and attributes are displayed in the results:

- Define an Entity's Default Sort Order
- Define Which Attributes to Include When Not Specified by the Utterance
- Define Which Attributes to Always Include in the Results
- Add a Custom Attribute

Typically, the database expert and the conversation designer work together on this task, as one has database schema expertise and the other has familiarity with user expectations.

You can test your changes by clicking **Preview** to open the conversation tester and entering an utterance to retrieve the appropriate data.

> **Tip:**
>
> Most of the changes that you make will require natural language parser (NLP) retraining. When you test your changes, if the **Train** icon has a red badge (
>
>), you'll first have to click **Train** and complete the training process.

Define an Entity's Default Sort Order

You can specify a default sort order for the skill to use whenever the user's utterance doesn't specify one. To set the default, go to the entity's **General** tab, click **Add Attribute Order**, select an attribute and select its order (Ascending or Descending). You can continue clicking **Add Attribute Order** to add more attributes to the sort order.

Define Which Attributes to Include When Not Specified by the Utterance

If the utterance doesn't name any attributes, then you probably want the results to include some essential fields. You can use **Default Attributes** in the entity's **Configuration** tab to specify these fields. For example, for an `invoices` entity, you might want to display `invoice_num`, `invoice_date`, and `invoice_amount` when no attributes are named.

Note that you can't add attributes of type query entity to the default attributes list.

Define Which Attributes to Always Include in the Results

When an utterance identifies specific attributes, you might want the result to include not only the requested attributes, but also some context. For example, if someone enters "show invoice amounts", the data won't make sense if it only shows the `invoice_amount` values, and not some identifying context like `invoice_num`. Use

Minimum Attributes on the entity's **Configuration** tab to identify the minimum attributes.

You cannot add attributes of type query entity to the minimum attributes list.

Add a Custom Attribute

You can add your own custom attributes to display additional information, such as derived or calculated values.

1. From the **Attributes** tab on the entity page, click **+ Add Attribute**, and provide a canonical name and type.
2. On the **Natural Language** tab, provide a primary name and optionally add synonyms.
3. On the **Backend Mapping** tab, select **SQL Expression** and add the expression.

If the expression references a column, use the column name from the physical model (database schema) and prepend `${alias}`. For example, for an `invoices` entity, you might add an `amount_to_pay` attribute with the expression `${alias}invoice_amount + ${alias}discount_taken`.

You can use this table to determine what type to use for the attribute:

Type	When to Use	Examples
Number	The values are only numeric and are not restricted to a set list.	Numeric employee ID, invoice amount
Date	The value is a date without a time.	Hire date
Date/time	The value can have both a date and a time.	Departure date and time
Entity	The attribute is associated with a value list entity. Note that if the value list enumerates all the valid values (that is, a closed list) and the values are rarely used in natural language utterances, you should add synonyms for the values in the list.	status (closed), supplier names (open)
String	Use for text that can contain numbers and characters where it doesn't make sense to associate with a value list.	Alpha-numeric invoice number, product description
Query entity	Only use when you need to link to another query entity.	No examples
Boolean	Do not use.	Not applicable

Define Query Rules

Here's how you use an entity's settings on the **Entities** page to control the ways in which end-users can ask about the data and how to evaluate the results.

You can test your changes by clicking **Preview** to open the conversation tester and entering an utterance to retrieve the appropriate data.

> **Tip:**
>
> Some of the changes that you make will require natural language parser (NLP) retraining. When you test your changes, if the **Train** icon has a red badge (

), you'll first have to click **Train** and complete the training process.

- **Identify Which Attribute to Use for Measuring or Comparing:** If the utterance asks to compare entity items to a number or asks to rank the entities using a superlative like *greatest* or *least*, which measurable attribute, if any, should the skill use to perform the comparison? Say, for example, the users ask about the greatest supplier, you might want the skill to use the `rating` attribute for comparisons. To specify which attribute to use for measuring or comparing, go to the entity's **General** tab and select the attribute from the **Measure By** drop-down. If the ranking is opposite of numerical order, such as 5 being better than 1, then you should also set the attribute's **Invert Comparison** to true on its **General Information** tab.
- **Specify How to Compare Measurable Attributes:** By default, measurable attribute values are compared using numerical order, where 1 is less than 5. However, sometimes it is more appropriate to invert the comparison where 1 is better than 5. For example, when looking at race results, the 5 best times are the lowest values in the results. To invert comparisons for an attribute, set the attribute's **Invert Comparison** to true on its **General Information** tab. Note that this setting also affects the attribute's sort order.
- **Allow Partial Matching for Strings:** If you expect that users will frequently leave out leading or trailing characters or values, such as "manager" instead of "department manager", then consider enabling partial matching. When partial matching is turned on, the generated SQL "where clause" uses `upper (<column-name>) LIKE UPPER(%<string>%)` instead of `= <string>`. You can enable partial matching on the attribute's **General Information** tab. Note that the partial matching behavior for entity attributes is different from fuzzy matching behavior for value lists.

Enable Natural Language Queries for Denormalized Columns

If you have a denormalized attribute with a name that uses a pattern to identify the attributes that the column represents, such as PTD_LBR_CST, you can make the denormalized attribute understandable to the natural language model by mapping a normalized entity to it through the use of a column expansion backend mapping.

For example, say that you have a costToSales query entity with the attributes PTD_LBR_CST, QTD_LBR_CST, YTD_LBR_CST, PTD_SUB_CST, QTD_SUB_CST, YTD_SUB_CST.

To enable the skill to associate natural language queries with these attributes, you create a Cost query entity that contains the uniquely-identifying attributes, such as project_num, plus period, type, and cost. The period and type attributes are of type entity and reference the period (PTD, QTD, YTD) and type (LBR, SUB) value lists. The cost attribute's backend mapping is a column expansion with the expression "$

{period}_${type}_CST". The final step is to add the cost attribute to the costToSales entity, which references the Cost query entity to link the two entities.

When the query is "what are my YTD labor costs", the backend column expansion mapping tells the skill to retrieve the value from the YTD_LBR_CST attribute, which is in the costToSales entity (assuming that the necessary primary names and synonyms are set).

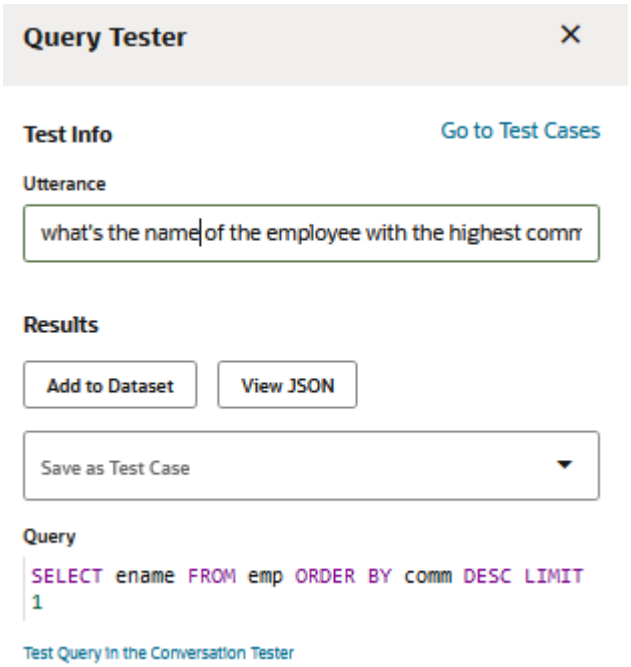
Test and Repair

As you define and add training data to your entities and attributes through names, synonyms, value lists, and the query entities dataset, you'll want to test how well the training data helps the natural language translates the end user's utterances into SQL queries.

> **Tip:**
>
> If the Train icon has a red badge (

), you'll have to click **Train** and complete the training process before you can test the utterances.

The **Entities** page has a **Test Queries** link that opens the query tester for trying out the your use-case utterances. In the tester, you can enter your test utterance and review the OMRQL query that the skill generated.



If the OMRQL query isn't correct, then you'll need to repair the skill by using the appropriate fix:

- Add synonyms for an entity or attribute. See Provide Training Data Through Names and Synonyms.
- Associate an attribute with a value list or add items to a value list. See Provide Training Data Through Value Lists.

- Add the utterance and corrected OMRQL to the query entities dataset to teach the model to associate words and phrases with attributes, entities, and OMRQL keywords in the context of a full utterance. See Provide Training Data Through Utterances.

If the query tester reports that there's *insufficient training data*, first check if you introduced a typo or your query is too vague. These can't be resolved by training. Otherwise, you might be able to resolve insufficient training data by adding the utterance to the query entities dataset. Here are some examples of the kinds of insufficient training data issues that you might be able to resolve by adding to the dataset.

- **Attribute confusion:** For example, does status refer to payment status or approval status.
- **Attribute-Value confusion:** For example, how many managers are there (is it referring to the `manager` attribute's value or the employee's job value?).
- **Search values that are also keywords or operators:** For example, distinguishing the synonym "total" from the operator `SUM`.

If the OMRQL is valid, you can test how the skill translates the OMRQL to SQL by clicking **Click to test this in the conversation tester**. The **Conversation Tester** displays with the results.

In the conversation tester, you can see the OMRQL and SQL statements on the **SQL Dialogs** tab.

Monitor and Improve

The skill's **Insights** page provides several metrics you can use to measure how well your SQL Dialog skill is performing and to determine where to make improvements.

As a **business analyst**, you might be interested in these data query metrics on the **Overview** tab:

- **Performance: Conversations Trend by Status (Line)** shows the number of conversations over a given time period and whether the traffic is tending up, down, or sideways.

- The ratio between **Correct Queries** and **Incorrect Queries** indicates how satisfied the bot users are with the accuracy of translating utterances to SQL queries.
- The ratio between **Completed** and **Incomplete** conversations shows the extent to which technical issues impact the users' experiences.
- The ratio between **Total Conversations** and **Unresolved (OOD/OOS) Queries** helps measure the extent to which the skill meets the end users expectations.
- Both **Conversations Trend by Type** and the ratio between **Total Conversations** and **Data Queries Conversations** show the proportion of utterances that are SQL queries.
- **Data Query Entities** show which entities are queried the most.

As an **AI trainer**, you can examine the user messages on the **Conversations** tab to discover areas for improvement. For example, you can review these user messages:

- **Type: Intent, Outcome: Incomplete** user messages indicate problems with translating the utterance to an SQL query. Often, you can fix these issues by adding synonyms or, for more complex queries, adding mapped utterances to the query entities dataset. Note that you also can see these messages by selecting **System Handled Errors** from the **Errors** drop-down list.
- **Type: Intent, Intent: unresolvedIntent** user messages indicate both out of scope utterances and utterances that the skill doesn't recognize as a data query utterance. For the utterances that are valid data queries but the skill doesn't recognize as such, you often can fix the problems by adding synonyms or mapping the utterances to OMRQL in the query dataset.
- **Type: Data Query, Entities** shows the user messages by query entity.
- **Type: Data Query, Outcome: Incorrect** shows the messages that the users thought returned incorrect results. You should verify that the results are incorrect, and, if so, add synonyms, value lists, and query dataset entries as appropriate.

3
OMRQL Reference

Here are the keywords that you can use when you define OMRQL queries for the utterances that you add to the query entities dataset. Note that you use the canonical names and not primary names and synonyms

Component	OMRQL Keywords	OMRQL Example	Constraints
Basic Components	• SELECT • * • FROM	SELECT * FROM Emp	The OMRQL can't name attributes that aren't referenced in the utterance.
Filtering	WHERE Currently, you can't add custom training data that filters by date or datetime.	SELECT * FROM Emp WHERE comm > 0	None.
Linking Entities (see the information about link attributes below)	. (period)	SELECT * FROM Emp WHERE Dept.loc = 'NYC'	None.
Ordering	• ORDER BY • LIMIT • ASC • DESC	SELECT name FROM Emp ORDER BY hiredate DESC LIMIT 10	The OMRQL can order data using ORDER BY <ATTR> [LIMIT N] only if the utterance includes the word order or its natural language synonyms such as sorted, ordered, highest, and smallest.
Aggregate Functions	• COUNT • DISTINCT • AVG • SUM • MIN • MAX	SELECT AVG(sal) from Emp	The OMRQL can contain DISTINCT only if the utterance contains that word or a natural language synonym such as different or unique.
Grouping	• GROUP BY • HAVING	SELECT COUNT(*) FROM Emp GROUP BY Dept.loc HAVING Dept.loc = 'NYC'	None.

Component	OMRQL Keywords	OMRQL Example	Constraints
Comparison Operators	• = • != • <> • > • >= • < • <= • LIKE • NOT LIKE • BETWEEN • IN • NOT IN	SELECT * from Dept WHERE name IN ('Sales', 'HR')	For the >, >=, <, and <= operators, the utterance must contain an equivalent natural language synonym such as greater than, at least, less than, and at most. If the utterance doesn't contain an operator synonym, then the OMRQL must contain =. The OMRQL can contain LIKE only if the utterance contains that word or a natural language synonym such as includes, contains, or substring. The OMRQL can contain BETWEEN only if the utterance contains that word or a natural language synonym such as in the range of.
Logical Operators	• AND • OR • NOT	SELECT name FROM Emp WHERE sal > 100000 AND role = 'VP'	None.

All the values in the OMRQL statement must exactly match the database value and format. Take, for example, the utterance "who is the employee whose name is Jones", If the database values for the name attribute are all capital letters, then the name value must also be all capital letters. That is "SELECT * FROM Emp WHERE name = 'JONES'".

When the utterance that you are mapping uses a synonym for the actual database value, then that synonym must be defined for the value in the value list, and the OMRQL must use the actual database value. For example, if the utterance is "show the department whose location is the big apple", then "big apple" must be defined in the dept_loc value list as a synonym for the value "NEW YORK", and the OMRQL must be "SELECT * FROM Dept WHERE loc = 'NEW YORK'".

Here are some examples of how to write OMRQL for your utterances:

Utterance	SQL	OMRQL	Comments
Show me all employees who work as a clerk	SELECT * FROM Emp WHERE job = 'CLERK'	SELECT * FROM Emp WHERE job = 'CLERK'	OMRQL is identical to SQL.

Utterance	SQL	OMRQL	Comments
Show me all employees who work in sales department	SELECT * FROM Emp AS T1 JOIN Dept AS T2 ON T1.deptno = T2.deptno WHERE T2.dname = 'SALES'	SELECT * FROM Emp WHERE **dept.dname** = 'SALES'	Instead of a JOIN, use "link_attribute.attribute_name" to refer to an attribute from another entity.
Adams is a member of what department?	SELECT * FROM Dept AS T1 JOIN Emp AS T2 ON T1.deptno = T2.deptno WHERE T2.ename = 'Adams'	SELECT * FROM Dept WHERE **emp.ename** = 'ADAMS'	Instead of a JOIN, use "link_attribute.attribute_name" to refer to an attribute from another entity.
What is the department location and job role of employee Adams	SELECT T1.LOC, T2.JOB FROM DEPT T1 JOIN EMP T2 ON T1.DEPTNO = T2.DEPTNO WHERE T2.ENAME = 'ADAMS'	SELECT loc, **emp.job** FROM Dept WHERE emp.ename = 'ADAMS'	Notice how the OMRQL is simpler to write than the SQL.
How many employees are there for every job role?	SELECT COUNT(*), job FROM Emp GROUP BY job	SELECT COUNT(*), job FROM Emp GROUP BY job	OMRQL is identical to SQL.
Which employee has the highest salary?	SELECT * FROM Emp ORDER BY salary DESC LIMIT 1	SELECT * FROM Emp ORDER BY salary DESC LIMIT 1	OMRQL is identical to SQL.
Show employee name and department name ordered by the salary in ascending order	SELECT T1.ename, T2.dname FROM Emp AS T1 JOIN Dept AS T2 ON T1.deptno = T2.deptno ORDER BY T1.sal ASC	SELECT ename, **dept.dname** FROM Emp ORDER BY salary ASC	Notice how the OMRQL is simpler to write than the SQL.

With the exception of linking entities, the OMRQL components are similar to SQL. Instead of an SQL JOIN, you use a pair of link attributes to link one entity to another. Attribute links have primary names and synonyms that define the relationship between the entities. For example an employee/department attribute link with a 1-1 relationship can have a primary name "department" and synonyms "works in", "belongs to", and "team". A department/employees attribute link with a 1-many relationship can have a primary name "employees" and synonyms "members", and "workers".

Besides the typical primary key/foreign key link attributes you also can have these types of link attributes:

- Multiple link attributes from one entity to another that define multiple semantic relationships.
- A link attribute from an entity to itself that implies a self join.
- A link attribute for an intersection table due to a many-to-many join