Published using Google Docs Report abuse Learn more

Tab 1

Skeleton Crew NL2SQL

Attempter Specifications

Document status: **WIP**
Last update: Feb 10, 2025

Welcome to the NL2SQL Project! We're excited to have you on board. This guide provides all the information you need to get started and make meaningful contributions by creating NL2SQL (Natural Language to SQL) pairs that consist of natural language questions and their corresponding SQL queries.

Project Resources:

- Please reach out directly to the project team here in our Skeleton Crew Community Channel: https://community.outlier.ai/chat/c/skeleton-crew-nl2sql/188987
- Please join our Attempter Zoom Room if you're onboarded and working
- Please join our Onboarding Zoom Room if you're new and need onboarding
- Retail Database Create Table Statements: retail_database_v2.sql

---

Updates and Change Log
Check for periodic updates and review any changes to stay successful on this task.

Change Log	
Date	Summary of Changes
Feb 10, 2025	**Updated the category requirement:** The category of the first turn is now predetermined and the other two categories are selected by the human annotator using this link:
Feb 11, 2025	**Added Section 5: Reviewers rubrics:** Added a table of rubrics to rate turns/tasks.
Feb 11, 2025	**Update regarding function changing:** The rewritten query must not use the current date function (this makes it

Published using Google Docs Report abuse Learn more

Table of Contents

🍎 What Should You Know After Finishing This Guide?

- **Project Overview:**
  Understand the purpose of generating NL2SQL examples using a provided retail database schema, existing data, and skeleton SQL queries.

- **Task Flow**
  Learn how you'll perform these tasks —from aligning skeleton queries to the DB schema, creating natural language questions, inserting sample data, and generating multiple distinct NL2SQL examples for each skeleton query.

1️⃣ Section 1: Project Objectives

In this project, you will work with a provided retail database schema, a set of skeleton SQL queries (with placeholder tokens), and existing database records for nine tables (those nine tables are read-only). Your goal is to produce **three distinct NL2SQL pairs** per skeleton query.

Rewriting from Skeleton Queries:

A **skeleton query** is a **query with placeholder/dummy names.**

- The same sequence of placeholder tokens, when used in different parts of the query, for example, "raofx" in the WHERE clause and in the SELECT clause, can be replaced with different column names.

Published using Google Docs Report abuse Learn more


```
  pynro.htrzu_g AS zfpbwdecoa,
  SUM(pynro.tzpnl_avfmyub) AS wyojqeviflthbrzs
FROM _enjirapkvdulhgyqxosbztmwcf AS pynro
```

Creating NL2SQL pairs:

- You'll create an executable SQL query with actual table and column names using the skeleton query as a starting point..
- A natural language question that requests some information that is pulled from the retail database by the skeleton SQL query rewrite.
- The category used to write your rewrite (each category has different relevant tables)
- Sample data insertions to ensure the query returns meaningful output and also used to verify the query's correctness.
- Execution results of the query against the consolidated database.

2️⃣ Section 2: Task Overview

- Each task will have 3 turns based on the same skeleton query.
  a. You'll change the placeholders of the skeleton to create 3 usable SQL queries and write 3 natural language (NL) questions for those queries. We will call these NL2SQL pairs.
  b. In Turn 1, you'll match the NL2SQL pair with the provided task category. You'll use this to figure out the topic of your Natural Language Question and SQL query.
  c. The NL2SL pairs in turns 2 and 3 **MUST be creative and NOT a simple variation of the previous NL2SQL pairs you have created**. In order to ensure this you'll see that:
     - The category Turn 1 is predetermined, but you cannot use this category in Turn 2 and Turn 3. See the category table.
     - Only use tables relevant to the category. You're not required to use all tables.
     - You must ensure table diversity across the three turns: Use different tables in the different turns.
     - Each rewrite (R1, R2, R3) must also use a unique table, not used in another rewrite. A combination of tables must also be unique.

Good Examples	Bad Examples
• #1: R1 uses table [x]; R2 uses table [y]; R3 uses table [z] • #2: R1 uses tables [x, y]; R2 uses tables [z, y]; R3 uses tables [a]	• #1 R1 uses table [x]; R2 uses table [y]; R3 uses table [x] • #2 R1 uses tables [x.y]; R2 uses tables [y, z]. R3 uses tables [z, y]

- Each turn will consist of 5 submissions:
  1. A natural language question (NLQ).

Published using Google Docs Report abuse Learn more

4.1. The insert statements are required to make the SQL rewrite produce a meaningful output in the provided workspace.
5. The output of running the SQL rewrite.

- **Each turn will include two workspace IDEs: One for testing, and one for production.**
  1. You will first test your rewritten queries in a testing environment until you're satisfied with the output returned. Besides the documentation in Step 0
     1.1. The rewrite needs to be a 1:1 match with the NLQ.
     1.2. **The rewrite is the SQL code necessary for an LLM to pull information from a database that answers an NLQ.**
  2. After testing that the query works for the provided DB Schema, and ensuring that the data will be consistent with the current DB, you will push the INSERTS you tested and submitted into the prod environment.

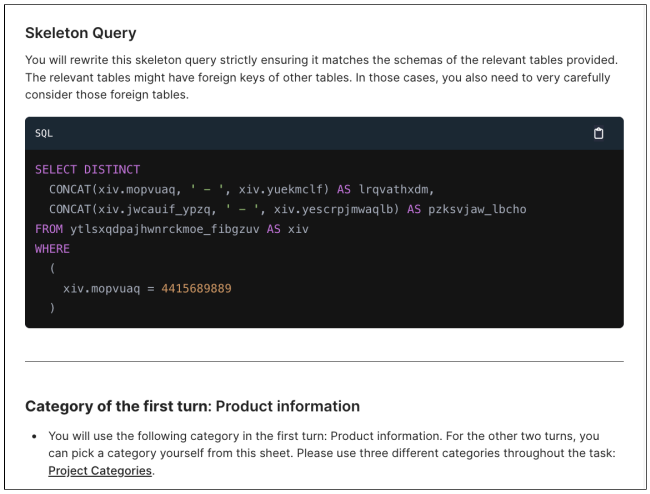
Step 0: Understand the Data Base Schema

- It is crucial that you fully understand the relations in the retail database in order to create logical queries.
- Please take a look at the database creation script to understand the data constraints of the Retail DB.
- You can also view the **online ERD** diagram of the retail database, or **download** the ERD diagram in different file formats:

Complete diagram	**Online**	**PNG**	**PDF**	**SVG**
Tables, PK and FK only		**PNG**	**PDF**	**SVG**

Step 1: Rewrite the Skeleton Query to Align it with the Retail DB Schema

💡 Before You Start: Check the Skeleton and Category of the First Turn



Skeleton Query

You will rewrite this skeleton query strictly ensuring it matches the schemas of the relevant tables provided. The relevant tables might have foreign keys of other tables. In those cases, you also need to very carefully consider those foreign tables.

SQL

```
SELECT DISTINCT
  CONCAT(xiv.mopvuaq, ' - ', xiv.yuekmclf) AS lrqvathxdm,
  CONCAT(xiv.jwcauif_ypzq, ' - ', xiv.yescrpjmwaqlb) AS pzksvjaw_lbcho
FROM ytlsxqdpajhwnrckmoe_fibgzuv AS xiv
WHERE
  (
    xiv.mopvuaq = 4415689889
  )
```

Category of the first turn: Product information

- You will use the following category in the first turn: Product information. For the other two turns, you can pick a category yourself from this sheet. Please use three different categories throughout the task: Project Categories.

You should align the skeleton with the **Retail DB Schema using the categories tables**. It's not a strict requirement to use all the relevant category tables. Try to use as many of those relevant tables as

Published using Google Docs Report abuse Learn more

column names from the provided **retail** database schema.

  - The same sequence of placeholder tokens but in a different location in the query, e.g. "raofx" in the WHERE clause and SELECT clause can be replaced with two different column names. It **should not be interpreted as a variable.**
  - The category of the first turn is predetermined. You can only use tables relevant to that category for the first turn.
  - For the other turns you can pick a category yourself. All three turns need to have a different category and use different tables.

- Example:

Skeleton Query With Placeholder Tokens For Two Columns and One Table

```
SELECT DISTINCT
  dffg.dfos
FROM pvywh AS dffg
WHERE
  (
    dffg.zgk_mus = 27460
  )
```

After Alignment With Retail DB (references the Customers table)

```
SELECT DISTINCT
  c.Name
FROM Customers c
WHERE
  (
    c.City = 'London'

  )
```

- **A SQL Rewrite Should:**

1. Ensure that the query is valid and specific to the db schema: It should reference existing columns from the Retail DB schema.
    a. The Retail Database contains 51 tables with a constant column count. No columns will be dropped or added. 9/51 tables are read-only.
    b. The # of columns per table varies greatly.
    c. You are only allowed to change these fields in the query: table names, column names, some of the functions used (e.g: CONCAT(), SUM(), operators(+/-), and constant values (e.g. constant values used in the SQL WHERE Clause to filter). ALWAYS replace the CURRENT_DATE() function with a date, a date range, or another function, since the ground truth varies over time.
    d. Maintain the complexity of the provided skeleton: If you delete constraints, such as ANDs inside of WHERE statements, the complexity of the query will change and this should be avoided at all costs.
2. MUST be executable against the database after data insertion.

Published using Google Docs Report abuse Learn more

    c. Think about possible insertions you may need to do in order to generate an output of at least one row.

3. The changed fields MUST make sense in the context of the query, and prompt.
4. The rewrite Should NOT include comments if these are not present in the skeleton query.
5. The rewritten query cannot use the current date function as it makes the NLQ time-sensitive (subject to change over time).
6. Instead, rewrite the query so it is anchored within two time periods. This makes the query more robust as querying it against the database always results in a meaningful result, even when time passes (given that the current data inserted or present is not deleted from the retail database).
7. The complexity of a query should be kept always.
8. Must reference a unique table/table combination.
9. Wrapped around backticks: ```\n[sql_code]\n```
    a. A markdown code black created will be rendered in.

Step 2: Create a Natural Language Question (NLQ)

- What You Do:
  Write a clear, concise, question in natural language that **directly corresponds** to the SQL query you've just created. There should be a one-to-one correspondence between the Natural Language Query and the SQL query.
- Example:
  What are the names of the customers who are based in London?
- **A Good NLQ Should:**
  - Write a natural language question that can be answered by executing the SQL and directly correspond to the logic of it.
    - NOTE: The NLQ should align with the complexity of the skeleton: If the NLQ requires fewer or more constraints to match the rewrite than to match the skeleton query the whole turn will fail on the grounds of reduced or increased complexity (see rubrics below).
  - Use natural, conversational language —Phrase the question to request the output data in plain language rather than describing the SQL query itself.
    - **Good Example:** *What are the names of the customers who are based in London?*

Published using Google Docs Report abuse Learn more

simply ask for the data, regardless of how it is presented.

- ex: if the query has: CONCAT(c.description, ' - ', c.category_id) AS category_description. You don't need to ask specifically for the '-' just ask for the description with the category id.

- Should NOT reference SQL functions or syntax. Ensure the questions sound like something a non-technical user would naturally ask. No overly technical, contrived, or grammatically awkward phrasing.
  - Do NOT use table names or **raw** column names (ex: customer_name)
  - Do NOT use words like column, table,rows or aliases.
- Unambiguous - the question should be clear, concise, and unambiguous. The information asked cannot be open to multiple interpretations. There should be only one correct answer.
- Should NOT include unnecessary fluff or pleasantries
  - Example of a **bad NLQ**: *"Hey! I need information on the "Electronics" category. Can you be so kind as to provide the category description along with the category ID, joined together by a dash?"*
- **Specify the historical time and range to ensure the query yields the same answer, even if asked in the future**: the NLQ should specify a specific time range and not use phrases like "last year", "last month". The returned ground truth should remain consistent over time.
- The query should have excellent grammar. ALWAYS use a spell checker like **grammarly** or **languagetool**.

Step 3: Populate the Database with Sample Data (INSERTs)

- **A Good Insert Should: (**Per insert statement):
  - You are ONLY allowed to add rows to a table.
  - Min of 2 insert statements and max of 10, **but if the tables you need are already populated and the query returns meaningful results, you must not insert extra data.**
  - You are NOT allowed to insert to these 9 tables (just reading):

```
Brands          Categories          Employees
LoyaltyProgram  Manufacturers       PromotionalEvents
Promotions      Stores              Subcategories
```

  - Each insert must populate all primary and foreign key columns, all selected colums, all columns with the NON NULL constraint. Columns designated as "Unique" in the **notes** or

Published using Google Docs Report abuse Learn more

- Have realistic and relevant data. The inserted data should reflect scenarios that are relevant to the retail domain. The data should align with the Retail DB schema and data types.
- Avoid unnecessary data insertions: only the necessary records to ensure the query has meaningful results. The goal is to insert enough data to test the query thoroughly without overwhelming the database with irrelevant records. **Only insert records that fit the filter conditions of the query.**
- Trigger useful results. This means populating the tables in a way that allows the query to calculate and aggregate the desired outcome.
- For primary keys inserted in the table, you **must** use an incremental approach based on existing data, like using `MAX(yyyyy_id)+1`. All primary keys are integers and increment in steps of 1.
  - Query the MAX(primary_key) of the table when inserting new rows and increment the primary +1 for each additional row inserted
- Foreign keys inserted in the table must exist in the relevant table.
- ⚠️ **DO NOT INSERT PII INTO THE DB** (Personally Identifiable Information). While the data must exemplify the workings of a real DB, you must always ensure not to include private information on it.

How Do We Create the INSERTs?:

- Make **a copy** of the following sheet: skeleton_crew_insert_into_creator_[template]
- In the first tab, named “schema”, you’ll find the database creation script. Use the database creation script as **documentation** on the constraints and data types used in the DB.
- You may want to use prompt nr 1 (“dummy insert data generator prompt”) to create data for your insert statements.
  - **Notice that you most probably will change the dummy generated data to match the database constraints and the data queried.**
  - **Warning: The dummy data generation tools are a very useful starting point but should not be blatantly copied. You HAVE TO validate the data inserted yourself.**
- You will only insert data into columns that are per the schema:
  - NOT NULL
  - FOREIGN KEYS
  - PRIMARY KEYS
  - Used in your query

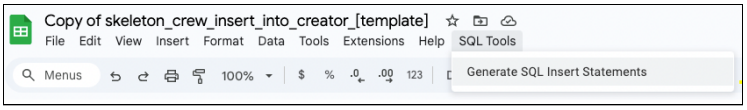
Published using Google Docs Report abuse Learn more

- **The first row of a table will have most columns filled out just for reference.**

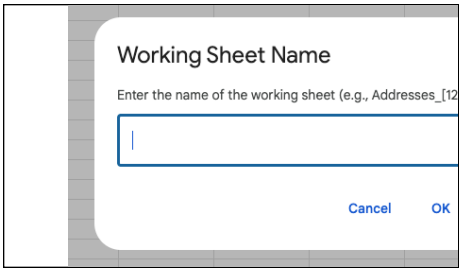
- You do NOT make INSERT statements in the 9 populated tables:
  - `Brands`
  - `Categories`
  - `Employees`
  - `LoyaltyProgram`
  - `Manufacturers`
  - `PromotionalEvents`
  - `Promotions`
  - `Stores`
  - `Subcategories.`

- Paste the data in the tab that corresponds to the table you will insert to.
  - You must double-check the data you provide in the sheet. The data should be in the correct format, the correct data types, and it should be aligned with the data that is already present in the DB.

- Once you are sure the data inserted in the sheet is in line with the previous instruction steps, go to **`SQL Tools -> Generate Insert Statements.`**



- Complete the pop-up with the name of the sheet (tab) that corresponds to the table you need to populate, and press **OK**.



- You will now get a file with the INSERT INTO statements you can use in the workspace IDE. Just copy over the insert into statements. Please still validate the data inserted. The Google sheet automatically infers the data type. Integers for example should be numbers and not wrapped around quotation marks.

Published using Google Docs Report abuse Learn more

- Often you will write insert into statements for multiple tables depending on the skeleton query. E.g. when the skeleton query contains a join or when you reference foreign keys. You have only one field for the insert into statements.
    - You would run the tool twice for a different tab to combine two code snippets and then combine the snippets together into a single insert.sql file.
    - Make sure you separate the insert into statements for different tables with a blank new line and group the insert into statements of the same table together.

- Important Notes:
    - Make sure you have enough sample data inserted into the DB so that your SQL query returns meaningful results.
    - Avoid adding unnecessary data to the tables of the DB unless it's absolutely necessary. Validate that the queried tables are populated with enough data to have the query return a useful result.

Example:

```
INSERT INTO Customers (CustomerID, Name, City) VALUES (1, 'Alice', 'London');
INSERT INTO Customers (CustomerID, Name, City) VALUES (2, 'Bob', 'Paris');
INSERT INTO Customers (CustomerID, Name, City) VALUES (3, 'Charlie', 'London');
INSERT INTO Customers (CustomerID, Name, City) VALUES (4, 'David', 'New York');
INSERT INTO Customers (CustomerID, Name, City) VALUES (5, 'Eve', 'London');
```

Step 4: Create Two Additional NL2SQL Pair Examples

- What You Do:
  Based on the same skeleton query, independently create two more NL2SQL pairs. These examples should:
    - For turns 2 and 3 you can pick a category yourself. All three turns need to have a different category.
    - Use a different SQL approach (e.g., altering the query logic or returning additional columns).
    - Feature a different natural language question that still aligns with the intended logic.
    - Possibly utilize alternative sample data insertions if needed.

- **A Subsequent Rewrite Should:**:
    - The NL question should request a different piece of information, ensuring the output data is not a direct 1:1 match while still aligning with the overall structure.

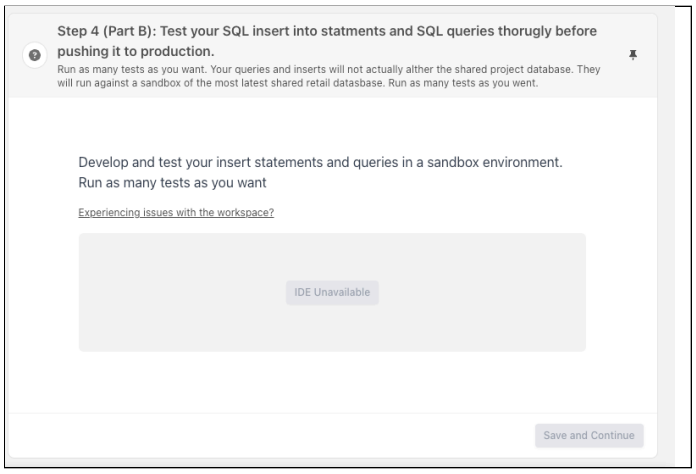
Published using Google Docs Report abuse Learn more

- Ex: Query1 uses tables a,b,c | Query2 uses tables a,c,d | Query3 uses tables a,b,c.

- No one query or insert statements should be the a direct 1:1 match

- Each new example(NL, Query, Insert) must maintain similar complexity.

Step 5: Test Your Queries Against the Production Data Base In The First Workspace / Sphere Engine



- What You Do:

  - Use the Sphere Engine Workspace in the task to test your queries and make sure they align with the current state of the DB.

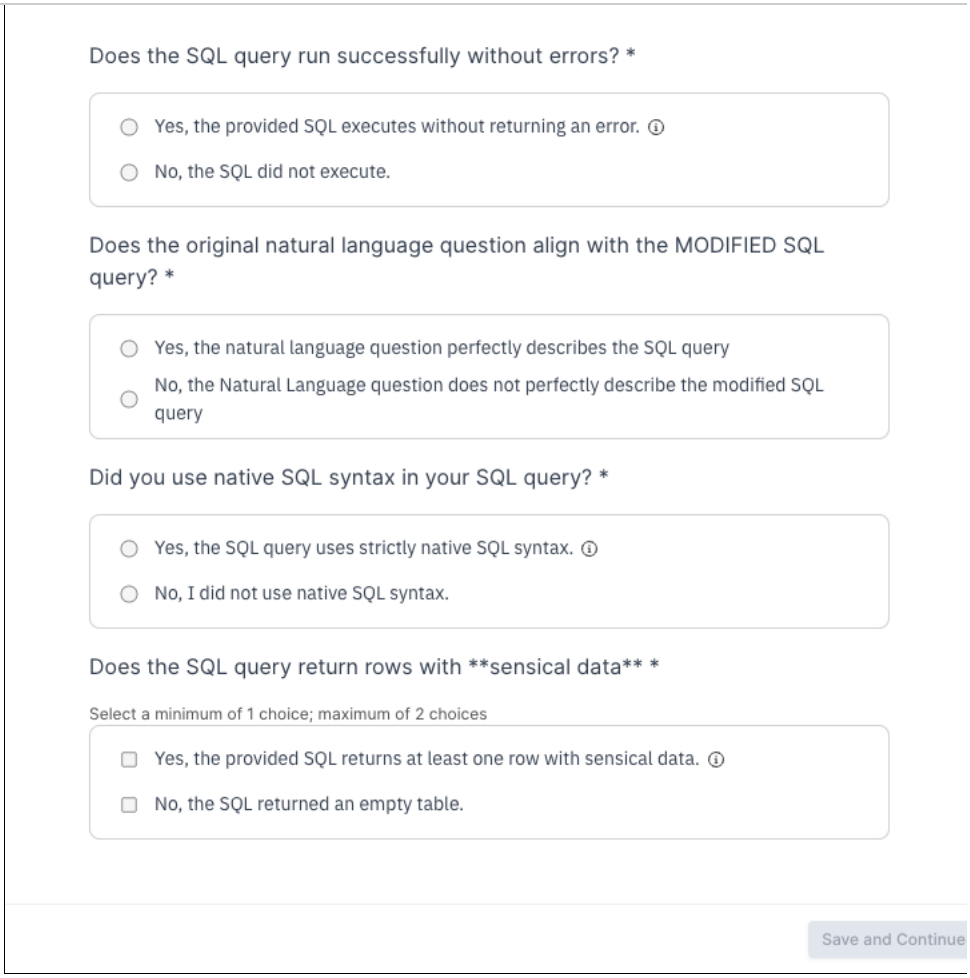
  - Delete the CSV file in between queries to make sure the output stays up to date.

- Important Notes:

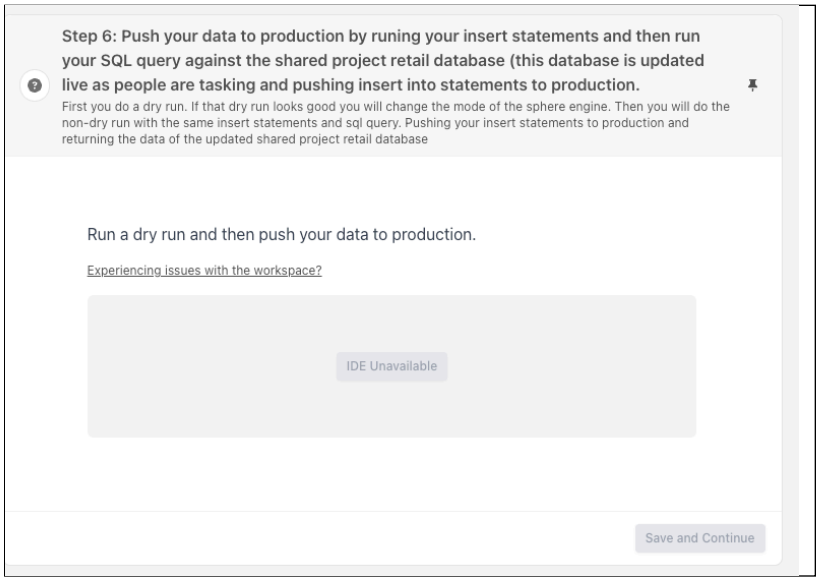
  - You may encounter that some insert statements may trigger reference errors. This can happen if the same record has already been inserted. Check that the data you insert is not conflicting with the current data in the DB

Step 6: Fill Out The Checklist: Fill out the checklist truthfully to validate you can push to production.

Published using Google Docs Report abuse Learn more



Step 7: Push Stage: Push Your Data into Production In The Second Workspace / Sphere Engine.



Published using Google Docs Report abuse Learn more

- First, do a dry run as a final sanity check (this runs both files) by pressing the button.
- Then do a prod run pushing your insert into statements to production and query the result.
  - The result will be saved to the results.csv file

Final results:

- You should submit **3 natural language questions** and **3 corresponding SQL queries**.

3️⃣ Section 3: Detailed Examples

Below are examples to illustrate the full task for one skeleton query.

Example #1

Provided Resources:

- **Database Schema:**

```
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    Name VARCHAR(255),
    City VARCHAR(255)
);

CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
    OrderDate DATE,

   FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

- **Skeleton Query:**

```
SELECT T1.C1 FROM T1 WHERE T1.C2 = 'VAL';
```

Aligned SQL Query:

```
SELECT c.Name FROM Customers c WHERE c.City = 'London';
```

Natural Language Question:

What is the name of the customer who lives in London?

Sample Data Insertions:

```
INSERT INTO Customers (CustomerID, Name, City) VALUES (1, 'Alice', 'London');
INSERT INTO Customers (CustomerID, Name, City) VALUES (2, 'Bob', 'Paris');
INSERT INTO Customers (CustomerID, Name, City) VALUES (3, 'Charlie', 'London');
INSERT INTO Customers (CustomerID, Name, City) VALUES (4, 'David', 'New York');
INSERT INTO Customers (CustomerID, Name, City) VALUES (5, 'Eve', 'London');
```

Published using Google Docs Report abuse Learn more

Example #2

Aligned SQL Query:

```
SELECT COUNT(*) AS LondonCustomerCount FROM Customers WHERE City = 'London';
```

Natural Language Question:

How many customers are located in London?

Sample Data:
Use the same consolidated sample data as in Example #1.

Execution Result (Expected):

LondonCustomerCount

3

Example #3

Aligned SQL Query:

```
SELECT Name, City FROM Customers WHERE City = 'London';
```

Natural Language Question:

Which customers and their cities are recorded for those living in London?

Sample Data:
Use the same consolidated sample data as in Example #1.

Execution Result (Expected):

Eve

4️⃣ Section 4: Deliverables and Final Notes

Your Deliverables

For each of the 85 provided skeleton queries, you will submit:

- **Three NL2SQL Pairs (Total: 255 pairs):**
  Each pair includes the aligned SQL query, a corresponding

Published using Google Docs Report abuse Learn more

- **Documentation of Execution Results:**
  Present the output of each SQL query execution in a clear format (e.g., tables or formatted text).

Additional Guidelines

- **Adhere to SQL Standards:**
  Use standard SQL syntax and avoid database-specific functions where possible.

- **Use Existing Data:**
  For the nine tables with pre-existing records, leverage that data as much as possible. Only add new data if absolutely necessary.

- **Maintain Consistency:**
  All data insertions across examples should contribute to a single, consolidated database state.

- ⚠️ **DO NOT INSERT PII INTO THE DATA BASE** (Personally Identifiable Information). While the data must exemplify the workings of a real DB, you must always ensure not to include private information on it.

- **Possible Error when 'INSERT INTO' DB:**
  While performing a task, it is possible that between executing your **INSERT INTO** statement on the sandbox database and running the corresponding statement in production, someone may have already inserted a record with the same primary key or unique ID. In that case, you will encounter an error. Therefore, you must modify the INSERT statement by changing the unique identifier and correcting the task, then run the query again to review the new output.

Categories table

- ail tables mapped to their category:

Category	Relevant Retail Database Tables
Salesforecast	- RetailPredictiveAnalytics - RetailAnalytics - Sales - SalesTransaction - Orders
Departmentperformance	- EmployeePerformance - StorePerformance
Salesdata	- Sales - SalesTransaction - SalesChannels - Sales_Representatives - Orders - Invoices
Inventorystatus	- InventoryLevels - InventoryAdjustments - InventoryManagement - Inventory_Movements
Dateinformation	- Orders - Invoices

Published using Google Docs Report abuse Learn more

Inventorygap	- InventoryAdjustments - InventoryLevels - InventoryManagement - Inventory_Movements
Gapanalysis	- RetailAnalytics - RetailPredictiveAnalytics - Sales - InventoryLevels - StorePerformance - CustomerFeedback - CustomerJourneyAnalytics
Productdetails	- ProductDetails - Products
Fiscalmonthnumber	- Financial_Reports - Sales - Transactions - Invoices
Productlocation	- InventoryLevels - Warehouses - Stores - Inventory_Movements
Inventorypipeline	- AdvancedInventoryManagement - InventoryAdjustments - InventoryLevels - InventoryManagement - Inventory_Movements - Warehouses
Daterange	- Orders - Invoices - Sales - Transactions - PromotionalEvents - MarketingCampaigns - SupplyChainEvents
Inventorylevels	- InventoryLevels - InventoryManagement - InventoryAdjustments - Inventory_Movements
Performancemetrics	For the 'Performance metrics' category, the most relevant table names from the list are: EmployeePerformance, StorePerformance.
Storepressure	For the 'Store pressure' category, the most relevant table names from the list would likely be: - StorePerformance - InventoryLevels - Sales - SalesChannels - EmployeePerformance
Minimumregistration	- Customers - CustomerProfile
Fiscalcalendar	- Financial_Reports
FinancialSummary	 - Financial_Reports - Invoices - Sales - Transactions
Orderinformation	- Orders - Invoices - Payment_Details - SalesTransaction - Transactions.
LowestISTSpercentage	- InventoryAdjustments - InventoryLevels - InventoryManagement - Inventory_Movements - SupplyChainManagement - Warehouses
Performancedata	- EmployeePerformance - StorePerformance

Published using Google Docs Report abuse Learn more

	- Stores
Storeperformance	- StorePerformance - Sale - SalesChannels - SalesTransaction - RetailAnalytics
Productinformation	- ProductDetails - Products - ProductBundles - ProductLifecycleManagement - ProductRecommendations - ProductReviews
Fiscalweek	- Financial_Reports - Sales - Transactions - Invoices
Fiscalyear	- Financial_Reports - Invoices, Sales - Transactions
Performancemetric	- EmployeePerformance - StorePerformance
Accountbalance	- Financial_Reports - Invoices - Transactions
Productpipeline	- ProductLifecycleManagement - ProductDetails - Products
Storelocation	- Addresses - Stores - Warehouses
Marketlocation	- Addresses - Stores - Warehouses
Inventorypressure	- InventoryAdjustments - InventoryLevels - InventoryManagement - Inventory_Movements - AdvancedInventoryManagement
Productclassification	- Categories - Subcategories - ProductDetails - Products
Performancegap	- EmployeePerformance - StorePerformance
Pricinginformation	- ProductDetails - Promotions - ProductBundles
Locationinformation	- Addresses - Stores - Warehouses
Classperformance	- EmployeePerformance - StorePerformance
Firmwarestatus	For the 'Firmware status' category, none of the table names provided in your list directly relate to firmware or its status. Firmware typically pertains to software that provides low-level control for a device's specific hardware. The tables in your list are more focused on retail operations, inventory, customer management, and sales.
Locationdata	- Addresses - Stores - Warehouses

Published using Google Docs Report abuse Learn more

Dummy insert data generator prompt	*- Create a table with strictly [amount_of_rows] rows populated with dummy data.* *- Just provide a plain rendered-in table and no SQL code.* *- Use original data and not just John Doe, Jane Smith, Alice, etc.* *-The min value for [primary_key] is[max_value_primary_key] and it should auto increment.* *- Create strictly dummy data for the tables: [Columns]* *- Also, ensure the data generated coheres with the existing data. Here are some existing rows:* *```sql* *[snippet_existing_rows]* *```*

5️⃣ Section 5: Reviewer Rubrics

- **Instructions Table Usage for Overall Task Scoring**

The overall turn score can only be as **high** as the **lowest** score of any of the 10 table criteria applied to a turn individually ignoring other turns.

The overall task score can only be as **high** as the **lowest** turn score.

Example 1: If you mark 1-2 (no Alignment) for the first criterion “Is SQL executable”, the overall task score is automatically reduced to a maximum of 2.

A score of 1 is reserved for cheating and spamming! Even if the task is horrendously executed you score it with a 1 only if you conclude a CB is spamming/cheating.

- **Rubrics**

The Below Table Is Applied Per Turn

Criterion	1-2 (No Alignment)	3 (Partial Alignment)	4-5 (Full Alignment)	Notes
Is SQL executable?	The SQL is not executable, containing syntax or logic errors that prevent execution.	The SQL has minor errors but is still executable with minor adjustments.	The SQL is fully executable without errors.	
Does the rewrite correctly align the skeleton query with the retail database	No, it uses columns or tables not present in the retail database.	N/A	Yes, it uses columns or tables not present in the retail database.	
Is the rewritten query of the same complexity as the skeleton query	- No, the rewritten query is notably less/more complex. - - Example of less complex: Less AND statements. - -	N/A	Yes, the complexity of the queries remains the same.	If the NLQ requires fewer or more constraints to match the rewrite than to match the skeleton query the turn fails on the grounds of reduced

Published using Google Docs Report abuse Learn more

	tables joined.			
Is the query time-sensitive as it uses a function like the current_date()	Yes, the query is time-sensitive as the output of the query changes over time	N/A	No, the query is not time-sensitive and if dates are used the query is anchored between two dates by using a constant date.	
Is the NLQ time-sensitive as it asks to count back or count forward from the current date	- Yes, the NLQ is time-sensitive as it asks for results counting forward or back from the current date. - - Example: What orders placed by Johannes in the last two weeks have a fee higher than 20 dollars?	N/A	No, the NLQ is not time-sensitive and when using dates the dates are anchored (the relevant dates referenced do not change when the NLQ is asked in the future)	Anchored means that the NLQ asks for the same dates always. Even if it would have been asked in a year from now. E.g: "What orders were placed between [date 1] and [date 2] by Customers located in New York?".
Are the three rewrites sufficiently diverse	No, they use the same three tables.	Yes, the rewrite uses different tables but not unique operators.	Yes, they all use unique tables and table combinations. They also use different operators.	Note that each rewrite (R) should also use two unique columns. This should happen automatically when using an unique table per R,
Is the returned table empty?	The query returns an empty table or NULL values when it shouldn't. The query selects count(*) and the output is 0, meaning the actual filters in the query returned no data.	The query returns some valid results, but some may be empty or incomplete.	The query returns the correct results without being empty or containing NULLs.	
Are the Insert Into Statements in the correct format	The insert into statements does not match the format requested.	N/A	Yes, the insert-into-statements format aligns with the format requested and are grouped together per table.	
Are the Insert Into Statements	The insert-into statements do not	The insert-into	The insert-into	

Published using Google Docs Report abuse Learn more

		used	used is reasonable	
Do the Insert Into Statements Cohere with existing data	No, the insert-into statements insert values that do not match the column's purposes	Yes, but the columns populated contain gaps.		
Are the wrong columns selected?	The query selects incorrect columns or omits key columns that should be included in the results.	The query selects some correct columns but omits or incorrectly selects some minor columns.	The query selects all the correct columns that match the NLQ without omissions or errors.	