# 1. Optimum Prompt for This Model and Task

The model you are using, "Llama Code 7B Instruct", is trained to follow instructions and respond to queries based on specific formatting and task-related prompts. Below is a refined prompt that should improve your model's performance for translating natural language (NL) queries to Postgres SQL queries:

**Optimized Prompt:**

You are an expert in PostgreSQL databases. Given a user's natural language (NL) question, your task is to:

1. Translate the user's question into a syntactically correct and efficient Postgres SQL query.

2. The SQL query should only retrieve the columns necessary to answer the question. Avoid retrieving unnecessary columns or entire tables.

3. Always wrap column names in double quotes (") to denote them as delimited identifiers.

4. If the question involves querying for "today's" data, use the `date('now')` function appropriately.

5. Use the `LIMIT` clause to restrict the result to at most 5 row, unless the user explicitly requests multiple examples.

6. Pay attention to which columns are available in the tables provided and ensure the query is restricted to those columns only. If a column does not exist, do not reference it in the SQL query.

7. Structure your response as follows:

Question: <Natural Language Question>

SQLQuery: <Generated SQL Query>

You are only allowed to use the following tables and their columns:

{LIST\_OF\_TABLES}

{LIST\_OF\_COLUMNS\_NAMES\_AND\_TYPES}

QUESTION: {USER\_NATURAL\_LANGUAGE\_INPUT}

SQLQuery:

**Key Changes:**

* Specified the inclusion of necessary details like the LIMIT clause for query size control.
* Emphasized proper column handling, especially with quoted identifiers and non-existent columns.
* Incorporated the handling of "today's" date properly with the date('now') function.
* Clarified that the query should only fetch relevant columns and rows.

# 2. Model Parameters That Can Help with the Quality of the Output

The quality of the output depends on several factors. Below are some recommended model parameters for the best performance:

1. **Temperature: 0.2 - 0.5**
   * Lower temperatures (0.2-0.3) will make the model's responses more deterministic and focused, avoiding unnecessary variation.
   * This is ideal for SQL generation, as you want consistency and correctness in the output, especially when generating code or queries.
2. **Top-p (nucleus sampling): 0.9**
   * A higher value allows the model to choose from a broader set of possible outputs, which can be useful when dealing with varied natural language queries.
   * This ensures the model considers a variety of solutions, but the deterministic temperature keeps it from becoming overly creative.
3. **Max Tokens: 150-200**
   * SQL queries can be concise, but adding room for potential complex queries (especially when using joins or nested selects) ensures the model doesn't truncate critical parts of the query.
4. **Stop Sequences:**
   * Use a stop sequence such as SQLQuery: to ensure that the model's response is terminated at the right point, preventing unnecessary continuations or irrelevant output.
5. **Repetition Penalty: 1.1 - 1.2**
   * This helps in reducing repeated patterns in the output, ensuring that the generated SQL query is unique and clear without excessive redundancy.

# 3. Tips and Tricks for Getting the Best Out of the Model for This Context

Here are several tips to fine-tune your interaction with the model for translating NL queries into SQL:

1. **Clear Table and Column Information:**
   * Always ensure that the model has access to accurate and up-to-date table structures (LIST\_OF\_TABLES and LIST\_OF\_COLUMNS\_NAMES\_AND\_TYPES).
   * If possible, provide sample data in your prompt to improve the model’s ability to contextualize the query.
2. **Structure User Questions Clearly:**
   * Break down complex natural language questions into simpler parts. For example, if a user asks for data involving multiple conditions, ensure the prompt allows the model to address each condition with an appropriately structured SQL query.
   * Avoid overly vague or ambiguous questions. The more specific the natural language input, the more precise the SQL query will be.
3. **Model Interaction:**
   * Use iterative questioning if the model fails to generate the correct query. If the initial response is incomplete or inaccurate, rephrase the natural language query and provide additional instructions or constraints.
   * Provide examples to the model when the behavior is not as expected. For example, if a user requests specific columns from multiple tables, guide the model to handle joins and ensure it only includes relevant columns. Add the following to the prompt:   
     “If the question requires data from multiple tables, make sure to join the relevant tables using appropriate keys and include only the necessary columns from each table.”
4. **SQL Formatting and Efficiency:**
   * Encourage the model to write efficient queries. For example, prompt the model to avoid querying for all columns with a wildcard (SELECT \*), instead ensuring it queries only what is necessary.
   * Ensure the model understands that performance is a key factor, especially when working with large databases. This can be done by stressing the use of LIMIT, WHERE, and ORDER BY clauses to make the queries more efficient.
5. **Handle "Today" or Time-Sensitive Data:**
   * If the question involves time-sensitive data (e.g., "today"), explicitly ask the model to use PostgreSQL’s date('now') function, as you have done in your prompt.
   * If the user asks for data between specific dates or times, make sure to remind the model to correctly implement date filtering.
6. **Limit Query Scope:**
   * If users have specified a scope (e.g., specific rows or a time frame), ensure that the model generates queries that respect those limits by using the WHERE and LIMIT clauses appropriately.
7. **Provide Context and Clarifications for Complex Questions:**
   * If a query requires interpreting natural language in multiple stages (e.g., filtering, joining, and selecting), consider breaking down the steps into smaller components, allowing the model to focus on generating one part of the query at a time.

By incorporating these strategies and fine-tuning the prompt, you will be able to achieve accurate, concise, and efficient SQL queries generated from natural language input.