**How to run the project:**

- Directly run the .ipynb file provided.
- After installing the 'happytransformer' library, uninstall the 'transformers' library which gets installed during the installation of 'happytransformer'.
- Restart the kernel.
- Install older version of transformer (4.28.0).
- This is due to the fact that the newer version of transformer has a dependency issue with PartialState import.
- Upload the .csv files of HR data for the model to query on it.

**Methodology:**

An overview of my approach is, I convert the natural language into SQL queries and then execute those queries with the help of sqlite3 in python.

**Models:**

- I have used a pre-trained t5 model for the task.
- There were a variety of choices for models like t5, BART, Alpaca, GPT4 SantaCoder etc.
- Amongst this I believe GPT4 or SantaCoder would have been the best model to use. GPT4 has shown to work well on natural language to SQL query tasks. SantaCoder model has been pre-trained for code generation on 'the stack' dataset. The dataset contains 358 programming languages and SQL is one of them.
- I didn't have access to GPT4, and models like SantaCoder & Alpaca require a lot of computational resources to be finetuned. Hence I had to choose between BART or t5.
- After doing literature review, I realized that t5 performed better for text-to-text generation tasks that BART. Hence I decided to go forward with t5.

**Dataset:**

- 2 datasets are publically available named: sql-create-context, sql-create-context_alpaca_style .
- Both of them have the same raw data, but the later one some prompting instruction. Hence I decided to use the later one.
- The T5 model requires the input data to have some fix token/prompt which is constant throughout all the data points.
- I have maintained the following format for the input:

The relevant table was constructed using the following SQL CREATE TABLE statement:  *Write Create Statement*
Write a SQL query that answers the following question:
*Write Natural Language Query*

- THis will help the model understand the context of our task better.

**Future Work:**

- **Use Better Model:**
  - To increase the accuracy of generating SQL queries, as mentioned above in the Models section we can use Alpaca, GPT4, Santacoder. These models are trained on large amounts of data which is closely related to our fine tuning task.
- **Use Better Dataset:**
  - Currently I'm training the model on just 10,000 data points due to computational limitations. Model trained on all 78577 data points will perform better.
  - Current dataset doesn't contain all the types of SQL queries, for example: there is a lack of queries which ask to calculate the percentage of a given column, or do aggregation between columns. There is a lack of complex queries. If we generate a few examples of such complex queries, the model will perform better.
  - While currently passing the input: when writing the create table statement, only single tables are created. If we generate some data points which create multiple tables, and have foreign key, primary key references in them. The model can understand relationships between different tables.
- **Improved Training Methods:**
  - **Chain of thoughts prompting:** In order to reduce error propagation and enhance results on text-to-SQL datasets, this paper suggests new prompting approaches that break down the original question and avoid providing specific information in reasoning steps.
  - **Conversational text-to-SQL:** Proposed text-to-SQL system consists of 3 parts:
    - Multi-Tasking with prompting
    - Constrained Decoding
    - N-Best List Reranking
  - **T5QL: Taming language models for SQL generation:** This work tries to bridge the gap of increasing accuracy of smaller LLMS like T5 for SQL query generation tasks. They show that dividing semantic parsing in two

tasks, candidate SQLs generation and candidate re-ranking, is promising and can reduce the need for large LLMs.

- **Reinforcement learning:** As GPT was fine tuned into ChatGPT using RLHF technique to perform better on conversational tasks, in a similar fashion we can fine tune our model by assigning positive & negative rewards upon query generation.
- **Fine Tuning on domain specific dataset:** This method can be used to overcome the problem of the models not being trained on internal/company data. We can create a small dataset which replicates the pattern of company data and use it for the finetuning process. I believe a small dataset can boost the performance significantly.