Data Base

Lab 1

AI-Powered ER Diagram Generator

Name / Yaseen Asaad Ahmed

Id / 22011349

- ## Part 1 : JSON to ER-Diagram Renderer

1. Code:

```python
import json
from graphviz import Graph
import sys

if len(sys.argv) > 1:
    path = sys.argv[1]
else:
    # If not passed, ask the user to input it
    path = input("Enter JSON path: ").strip()

with open(path, "r") as file:
    data = json.load(file)

ER = Graph(comment="ER Diagram", format="png")
ER.attr(splines="spline", bgcolor="white")

for entity in data["entities"]:
    entity_name = entity["name"].strip()
    ER.node(entity_name,entity_name, shape="box", style="filled", fillcolor="#d1e8ff")  #No two entites have same name so name = unique_id

    for attribute in entity["attributes"] :
        attribute_name = attribute["name"]
        attribute_id = f"{entity_name}+{attribute_name}"         #to uniquely identify each attribute by id = entity_name+attribute_name
        is_pk = attribute.get("isPrimaryKey", False)
        composite = attribute.get("composite", [])
        ismulti = attribute.get("isMultiValued", False)

        if is_pk:
            attribute_name =  f"<<u>{attribute_name}</u>>"
            ER.node(attribute_id, attribute_name, shape="ellipse", fillcolor="#fff2cc", style="filled")
        elif ismulti:
            ER.node(attribute_id, attribute_name, shape="ellipse", peripheries="2", fillcolor="#fff2cc", style="filled")
        else :
            ER.node(attribute_id, attribute_name, shape="ellipse", fillcolor="#fff2cc", style="filled")

        ER.edge(entity_name, attribute_id)


        for sub_attribute in composite :
            sub_attribute_id = f"{entity_name}+{attribute_name}+{sub_attribute}"
            ER.node(sub_attribute_id, sub_attribute, shape="ellipse", fillcolor="#f9daaf", style="filled")
            ER.edge(attribute_id, sub_attribute_id)


for relations in data.get("relationships", []):
    first_entity =  relations["entity1"]
    second_entity = relations["entity2"]
    relation_name = relations["name"]
    cardinality = relations["cardinality"]
    left_cardinality, right_cardinality = cardinality.split(":")

    ER.node(relation_name , relation_name , shape="diamond", style="filled", fillcolor="#f9bcbc") #No 2 relations have same name
    ER.edge(first_entity , relation_name , label = left_cardinality)
    ER.edge(relation_name , second_entity , label = right_cardinality)

    for attribute in relations.get("attributes", []):  #relationships can have attributes
        attribute_name = attribute["name"]
        attribute_id = f"{relation_name}+{attribute_name}"
        is_pk = attribute.get("isPrimaryKey", False)
        composite = attribute.get("composite", [])
        ismulti = attribute.get("isMultiValued", False)

        if is_pk:
            ER.node(attribute_id, f"<u>{attribute_name}</u>", shape="ellipse", fillcolor="#fff2cc", style="filled")
        elif ismulti:
            ER.node(attribute_id, attribute_name, shape="ellipse", peripheries="2",  fillcolor="#fff2cc", style="filled")
        else :
            ER.node(attribute_id, attribute_name, shape="ellipse", fillcolor="#fff2cc", style="filled")

        ER.edge(relation_name, attribute_id)

        for sub_attribute in composite :
            sub_attribute_id = f"{relation_name}+{attribute_name}+{sub_attribute}"
            ER.node(sub_attribute_id, sub_attribute, shape="ellipse", fillcolor="#f9daaf", style="filled")
            ER.edge(attribute_id, sub_attribute_id)



ER.render(filename="ER-diagram/ER Diagram", view=True, cleanup=True)
```

2. Full description of code:

This Python script automatically generates an Entity-Relationship (ER) diagram from a JSON specification using the **Graphviz** library. It provides a visual representation of database schemas based on structured input data.

Core Functionality

1. Input Handling

- Accepts JSON file path as command-line argument or interactive input **(useful in part2)**
- Loads and parses JSON data containing entity and relationship definitions
- Provides flexible input methods for different usage scenarios

2. Diagram Configuration

- Initializes Graphviz diagram with PNG format output
- Sets professional styling: splined edges and white background
- Configures visual properties for optimal readability

3. Entity Processing

For each entity in the JSON data:

- Creates rectangular nodes with blue fill color
- Generates unique identifier using entity name
- Processes entity attributes with specialized rendering :
    - **Primary Keys**: Underlined text formatting
    - **Multi-valued Attributes**: Double-bordered ellipses
    - **Composite Attributes**: Hierarchical structure with connected sub-attributes
    - **Regular Attributes**: Standard elliptical nodes with yellow fill

4. Relationship Processing

For each relationship definition:

- Creates diamond-shaped nodes with red fill color
- Parses cardinality notation (e.g., "1:N", "M:N") into left/right labels
- Establishes connections between entities and relationships
- Supports relationship attributes with same styling rules as entity attributes

5. Visual Output

- Renders final diagram to "ER Diagram.png"
- Automatically opens the generated image for immediate viewing
- Cleans up temporary files after rendering

6. Key Features

Unique Identification System

- **Entities**: Use entity names as unique identifiers (because no 2 entites with same name)
- **Attributes**: Composite keys using entity+attribute format (to prevent any error occurs if 2 attribute with same name)
- **Relationships**: Relationship names as unique identifiers (because no 2 entities with same name)

Comprehensive Attribute Support

- Primary key highlighting
- Multi-valued attribute visualization
- Composite attribute decomposition
- Consistent color coding across all element types

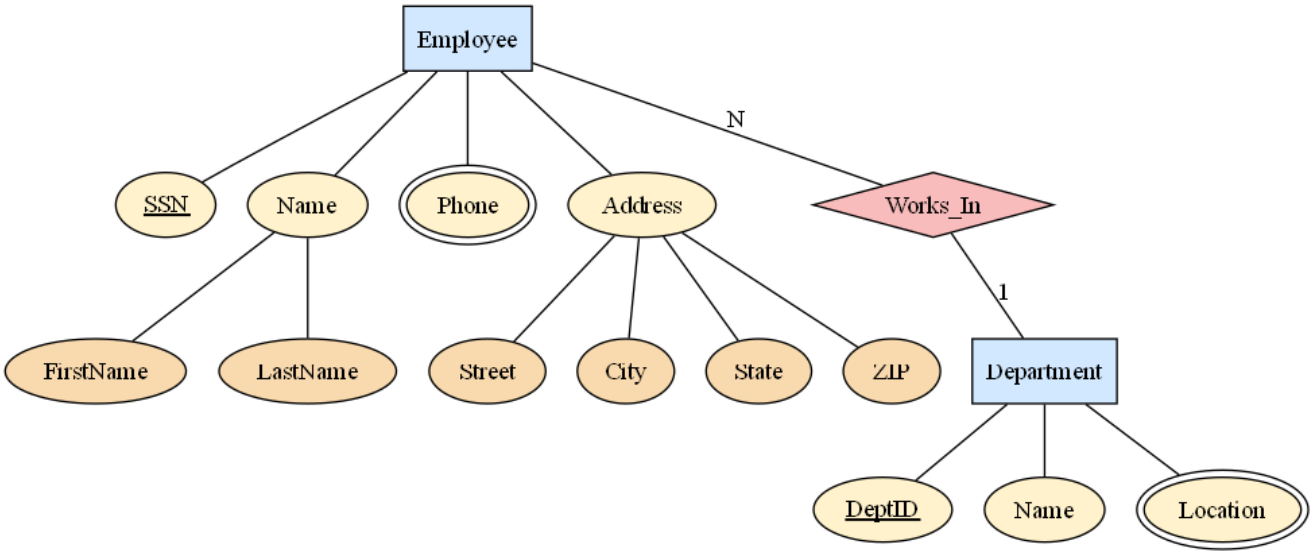
Cardinality Representation

- Automatic parsing of standard cardinality notation
- Support for all common cardinality types (1:1, 1:N, M:N)

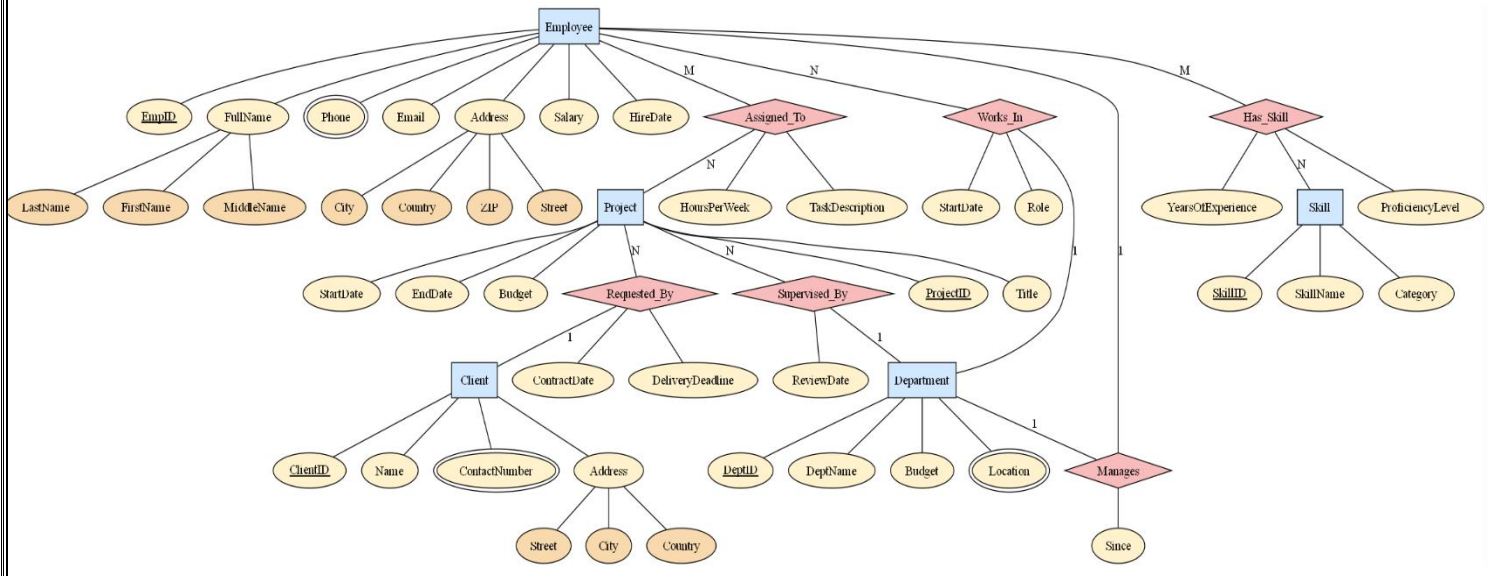
Technical Implementation

- **json**: For parsing input data
- **graphviz**: For diagram generation and rendering
- **sys**: For command-line argument handling

3. Output for given sample input:



4. Output for other complicated system:



- ## Part 2 : AI Integration with Gemini API

1. Code:

```python
import json
import re
import subprocess
import google.generativeai as genai


genai.configure(api_key="AIzaSyDGncTqq8760A9WUFLJWkxMS099mHFye_E")
model = genai.GenerativeModel("models/gemini-2.5-flash")

description = input("Enter your system description: ")

prompt = """from the given detailed english description for the system i want a json file that must strictly follow this schema :
    {
  "entities": [
    {
      "name": "Student",
      "attributes": [
        { "name": "StudentID", "isPrimaryKey": true },
        { "name": "Name", "composite": ["FirstName", "LastName"] },
        { "name": "Email" },
        { "name": "Phone", "isMultiValued": true }
      ]
    },
    {
      "name": "Course",
      "attributes": [
        { "name": "CourseID", "isPrimaryKey": true },
        { "name": "Title" },
        { "name": "Credits" }
      ]
    },
    {
      "name": "Instructor",
      "attributes": [
        { "name": "InstructorID", "isPrimaryKey": true },
        { "name": "Name" },
        { "name": "Office" }
      ]
    }
  ],
  "relationships": [
    {
      "entity1": "Student",
      "entity2": "Course",
      "name": "Enrolls",
      "cardinality": "M:N",
      "attributes": [
        { "name": "Grade" },
        { "name": "Semester" }
      ]
    },
    {
      "entity1": "Instructor",
      "entity2": "Course",
      "name": "Teaches",
      "cardinality": "1:N"
    }
  ]
}

 with some rules :
    1- no weak entites
    2- relationships can have attributes
    3- use only valid json without any comments
    """


prompt = description + prompt
response = model.generate_content(prompt)  # markdown + json part

json_match = re.search(r'\{.*\}', response.text, re.DOTALL)
if json_match:
    json_str = json_match.group()  #raw stirng json
    generated_json = json.loads(json_str)  #convert JSON string into a Python dictionary/list

    with open("ER-diagram/generated_ER.json", "w") as f:
        json.dump(generated_json, f, indent=2)
    print("ER JSON generated and saved as 'generated_ER.json'")
    subprocess.run(["python", "ER-diagram/ERdiagram.py", "ER-diagram/generated_ER.json"])
```

2. Full description of code:

This Python script implements an intelligent Entity-Relationship (ER) diagram generation system that leverages Google's Gemini AI to automatically convert natural language system descriptions into structured JSON schemas, which are then visualized as professional ER diagrams.

1. AI-Powered Natural Language Processing

- **Integration**: Utilizes **Google's Gemini 2.5 Flash model** through the Generative AI API
- **Configuration**: Secure API key authentication for AI service access

2. Intelligent Prompt Engineering

The system uses a sophisticated prompt structure containing:

Schema Template:

- Complete JSON schema with example entities (Student, Course, Instructor)
- Detailed attribute specifications:
  - Primary key identification (isPrimaryKey)
  - Composite attributes (e.g., Name → FirstName, LastName)
  - Multi-valued attributes (isMultiValued)
- Relationship definitions with cardinality constraints (1:N, M:N)
- Support for relationship-level attributes

Strict Validation Rules for JSON:

- Exclusion of weak entities
- Mandatory relationship attribute support
- Pure JSON output without comments or markdown

3. Input Processing Pipeline

User Interaction:

- Interactive prompt for natural language system description
- Flexible input accepting various description formats and detail levels

AI Processing:

- Concatenates user description with structured prompt template
- Submits to Gemini AI for intelligent schema extraction
- Handles mixed response formats (markdown + JSON)

4. Response Processing & Validation

JSON Extraction:

- Advanced regex pattern matching (r'\{.*\}') with DOTALL flag
- Robust extraction of JSON content from AI responses
- Handles various response formats and whitespace variations

Data Processing:

- Raw JSON string extraction from matched patterns
- Conversion to Python dictionary objects using json.loads()
- Structured data validation during parsing

5. File Management & Workflow Automation

Output Generation:

- Saves generated JSON to **"generated_ER.json"**
- Maintains human-readable formatting with 2-space indentation

Automated Visualization:

- Seamless integration with ER diagram generator
- Subprocess execution of visualization script
- Passes generated JSON file path as parameter
- End-to-end automation from text to visual diagram

Dependencies

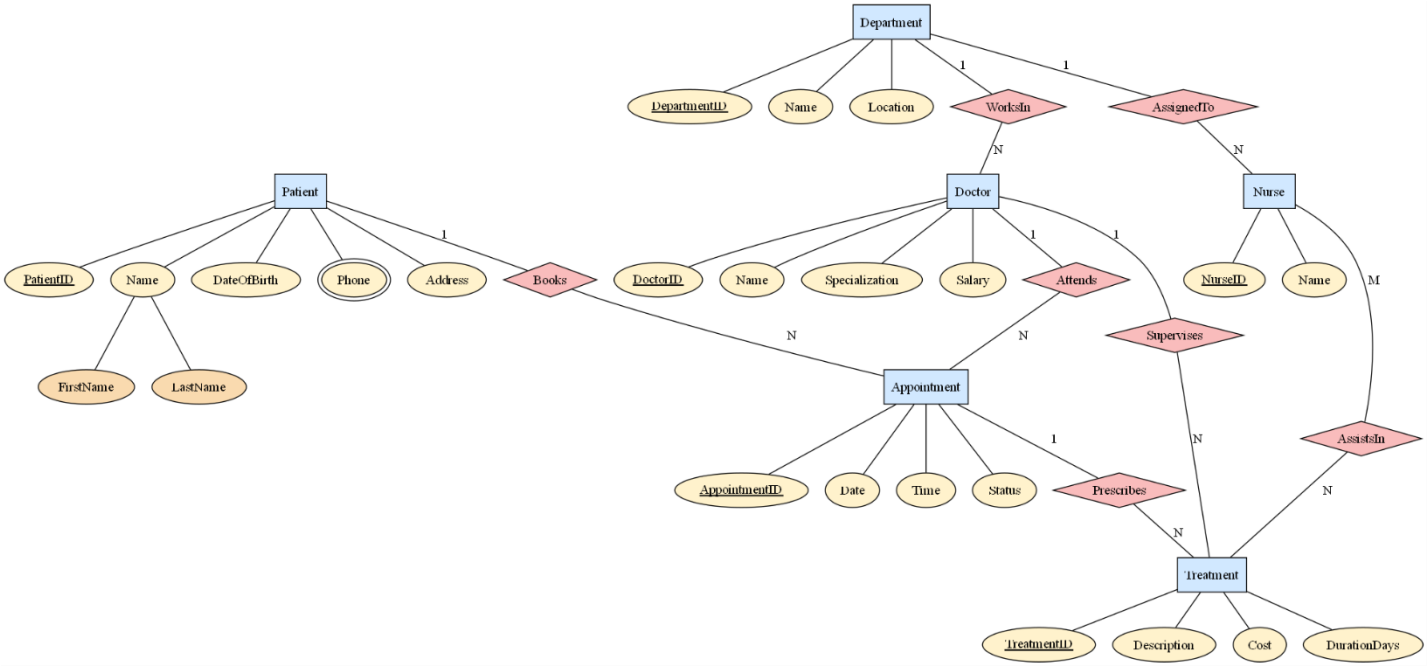
- **google.generativeai**: For AI-powered schema generation
- **json:** For data serialization/deserialization
- **re:** For robust JSON pattern matching
- **subprocess**: For workflow automation

3. Sample input :

Json description:

A hospital management system keeps track of patients, doctors, nurses, departments, appointments, and treatments. Each patient has a unique patient ID, full name (first and last), date of birth, phone numbers (a patient can have multiple), and address. Each doctor has a unique doctor ID, name, specialization, and salary. Each nurse has a nurse ID, name, and assigned department. Departments have a unique department ID, name, and location. Each doctor works in exactly one department, but a department can have many doctors. Patients book appointments with doctors. Each appointment has a date, time, and status (scheduled, completed, canceled). A treatment is prescribed during an appointment and has a treatment ID, description, cost, and duration in days. Nurses assist in treatments — a treatment can have several assisting nurses, and a nurse can assist in multiple treatments. Doctors supervise treatments, and each treatment is supervised by exactly one doctor.

ER_MODEL :


Department
DepartmentID
Name
Location
WorksIn
AssignedTo
1
1
N
N
Patient
Doctor
Nurse
PatientID
Name
DateOfBirth
Phone
Address
Books
DoctorID
Name
Specialization
Salary
Attends
NurseID
Name
M
1
1
1
FirstName
LastName
N
N
Supervises
Appointment
AssistsIn
1
N
N
AppointmentID
Date
Time
Status
Prescribes
N
Treatment
TreatmentID
Description
Cost
DurationDays