Ficha Técnica — Módulo 4: Base Orientada a Grafos (Neo4j)

Propósito del componente

Este módulo implementa el **modelo lógico y conceptual del dominio** dentro de una base de datos orientada a grafos (Neo4j).
Su propósito es:

- representar Frames, Criterios, Dependencias Funcionales, Atributos y Esquemas como nodos y relaciones;
- ejecutar evaluaciones reales de Formas Normales (1FN, 2FN, 3FN);
- soportar consultas estructuradas desde la API y el LLM;
- almacenar instancias concretas del dominio, no solo definiciones.

Es el **núcleo ejecutable del sistema experto**, donde se materializan reglas, instancias y razonamiento.

Se siguió utilizando los casos de usos planteados en el módulo 3 – Red lógica:
CU1 — Sistema evalúa una forma normal deseada (1FN, 2FN, 3FN) e informa su cumplimiento
CU2 — Usuario pide información sobre una forma normal

Entradas

Este módulo recibe datos estructurados que se transforman en nodos y relaciones:

- **Esquema a evaluar**
  - nombre del esquema
  - lista de atributos
  - atributos PK/no PK
- **Dependencias funcionales**
  - determinante → dependiente
  - tipo: Plena / Parcial / Transitiva
- **Forma normal solicitada**
  - 1FN / 2FN / 3FN (almacenada como propiedad en la instancia EVALUAR_FORMA_NORMAL)
- **Parámetros calculados**
  - cantidad de PK
  - multivaluados
  - cantidad de parciales
  - cantidad de transitivas

Salidas

La base de grafos permite obtener:

- Resultado **CUMPLE** / **NO_CUMPLE** para la forma normal solicitada.
- Relación explícita:
  - (Esquema)-[:CUMPLE]->(FN)
  - (Esquema)-[:NO_CUMPLE]->(FN)
- Motivos de incumplimiento
- Estado completo de atributos, DF y criterios.
- Consultas sobre criterios teóricos de cada FN.

Herramientas utilizadas y entorno

- **Neo4j AuraDB Free** (servicio en la nube).
- **Cypher Query Language** para:
  - creación de nodos
  - creación del metamodelo (FrameClass, Slot, Daemon)
  - instanciación
  - consultas de inferencia
- **Constraints** para evitar nodos duplicados.
- **Transacciones idempotentes** para ejecutar scripts varias veces sin errores.

Arquitectura o funcionamiento interno

La base está organizada en **tres niveles**:

🔹 A. Metamodelo (estructura general del sistema experto)

Se crean nodos de tipo:

- FrameClass
- Slot
- Daemon

Estos permiten representar:

- 1FN, 2FN, 3FN
- Criterios
- Esquemas
- Dependencias funcionales
- Slots como: sin_dependencias_parciales, pk_compuesta, etc.
- Daemons encargados de activar reglas

El metamodelo reproduce exactamente la lógica del Módulo 3.

🔹 B. Modelo del dominio (instancias reales)

Ejemplo: Esquema **Pedido**

Nodos creados:

- (:Esquema {name:'Pedido'})
- (:Atributo {name:'IDProducto', es_pk:true})
- (:Atributo {name:'IDPedido', es_pk:true})
- (:Atributo {name:'NombreProducto'})
- (:Atributo {name:'NroPedido'})
- (:Atributo {name:'Cantidad'})

Relaciones creadas:

- (:Esquema)-[:TIENE]->(:Atributo)
- (:Atributo)-[:DF {tipo:'Parcial'}]->(:Atributo)
- (:Atributo)-[:DF {tipo:'Plena'}]->(:Atributo)

Esto permite detectar automáticamente:

- multivaluados
- parciales
- transitivas
- PK compuesta

🔹 **C. Inferencia lógica (evaluación de FN en grafos)**

1. Se limpian evaluaciones anteriores (CUMPLE / NO_CUMPLE).
2. Se calculan flags:
   - pk_compuesta
   - sin_atributos_multivaluados
   - cant_df_parciales
   - cant_df_transitivas
3. Se aplican reglas:
   - Si se cumple 1FN → crear relación CUMPLE
   - Si no → NO_CUMPLE con motivo
4. Se evalúa 2FN y 3FN usando los resultados previos.
5. Se retorna una estructura como:


```
{
"esquema": "Pedido",
"forma_normal": "2FN",
"resultado": "NO_CUMPLE",
"detalle": {
"motivos": ["Tiene dependencias parciales"],
"parciales": 2
}
}
```


Ver anexo: Captura del grafo neo4j

Código relevante

El módulo se implementa mediante un archivo de Cypher.
Incluye:

- constraints
- creación del metamodelo
- generación de nodos
- creación de DF
- instanciación de esquemas
- evaluación de FN
- consultas finales

Repositorio del proyecto que contiene los comandos utilizados:
https://github.com/PauRodriguezz/EduDB-ProyectoIA/blob/main/Neo4j/setup.cypher

Ejemplo de instancia dentro del modelo conceptual

Ejemplo: Evaluar 2FN en "Pedido"
Dependencias detectadas:

- (IDProducto, IDPedido) → Cantidad (Plena)
- IDProducto → NombreProducto (Parcial)
- IDPedido → NroPedido (Parcial)

Proceso en Neo4j:

1. Se calcula que la PK es compuesta.
2. Se detectan 2 dependencias parciales.
3. Se evalúa 1FN (cumple).
4. Se evalúa 2FN → ❌ No cumple.

5. Se genera relación (Pedido)-[:NO_CUMPLE]->(2FN).
6. Se adjuntan motivos: ["Tiene dependencias parciales"].

Resultado final devuelto:

- NO CUMPLE 2FN

Resultados obtenidos

Durante las pruebas se verificó que:

- El metamodelo se creó correctamente y sin duplicados.
- La instanciación del esquema Pedido coincidió con el diseño conceptual.
- La clasificación de DF funcionó como se esperaba.
- Las relaciones CUMPLE / NO_CUMPLE se generaron con argumentos correctos.
- El motor pudo responder consultas sobre criterios teóricos.

Observaciones y sugerencias

- La lógica difusa podría integrarse como nodos adicionales o como propiedades calculadas.
- Sería útil agregar triggers automáticos que detecten claves candidatas.
- Recomendable validar DF automáticamente en el futuro (en lugar de ingresarlas manualmente).
- El grafo está preparado para escalar a mayor cantidad de esquemas y evaluaciones.

Anexo

📌 **Captura del grafo en Neo4j**