Trabajo Final BD II

Alumnos: Mauro Barrales, Nicolas Gonzalez e Ignacio Rodriguez

Institución: UCU

Curso: Analista en Informática

Profesores: Bernardo Rychtenberg y Javier Yannone

Maldonado, Uruguay

Fecha: 09/07/2025

INFORMACIÓN TEÓRICA

El diseño de bases de datos es un proceso en el que los requisitos de información de una organización o cliente se transforman en un esquema de base de datos eficiente. Este proceso se divide en tres partes, Diseño Conceptual (MER), Diseño Lógico (LOD) y Diseño Físico (DB Engine Oriented).

Diseño conceptual

El objetivo del diseño conceptual es crear una representación de alto nivel de la información necesaria para resolver un problema, que es independiente de cualquier software o hardware específico. Se centra en qué datos se necesitan y cómo se relacionan entre sí, sin preocuparse por los detalles técnicos.

- Proceso:
    1. **Recopilación de Requerimientos:** Entender a fondo las necesidades de información de los usuarios y las reglas de negocio. Esto implica entrevistas, análisis de documentos existentes, observación, etc.
    2. **Identificación de Entidades y Atributos:** Reconocer los objetos de la realidad del modelo de dominio.
    3. **Identificación de Relaciones y Cardinalidades:** Determinar cómo se asocian las entidades y especificar las restricciones de cardinalidad para cada relación.
    4. **Uso de Modelos de Datos Conceptuales:** El Modelo Entidad-Relación (MER) es la herramienta estándar para esta etapa.

5. **Resultado:** Un diagrama MER detallado, comprensible tanto para los analistas como para los usuarios finales. Este modelo sirve para seguir adelante con las fases posteriores del diseño.

Diseño Lógico

La fase de diseño lógico toma el esquema conceptual MER y lo transforma en un esquema de base de datos como el modelo relacional. Aunque es independiente de un SGBD particular, ya está atado a las estructuras y reglas de un modelo de datos.

- **Proceso:**
    1. Mapeo del Modelo Conceptual al Modelo Lógico: Este es el paso central. Se aplican reglas de transformación para convertir entidades, atributos y relaciones del MER en tablas (entidades y relaciones), columnas (atributos), claves primarias y claves foráneas del modelo relacional.
        - Cada tipo de entidad fuerte se convierte en una tabla.
        - Cada tipo de entidad débil se convierte en una tabla con una clave foránea a su entidad propietaria.
        - Los atributos simples se convierten en columnas.
        - Los atributos compuestos se mapean a múltiples columnas simples.
        - Los atributos multivaluados generalmente requieren una nueva tabla.

- Las relaciones 1:1 y 1:N se pueden mapear añadiendo una clave foránea a una de las tablas.
- Las relaciones M:N generalmente se mapean a una nueva tabla de enlace que contiene las claves primarias de las entidades participantes como claves foráneas.
- La generalización/especialización se puede mapear de varias maneras (una tabla por jerarquía, una tabla por superclase, una tabla por subclase).

2. **Normalización del Esquema:** Una vez que el esquema inicial del modelo relacional ha sido generado, se aplica el proceso de normalización para asegurar que las tablas resultantes cumplan con las formas normales deseadas (generalmente 3FN o FNBC). Esto implica analizar las dependencias funcionales y descomponer las tablas si es necesario para eliminar redundancias y anomalías.
3. **Selección de Tipos de Datos:** Asignar tipos de datos apropiados a cada columna (ej., VARCHAR, INT, DATE, BOOLEAN).
4. **Definición de Restricciones de Integridad:** Establecer restricciones adicionales como NOT NULL, UNIQUE, CHECK y restricciones de clave foránea.

Diseño Físico:

El diseño Físico es la etapa final de diseño, en esta se toma el esquema del diseño lógico y se transforma en una implementación orientado a la eficiencia del SGBD que se va a utilizar. Se ocupa de maximizar el rendimiento y como los datos se almacenan y manipulan utilizando los recursos del SGBD.

Se tienen en cuenta los Índices, los tipos de datos para almacenarlos en las tablas, vistas, particionamiento de disco y diferentes restricciones de seguridad e Integridad (además de las del negocio) para la base de datos como por ejemplo la unicidad y no nulidad.

Modelo Entidad-Relación (MER)

El Modelo Entidad-Relación es una herramienta fundamental en el diseño de bases de datos, utilizada en la etapa de diseño conceptual. Su propósito principal es modelar la estructura de los datos de una organización. Representa el mundo real mediante componentes: entidades, atributos y relaciones.

Entidad Fuerte

Una entidad representa un objeto de la realidad. Estos pueden ser tangibles, como un EMPLEADO, PRODUCTO o DEPARTAMENTO, o intangibles, como una ORDEN de compra o un CURSO universitario. En el MER, se trabaja con tipos de entidades, que son un grupo de instancias que comparten las mismas características. Por ejemplo, EMPLEADO es un tipo de entidad que agrupa a todos los empleados de una empresa.

Entidad Débil

Una entidad débil es una entidad que no se puede identificar por sí misma, y necesita de otra entidad para ser identificada. No tiene clave primaria propia, si no que necesita la de la entidad fuerte con la que tiene relación. Por ejemplo. HOSPITAL tiene HABITACIONES, HABITACIÓN es la entidad débil. Si una HABITACIÓN se identifica solo por un número (ej., "Habitación 101"), ese número por sí solo no es único, ya que podría haber una "Habitación 101" en el Hospital A y otra "Habitación 101" en el Hospital B.

Atributos

Los atributos son las propiedades o características que describen un tipo de entidad o un tipo de relación. Cada atributo tiene un dominio, que es el conjunto de valores permitidos para ese atributo.

- **Atributo Clave Primaria (PK - Primary Key):** Es un atributo o grupo de atributos que permiten identificar de forma única cada entidad. Deben ser únicos y no nulos. Por ejemplo, DNI para una Persona.
- **Atributo Compuesto:** Es un atributo que puede ser separado en varios atributos más simples, cada uno con su propio significado. Por ejemplo, el atributo Dirección podría estar compuesto por Número_Casa, Calle, Ciudad, Provincia y Código_Postal. Al trabajar con atributos compuestos, se puede acceder a sus componentes de forma individual si es necesario.
- **Atributo Multivaluado:** Es un atributo que puede tener muchos valores para una sola entidad. Permite almacenar colecciones de

valores. Por ejemplo, un EMPLEADO puede tener muchos Numeros_De_Telefono. En el modelo relacional, suelen tener una tabla aparte para su almacenamiento.

- **Atributo Derivado:** Es un atributo que puede ser calculado en base a otros atributos. Por ejemplo, la edad de una persona se puede calcular mediante la fecha de nacimiento y la fecha actual.

Relaciones y Relaciones Ternarias

Las relaciones representan las asociaciones significativas entre dos o más tipos de entidades. Un tipo de relación define la naturaleza de la asociación. Por ejemplo, TRABAJA_EN es un tipo de relación entre EMPLEADO y DEPARTAMENTO.

- **Grado de una Relación:** Se refiere al número de tipos de entidades que participan en una relación.
    - **Relaciones Binarias:** Son las más comunes, involucrando dos tipos de entidad (ej., EMPLEADO TRABAJA_EN PROYECTO).
    - **Relaciones Ternarias:** Involucran tres tipos de entidad. Son útiles cuando no se puede representar la lógica mediante relaciones binarias. Un ejemplo es PROVEEDOR que SUMINISTRA PIEZA a PROYECTO. La cantidad de una pieza suministrada depende de la combinación específica de proveedor, pieza y proyecto.

- **Relaciones de Grado Superior:** Aunque menos comunes, pueden existir relaciones cuaternarias o de mayor grado.

Cardinalidad

La cardinalidad define el número de instancias de un tipo de entidad que pueden o deben asociarse con una instancia de otro tipo de entidad. Se expresa comúnmente como un par (min, max), donde:

Las restricciones más comunes son:

- **Uno a Uno (1:1):** Cada instancia de una entidad A se asocia con un máximo de una instancia de la entidad B, y viceversa. Ejemplo: EMPLEADO DIRIGE DEPARTAMENTO (un departamento es dirigido por un solo empleado, y un empleado dirige un solo departamento).
- **Uno a Muchos (1:N):** Una instancia de la entidad A se asocia con múltiples instancias de la entidad B, pero cada instancia de B se asocia con un máximo de una instancia de A. Ejemplo: DEPARTAMENTO TIENE EMPLEADOS (un departamento tiene muchos empleados, pero un empleado trabaja en un solo departamento).
- **Muchos a Muchos (M:N):** Una instancia de la entidad A se asocia con múltiples instancias de la entidad B, y una instancia de B se asocia con múltiples instancias de A. Ejemplo: ESTUDIANTE SE_MATRICULA_EN CURSO (un estudiante se matricula en muchos cursos, y un curso tiene muchos estudiantes matriculados).

Generalización y Especialización

Generalización: Es cuando se identifican las características comunes (atributos y/o relaciones) entre varios tipos de subentidades y se combinan en una superclase. La superclase "generaliza" las características compartidas. Por ejemplo, CARRO, CAMIÓN y MOTOCICLETA pueden generalizarse en VEHÍCULO, que tendrá atributos y relaciones comunes a todos ellos (como Número_De_Serie o COLOR).

Especialización: Es el proceso inverso a la generalización. Implica la definición de subtipos de un tipo de entidad existente (superclase) basándose en atributos distintivos o roles específicos. Las subclases heredan todos los atributos y relaciones de su superclase, y pueden tener atributos o relaciones adicionales que son específicas de ellas. Por ejemplo, EMPLEADO puede especializarse en EMPLEADO_FIJO, EMPLEADO_POR_HORAS o CONSULTOR, cada uno con sus propias características.

Normalización

La normalización es un proceso para diseñar bases de datos relacionales eficientes y con integridad de datos. Su objetivo es reducir la redundancia de datos, eliminar anomalías de actualización (inserción, borrado, modificación) y asegurar la consistencia de los datos. Se logra descomponiendo relaciones (tablas) problemáticas en relaciones más pequeñas y Formas Normales (FN)

Las formas normales son un conjunto de reglas o condiciones progresivamente más estrictas que una relación debe satisfacer para ser considerada "normalizada".

1. Primera Forma Normal (1FN): Una relación está en 1FN si y sólo si todos los dominios de sus atributos contienen únicamente valores atómicos. Esto significa que no hay atributos multivaluados ni atributos compuestos con anidamientos complejos dentro de una celda. Cada celda de la tabla debe contener un único valor.
2. Segunda Forma Normal (2FN): Una relación está en 2FN si está en 1FN y todos sus atributos son completamente dependientes de la clave primaria. Esto significa que si la clave primaria es compuesta, ningún atributo fuera de la clave primaria puede depender solo de una parte de la clave primaria.
3. Tercera Forma Normal (3FN): Una relación está en 3FN si está en 2FN y no contiene dependencias funcionales transitivas de atributos con respecto a la clave primaria. Una dependencia transitiva ocurre cuando un atributo no clave depende de otro no clave , que a su vez depende de la clave primaria. Por lo tanto, ningún atributo no clave debe depender de otro atributo no clave.
4. Forma Normal de Boyce-Codd (FNBC): Una relación está en forma normal Boyce/Codd (BCNF) si y sólo si todo determinante es una clave candidata

5. Cuarta Forma Normal (4FN): Una tabla está en Cuarta Forma Normal (4FN) si, además de cumplir con la Forma Normal de Boyce-Codd (FNBC), elimina las dependencias multivaluadas (DMV) no triviales. Si un atributo A determina múltiples valores para otro atributo B de forma independiente de otros atributos en la tabla, entonces A debe ser la clave completa de la tabla.
6. Quinta Forma Normal (5FN): Una relación está en 5FN si está en 4FN y no puede ser descompuesta en tablas más pequeñas sin perder información ni crear datos falsos al volver a unirlas.

El objetivo de la normalización es alcanzar una forma normal que sea adecuada para la aplicación, típicamente 3FN o FNBC, ya que ir más allá puede aumentar la complejidad y afectar el rendimiento.

Patrones: DTO, Mapper, DAO y MVC

DTO (Data Transfer Object)

El patrón DTO se usa para enviar datos entre capas como cliente-servidor, ayuda a que al devolver un objeto por ejemplo no se muestre toda su estructura, de esta forma, un backend puede enviar un DTO con solo los datos necesarios al frontend.

Mapper

Los mappers convierten objetos a diferentes formatos, como entre las entidades que se usan en la base de datos y los DTOs que se envían. Hay librerías que automatizan el trabajo, como MapStruct.

DAO (Data Access Object)

DAO es un patrón que encapsula el acceso a la base de datos. Proporciona métodos para crear, leer, actualizar y borrar datos, pero sin que la lógica del negocio tenga que preocuparse por cómo se hacen esas operaciones.

Esto permite cambiar la base de datos o la forma de acceso sin afectar otras partes del sistema.

MVC (Modelo-Vista-Controlador)

Modelo-Vista-Controlador (MVC) es un patrón de arquitectura utilizado para separar la lógica de negocio y la gestión de datos de la forma en que se muestra la información al usuario. Es una forma de organizar una aplicación para que cada parte se centre en una tarea específica.

El MVC se compone de tres elementos clave:

- **Modelo:** este componente representa los datos y toda la lógica de negocio. Su tarea es gestionar y conocer todo sobre la información: cómo se accede a ella, cómo se actualiza y cuáles son las reglas que la rigen.

- **Vista:** La Vista se encarga de la presentación visual del Modelo. Su función es definir exactamente cómo deben mostrarse los datos al usuario. Si los datos del Modelo cambian (por ejemplo, si cambia el precio de un producto), la Vista se encarga de reflejar esos cambios en la pantalla. Esto puede ocurrir de dos maneras: la Vista puede "escuchar" al Modelo para alertarle de los cambios, o puede pedir al Modelo la información más reciente cuando la necesite.
- **Controlador:** Actúa como intermediario. Su trabajo es tomar las interacciones del usuario con la vista y convertirlas en acciones a realizar por el Modelo. Por ejemplo, en una interfaz gráfica, un clic en un botón o la selección de un menú son interacciones. En una aplicación web, estas interacciones suelen ser peticiones HTTP (GET y POST).

Tecnologías Web

React

React es una biblioteca JavaScript para construir interfaces de usuario. Se basa en componentes reutilizables y utiliza un DOM virtual para mejorar el rendimiento.

React Router DOM

Es una librería que permite gestionar rutas dentro de una aplicación React, para poder navegar entre diferentes pantallas sin recargar la página.

Tailwind CSS

Tailwind es un framework CSS que ofrece clases predefinidas para diseñar de forma rápida y responsive sin tener que escribir CSS.

Axios

Axios es una librería JavaScript que permite realizar peticiones HTTP asíncronas a APIs. Facilita el envío y la recepción de datos, la gestión de errores y el trabajo con promesas.

Figma

Figma es una herramienta de diseño colaborativo basada en la nube. Permite prototipar, compartir y trabajar en equipo al mismo tiempo, lo que es útil para proyectos de desarrollo UX/UI.

ALTERNATIVAS REALIZADAS

Distintos MER, JPA/SIN JPA

Diferentes alternativas fueron surgiendo cuando iniciamos el proyecto y sobre la marcha del mismo. Primero intentamos adquirir toda la información que consideramos relevante en la consigna para poder llevar a cabo un registro de todas las entidades y posibles relaciones que podría tener nuestro primer MER.

Luego realizamos nuestro primer MER donde nosotros creíamos que abarcamos gran parte del problema a solucionar, para luego ir evolucionandolo.

MER 1:

https://github.com/NicooGon/INFORME_BDII/blob/dev/MER/MER%201.png

Este modelo no fue la solución porque no abarcaba todo lo necesario, sobraban generalizaciones. Además, hay relaciones redundantes como ELECCION - PAPELETA o ELECCION-LISTA. También nos dimos cuenta que a través de algunas relaciones, no podíamos hacer consultas requeridas por el obligatorio.

MER 4:

https://github.com/NicooGon/INFORME_BDII/blob/dev/MER/MER%204.png

Aunque habíamos solucionado ciertos problemas con respecto a generalizaciones y entidades que no hacían falta o eran irrelevantes, todavía tenía ciertas fallas para llegar a adquirir los resultados de los votos que te pedía la consigna (por departamento y circuitos), errores de cardinalidades para representar los circuitos de una elección, loop entre ciudadano - elección - credencial - circuito.

También en este diagrama incluimos una especie de Modelo Relacional debajo del MER para organizarnos con las tablas que teníamos que realizar con sus respectivos atributos, Primary Keys y Foreign Keys. Esto para que luego sea un poco más simple realizar el Script SQL.

MER 5:

https://github.com/NicooGon/INFORME_BDII/blob/dev/MER/MER5_Elecciones.drawio.png

Este fue el diseño final y el implementado. Este diseño fue ideal debido a que nos permitió cumplir con todos los requerimientos del sistema del obligatorio. La tabla lista_politico nos permitió obtener los resultados de las elecciones (algo que con el MER anterior no se lograba). Además lista ahora sería una entidad débil con respecto a elección y departamento. Por lo tanto, para poder identificarla se necesitaba la eleccion_id y el departamento_id). Otro cambio significativo, el cambio de circuito a entidad débil, nos permitió saber a qué elección pertenecía, relacionando los votos con el mismo, lo cual es más coherente ya que las personas votan en un circuito y los resultados se obtienen por circuito. Es importante mencionar que el MER cumple con las cinco formas normales, garantizando un diseño eficiente, sin redundancias ni inconsistencias.

El diseño 2 y 3[1] No lo incluimos en el informe debido a que el 2 es similar al 3 pero menos completo. Lo mismo pasa con el 4 y 5, el 5 tiene algunas correcciones más y es el que utilizamos durante el 70% del proyecto.

En base al MER 1 realizamos un Script SQL para probar algunas funcionalidades básicas, después a medida que fuimos modificando nuestros MER también implementamos los cambios de las tablas a nuestro Script, cabe recalcar que este Script está realizado en el Gestor de BD MySQL.

En este caso no tenemos toda la evolución del Script ya que lo fuimos sobreescribiendo.

[1] En el caso de querer ver todos los MER: https://github.com/NicooGon/INFORME_BDII/tree/dev/MER

ScriptSQL:

https://github.com/NicooGon/INFORME_BDII/blob/dev/Scripts%20SQL/Script-Elecciones-Completo.sql

Este Script tiene datos ya precargados de ciudadanos, circuitos, elecciones, credenciales, listas, partidos políticos, etc. Esto porque nos pareció más fácil para probar la interfaz gráfica de la que vamos a hablar a continuación, aunque claramente en nuestro sistema se pueden crear todas estas cosas. Por ejemplo, los datos de los ciudadanos para ingresar son los que se encuentran ahí, siendo el ciudadano con poder de gestionar todo el sistema el de la credencial con Serie '1' y Número 12345.

Frontend

La alternativa en cuanto a interfaz gráfica, desarrollamos un prototipo en Figma[2], el cual tuvo cambios con respecto al mismo. Agregamos algunas páginas más para representar resultados y algunos botones más que nos dimos cuenta que necesitábamos a medida que realizamos la interfaz gráfica y revisamos que cumpliera con la letra del obligatorio.

Para la solución de la interfaz, utilizamos React debido a que es fácil la navegación entre páginas y la reutilización de componentes como botones, tablas, etc. Usamos tailwind para poder estilizar de forma más sencilla (ya que CSS a veces se complica).

---

[2] https://www.figma.com/design/m7B90e71parhlDiRgo7Ihy/BD-II-TP?node-id=0-1&p=f&t=YSIcBegkS6xyuq94-0

Para la navegación entre páginas usamos react router dom, el cual permite crear rutas dinámicas pasando variables por el path, lo cual es útil para acceder a páginas que tienen que ver con otras. Por ejemplo: Entrar al circuito 1 de la elección 2025.

Para las tablas utilizamos librerías que nos daban el componente hecho, por lo cual solo debíamos insertarles los datos necesarios. Lo mismo con los botones que abren “Popups”, son sacados de una librería que dan el componente hecho.

Backend:

Implementamos una solución siguiendo el patrón de arquitectura MVC. Además, utilizamos otros patrones como DTO, Mapper y DAO. Usamos Spring Boot como framework y trabajamos con JDBC para conectarnos directamente a la base de datos, evitando el uso de ORMs como JPA.

Aquí en realidad nosotros nos encontramos con algo que nos parecía un problema pero tal vez no era uno, ya que básicamente en nuestro frontend consumiamos los endpoints del back y estos devolvian los objetos que queríamos pero con los ids del objeto relacionado, es decir, no venía la data del objeto entero que estaba relacionado, lo que podía causar bastantes llamadas al back en ciertas páginas del front, es por esto que de JDBC pasamos a utilizar el ORM JPA con el @Entity en cada entidad, pero después de refactorizar una gran mayoría del código nos encontramos con problemas en cuanto a utilizar claves compuestas, realizar relaciones hacia entidades con claves compuestas y la herencia la cual no

pudimos implementar de ninguna manera, entonces básicamente volvimos los commits del ORM hacia atrás y seguimos con el JDBC, por suerte, habíamos creado una rama llamada jpa para el cambio de JDBC al ORM, por lo que en dev no realizamos ningún commit. Todas las consultas SQL fueron escritas “a mano” dentro de los DAOs. Cada DAO implementa su propia interfaz, lo que facilitó la inyección de dependencias y el cambio de implementación si fuera necesario. Los servicios, por su parte, también cuentan con una interfaz propia, y son los encargados de contener la lógica de negocio. Los controladores utilizan esas interfaces de los servicios para manejar las solicitudes del usuario.

En cuanto a los model, cada uno tiene asociado un DTO y un Mapper realizado con la dependencia MapStruct. Esto nos permitió manejar los datos de forma más segura, evitando exponer directamente las entidades y manteniendo el control sobre qué información se envía o recibe.

Conclusión

En conclusión, creemos que cumplimos con los requerimientos del obligatorio, desarrollando una solución adecuada a la propuesta. Durante el proyecto atravesamos etapas de análisis, diseño, error, implementación y prueba, lo que nos permitió mejorar la arquitectura de la aplicación. Además, el feedback dado por los profesores a las propuestas que mostrabamos fue muy útil para corregir nuestra solución.

Algunas cosas que nos hubiera gustado agregar, pero quedaron pendientes a futuro (como mejora de este “prototipo”), es poder registrar al usuario completo

(por ahora solo está la funcionalidad de loguearse con credenciales pre-cargadas en la base de datos), y poder hacer distintos tipos de elecciones.

Webgrafía

- Elmasri, R., & Navathe, S. B. (2025). *Fundamentos de sistemas de bases de datos* (5a ed.). Recuperado de https://ia802808.us.archive.org/8/items/fundamentosdesistemasdebasesdedatos/Fundamentos-de-Sistemas-de-Bases-de-Datos.pdf
- React. (2025). *Getting started*. Recuperado el 9 de julio de 2025, de https://reactjs.org/docs/getting-started.html
- React Router. (2025). *Main documentation*. Recuperado el 9 de julio de 2025, de https://reactrouter.com/en/main
- Tailwind CSS. (2025). *Documentation*. Recuperado el 9 de julio de 2025, de https://tailwindcss.com/docs
- Axios. (2025). *Introduction*. Recuperado el 9 de julio de 2025, de https://axios-http.com/docs/intro
- Figma. (2025). *Getting started with Figma*. Recuperado el 9 de julio de 2025, de https://help.figma.com/hc/en-us/articles/360040529373-Getting-started-with-Figma