BASES DE DATOS 2

Práctica II

Grupo Miércoles A

28/02/2024

PABLO MORENO MUÑOZ 841972@unizar.es

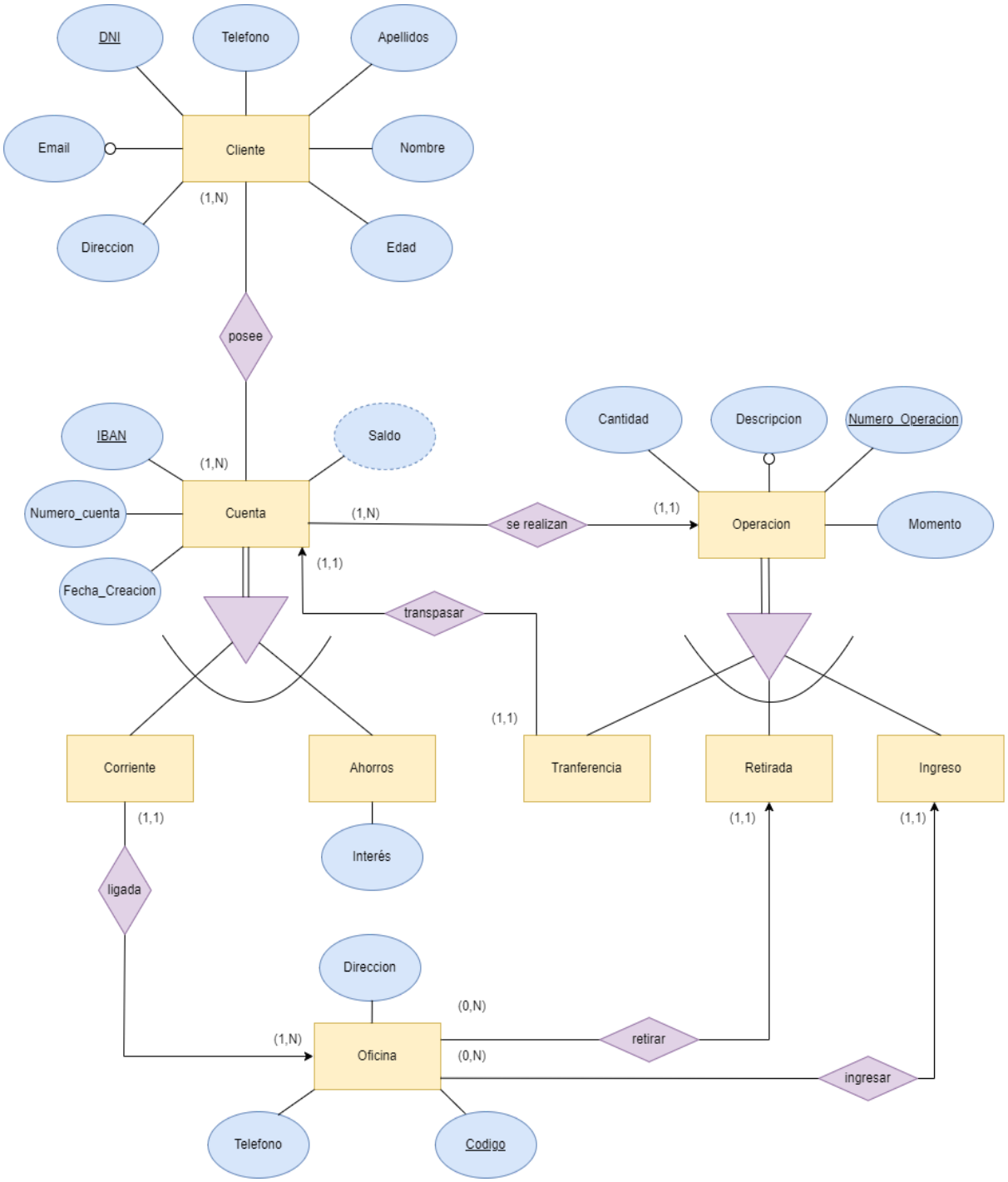
ANDREI DUMBRAVA LUCA 844417@unizar.es

GUILLERMO BAJO LABORDA, 842748@unizar.es

Índice

Diseño conceptual de la base de datos

El diseño conceptual de una base de datos es un proceso crucial donde se establecen las entidades clave y sus relaciones de manera abstracta. En este contexto, el modelo Entidad-Relación (E/R) emerge como una herramienta fundamental para representar la estructura conceptual de la base de datos. A continuación, se muestra el diseño conceptual de la base de datos:



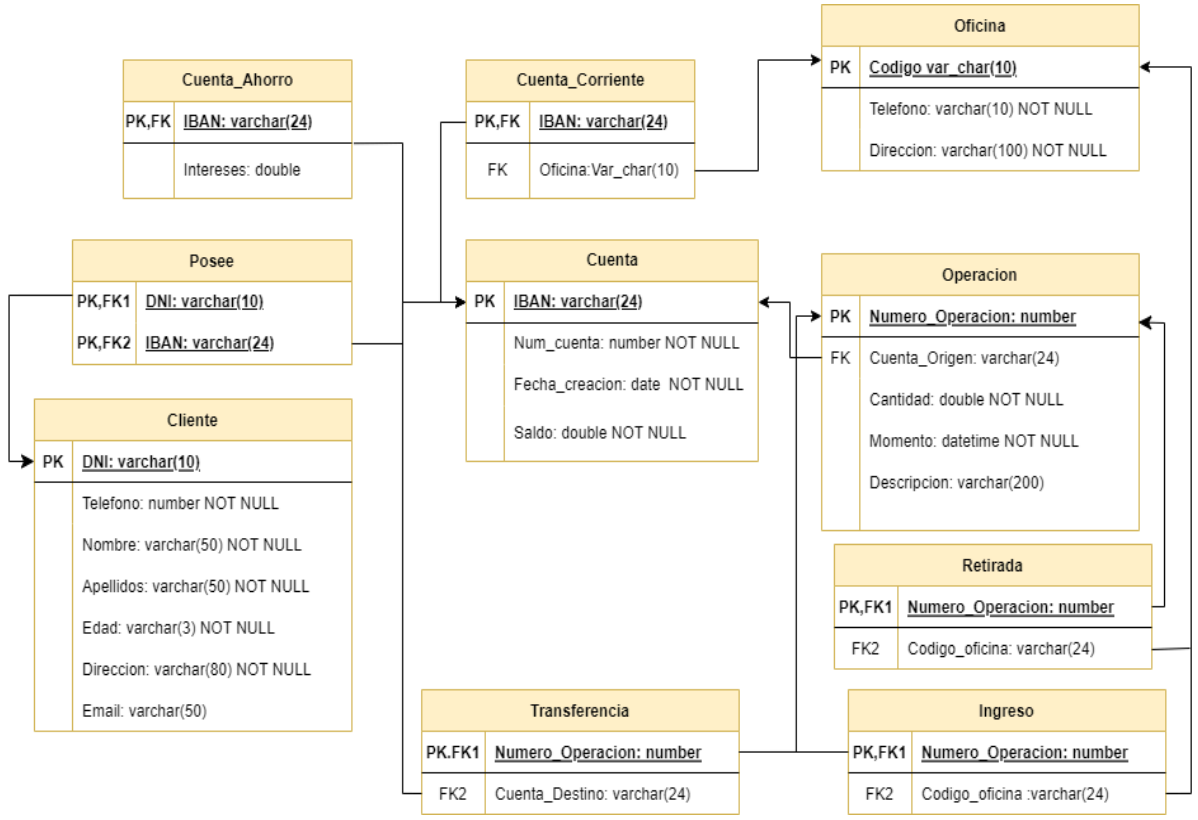
Como se puede apreciar en el esquema, aparecen un total de 9 entidades, cada una con sus respectivos atributos y relacionadas entre sí. Los atributos subrayados son únicos y representan la clave primaria de cada entidad. Aquellos ligados con un circulo son atributos opcionales. En este diseño aparecen dos especializaciones, por una parte, distinguimos dos tipos de Cuentas (Corrientes y de Ahorros). Apreciamos que la especialización es disjunta, porque una cuenta no puede pertenecer a ambas especializaciones a la vez, y de cobertura total, puesto que toda cuenta debe ser o bien de ahorros, o bien corriente. La entidad Operación también se trata de una especialización disjunta de cobertura total, en la que disponemos de las especializaciones Transferencia, Retirada e Ingreso.

Respecto a las decisiones de diseño, ha habido ciertos aspectos que han causado duda a la hora de realizar este primer diseño inicial. Por una parte, se consideró que no era necesario relacionar la entidad *Oficina* con *Retirada* e *Ingreso*, pero finalmente se hizo porque la oficina asociada a la cuenta no tiene por qué ser la sucursal desde la cual se retira/ingresa el dinero, sino que puede ser otra distinta. Por otra parte, respecto al atributo *Saldo* de la entidad *Cuenta* se ha consensuado que debe ser un atributo derivado, puesto que puede ser calculado a partir de las diferentes entradas de las tablas *Operación*, *Transferencia*, *Retirada…*

Otro aspecto que dio lugar a ciertas dudas inicialmente fue la opción de añadir a la entidad Transferencia el atributo Cuenta_Origen, lo cual podría simplificar las consultas. Sin embargo, esto solo evitaría la realización de un join, y dado que significaría la existencia de redundancia en la base de datos, esta opción finalmente se descartó.

Diseño lógico de una base de datos relacional

El diseño lógico de una base de datos es la etapa en la que se traduce el modelo conceptual, basado en el modelo Entidad-Relación (E/R), a un modelo de datos específico para el sistema de gestión de bases de datos (SGBD) que se utilizará. En esta fase, se definen las tablas, los atributos, las claves primarias y foráneas, así como las restricciones de integridad necesarias para representar con precisión la estructura conceptual en un esquema que pueda ser implementado en un SGBD. Este proceso de diseño lógico es fundamental para garantizar la eficiencia, la integridad y la consistencia de los datos dentro del sistema de base de datos. A continuación, se muestra el diseño lógico de la base de datos, así como diversas decisiones de diseño tomadas:



Como se puede observar en el diseño, se ha optado por el mismo método a la hora de pasar a relacional ambas especializaciones. Se ha creado la entidad padre con un identificador único y los atributos correspondiente, y por otra parte, las entidades hijas con ese identificador como clave foránea (que a su vez es clave primaria de la entidad hija), y atributos específicos de cada entidad hija, como bien puede ser la cuenta destino en la entidad *Transferencia* o el código de oficina en la entidad *Ingreso*.

Implementación con el modelo relacional

La implementación con el Modelo Relacional marca una etapa crucial en el desarrollo de una base de datos, donde el diseño lógico conceptual se traduce en estructuras concretas de tablas, columnas y relaciones entre estas, utilizando el lenguaje estándar SQL (Structured Query Language). Esta fase no solo implica la creación de tablas que representen las entidades y relaciones definidas en el modelo conceptual, sino también la incorporación de restricciones de integridad referencial y otros elementos que garanticen la consistencia y la fiabilidad de los datos.

Una de estas restricciones se ha aplicado en la tabla Cuenta_Ahorro, donde se ha agregado un check al campo Interes. Este check asegura que el valor del interés sea mayor o igual a cero, evitando valores negativos y garantizando la coherencia de los datos financieros.

Además, se han implementado triggers para mantener la integridad de los datos en la base de datos. Por ejemplo, el trigger verificar_iban se activa antes de insertar o actualizar una fila en la tabla Cuenta_Corriente. Su función es verificar si el IBAN proporcionado ya está asociado a una cuenta de ahorro. Si se encuentra una cuenta de ahorro asociada al mismo IBAN, se lanza un error para evitar asociar una cuenta corriente a un IBAN que ya tiene una cuenta de ahorro.

De manera similar, el trigger verificar_iban_ahorro se activa antes de insertar o actualizar una fila en la tabla Cuenta_Ahorro. Su objetivo es verificar si el IBAN proporcionado ya está asociado a una cuenta corriente. Si se encuentra una cuenta corriente asociada al mismo IBAN, se lanza un error para evitar asociar una cuenta de ahorro a un IBAN que ya tiene una cuenta corriente.

Estos triggers aseguran que no se produzcan conflictos al asociar cuentas corrientes y cuentas de ahorro al mismo IBAN, contribuyendo así a mantener la coherencia y fiabilidad de los datos en la base de datos.

Esta implementación conjunta de estructuras de tablas, restricciones y triggers fortalece la integridad de los datos y su coherencia dentro del sistema de gestión de la base de datos.

Diseño lógico de una base de datos objeto/relacional

Para el diseño lógico de una base de datos siguiendo el enfoque objeto/relacional se han seguido las pautas vistas en clase, utilizando el estándar de SQL:99, y aprovechando todas las ventajas que nos permite realizar, así como la herencia de tipos o el uso de tablas tipadas con arrays de referencia a otros atributos de objetos distintos.

Para ello se ha diseñado el siguiente esquema:

Cabe destacar que para especificar el tamaño del array se dejará vacío, siendo un tamaño variable, dependiendo del SGBD se podrá aplicar un array dinámico o no, siendo en caso contrario necesario especificar un valor para el array. Notación: '[]'

TIPOS:

```
Create type cuentaUDT as (
        iban varchar(34),
        fecha_creacion date,
        saldo double,
        refCliente ref(clienteUDT) scope cliente array[]
                references are checked on delete no action
) instantiable not final ref is system generated;

Create type oficinaUDT as (
        codigo varchar(10),
        direccion varchar(100),
        telefono varchar(9)
) instantiable not final ref is system generated;

Create type clienteUDT as (
        dni varchar(9),
        nombre varchar(50),
        apellidos varchar(50),
        edad integer,
        email varchar(50),
        telefono varchar(9),
        refCuenta ref(cuentaUDT) scope cuenta array[]
                references are checked on delete restrinct
) instantiable not final ref is system generated;
```

```
Create type corrienteUDT under cuentaUDT as (
        refOficina ref(oficinaUDT) scope oficina
                references are checked on delete set null
) instantiable not final ref is system generated;

Create type ahorroUDT under cuentaUDT as (
        interes double
) instantiable not final ref is system generated;

Create type operacionUDT as (
        numero_operacion integer,
        fecha date,
        cantidad double,
        descripcion varchar(100)
)  instantiable not final ref is system generated;

Create type ingresoUDT under  operacionUDT as (
        refOficina ref(oficinaUDT) scope oficina
                references are checked on delete set null
)  instantiable not final ref is system generated;

Create type retiradaUDT under  operacionUDT as (
        refOficina ref(oficinaUDT) scope oficina
                references are checked on delete set null
)  instantiable not final ref is system generated;

Create type transferenciaUDT under  operacionUDT as (
        refCuenta ref(cuentaUDT) scope cuenta
                references are checked on delete set null
)  instantiable not final ref is system generated;


## TABLAS:
Create table oficina of oficinaUDT (
        codigo primary key,
        direccion not null,
        telefono not null,
        ref is oficinaID system generated
);
```

```
Create table cliente of clienteUDT (
        dni primary key,
        nombre not null,
        apellidos not null,
        edad not null,
        email not null,
        telefono not null,
        ref is clienteID system generated
);

Create table cuenta of cuentaUDT (
        iban primary key,
        fecha_creacion not null,
        saldo not null,
        ref is cuentaID system generated
);


Create table cuentaCorriente of corrienteUDT under cuenta;
Create table cuentaAhorro of ahorroUDT under cuenta(
        interes not null
);

Create table operacion of operacionUDT (
        numero_operacion primary key,
        fecha not null,
        cantidad not null,
        ref is operacionID system generated
);
Create table transferencia of transferenciaUDT under operacion;
Create table retirada of retiradaUDT under operacion;
Create table ingreso of ingreso UDT under operacion;

## En algunos SGBD la generación de referencias es manejada directamente y no hace falta especificarla de manera explícita.
```

Fuera de la declaración de los arrays, no se han tomado decisiones de mayor importancia, pues el estándar permite diseñar el modelo objeto/relacional de manera más directa y sencilla, permitiendo declarar la mayoría de principales restricciones.

Bajo este diseño se intentará realizar una traducción a cada SGBD para intentar reproducir el mismo comportamiento de la mejor manera posible.

Implementación con el modelo objeto/relacional

La implementación del modelo objeto/relacional se ha realizado con los tres gestores indicados en la práctica. Para cada gestor se ha debido de tener en cuenta las características de éste y las posibles limitaciones que poseían para soportar la orientación a objetos, por lo que se comentarán los aspectos más relevantes a continuación.

Oracle

En la implementación del sistema de gestión bancaria en Oracle, se utilizan varias funcionalidades del modelo objeto/relacional que ofrece Oracle Database. Entre las funcionalidades clave destacan las siguientes:

Herencia de Tipos: Oracle permite la creación de jerarquías de tipos, lo que facilita la representación de relaciones de herencia entre entidades. En el código proporcionado, se observa la aplicación de herencia mediante el uso de tipos definidos por el usuario y subtipos que heredan atributos y comportamientos de tipos base. Por ejemplo, los tipos Cuenta_AhorroUDT y Cuenta_CorrienteUDT son subtipos de CuentaUDT, lo que permite representar la relación de herencia entre diferentes tipos de cuentas bancarias.

Tipos Definidos por el Usuario (UDTs): Oracle permite la creación de tipos de datos personalizados mediante el uso de tipos definidos por el usuario (User Defined Types). En el código SQL implementado, se han definido varios tipos de objetos como ClienteUDT, CuentaUDT, OperacionUDT, etc., para representar las entidades del sistema bancario. Estos tipos de datos personalizados encapsulan tanto los atributos como el comportamiento asociado a cada entidad, lo que facilita la gestión de la información de la base de datos.

Referencias (REFs): Oracle proporciona el tipo de datos REF para manejar referencias a objetos en la base de datos. Estas referencias se utilizan para establecer relaciones entre entidades, como la relación entre un cliente y sus cuentas bancarias, o la relación entre una operación y la cuenta asociada. Esto permite representar las relaciones complejas entre entidades de manera eficiente y consistente en la base de datos.

Restricciones de Integridad: Oracle permite la aplicación de restricciones de integridad para garantizar la coherencia y la consistencia de los datos en la base de datos. Se aplican diversas restricciones como

claves primarias, restricciones de nulidad, y restricciones de integridad referencial para mantener la integridad de los datos y prevenir inconsistencias.

En resumen, la implementación del modelo objeto/relacional en Oracle ofrece una solución robusta y escalable para el diseño y gestión de bases de datos complejas, aprovechando las funcionalidades avanzadas proporcionadas por el sistema de gestión de bases de datos Oracle Database.

IBM DB2

En IBM DB2, los arrays de tipo referencia no se pueden definir directamente como en algunos otros sistemas de gestión de bases de datos. Sin embargo, puedes lograr una funcionalidad similar utilizando estructuras de datos y relaciones entre tablas.

Al igual que ocurría con PostgreSQL, carece de arrays de tipo referencia para representar las asociaciones con cardinalidad N, por lo tanto se ha utilizado una tabla tipada para la representación de la relación N:M entre Cliente y Cuenta, y en el caso de las asociaciones 1:N, únicamente se ha hecho referencia a la clave primaria de la tabla referenciada

La principal peculiaridad a la hora de generar los tipos de datos UDT sería el uso de las siguientes sentencias al final de cada uno de ellos:

INSTANTIABLE NOT FINAL: Esta parte de la declaración indica que el tipo de dato es instantiable, lo que significa que se puede utilizar para crear instancias con valores específicos. Además, NOT FINAL significa que este tipo de datos no es final y puede ser heredado por otros tipos de datos.

REF USING INTEGER mode db2sql: Esta parte de la declaración especifica cómo se referencia este tipo de datos en la base de datos. En este caso, se utiliza una referencia (REF) que utiliza un identificador único de tipo entero (INTEGER) y se utiliza el modo db2sql, que es el modo predeterminado para manipular tipos de datos estructurados en IBM DB2.

Otro problema encontrado ha sido a la hora de intentar generar los oid’s de las distintas instancias de las tablas tipadas, en este caso no pueden ser generadas por el sistema con la sentencia REF IS oid SYSTEM GENERATED, este sistema de bases de datos te obliga a establecer el oid cómo : REF IS oid USER GENERATED, tras leer la documentación se ha llegado a la conclusión que la opción más similar sería usar la función GENERATE_UNIQUE() en los inserts, para ello en la creación de los

tipos hay que poner la sentencia REF USING VARCHAR(16) que es el tipo adecuado ya que la función comentada previamente devuelve CHAR (13) FOR BIT DATA .

Finalmente para establecer la relación TENER y OPERACIÓN se ha establecido que el oid de las tablas Cliente ,Cuenta y Oficina se usará la sentencia REF USING INTEGER y se introducirá a mano un número secuencialmente incrementado.
Por otro lado en DB2 comentar una dificultad a la hora de implementar los triggers, debido
a que es necesario cambiar el carácter separador de los triggers tras la cláusula END
para que pueda ser entendido por el gestor.
Debido a este desconocimiento se perdió una gran cantidad de tiempo buscando la solución, la cual consiste en establecer como separador en nuestro caso el carácter '@' y en el comando establecer dicho separador en vez de ';' que usa por defecto de la siguiente manera:
db2 -td@ -f trigger.sql

PostgreSQL

Este gestor de bases de datos intenta realizar un modelo objeto/relacional de manera bastante ineficiente, intentando parecerse lo máximo posible pero sin mucho éxito al estándar *SQL 1999*.

A pesar de esto, *Postgres* acepta la herencia de tablas mediante la cláusula *INHERITS*, pero esto no significa que sea perfecto, pues no se puede realizar en tablas tipadas, y si se considera el uso de esta claúsula se deben tener en cuenta un par de aspectos muy importantes: se heredan las restricciones de la tabla padre hacia las tablas hijas, pero si se tiene una clave extranjera que referencia a un valor de la tabla padre, esta no podrá ser encontrada si se han introducido los valores directamente en las tablas hijas. Tiene sentido que directamente se inserte en las tablas hijas, pues se tienen las mismas columnas que en la tabla padre más las columnas propias de cada tabla hija. Lo más sorprendente es que si se hace una consulta a esta tabla padre encuentra todos esos atributos que se introdujeron en las tablas hijas, pero una clave extranjera no la encuentra. Esto no tiene mucho sentido, y marca una gran deficiencia en este gestor, pues si se opta por realizar la herencia se deberá renunciar al control directo mediante claves extranjeras e implementar triggers para controlar la existencia de las claves antes de insertarlas. Por ejemplo, la tabla posee contiene los números *IBAN* de algunas cuentas, pero al insertar estas cuentas como corrientes o de ahorro no se pueden referenciar a la tabla cuenta desde la tabla Posee, pues no las encuentra. Además las tablas hijas no heredan la unicidad de las claves primarias de las tablas padre, por lo que se podría duplicar la clave en dos tablas hijas distintas. Este problema se soluciona con la disyunción de las tablas, mediante triggers.

Como bien se ha comentado anteriormente tampoco existe la herencia de tablas tipadas, pues a pesar de eso se ha adaptado una herencia con tablas no tipadas pero que heredan de tablas tipadas, un pequeño "apaño" que nos sirve para salir del paso.
Otro aspecto importante a tener en cuenta es la inexistencia de referencias mediante arrays, además que desde nuestro gestor tampoco dejaba declarar arrays directamente, por lo que se optó por realizar la relación *N:M* de cliente con cuenta mediante una tabla intermedia llamada Posee. De haberse podido incluir arrays, se habrían usado y añadido triggers para poder comprobar la restricción de la existencia del cliente o cuenta añadidos.

Algunas restricciones básicas se han podido gestionar directamente en la creación de las tablas, como las cláusulas *CHECK* para comprobar que el número de teléfono tiene tamaño 9,… Otra restricciones son la elección de las claves primarias y de los atributos *NOT NULL* de las tablas. Una de las restricciones más importantes a tratar ha sido la creación de una secuencia para el número de operación asociado a dicha cuenta. Esta gestión se ha debido solucionar con un trigger, modificando el valor del número de operación al número de operaciones de esa cuenta +1.

También es muy importante explicar que muchas de las restricciones se han debido manejar con triggers, como la cobertura total y la disyunción de las clases hijas, pues no existe mecanismo directo dentro de Postgre que gestione esto de manera más sencilla. En el archivo .sql se pueden ver todos los triggers creados que han sido necesarios para cumplir con el problema planteado, entre los que se encuentran los mencionados anteriormente y los de comprobación de existencia de los atributos que guardan la clave de las clases padre en una herencia.

En resumen, se ha tratado de adaptar lo máximo posible el modelo al esquema objeto/relacional y aprovechar las ventajas de la orientación a objetos, pero este SGBD se considera que no es la mejor opción para implementar este tipo de esquemas.

Generación de datos y pruebas

Oracle

En el proceso de preparación de datos de prueba para la base de datos, se ha empleado la herramienta Data Generator. Data Generator resulta una solución eficaz para generar conjuntos de datos ficticios de manera automatizada. La herramienta permite especificar la cantidad de datos necesarios y definir reglas para su generación, lo que puede resultar útil para la creación de conjuntos de datos realistas y variados.

Se ha procurado crear e insertar datos variados para verificar el correcto funcionamiento de la base. A su vez, se han realizado unas consultas simples para corroborar la correcta inserción de los datos en la base.

IBM DB2

Para IBM, la mejor opción ha sido utilizar la herramienta Mockaroo para generar datos de prueba. Mockaroo es una herramienta en línea que permite generar datos de prueba de manera rápida y sencilla. Ofrece una amplia gama de tipos de datos y opciones de personalización, lo que la hace adecuada para simular una variedad de escenarios de datos. Además, Mockaroo permite exportar los datos generados en diversos formatos, incluido un archivo .sql, lo que facilita la carga de datos en la base de datos utilizando comandos de terminal. Esto proporciona a IBM una solución eficiente y escalable para generar conjuntos de datos de prueba que pueden ser utilizados para validar y mejorar el rendimiento y la funcionalidad de sus sistemas.

PostgreSQL

Para este SGBD se ha considerado como mejor opción la utilización de la herramienta Copilot para generar datos de prueba. Copilot es una herramienta de IA que devuelve una respuesta en formato de texto a una pregunta que le hagamos, por lo que se puede personalizar de muchas maneras como quieres que te genere los datos, por lo que es bastante sencillo. Al devolver todos los resultados esperados se han guardado en un archivo .sql para importarlo desde la herramienta adminer, la cuál facilitaba la carga de datos.

De forma adicional se han realizado un par de consultas que han servido de ayuda para comprobar la correcta implementación de nuestro esquema.

Implementación con db4o

Tal y como se especificaba explícitamente en el guión de la práctica, se ha decidido implementar un pequeño esquema de una base de datos en db4o. En concreto, se ha decidido implementar la relación entre una cuenta y un cliente, pudiendo tener listas de clientes asociados a una cuenta y viceversa. Para lograr esto se ha creado un proyecto en Java bastante sencillo que usa el paquete dado como ejemplo y se ha editado para lograr el objetivo deseado. Cabe destacar que esta implementación se ha tomado como una pequeña toma de contacto, y no es una implementación muy compleja.

Se han creado varias clases con sus respectivos atributos, constructor, setters, getters y el método toString para poder mostrarlo por pantalla.

Uno de los aspectos más importantes a tener en cuenta es la ventaja de implementar un esquema orientado a objetos de manera directa, usando un lenguaje de programación que directamente hace uso de esta característica. Si se quisiera hacer uso de restricciones es importante comentar que db4o no usa este tipo de restricciones, teniendo que añadir una capa extra si se quisiera hacer. Un claro ejemplo está en el momento que uno desea añadir una restricción de disyunción o una restricción del tamaño del número introducido, siendo necesaria una función para comprobar este valor antes de crear el cliente.

La base de datos embebida se almacena en un archivo físico con extensión .db4o, siendo necesario crear una clase para añadirla a la base con el método store que proporciona ésta después de haberse configurado. Para trabajar con la base de datos correctamente se ha necesitado hacer uso de algunas clases bastante abstractas como ObjectContainer, que permite realizar las operaciones de inserción y eliminación de la base de datos, además de la clase ObjectSet, que es una colección de objetos que guarda los resultados de una consulta realizada, utilizada para comprobar el correcto funcionamiento del sistema.

Comparación de los SGBD

Soporte del Modelo Relacional:

Oracle destaca por su gran soporte al modelo relacional, ofreciendo una amplia gama de funcionalidades que garantizan la integridad y eficiencia de los datos. Más allá de las características estándar de SQL para la definición de tablas y restricciones de integridad, Oracle proporciona herramientas avanzadas para la optimización del rendimiento, como la gestión de índices y particiones de tablas. Estas funcionalidades permiten adaptar la base de datos a las necesidades específicas del sistema, mejorando tanto la velocidad de acceso como la escalabilidad del sistema. Además, Oracle Database ofrece una gestión completa de transacciones ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad), garantizando la fiabilidad y consistencia de los datos en todo momento.

IBM DB2 tiene un sólido soporte para el modelo relacional, proporcionando características estándar de SQL que cumplen con las especificaciones del modelo. Esto incluye la definición de tablas, claves primarias y extranjeras, y restricciones de integridad. Sin embargo, es importante destacar que IBM DB2 carece de ciertas características avanzadas presentes en otros gestores, como los arrays de tipo referencia para representar asociaciones con cardinalidad máxima N. Asimismo para la ejecución de los triggers es necesario modificar el carácter de compilación a uno distinto del ‘;’ que es el usado por defecto.

PostgreSQL es un sistema de gestión de bases que destaca por un gran soporte para el modelo relacional, incluyendo todas las características propias de este mismo, como la definición de tablas, referencias mediante claves extranjeras y uso de claves primarias. Postgre por su parte incluye también varias características muy avanzadas, pero tienen sus desventajas. Por ejemplo, se ofrecen opciones de replicación y escalabilidad, como la partición de tablas, pero se reporta de manera bastante general que no son tan intuitivas de usar como las de otros gestores. Además, PostgreSQL está diseñado para ambientes de alto volumen, siendo bastante lento en inserciones y actualizaciones de bases de datos pequeñas; y su sintaxis para algunas sentencias o comandos no es tan intuitiva.

Modelo Objeto/Relacional y Limitaciones:

En lo que respecta al modelo objeto/relacional, **Oracle** ofrece una flexibilidad notable. Permite la definición de tipos de datos complejos y estructuras de objetos, lo que facilita la representación de asociaciones N:M y otras relaciones complejas. Además, Oracle proporciona la capacidad de definir métodos y funciones por el usuario en lenguajes de programación, que pueden ser invocados desde

dentro de las consultas SQL. Esto amplía las capacidades del modelo objeto/relacional y ofrece opciones adicionales para la implementación de lógica de negocio dentro de la base de datos.

En cuanto al modelo objeto/relacional, **IBM DB2** ha evolucionado para ofrecer algunas características avanzadas, pero presenta ciertas limitaciones. Por ejemplo, la ausencia de arrays de tipo referencia dificulta la representación directa de relaciones N:M entre entidades. Para superar esta limitación, se ha optado por utilizar una tabla tipada para la representación de estas relaciones. En el caso de las asociaciones 1:N, se hace referencia únicamente a la clave primaria de la tabla referenciada. Una de las características de este gestor al igual que en oracle es que requiere la especificación del tipo de dato del OID para cada tipo. Otra característica sería la capacidad de definir Métodos y funciones por el usuario en lenguajes de programación que pueden ser invocados desde dentro de las consultas SQL

En relación a **PostgreSQL** y el modelo objeto/relacional, este sistema de gestión de bases de datos intenta ofrecer una solución flexible y potente a este modelo, ofreciendo tanto las ventajas del modelo relacional como las del modelo orientado a objetos. Algunas características de este modelo que nos permite aclarar por qué se declara como un intento y no un logro son la imposibilidad de heredar de tablas tipadas o de usar referencias mediante claves extranjeras a atributos de una clase padre que guarda todas las claves de sus clases hijas. Esto nos ha complicado demasiado la implementación del modelo, por lo que no se considera una de las mejores opciones. Pero también cabe destacar que si se tiene un modelo muy sencillo que no contiene muchas relaciones este gestor es bastante sencillo de usar y aunque, no demasiado intuitivo, no sería una muy mala opción.

Conclusiones:

En resumen, **PostgreSQL** destaca por su gran utilidad para implementar un modelo relacional y su rapidez, además de ofrecer una gran cantidad de funcionalidades avanzadas y escalabilidad. Aunque tiene ciertas limitaciones, se considera una opción atractiva para utilizar en una variedad de aplicaciones. **Oracle**, por otro lado, se destaca por su robustez y amplio soporte para ambos modelos utilizando una variedad de características avanzadas, ofreciendo un alto rendimiento en sistemas que requieren una gestión avanzada de los datos. En cuanto a **IBM DB2**, ofrece una solución sólida en el modelo relacional, mientras que se queda algo corto en el modelo objeto/relacional por sus limitaciones en la representación de asociaciones complejas y características avanzadas. A pesar de eso, su rendimiento y escalabilidad lo convierten en una buena alternativa para entornos empresariales. La elección del sistema dependerá de las necesidades específicas de nuestro proyecto, pero en general si se puede usar Oracle, es la mejor opción bajo nuestro punto de vista.

Esfuerzos invertidos

	Miembros del grupo		
Tarea	Guillermo Bajo	Andrei Dumbrava	Pablo Moreno
Diseño conceptual	2h	1h	2h
Diseño Lógico Relacional	30 min	30 min	30 min
Implementación modelo relacional	1h 30 min	30 min	1h 30 min
Diseño Lógico Objeto/Relacional	30 min	1h 30 min	30 min
Implementación modelo objeto/relacional	4h	3h	3h 30 min
Generación datos y pruebas	30 min	30 min	1h
Implementación db4o	30 min	2h 30 min	30 min
Memoria	2h	1h 30 min	1h 30 min
Total	**11h 30 min**	**11h**	**11h**