Java: cómo comprobar si existe o no un archivo o una carpeta en el disco duro | 5 razones que convencerán a tu jefe que debes teletrabajar

¿Qué es un ORM?

*Por **José Manuel Alarcón***

Tradicionalmente, para realizar **acceso a datos desde un lenguaje orientado a objetos** (POO) como pueden ser .NET o Java, era necesario **mezclar código y conceptos muy diferentes**.

Por un lado teníamos **la aplicación** en sí, escrita en alguno de estos lenguajes como C# o Java. Dentro de éste programa hacíamos uso de **ciertos objetos especializados** en conectarse con bases de datos y lanzar consultas contra ellas. Estos objetos de tipo `Connection`, `Query` y similares, eran en realidad conceptos de la base de datos llevados a un programa orientado a objetos que no tenía nada que ver con ellos. Finalmente, para lanzar consultas (tanto de obtención de datos como de modificación o de gestión) se introducían **instrucciones en lenguaje SQL**, en forma de cadenas de texto, a través de estos objetos.

Además estaba el problema de **los tipos de datos**. Generalmente el modo de representarlos y sus nombres pueden variar entre la plataforma de desarrollo y la base de datos. Por ejemplo, en .NET un tipo de datos que almacena texto es simplemente un `string`. En SQL Server para lo mismo podemos utilizar cadenas de longitud fija o variable, de tipo Unicode o ANSI, etc... Y lo mismo con los otros tipos de datos. Algunos ni siquiera tienen por qué existir en el lenguaje de programación. Y tampoco debemos olvidarnos de **los valores nulos** en la base de datos, que pueden causar todo tipo de problemas según el soporte que tengan en el lenguaje de programación con el que nos conectamos a la base de datos.

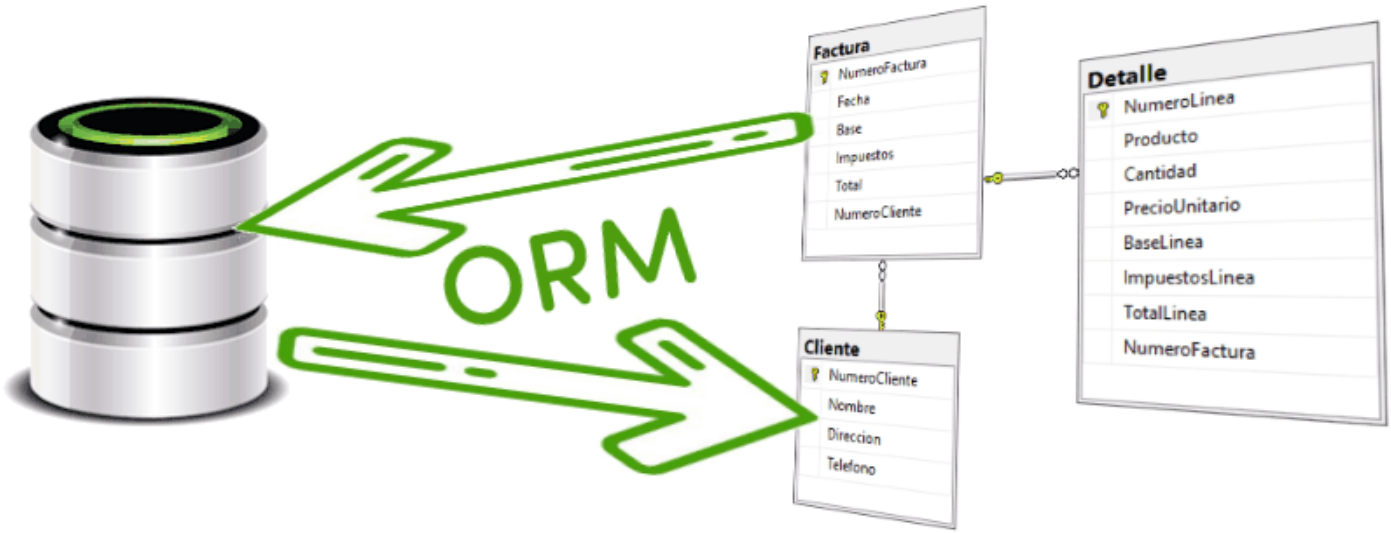
Otros posibles problemas y diferencias surgen del modo de pensar que tenemos en un lenguaje orientado a objetos y una base de datos. En POO, por ejemplo, para representar una factura y sus líneas de factura podrías definir un objeto `Factura` con una propiedad `Lineas`, y si quieres consultar las líneas de una factura solo debes escribir `factura.Lineas` y listo, sin pensar en cómo se relacionan o de dónde sale esa información. En una base de datos, sin embargo, esto se modela con dos tablas diferentes, una para las facturas y otra para las líneas, además de ciertos índices y claves externas entre ellas que relacionan la información. Si además se diese el caso de que una misma línea de factura pudiese pertenecer a más de una factura, necesitas una tercera tabla intermedia que se relaciona con las dos anteriores y que hace posible localizar la información. Como vemos **formas completamente diferentes de pensar sobre lo mismo**.

La complejidad no termina aquí ya que las bases de datos tienen **procedimientos almacenados** (que son como pequeños programas especializados que se ejecutan dentro de la base de datos), **transacciones**, y otros conceptos que desde el punto de vista de un lenguaje POO son completamente extraterrestres y ajenos.

Estas diferencias entre conceptos, tipos de datos, y modos de trabajar pueden causar muchos problemas de lo que se dio en llamar **"desfase de impedancia"** o "impedance mismatch" en inglés, en una clara analogía los circuitos eléctricos y al flujo eléctrico (en este caso aplicado a flujo de información). El concepto se refiere a la dificultad para hacer fluir la información entre la base de datos y las diferentes capas del programa en función de la diferencia existente entre cada una de estas partes.

Este desfase de impedancia hace que pueda llegar a ser muy complicado trabajar contra una base de datos desde un lenguaje POO si queremos sacar partido a los conceptos habituales que usamos en éstos, y huir de bibliotecas de funciones que nos fuerzan a trabajar con los conceptos de la base de datos.

"Mapeadores" al rescate



Como acabamos de ver, **lo ideal** en una aplicación escrita en un lenguaje orientado a objetos sería **definir una serie de clases, propiedades de éstas que hagan referencia a las otras, y trabajar de modo natural con ellas**.

¿Qué quieres una factura? Simplemente instancias un objeto `Factura` pasándole su número de factura al constructor. ¿Necesitas sus líneas de detalle? LLamas a la propiedad `Lineas` del objeto. ¿Quieres ver los datos de un producto que está en una de esas líneas? Solo lee la propiedad correspondiente y tendrás la información a tu alcance. Nada de consultas, nada de relaciones, de claves foráneas...

> En definitiva **nada de conceptos "extraños" de bases de datos en tu código orientado a objetos.** En la práctica para ti la base de datos es como si no existiera.

Eso precisamente es lo que intenta conseguir el software llamado **ORM**, del inglés **Object-Relational Mapper** o "Mapeador" de relacional a objetos (y viceversa). Un ORM es una biblioteca especializada en acceso a datos que genera por ti todo lo necesario para conseguir esa abstracción de la que hemos hablado.

Gracias a un ORM ya no necesitas utilizar SQL nunca más, ni pensar en tablas ni en claves foráneas. Sigues pensando en objetos como hasta ahora, y te olvidas de lo que hay que

hacer "por debajo" para obtenerlos.

El ORM puede **generar clases a partir de las tablas** de una base de datos y sus relaciones, **o hacer justo lo contrario**: partiendo de una jerarquía de clases crear de manera transparente las entidades necesarias en una base de datos, ocupándose de todo (ni tendrás que tocar el gestor de bases de datos para nada).

¿Qué ORMs tenemos disponibles?

- En el mundo **Java** el ORM más conocido y utilizado es **Hibernate** que pertenece a Red Hat aunque es gratuito y Open Source. Hay muchos otros como **Jooq** , **ActiveJDBC** que trata de emular los Active Records de Ruby On Rails, o **QueryDSL** , pero en realidad ninguno llega ni por asomo al nivel de uso de Hibernate. Si necesitas un ORM en Java debes aprender Hibernate, y luego ya si quieres te metes con otro, pero este es indispensable.

- En la **plataforma .NET** tenemos varios conocidos, pero el más popular y utilizado es **Entity Framework** o **EF**, que es el creado por la propia Microsoft y que viene incluido en la plataforma .NET (tanto en la "tradicional" como en .NET Core). También existe un "port" de Hibernate para .NET llamado **NHibernate** y que a mucha gente le gusta más que EF. Hay otros como **Dapper** que está creado por la gente de Stack Exchange y es mucho más sencillo que los anteriores, lo cual es considerado una gran virtud por mucha gente (entre los que me incluyo), y también es muy utilizado. Y es muy conocido **Subsonic** , que lleva casi diez años en activo pero que puede llegar a ser bastante complicado (a mí personalmente no me gusta nada).

En PHP tienes **Doctrine** , tal vez el más conocido y recomendado (utilizado por el *framework* **Symfony**), pero también se usan bastante **Propel** , **RedbeanPHP** y uno muy popular pero ya en desuso es **Xyster** (pero te lo encontrarás aún bastante por ahí).

En Python el híper-conocido *framework* **Django** (así sinónimo de desarrollo web con este lenguaje) incluye su propio ORM, pero también tenemos SQLAlchemy por el que muchos beben los vientos y lo califican como el mejor ORM jamás hecho (no tengo

experiencia con él como para saberlo). También están **Peewee** o **Pony ORM** entre otros.

Prácticamente todas las plataformas tienen el suyo, así que búscalos para la tuya y mira cuál es el más popular y el que más comunidad reúne.

Ventajas e inconvenientes de un ORM

Los ORM ofrecen enormes **ventajas**, como ya hemos visto, al reducir esa "impedancia" que impide el buen flujo de información entre los dos paradigmas POO-Relacional. Pero además:

- No tienes que escribir código SQL, algo que muchos programadores no dominan y que es bastante complejo y propenso a errores. Ya lo hacen por nosotros los ORM.
- Te dejan sacar partido a las bondades de la programación orientada a objetos, incluyendo por supuesto la herencia, lo cual da mucho juego para hacer cosas.
- Nos permiten aumentar la reutilización del código y mejorar el mantenimiento del mismo, ya que tenemos un único modelo en un único lugar, que podemos reutilizar en toda la aplicación, y sin mezclar consultas con código o mantener sincronizados los cambios de la base de datos con el modelo y viceversa.
- Mayor seguridad, ya que se ocupan automáticamente de higienizar los datos que llegan, evitando posibles ataques de inyección SQL y similares.
- Hacen muchas cosas por nosotros: desde el acceso a los datos (lo obvio), hasta la conversión de tipos o la internacionalización del modelo de datos.

Pero no todo va a ser alegría. También tienen sus **inconvenientes**:

- Para empezar pueden llegar a ser muy complejos. Por ejemplo, NHibernate tiene ya más de medio millón de líneas de código ahora mismo, y muchas clases y detalles que hay que saber. Eso hace que su aprendizaje sea complejo en muchos casos, y hay que invertir tiempo en aprenderlos y practicar con ellos hasta tener seguridad en su manejo diario.

- No son ligeros por regla general: añaden una capa de complejidad a la aplicación que puede hacer que empeore su rendimiento, especialmente si no los dominas a fondo. En la mayor parte de las aplicaciones probablemente no te importe, pero en ocasiones su impacto en el rendimiento es algo a tener muy en cuenta. En general una consulta SQL directa será más eficiente siempre.

- La configuración inicial que requieren se puede complicar dependiendo de la cantidad de entidades que se manejen y su complejidad, del gestor de datos subyacente, etc...

- El hecho de que te aísle de la base de datos y no tengas casi ni que pensar en ella es un arma de doble filo. Si no sabes bien lo que estás haciendo y las implicaciones que tiene en el modelo relacional puedes construir modelos que generen consultas monstruosas y muy poco óptimas contra la base de datos, agravando el problema del rendimiento y la eficiencia.

En definitiva, los ORM son una herramienta que puede llegar a ser maravillosa, pero que en aplicaciones pequeñas pueden ser como "matar moscas a cañonazos". En aplicaciones más grandes donde el mantenimiento y la homogeneidad sean importantes, son indispensables. Eso sí, no te eximen de aprender bien el lenguaje SQL o las maneras más tradicionales de realizar acceso a datos, ya que conocerlas puede marcar la diferencia cuando surjan problemas o haya que determinar por qué se produce una merma de rendimiento, etc...

Fecha de publicación: *26 de febrero de 2018*

Fundador de campusMVP, es ingeniero industrial y especialista en consultoría de empresa. Ha escrito diversos libros, habiendo publicado hasta la fecha cientos de artículos sobre informática e ingeniería en publicaciones especializadas. Microsoft lo ha reconocido como MVP (Most Valuable Professional) en desarrollo web desde el año 2004 hasta la actualidad. Puedes seguirlo en Twitter en @jm_alarcon o leer sus blog técnico o personal.

Ver todos los posts de José Manuel Alarcón

Archivado en: Acceso a Datos
13 comentarios

Comentarios (13)

Danny
02/03/2018 20:06:38

Muy buen POST. Creo que es importante como programador aprender las dos formas de desarrollo. En mi caso trabajo mucho con EF en .NET pero siempre hay situaciones donde le metemos un SQL ya sea por rendimiento o claridad. Saludos desde Paraguay. Siempre los sigo.
Responder

Javier
16/03/2018 17:21:21

Gracias José Manuel.
Actualmente estoy trabajando con EF Core y la verdad que doy fe de las desventajas que listas en el artículo. Definitivamente, hay que aprender a usar el ORM que estés usando en tu proyecto.

Por otro lado, ¿recomiendas alguna bibliografía/webgrafía en particular? Me gustaría profundizar más en el tema.

Saludos!
Responder

José Manuel Alarcón
16/03/2018 20:45:17

Gracias por comebtar.

Pues la verdad es que los libros van de capa caída en todo el mundo y no tengo nada que recomendarte. Pero si nos das tiempo quizá tengamos el mejor recurso de aprendizaje sobre esto en algún momento del futuro ;-)

Saludos!
Responder

Tiago Barro
14/09/2018 11:15:56

Gracias por el artículo José Manuel.

Por si a alguien le sirve nuestra experiencia:
- En 2007 empezamos el desarrollo de nuestro propio ORM. Todavía no existía ni EF ni Dapper. :-D

- El rendimiento de nuestra plataforma en Windows Forms tenía que ser muy alto por lo que decidimos desarrollar un ORM propio optimizado al máximo en función de nuestras necesidades y aprovechando la potencia de Sql Server. Teníamos claro que no nos sería necesario utilizar otras bbdd, con lo que se desarrolló pensando siempre en las oportunidades de rendimiento que ofrecía Sql Server.

- Utilizamos CodeSmithTools para la generación automática de código. Es decir, creamos la estructura de nuestras entidades en la base de datos y ejecutamos las plantillas de CodeSmith que atacan a estas tablas, con lo que se nos generan automáticamente los procedimientos almacenados necesarios (CRUD) sin tener que escribir ninguna línea de código y totalmente optimizados.

- También desarrollamos plantillas que nos generan las clases en C# de todas las entidades con lo que, sin escribir código, conseguimos: Las ventajas que has comentado al ser un ORM, alto rendimiento, control automático de concurrencia, auditoría automática, etc... Además ya se generan automáticamente las funciones más habituales de entidades: Carga de registro por indice principal, por filtros, ordenación, guardado automático con información del usuario, terminal, etc.., eventos de antes y después de guardar, antes y después de eliminar, etc...

- Actualmente estamos integrando Dapper con nuestro propio ORM aprovechando su ligereza y optimización.

En resumen: Con un desarrollo propio de ORM, obtienes un control mucho mayor y un rendimiento totalmente optimizado para tus necesidades.
Desventajas: Un curro impresionante hasta que no lo tienes totalmente desarrollado. :-D

Saludos!
Responder

José Manuel Alarcón
14/09/2018 11:25:06

Hola Tiago:

Muy interesante vuestra experiencia. Igual un día os interesa escribir un artículo para contar lo que habéis aprendido, bueno y malo, de crear algo así. Si es así, ya sabes: mándanoslo para publicarlo :-)

Saludos!
Responder

Tiago Barro
14/09/2018 11:40:07

Hecho José Manuel.
Estaré encantado de escribir un artículo con nuestra experiencia.
Espero estar a la altura de vuestros posts! :-D

Un abrazo
Responder

Joseba
24/05/2019 10:40:43

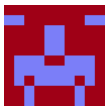
Hola!

¿Y ese fantástico artículo se escribió en algún momento?

Saludos.
Responder

Tiago Barro
24/05/2019 15:21:34

Gracias por el interés Joseba. Estamos en ello. :-D
Saludos!
Responder



Jose Andrés
16/07/2019 9:39:40

Fundador de campusMVP, es ingeniero industrial y especialista en consultoría de empresa[...]

Cuando un sujeto habla en tercera persona para referirse a sí mismo evidencia un sentimiento de despersonalización.

La despersonalización es una alteración de la percepción o la experiencia de uno mismo de tal manera que uno se siente "separado" de los procesos mentales o cuerpo, como si uno fuese un observador

externo a los mismos.

Esto implica que no puede reconocerse en eso que dice o describe de sí mismo.

Es un trastorno psicosomático que viene desencadenado por la ansiedad, el estrés, o el consumo de algunas drogas. Algunos síntomas son mareos, dolores de cabeza y falta de concentración.

Si se refiere en términos de chiste o humor o en serio no cambia esta cuestión.

Si esto se hace en forma ocasional o como excepción no indicaría gran patología, pero si es la manera habitual de la persona de comunicarse es un claro signo de inestabilidad mental.

Es cierto que es un rasgo típico de la personalidad esquizoide porque hay una escisión del yo muy clara. El no poder decir YO... indica que algo se vive como ajeno o fuera de uno.
Responder

José Manuel Alarcón
16/07/2019 11:09:07

jajaja, me parto contigo :-)

Si vieses con detenimiento el blog verías que esa descripción está escrita de la misma manera para todos los autores, porque no se están describiendo a sí mismos, sino que el blog (sí, ese ente impersonal) está diciendo quién es el autor de cada post. Pero gracias por invertir tu tiempo en comentarlo.

Saludos!
Responder

Joseba
16/07/2019 11:37:22

No se lo tengas en cuenta,. El chaval estará haciendo algún cursillo de psicoanálisis y se ha emocionado imitando a Sigmund Freud. Ya ha aprendido que antes de hacer un psicoanálisis es recomendable hacer un análisis del entorno del sujeto.
Responder

jcarlos
04/11/2019 10:33:42

Llevo programando mas de 20 años sobre todo con access-vba y ahora estoy aprendiendo vs2019, c#, sql server y devexpress. Es un cambio muy grande pero para mejorar. Estoy "pegándome" con las clases y ahora estoy con una clase para gestionar los albaranes. He creado 2, una para la cabecera y otra para las líneas pero en su artículo he visto la idea de tener una para las 2 cosas (propiedad factura.lineas), lo cual me parece muy bueno. ¿alguien tiene algún ejemplo, artículo, tutorial de como hacer esto? me gustaría aprenderlo para hacer las cosas lo mejor posible. Gracias.
Responder

campusMVP
04/11/2019 12:35:41

> Hola J. Carlos:
>
> Si quieres aprender a fondo .NET y especialmente .NET Core que es el presente y el futuro de .NET te podemos recomendar nuestro curso:
>
> www.campusmvp.es/.../....NET-Core-3-y-C-8_242.aspx
>
> No vas a encontrar nada tan "potente" en el mercado, con soporte tutorial por el mismo experto que ha creado los contenidos, teoría, vídeos prácticos, multitud de ejemplos...
>
> Incluye un módulo de Entity Framework, así como LINQ, acceso a datos tradicional, etc...
>
> Saludos.
> Responder

Huevo, perro, abeja o viernes, ¿cuál es un día de la semana?:

¿Qué vas a aprender a programar hoy?

Curso Java (Plataforma + Lenguaje)
Curso Kubernetes y Docker (DevOps)
Curso ASP.NET Core MVC (Backend)
Curso de Plataforma .NET y C#
Curso Creación de Aplicaciones De Gestión
Curso SQL Server (Especialista Base Datos)

Curso HTML5 y CSS3 para programación
Curso JavaScript/ECMAScript (Avanzado)
Curso Responsive Web Design y Bootstrap
Curso Angular (Aplicaciones Web)
Cursos preparación de Exámenes Oficiales
Ver todos los cursos

¿Te ayudamos a decidir?

MVP

¿Por qué campusMVP?
Recursos para programadores
Bonifica tu curso
Servicios para empresas
Descuentos
Preguntas frecuentes
Contacto