Core

Volgens TheServerSide raken OO-databases weer in de mode. Tijd voor een gesprek met één van de volhouders op dit gebied: Intersystems. Intersystems heeft behalve onder meer de OO-database Caché, ook een tool ontwikkeld voor Pojo-persistance, met de onmogelijke naam Jalapeño. Een gesprek met Olivier Caudron en Aleks Djakovic van Intersystems.

Pojo-persistance en OO-databases

Interview

OO-databases zijn nooit echt aangeslagen, waarom niet?
Olivier Caudron: "Er is altijd een tendens in de Javawereld geweest om naar OO-databases te gaan, maar het is nooit een succesverhaal geweest. Neem JDO, dat was eerst bedoeld was voor OO-databases, maar is ineens veranderd in een SQL-mappingtool. Ik denk dat het vooral aan de onbekendheid ligt."
Aleks Djakovic: "Daar ben ik het mee eens, ik ben naar vele Java-conferenties geweest. Degenen aan wie je het product toont zijn aanvankelijk heel erg sceptisch. Dat ligt echt aan de onbekendheid. Na een demonstratie verandert die houding. We moeten een manier vinden om de voordelen van OO-databases beter te communiceren, en het product en de capaciteiten ervan te laten zien. Laten zien dat je simpele *mapping* kunt doen, of complete *mapping*, maar nog steeds met de voordelen van het feit dat onze omgeving perfect is voor rapid development (RAD) met de voordelen van een schaalbare en goed performende database."

U schreef ooit de JDBC-driver voor Caché. Waarom heb je eigenlijk een JDBC-driver nodig bij een OO-database?
Aleks Djakovic: "Caché is in de kern een hiërarchische database, we slaan alles op in boomachtige structuren. Maar de engine voor object-projectie en sql-projectie zitten naast elkaar. Als je een SQL-query doet ga je naar de SQL-engine en dan naar de data, als je dat via de objectkant doet, kies je de andere weg. Als het mixt ga je van de een kant naar de andere en terug.


Olivier Caudron: JDO is ineens veranderd in een SQL-mappingtool

Dré de Man
tekst en foto's

We hebben een beperkte hoeveelheid van *mapping* maar we doen het op database-niveau in tegenstelling tot de rest van de wereld die het aan de Java-kant, op applicatieniveau doet. Dat is een groot verschil tussen ons en de andere spelers. We zijn echt object- en relationele database tegelijkertijd. De grote meerderheid van de gebruikers gebruiken echter de relationele toegang, JDBC en ODBC. Die gebruikers willen gewoon een bestaande relationele database vervangen en zijn niet op zoek naar de specifieke voordelen van een OO-database."
Olivier Caudron: "Maar na verloop van tijd komt er een moment waarop er toch de behoefte aan de objectgeoriënteerde toegang komt en vaak gebeurt dan allebei. Voor mij is het mooi dat je beide mogelijkheden hebt; het wordt nooit verwarrend, je weet altijd waar je staat."

Er zijn nu heel veel O/R tools, er zijn ook relationele databases die een OO-toegang hebben, ik zou verwachten dat dat veel concurrentie betekent.
Aleks Djakovic: "Nee niet echt, Hibernate is onze partner. Wij zijn een van de gecertificeerde databases. Dat gaat dan meer specifiek over onze Java-producten. Als je van de relationele wereld komt en je moet de mapping doen, of je hebt een legacy-applicatie die al de mapping doet maar je wil het gewoon hosten op onze database, dan voldoet Hibernate prima. Of je gebruikt ons product dat erg lijkt op Hibernate, maar dat ook de mapping simplificeert. Joined tables, inheritance etc, voor ons is het heel natuurlijk. Object naar object. Er zijn dus meerdere opties, je gebruikt Hibernate en onze JDBC-driver, of je gebruikt Jalapeño, of de derde optie, je gebruikt onze eigen applicaties, die object-object-mapping doen, dat is waarschijnlijk de geprefereerde, de simpelste manier om een applicatie gedeployd te krijgen."

Concreet, als je een klantobject heb en je stopt dat in de database?
Aleks Djakovic: "In dat geval zou je Jalapeño en Jalapeño-notaties gebruiken. Je zou ook JPA-notaties kunnen gebruiken, omdat we JPA volledig ondersteunen. In een concreet voorbeeld, een klant met een string naam, data objecten en misschien een relatie, naar accounts bijvoorbeeld, hoef je eigenlijk niet veel te doen, je hoeft niets te annoteren. We hebben dus de omgekeerde manier van werken van JPA: we nemen aan dat je alle klassen wilt persisteren. Wil je een klasse niet persisteren, dan moet dat aangeven. Op klasseniveau zijn er verschillende annotaties voor het geval je het wil gebruiken als een relationele tabel, kun je de tabel naam specificeren. Dat is echter absoluut niet noodzakelijk. We doen het door de klassen te introspecten, gebruik makende van Java Introspection. We doen eenvoudige mapping: string naar string, data als data, als er een relatie is annotatie voor relaties – die overigens heel erg veel lijken op JPA. Wat weer anders is dan bij JPA: wij hebben geen annotaties nodig om de data te vullen op een slimme, referentieel integere manier. Dingen worden als objecten vastgelegd worden maar ook als relationele tabellen. Met een keer klikken kun je grote hoeveelheden dat aan de serverkant genereren, zonder een regel SQL. Dat maakt ons heel populair, want zo kun je snel testen, snel een databaseschema opleveren en het bevolken met *random* data. Je kunt dan weer snel testen een echt schema reconfigureren, dat triggert weer acties aan de serverkant en je schema verandert daar ook. Maar de data blijft ondertussen steeds op dezelfde plaats."

We hebben dus de omgekeerde manier van werken van JPA: we nemen aan dat je alle klassen wilt persisteren

Aleks Djakovic: "Het overervingsmodel voor onze database lijkt heel erg op dat binnen Java. Je kunt data ook automatisch refactoren. Als een superklasse van Java extraheert dan doen we dat op dezelfde manier. De data zit niet in een object, het zit daaronder. Als je van één klasse twee klassen maakt, dan betekent dat alleen dat de data tot een ander klasse behoort, maar die blijft wel op dezelfde plaats. Een suiperklasse extraheren, een klasse omlaagdrukken of naar boevn halebn, het kan allemaal. Sommige refactoring-dingen zijn niet voor ons niet mogleijk, encapsulate fields bijvoorbeeld, dat is echt de Java-kant. Maar niettemin is er een grote lijst van interessante refactoring-mogelijkheden die we volledig ondersteunen."

Jalapeño, wat is dat nu behalve een heel moeilijke naam?
Aleks Djakovic: "In wezen gaat het om het versimpelen van Java-persistance. Als doel kun je een object-database of een relationele database gebruiken. Jalapeño is een tool om je database-schema mee te genereren en we hebben ook tools om dat schema te migreren naar welke relationele database dan ook. Dat laatste kwam voort uit vragen van klanten, die vast wilden houden aan een bepaalde database. We doen dus object naar object-mapping gebruikmakende van Jalapeño notatataties, en we exporteren dat schema naar bijvoorbeeld Oracle. Jalapeño is werkelijk een RAD-tool, dat Java-persistance met testdata oplevert, en databewust refactoring, om het in een paar zinnen te zeggen. We doen echter ook alles wat JPA doet."

Waar vindt het zijn toepassing?
Aleks Djakovic: "Het belangrijkste punt is de complexiteit van het objectmodel. Als je een heel complex objectmodel hebt, dan heb je echt een voordeel. We kennen klanten die moeilijkheden hadden met Hibernate vanwege de complexiteit van het objectmodel. De mapping kost veel tijd, het is beter dan de code zelf te moeten schrijven, maar het kost nog steeds veel tijd. Wanneer het eenmaal gedeployed is moet het ook onderhouden worden. Als je in een zo gecompliceerd model ergens een peformanceprobleem hebt, dan is het heel moeilijk om het te vinden en op te lossen. Wij hebben ons model en je hoeft alleen je databasespecifiek dingen als indexen en relaties aan te geven. Wanneer je dat eenmaal hebt hoef je je model niet te veranderen, je

Aleks Djakovic: “We doen annotatie by execption, dat is typisch de trend in de industrie

hoeft niet te denken aan overerving, alles wordt gemapped, je drukt op de knop en het zit in de database. Je werkt gewoon object naar object dus het heeft ook geen consequenties voor de performance. Het is geen mapping-tool structuur, het is essentieel iets anders en daardoor ook gemakkelijker om te tunen, te onderhouden, om te deployen.
Zo gauw het datamodel complex is, wordt het interessant onze technologie te gebruiken. Als je maar één object hebt, dan is er maar een heel klein voordele in het gebruiken van Jalapeño. Maar met duizend objecten waarbinnen je overerving hebt, is er een heel duidelijk voordeel.”

Aleks Djakovic: “We doen annotatie *by execption*, dat is typisch de trend in de industrie. Met JPA heb je geen keus wanneer het een complexe structuur is. Als je maar drie of vier klassen hebt, gebruik gewoon Hibernate en dan word je ook gelukkig. Dat beantwoordt ook de vraag van eerder: ‘waarom zou je ons boven Hibernate kiezen’: we zijn partners dus ieder heeft zijn situaties waar je het goed kunt gebruiken.
Een ander punt in een complex datamodel is dat we bij refactoring niet de data, maar alleen de definitie veranderen. In het typische Hibernate-model met een relationele database is het echt heel erg moeilijk. Op het moment dat je een Java-refactoring doe, is je datamodel helemaal verprutst. Niet alleen aan de serverside maar ook aan de clientkant, want je annotaties zijn overal. Sommige tools zullen de annotaties juist verplaatsen, maar gerelateerd aan de mechaniek van het verplaatsen. Als je een klasse hebt die gerelateerd is aan een veld en je verplaatst dat, dan verplaatsen ze die klasse ook. Maar wanneer dat veld in feite een relatie is tot een andere klasse, van de originele klasse, dan wordt het echt complex. Je moet dan niet alleen je Java-klassen herschrijven, maar ook je annotaties, of Hibernate descriptors. Je moet het redeployen op je database en dan meestal moet je al je data weggooien en met een migratietool over kopiëren enzovoorts. Dat is een groot en ingewikkeld proces. Bij ons is het een druk op een knop in Eclipse, je hoeft niet eens te recompilen. We hebben het sinds anderhalf jaar en we dachten aanvankelijk dat er snel een concurrerend product op de markt zou verschijnen. Maar niemand heeft het nagevolgd. Dat komt waarschijnlijk omdat het veel te complex is wanneer je uit de relationele wereld komt.” «