BC4J of TopLink?

Ontwikkelen vanuit datamodel of classmodel

Alles is een object in Java" zo luidt een veelgehoorde uitspraak. Een Java applicatie bestaat uit het creëren van objecten (instanties van classes) om vervolgens iets zinnigs met deze objecten te doen. In de meeste gevallen is het niet zo handig om objecten telkens opnieuw aan te maken op het moment dat de applicatie actief wordt. Al snel ontstaat de behoefte om op een gegeven moment de staat waarin een verzameling van objecten zich bevindt, (blijvend) op te slaan of opnieuw te construeren. Oracle biedt BC4J en sinds kort TopLink, om de Java-applicatie-ontwikkeling op dit punt te ondersteunen.

Het bewaren en opnieuw construeren van een verzameling objecten *(object graph)* maakt onderdeel uit van de persistency functionaliteit van een Java applicatie. Momenteel wordt voor het overgrote deel van Java/J2EE applicatieontwikkeling voor wat betreft deze persistency een relationele database gebruikt als primaire *datasource*. Om deze relationele databases in te kunnen zetten, is het dus wel een eerste vereiste dat een Java applicatie een database kan benaderen en gebruiken.

Java Database Connectivity – JDBC

JDBC is een Application Programming Interface (API) met als doel om in Java geschreven programma's te kunnen laten werken met relationele databases. De API bestaat uit een verzameling van Java classes en interfaces, waarvoor de standaard gedefinieerd is door Sun Microsystems. Een individuele provider of leverancier van een database, kan deze standaard uitbreiden met zijn eigen implementatie van de JDBC driver. Van deze driver implementaties wordt minimaal verlangd dat zij de ANSI SQL-92 *Entry Level* specificatie ondersteunen. Op deze manier ontstaat de mogelijkheid om SQL statements tegen een bepaalde database uit te voeren.

De introductie van SQLJ, als laagje op JDBC, maakt het eenvoudiger voor ontwikkelaars om SQL op te nemen in Java code. Vergelijkbaar met Oracle RDBMS precompiler oplossingen als Pro*C en Pro*Cobol kunnen SQL statements, *embedded* in Java worden gecodeerd. De SQLJ precompiler zorgt vervolgens voor de vertaling van deze embedded statements naar JDBC code. JDBC/SQLJ is echter een relatief *low-level* manier om relationele database te gebruiken in Java applicaties. JDBC biedt slechts een dun toegangslaagje naar een database. Dit manifesteert zich vooral door de hoeveelheid code, die nodig is om "iets" gedaan te krijgen. Voorts wordt het directe gebruik van JDBC voor *persistency* vaak beschouwd als minder geschikt binnen een verder sterk objectgeoriënteerde (OO) aanpak. Dit vormt de zogenaamde *impedance mismatch*. Al snel ontstaat de behoefte aan een hoger abstractieniveau.

Patterns

JDBC abstractie begint meestal met de realisatie van een *class-model*. De bedoeling hiervan is om een brug te slaan tussen de relationele en de objectgeoriënteerde wereld. Een verzameling van interfaces en classes is het gevolg, waarin zoveel mogelijk low-level zaken zijn afgeschermd. Dit is in de meeste gevallen echter niet het eindstation.

De mate van abstractie vult op dat moment nog maar een klein gedeelte in van de noodzakelijke, ondersteunende logica *(application plumbing)*. Hoe minder de uiteindelijke applicatie zich hoeft aan te trekken van de opslag, des te beter. In diverse iteratieve slagen zal abstractielaag op abstractielaag gezet worden. Gedurende dit proces blijkt al snel dat bepaalde gelijksoortige problemen telkens weer opdoemen en opgelost moeten worden. De implementatie van deze patterns vormen samen met de JDBC abstractie uiteindelijk een *persistency framework*.

> ***Hoe minder de uiteindelijke applicatie zich hoeft aan te trekken van de opslag, des te beter***

Persistency Framework

Een persistency framework biedt een hoger abstractieniveau, waardoor de details van de toegang tot objecten in een datasource verborgen zijn. Met behulp van een dergelijk framework kan een *persistency layer* geïmplementeerd worden. Deze laag beschrijft de domeinobjecten, het (data)Model van de uiteindelijke applicatie. Aan de onderkant van deze laag wordt via JDBC de database ontsloten en worden resultaten opgeleverd in een relationeel formaat. Aan de bovenkant ervan wordt door het persistency framework de "vertaling" gemaakt naar de domeinspecifieke objecten van de applicatie. Bij J2EE applicaties wordt deze persistency laag gebruikt als onderdeel van een Model-View-Controller *(MVC)* architectuur.

Oracle's Persistency Frameworks

Er is een groot aantal commerciële en open source frameworks beschikbaar die een deel van Java/J2EE applicatieontwikkeling ondersteunen. Ook Oracle levert een aantal van dit soort 'applicatieraamwerken', op het gebied van persistency zelfs twee. Business Components for Java (BCJ4) maakt al geruime tijd onderdeel uit van de Oracle toolstack. Oracle9*i*AS TopLink is daar nog niet zo heel lang geleden als nieuw alternatief aan toegevoegd.

Business Components for Java (BC4J)

BC4J is meer dan alleen een persistency framework. Het kan een bredere en betere ondersteuning bij het applicatieontwikkelingsproces bieden en beperkt zich hierbij niet alleen tot persistency aspecten. Aan de basis van deze ondersteuning staat de implementatie van diverse "andere" design patterns, aangevuld met onder andere een uitgebreide JSP tag library. Ook is een efficiënte integratie met de overige Oracle-infrastructuurcomponenten gewaarborgd, zonder dat daar database- of applicatieserver-afhankelijkheden aan ten grondslag liggen. Zo kan BC4J optimaal gebruik maken van de Oracle database en wordt vanuit de ontwikkelomgeving, Oracle JDeveloper, uitgebreide ondersteuning geboden. BC4J is in het bijzonder geschikt voor de ontwikkeling van J2EE applicaties, waarbij gebruik gemaakt wordt van een bestaand datamodel. Deze *bottom-up* aanpak wordt echter niet door het grootste gedeelte van de OO community onderschreven. Die groep staat vooral een aanpak voor die redeneert vanuit een *classmodel.* Ondanks dat een dergelijke aanpak ook met BC4J goed mogelijk is, is voornamelijk de meer 'natuurlijke' manier waarop TopLink deze ondersteunt een reden waarom Oracle dit framework als alternatief aanbiedt.

Afbeelding 1. Dit classmodel wordt in de beschreven applicatie gebruikt.

Oracle9iAS TopLink

Aan de basis van Oracle9iAS TopLink ligt het vroegere TopLink. Dit framework, ooit een product van ***T**he **O**bject **P**eople Ltd.* is in juni 2002 door Oracle overgenomen van WebGain. TopLink heeft een behoorlijk groot marktaandeel in de Java community. Een groot deel van deze community prefereert het werken vanuit een *classmodel* boven het werken vanuit een *datamodel.* Het is vooral deze *top-down* benadering die Oracle9iAS TopLink interessant maakt voor deze groep ontwikkelaars. In de nu volgende paragrafen zullen we nader ingaan op Oracle9iAS TopLink.

De afzonderlijke object-attributen worden beschreven in termen van fysieke opslag

Oracle9iAS TopLink Onderdelen

De onderdelen van Oracle9iAS TopLink worden gevormd door een verzameling van Java-classes en -library's en de Oracle9iAS TopLink Workbench.

De classes en library's worden door de uiteindelijke Java applicatie op runtime gebruikt om op abstracte manier toegang te verkrijgen tot een datasource. Met behulp hiervan kan *persistency* worden toegevoegd aan Java applicaties. Hoe dit dient te gebeuren is voor een belangrijk deel beschreven in een persistency blauwdruk van het classmodel. De *workbench* biedt, als onderdeel van de Oracle9iAS TopLink oplossing, de functionaliteit om een dergelijke persistency blauwdruk te maken.

In tegenstelling tot BC4J is de ondersteuning voor Oracle9iAS TopLink vanuit Oracle JDeveloper iets beperkter. Zo is de workbench op dit moment nog een aparte applicatie en niet standaard geïntegreerd met de ontwikkelomgeving. De TopLink Java-classes en - library's (noodzakelijk bij het programmeren en compileren) kunnen zoals gebruikelijk eenvoudig worden toegevoegd aan de verzameling van library's binnen Oracle JDeveloper.

Oracle9iAS TopLink Concepten

De Oracle9iAS TopLink concepten zijn redelijk eenvoudig en worden op hoofdlijnen gevormd door de volgende onderdelen:

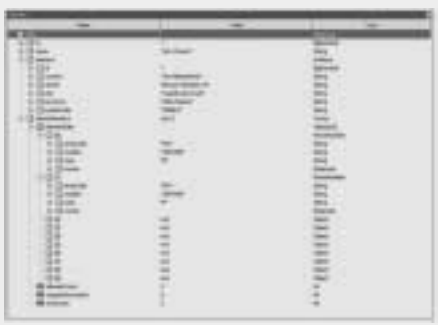
- Descriptors
- Mappings
- Database Sessions
- Units of Work

De *descriptors en mappings* maken deel uit van de al eerder genoemde blauwdruk van het classmodel. Database Sessions en Units of Work zijn voornamelijk Oracle9iAS TopLink runtime concepten. Laten we kort bij deze onderdelen stil staan.

Descriptors

Een descriptor bevat informatie over hoe een object, inclusief eventuele relaties naar andere objecten, afgebeeld moet worden op een datasource. De volgende zaken maken deel uit van een descriptor:

- *Java class naam*
  Informatie over de class die beschreven wordt en de tabellen waarin instanties van deze class opgeslagen moeten worden.
- *Primaire sleutel*
  De te gebruiken primaire sleutel van de tabel met instanties.
- *Query sleutels*
  Een lijst van aliases voor de veldnamen.
- *Mappings*
  Waarin de attributen van de class beschreven worden, inclusief eventuele relaties van attributen met andere classes.



Afbeelding 2. Een object graph

Afbeelding 3. De Workbench toont welke domein classes onderwerp moeten zijn van het project

- *Property's*
  Met daarin aanvullende informatie over (het gebruik van) de descriptor. Denk hierbij aan zaken als; autorisatie, validatie.

Mappings

Een mapping, als onderdeel van een descriptor, wordt gebruikt om het toegangspad voor een objectattribuut aan te geven. Met andere woorden hoe wordt een objectattribuut afgebeeld in de uiteindelijke datasource. Hierbij wordt onderscheid gemaakt tussen twee typen mappings, namelijk:

- *Direct mappings*
  Het betreft hier een objectattribuut wat **1:1 ("direct-to-field")** of **1: 1' ("transformation type")** afgebeeld wordt in de datasource. De eerste groep kenmerkt zich doordat een Java type eenvoudig (direct) kan worden afgebeeld op het typesysteem van de datasource, bijvoorbeeld op een NUMBER of VARCHAR2 in geval van Oracle. De tweede groep vergt een vertaling van het Java type naar een ander, voor de datasource bruikbaar, type alvorens tot opslag kan worden overgegaan.

- *Relationship mappings*
  Dit zijn objectattributen die gerepresenteerd worden door referenties naar andere (TopLink enabled) classes op te nemen. Hierbij zijn twee varianten mogelijk, namelijk: **1:1** (een link van 1 class naar een andere class), of **1:n, m:n** (een verzameling van links).

Net zoals bij descriptors het geval is, geldt bij mappings dat ze aangemaakt en onderhouden kunnen worden met behulp van de workbench. Hiermee wordt de blauwdruk van het class-model gemaakt ten behoeve van de persistency laag.

Database Sessions

Een *Database Session* beschrijft het verband tussen de applicatie en de database en vormt daarmee de link naar de datasource. Dit object wordt ook gebruikt voor het lezen van gegevens uit de datasource.

Unit of Work

Een *Unit of Work* (letterlijk: *"eenheid van werk"*) wordt in de Oracle9iAS TopLink context gebruikt om operaties die gegevens manipuleren (insert/update/delete) te kunnen groeperen.

Werken met TopLink

Nu de basisconcepten van Oracle9iAS TopLink duidelijk zijn, kunnen we aan de hand van een klein voorbeeld het werken met Oracle9iAS TopLink verder beschrijven. Hierbij zal aan een programma waarin gewerkt wordt met een bestaand *class-model,* functionaliteit toegevoegd worden om gegevens weg te

schrijven en op te halen. Het *classmodel* beschrijft de diverse domeinobjecten die weggeschreven en opgehaald dienen te worden (zie afb. 1). Let op: deze applicatie is niet in staat om runtime gegevens op te slaan en biedt dus geen persistency.

```
package com.cumquatit.acme;

public class TestRun  {
  public static void main(String[] args) {
    Employee emp = new Employee();

    Address empAddress = new Address();
    empAddress.setCity("Capelle a/d IJssel");
    empAddress.setCountry("The Netherlands");
    empAddress.setId(new java.math.BigDecimal("1"));
    empAddress.setPostalCode("2909LD");
    empAddress.setProvince("Zuid-Holland");
    empAddress.setStreet("Rivium Westlaan 19");

    PhoneNumber businessPhone = new PhoneNumber("B","010","2881690");
    PhoneNumber businessFax   = new PhoneNumber("F","010","2881688");

    emp.setAddress(empAddress);
    emp.addPhoneNumber(businessPhone);
    emp.addPhoneNumber(businessFax);
    emp.setId(new java.math.BigDecimal("1"));
    emp.setName("Jan Vissers");

    System.out.println(emp);
  }
}
```

Indien we dit programma uitvoeren, zal op een gegeven moment een situatie ontstaan waarbij de object graph aanwezig is. De Oracle JDeveloper debugger is gebruikt om de uitvoering van het programma te onderbreken en vervolgens deze runtime gegevens te kunnen bekijken (zie afb. 2). Als we hier niets aan doen, dan gaan de hierboven getoonde gegevens verloren op het moment dat het programma afgelopen is. Er zijn immers geen voorzieningen aanwezig om de gegevens vast te leggen.

Opslag en toegang

Om deze functionaliteit wel te bieden kan het *classmodel* gebruikt worden als basis voor een *persistency layer*. Met behulp van de Oracle9iAS TopLink Workbench kunnen we een project aanmaken met daarin alle, voor opslag en toegang, belangrijke aspecten. Deze informatie kan vervolgens op runtime gebruikt worden om de gewenste persistency functionaliteit in te vullen (zie afb. 3).

Om te beginnen kunnen we naast dat we informatie geven over de doelomgeving van de datasource, specificeren welke classes van het *classmodel* persistent gemaakt moeten worden. De hierboven getoonde afbeelding geeft aan dat de domein classes *Address, Employee* en *PhoneNumber* onderwerp moeten

Deze groep houdt zich het liefst bezig met Java en stelt de uiteindelijke opslag van gegevens vaak tot het laatst uit

zijn van het project. Vervolgens kan nadere persistency informatie aan deze *descriptors* worden toegevoegd.
Een van de zaken die hierbij aan de orde komt is hoe een class afgebeeld moet worden in de specifieke datasource. Dit is afhankelijk van het al dan niet aanwezig zijn van een bestaand datamodel in de vorm van (database) tabellen.

Point and click

Wanneer een datamodel al aanwezig is en gebruikt moet worden, dan kunnen de classes en bestaande tabellen via 'point-and-click' aan elkaar gekoppeld worden. Meestal zal het gebruik van TopLink echter direct verband houden met het feit dat een top-down benadering gekozen is voor persistency en zullen er geen bestaande tabellen aanwezig zijn. In dat geval kunnen met behulp van de Workbench de benodigde tabellen worden aangemaakt, op basis van de te persisteren *classes* (zie afb. 4). De volgende stap in het definitieproces is het vastleggen van afzonderlijke *mappings* per descriptor. Zoals al eerder vermeld worden hierbij de afzonderlijke objectattributen beschreven in termen van fysieke opslag. Ook deze stap kan worden uitgevoerd met behulp van de workbench (zie afb. 5).

Afbeelding 4. Met behulp van de Workbench kunnen de benodigde tabellen worden aangemaakt op basis van de te persisteren classes

Afbeelding 5. De afzonderlijke objectattributen worden beschreven in termen van fysieke opslag

Omvangrijk arsenaal

Voor de volledigheid wordt hier opgemerkt dat Oracle9iAS TopLink over een omvangrijk arsenaal van geavanceerde technieken beschikt om de persistency definitie van een *classmodel* vorm te geven. Een voorbeeld hiervan is de mogelijkheid om *relationship mappings* te definiëren op basis van *indirection*. Hiermee wordt voorkomen dat bij het teruglezen van een object, de complete *object graph* meteen wordt opgehaald. Eventuele referenties (naar andere objecten) kunnen worden opgehaald, op een "need to know basis". Deze geavanceerde technieken, waaronder ook geoptimaliseerde caching en dergelijke behoren, vallen buiten het bereik van dit artikel.

Blauwdruk

Op het moment dat het totale *classmodel* vastgelegd is in het project, kan de informatie klaargemaakt worden voor gebruik. Deze *deployment* stap resulteert in een XML of Java bestand. Dit eindresultaat bevat de complete blauwdruk van de *persistency laag* en wordt op runtime gebruikt. De volgende onderdelen verduidelijken het gebruik van deze blauwdruk in een Java applicatie.

Database login

In de blauwdruk, in dit geval opgeslagen in een XML bestand **(JV001.xml)**, kan informatie zijn vastgelegd over de doelomgeving van de datasource. Hierdoor is het bijvoorbeeld mogelijk om in te loggen in de concrete database.

```
package com.cumquatit.acme_tl;

import oracle.toplink.tools.workbench.XMLProjectReader;
import oracle.toplink.sessions.Project;
import oracle.toplink.sessions.DatabaseSession;

public class LoginTest  {
  public static void main(String[] args) {

    Project builderProject =
XMLProjectReader.read("E:/Projects/Research/TopLink/JV001.xml");
    DatabaseSession session = builderProject.createDatabaseSession();

    session.login("rnd_toplink","rnd_toplink");
    try {
      Thread.sleep(20000); //Gives us time to check whether an actual
login occurs
    } catch( InterruptedException ignore ) { }

    session.logout();
  }
}
```

Vastleggen van gegevens in een database

Ook hierbij wordt de eerder genoemde blauwdruk gebruikt. Merk op dat de hieronder afgebeelde broncode, dezelfde is als de oorspronkelijke broncode. Dit keer is echter de mogelijkheid tot opslaan van gegevens toegevoegd.

```
package com.cumquatit.acme;

import oracle.toplink.sessions.DatabaseSession;
import oracle.toplink.sessions.Project;
import oracle.toplink.sessions.UnitOfWork;
import oracle.toplink.tools.workbench.XMLProjectReader;

public class TestRun  {
  public static void main(String[] args) {
    Employee emp = new Employee();

    Address empAddress = new Address();
    empAddress.setCity("Capelle a/d IJssel");
    empAddress.setCountry("The Netherlands");
    empAddress.setId(new java.math.BigDecimal("1"));
    empAddress.setPostalCode("2909LD");
    empAddress.setProvince("Zuid-Holland");
    empAddress.setStreet("Rivium Westlaan 19");

    PhoneNumber businessPhone = new PhoneNumber("B","010","2881690");
    PhoneNumber businessFax   = new PhoneNumber("F","010","2881688");

    emp.setAddress(empAddress);
    emp.addPhoneNumber(businessPhone);
    emp.addPhoneNumber(businessFax);
    emp.setId(new java.math.BigDecimal("1"));
    emp.setName("Jan Vissers");

    Project builderProject
      =
XMLProjectReader.read("E:/Projects/Research/TopLink/JV001.xml");
    DatabaseSession session = builderProject.createDatabaseSession();

    session.login("rnd_toplink","rnd_toplink");
    UnitOfWork unitOfWork = session.acquireUnitOfWork();
    Employee tempconsultant = (Employee)unitOfWork.registerObject(emp);
    unitOfWork.commit();
    session.logout();
  }
}
```

De vetgedrukte regels tekst, is de toegevoegde code om opslag van gegevens mogelijk te maken. Details over hoe de diverse objecten inclusief attributen, opgeslagen moeten worden in een bepaalde database is onderdeel van de in de blauwdruk opgenomen definities.

Het is vooral de werkwijze van de ontwikkelaar, die bepalend zal zijn of voor BC4J dan wel voor TopLink gekozen wordt

Opvragen van gegevens uit een database

Hierbij zijn andermaal de vetgedrukte regels tekst, de specifiek toegevoegde code om gegevens uit een database te halen. Hoe dit verder gebeurt, is verder voor de applicatie niet interessant en is vastgelegd in de blauwdruk.

```
package com.cumquatit.acme;

import oracle.toplink.sessions.Project;
import oracle.toplink.sessions.DatabaseSession;
import oracle.toplink.expressions.ExpressionBuilder;
import oracle.toplink.expressions.Expression;
import oracle.toplink.tools.workbench.XMLProjectReader;

import java.util.Vector;
import java.util.Iterator;

public class ReadDB  {
  public static void main(String[] args) {
    Project builderProject =
XMLProjectReader.read("E:/Projects/Research/TopLink/JV001.xml");
    DatabaseSession session = builderProject.createDatabaseSession();

    session.login("rnd_toplink","rnd_toplink");
    ExpressionBuilder builder = new ExpressionBuilder();
    Expression expression = builder.get("name").equal("Vissers");
    Employee employee =
(Employee)session.readObject(Employee.class,expression);

    System.out.println(employee);
    System.out.println(employee.getAddress());
    Vector phoneNumbers = employee.getPhoneNumbers();
    Iterator phoneIter  = phoneNumbers.iterator();
    while( phoneIter.hasNext() ) {
      System.out.println((PhoneNumber)phoneIter.next());
    }

    session.logout();
  }
}
```

Heeft als resultaat:

```
Employee: Vissers
Address: Rivium Westlaan 19, Capelle a/d IJssel, ZH, The Netherlands
PhoneNumber [BP]: (010) 288-1693
PhoneNumber [HP]: (010) 288-1690
```

BC4J versus TopLink

Met BC4J en TopLink biedt Oracle twee alternatieven aan voor het realiseren van de persistency laag ten behoeve van een Java/J2EE applicatie. Op hoofdlijnen zijn beide producten in staat om dezelfde functionaliteit in te vullen. Daarom is de vraag gerechtvaardigd waarom Oracle TopLink heeft opgenomen in het productengamma.
Het is voornamelijk de werkwijze van de ontwikkelaar, die bepalend zal zijn of voor BC4J dan wel voor TopLink gekozen wordt (indien dit de twee opties zijn). Zoals al eerder werd opgemerkt zal vooral de OO community een aanpak prefereren die uitgaat van een classmodel. In veel gevallen (maar niet in alle!) kenmerkt zich deze groep door het feit dat ze van Java houden, maar een broertje dood hebben aan SQL. Deze groep wil zich het liefst uitsluitend bezighouden met Java en stelt (vaak) de uiteindelijke opslag van gegevens tot het laatst uit. In potentie een zeer gevaarlijke aanpak overigens, daar deze verantwoordelijk gehouden kan worden voor een groot gedeelte van de vaak voorkomende performance problemen van (J2EE) applicaties.

BC4J hanteert een andere filosofie en zal vooral ontwikkelaars met een "traditionele" SQL achtergrond aanspreken. Deze groep is gewend om terecht veel aandacht te besteden aan de manier waarop data opgeslagen en bevraagd worden. In veel gevallen zullen in het bijzonder organisaties met een bestaande Oracle infrastructuur, die zich willen begeven op het Java pad, gecharmeerd zijn van BC4J. Zij weten immers dat er geen betere database bestaat dan Oracle en hebben doorgaans al veel tijd en geld geïnvesteerd in het opzetten van een robuuste database. Het feit dat BC4J, naast persistency, nog tal van andere design patterns implementeert maakt dit framework dan tot een zeer aantrekkelijke optie. In de toekomst is wel een combinatie van beide mogelijk. Hierbij is het denkbaar dat BC4J gebruik maakt van de faciliteiten die door TopLink, als meer neutrale "mapping" oplossing, geboden worden.

Conclusie

Met Oracle9iAS TopLink komt Oracle tegemoet aan de Java community, met als doel om het marktaandeel voor de complete Java toolstack (Oracle9i Aplication Server en Oracle9i JDeveloper) te vergroten. Deze groep werkt vanuit een Java classmodel en wil zich liever niet met SQL bezighouden. Met Business Components for Java is deze manier van werken ook mogelijk, maar ligt minder voor de hand. Bovendien wordt BC4J, deels ten onrechte, gezien als een proprietary oplossing van Oracle. Mogelijk kan dit voor potentiële gebruikers een drempel vormen.

Jan Vissers
is senior consultant bij Cumquat Information Technology. Cumquat richt op het bieden van oplossingen op het gebied van Self-Service applicaties, Portals, Webservices en Business-to-Business integratie. Hierbij gebruikmakend van het toepassen van XML en Java technologie op basis van het Oracle platform.