JEE SQL

Vilka typer av SQL finns i JEE

För att persistera data till en SQL databas, måste ju givetvis SQL användas. Men hur bör man skriva sina SQL frågor för optimering?

Det finns ett antal olika typer av SQL varianter i Java och vi ska titta närmare på dessa nu.

- Native Query
- JPQL
  - Dynamic Query
  - Named Parameters
  - Named Query
  - Typed Query

Native Query

Native Query är ren SQL. Man skriver en sträng direkt till EntityManager som sen skickar strängen till servern som parsear den till sin databas koppling.

Fördelen med Native Query är att den alltid kommer fungera. Har man skrivit den rätt i sitt shell, kommer den bli rätt i Java med.

SQL arbetar som bekant med tabell och kolumn namn i sina frågor.

JPQL är JPA's egna Query språk. Skillnaden mot Native Query är att JPQL arbetar med Entiteter.

JPQL är i sin tur uppdelat i olika typer av frågor:

Dynamic, Typed, Named och *Criteria*.

Dynamic Query

Dynamiska frågor skapas vid Runtime, när en SQL sträng skickas in i createQuery metoden i EntityManager klassen.

Fördelen är i situtationer när frågan inte är helt klarställd förräns vid Runtime utan har krav på en användares val.

Nackdelen är säkerheten. Eftersom frågorna lätt kan speglas I Stacktrace, kan en Hacker utan större problem kunna ta reda på olika paramterar och därefter göra SQL Injections.

Named Parameters

Named Parameters tar bort den största risken med att exponera sina förväntade parametrar. Genom att använda parametrar istället som genom *marshalling* blir en del av frågan, kan inte frågan förändras, utan istället måste man använda *.setParameter()* metoden i Query för att binda användarens val.

Man kan använda sig av *Positional Parameters* om man hellre vill det, där frågan istället för att fråga efter en parameter tar in ett Query Param med en siffra som man sen i *.setParameter()* metoden refererar till.

Named Query

@NamedQuery skrivs i Entitetsklassen. Varje fråga man vill den entiteten ska ha, måste annoteras med @NamedQuery, men de kan nästas in i *@NamedQueries({ })*. I transaktionslagret använder man då *.createNamedQuery* och anropar namnet man skrivit i NamedQuery *samt* klassen den finns i.

Fördelen med dessa är att entitets specifika frågor hamnar i rätt klass, vilket gör det mer underhållsbart för utvecklare.

Typed Query

Typed Query låser frågan till en entitet, vilket gör att en fråga inte returnerar ett rent Java Objekt, utan ett Objekt av entitetstypen.

Fördelen med detta är att typsäkerheten blir betydligt högre, när man slipper göra ett cast. Detta genererar *mindre* mängd Exceptions att ta hand om och gör det betydligt enklare att skriva tester.

Criteria API

Criteria API är ett bra tillägg när man måste använda sig av AND / OR i sina frågor.

Istället för att skriva långa frågesträngar, kan man använda sig av en *Root* som är ens Entitet. Därefter använder mansig av *Predicate* och bygger med frågot gentemot sin Root.

T.ex Root.get(Student.enrolled), date;

På detta sätt kommer variablen enrolled frågas, med datumet i variablen date.