Posudek vedoucího diplomové práce

Michal Pravda: Výhody a nevýhody použití perzistence objektů v jazyce Java

Cílem této práce byla analýza možností nástroje *Hibernate* pro perzistenci objektů v jazyce Java a srovnání výhod a nevýhod, které jeho použití přináší v porovnání se standardním přístupem prostřednictvím JDBC rozhraní. Důležité tedy bylo navrhnout, vytvořit a vyhodnotit testy zaměřené na efektivitu přístupu k relačním a objektově relačním databázím z programů psaných v jazyce Java. Porovnány měly být přímý přístup k datům přes JDBC rozhraní a přístup, využívající objektově–relační mapování nástroje Hibernate pro dosažení perzistence objektů.

Začátek práce je věnován seznámení s problematikou perzistence obecně a úvodu do práce s oběma prostředími, knihovnou *Hibernate* a JDBC.

Návrhu srovnávacích hledisek je věnována pátá kapitola práce. Za hlavní kritéria byly označeny doba vývoje nutné pro implementaci manipulace s databází, výkon, kterého aplikace při použití toho kterého přístupu dosahuje a zpracovatelnost výsledné aplikace. Tyto kritéria byla testována i s ohledem na komplexitu dat, která mají být v databázi ukládána.

Zajímavé jsou dle mého názoru ty výsledky, kde realizované testy narážejí na hranice možností jednotlivých prostředí. Jedná se o případy použití objemných dávek příkazů (str. 51), kde dochází k chybám v implementaci, manipulace s objemnými rekurzivními strukturami, které vyžadují pro rekurzi příliš velký objem volné paměti (str. 53), vytváření komplikovaných hierarchií objektů, které mohou narážet na limity pro relační tabulky (str. 62), zpomalovat čtení dat z hierarchií o velké hloubce (str. 63) a podobně.

K testům a jejich výsledkům bych měl několik dotazů:

- Na straně 44 je srovnání času, který zabralo dosažení perzistence pro různě složité objekty v databázi. Až na první, nejjednodušší případ, jako výhodnější a časově méně náročné vychází *Hibernate*. V prvním případě *Hibernate* prohrává kvůli menší strmosti učící křivky. Tento graf zřejmě odpovídá prvnímu většímu projektu psanému v tom kterém prostředí. Dá se odhadnout, nakolik jsou časy pro JDBC způsobené „hledáním té správné cesty“, a nakolik její implementací?
- V popisu na straně 54 je uvedeno, že Hibernate vítězí nad JDBC při manipulaci s mnoha nezávislými objekty ve většině případů vítězí díky tomu, že shromažďuje více příkazů do dávek a odesílá je na server vždy po několika najednou. Předpokládám ale, že toto řešení je dosažitelné i přes JDBC za cenu (možná podstatně) delšího vývoje aplikace samotné.
- Výsledky jsou uváděny zvlášť pro SELECT, INSERT a pro UPDATE. Budou výsledky obdobné i v případě, že budou všechny operace navzájem promíchané, jako je tomu u reálných aplikací, nebo bude *Hibernate* znevýhodněn tím, že (možná) nedokáže operace různých typů seskupovat do dávek, a bude nucen je provádět samostatně?
- Na obr. 10 na str. 55 je uveden graf pro načítání acyklických grafů. Knihovna *Hibernate* vítězí díky implementaci líného nahrávání. Na straně 56 je uveden graf pro případ, kdy si aplikace vynutí načtení všech objektů pomocí rekurzivního průchodu grafem. Přesto jsou výsledky pro *Hibernate* podstatně rychlejší. Čím je rozdíl v rychlostech způsoben?

V práci bych uvítal podrobnější informace o schématu (kromě těch uměle generovaných), nad kterým byly testy prováděné. Zdá se, že se nejedná o třídy Address, BillingDetails a podobné, uváděné v příkladech v první polovině práce. Sjednocení příkladů by prospělo konzistenci práce. U názvosloví jednotlivých typů mapování bych dále upřednostnil spíše pojmy „horizontální“, „vertikální“, „filtrované“ uvedené na str. 33 jako alternativní, před názvoslovím „tabulka pro každou třídu“, „tabulka pro každou konkrétní třídu“ a podobně.

Celkově se domnívám, že práce splnila cíle, se kterými byla zadána, a doporučuji ji uznat jako práci diplomovou.

V Praze dne 15. 5. 2007