Kapitola 31

Uložené rutiny

V průběhu práce s knihou jste viděli poměrně dost příkladů, v nichž se dotazy MySQL vkládaly přímo do skriptu PHP. U menších aplikací je to jistě v pořádku, jak ale složitost a velikost aplikací narůstá, mohlo by pokračování v takové praxi skončit až fiaskem. Například co když máte rozmístit dvě podobné aplikace, jednu desktopovou a druhou webovou, které obě používají databázi MySQL a provádějí mnoho stejných úkolů? Když je tu a tam třeba změnit nějaký dotaz, budete muset provést patřičné modifikace všude, kde se daný dotaz objevuje, a to ne v jedné, ale ve dvou aplikacích!

Jinou výzvou, které čelíte, když pracujete se složitými aplikacemi, zvlášť když je vyvíjíte v týmu, je poskytnout každému členu týmu příležitost, aby mohl přispívat ke zdaru projektu na základě svých odborných znalostí a kvalifikace, aniž by přitom šlapal po úsilí ostatních. Typické je, že ti lidé, kteří jsou zodpovědní za vývoj databáze a údržbu (známí jako databázoví architekti), jsou neobyčejně erudovaní v psaní efektivních a bezpečných dotazů. Ale jak může takový databázový architekt psát a udržovat tyto dotazy, aniž by nelezl do zelí vývojáři aplikace, když jsou dotazy vložené přímo do kódu? Navíc, jak se může databázový architekt spolehnout na to, že vývojář následně „nezdokonalí" tyto dotazy, a potenciálně tím otevře dveře infiltracím v podobě útoků injektáží SQL (které spočívají v tom, že se modifikují data odesílaná do databáze se záměrem, aby se pak dal spustit nějaký škodlivý kód SQL)?

Jedním z nejběžnějších řešení těchto výzev je databázová schopnost, které se říká *uložená rutina – stored routine*. Uložená rutina je nějaká sada příkazů SQL, která je uložená na databázovém serveru a vykonává se tak, že se zavolá prostřednictvím dotazu názvem, který jí byl přiřazen. V mnohém je to obdoba funkce, protože ta také zapouzdřuje nějakou sadu příkazů, které se vykonají poté, co se funkce zavolá svým názvem. Uložená rutina se dá udržovat za bezpečnými zdmi databázového serveru a na kód aplikace nemusíte ani sáhnout.

Od verze 5.0 už MySQL konečně podporuje tuto dlouho toužebně očekávanou schopnost. V této kapitole se dozvíte vše potřebné o tom, jak MySQL implementuje uložené rutiny. Probereme syntax a uvidíte, jak se uložené rutiny vytvářejí, spravují a vykonávají. Také se naučíte začleňovat uložené rutiny do svých webových aplikací. Na začátku se však chvilku zastavme u formálnějšího souhrnu jejich výhod a nevýhod.

Měli bychom používat uložené rutiny?

Není žádoucí, abyste se naslepo nadšeně vrhli do módních vln uložených rutin. Vyplatí se, když v krátkosti posoudíme jejich přednosti a stinné stránky, zejména proto, že jejich užitečnost je v databázové komunitě předmětem vášnivých debat. V tomto oddílu najdete souhrn pro a proti, uvažujete-li o začlenění uložených rutin do své vývojářské strategie.

Přednosti uložených rutin

Uložené rutiny mají mnoho předností, nejvýznačnější z nich jsou zdůrazněné zde:

- **Eliminuje se redundance**. Když mnoho aplikací napsaných v různých jazycích vykonává stejné databázové úkoly, tak shromáždění těchto úkolů do uložených rutin podobných funkcím redukuje jinak běžnou redundanci vývojových procesů.
- **Vyšší výkon**. Kompetentní databázový administrátor je pravděpodobně ten nejkompetentnější člen týmu, když jde o to, jak psát optimalizované dotazy. Proto je rozumné, když se tvorba velmi komplikovaných operací, které se týkají databáze, přenechá tomuto individuu tím, že se takové operace udržují jako uložené rutiny.
- **Větší bezpečnost**. Když pracujete ve zvlášť citlivých prostředích, jako jsou bankovnictví, zdravotnictví nebo obrana státu, je někdy oficiálně nařízeno, aby byl přístup k datům přísně restriktivní. Uložené rutiny nabízejí skvělý způsob, jak zajistit, aby měli vývojáři přístup jen k těm informacím, které nezbytně potřebují k řešení svých úkolů..
- **Snazší údržba rozsáhlých aplikací**. Přestože diskuse o výhodách vícevrstevných architektur přesahuje rámec této knihy, používání uložených procedur v součinnosti s vrstvou dat může dost usnadnit údržbu rozsáhlých aplikací. Chcete-li se o tomto tématu dozvědět víc, zadejte ve svém webovém vyhledávači termín „n-tier architecture“.

Nevýhody uložených rutin

Přestože vás předchozí výčet předností přesvědčil, že jsou uložené rutiny ta cesta, kterou se vydáte, zamyslete se chvilku nad jejich nevýhodami:

- **Vyšší spotřeba prostředků**. Mnozí argumentují tím, že jediný účel databáze je ukládat data a udržovat relace mezi nimi, ne vykonávat kód, který by jinak vykonala aplikace. Kromě toho, že se tím narušuje jediný hlavní účel databáze, vykonávání takové logiky z databáze spotřebovává také dodatečné prostředky procesoru a paměti.
- **Menší zdatnost**. Jak se brzy dozvíte, jazykové konstrukce SQL nabízejí bohatou výbavu i značnou flexibilitu; většina vývojářů však přišla na to, že budovat tyto rutiny je snadnější i pohodlnější, když se to dělá v nějakém vyspělém plnohodnotném jazyku, jakým je PHP.
- **Horší udržovatelnost rutin**. Přestože se dají pro správu uložených rutin používat utility s grafickým uživatelským rozhraním , jako je MySQL Query Browser (viz kapitola 26), psaní kódu i ladění uložených rutin je o hodně obtížnější, než když píšete funkce PHP ve zdatném integrovaném vývojovém prostředí.

- **Obtížnější přenositelnost**. Protože uložené rutiny často používají syntax, která je specifická pro danou databázi, určitě se vynoří nějaké potíže s přenositelností, budete-li potřebovat používat aplikaci v součinnosti s jiným databázovým produktem

Nuže, i potom, co jste se seznámili s výhodami i nevýhodami, možná si stále nejste jisti, zda jsou uložené rutiny pro vás to pravé. Asi nejlepší rada, kterou je možno v tomto ohledu poskytnout, je, abyste si přečetli tuto kapitolu a podnikli všelijaké experimenty s četnými příklady, které v ní najdete.

Jak MySQL implementuje uložené rutiny

Přestože se všude okolo omílá termín *uložené procedury*, MySQL ve skutečnosti implementuje dvě varianty procedur, na které se odkazuje společně jako na *uložené rutiny*:

- **Uložené procedury**. Uložené procedury podporují vykonávání příkazů SQL jako jsou `SELECT`, `INSERT`, `UPDATE` a `DELETE`. Mohou také připravovat parametry, na které se dá odkazovat později, vně procedury.
- **Uložené funkce**. Uložené funkce podporují vykonávání pouze příkazů `SELECT`, akceptují jen vstupní parametry a musejí vracet právě jednu hodnotu. Kromě toho můžete vložit uloženou funkci přímo do příkazu SQL tak, jak to děláte se standardními funkcemi MySQL, jako jsou `count()` a `date_format()`.

Všeobecně řečeno, uložené procedury byste měli používat tehdy, když potřebujete pracovat s daty nacházejícími se v databázi, třeba získávat řádky, vkládat, odstraňovat nebo aktualizovat hodnoty, zatímco s pomocí uložených funkcí byste měli s těmito daty manipulovat nebo provádět speciální výpočty. Syntax prezentovaná v průběhu kapitoly je ovšem pro obě varianty prakticky identická, kromě toho, že se přepíná termín „procedura“ a „funkce“. Například, příkazem `DROP PROCEDURE název_procedury` se odstraní existující uložená procedura, kdežto příkazem `DROP FUNCTION název_funkce` se odstraní existující uložená funkce.

Tabulky přístupových oprávnění uložených rutin

Ti z vás, kdo už s MySQL nějaký čas pracujete, dobře víte, že v databázi mysql přibyly nové tabulky. Dvě z nich, `proc` a `procs_priv`, slouží ke správě uložených rutin a přístupových oprávnění, která se požadují, chcete-li je vytvářet, vykonávat, měnit nebo odstraňovat.

proc

V tabulce `proc` jsou uložené informace o uložené rutině, mezi něž patří její syntax, datum, kdy byla vytvořena, seznam jejích parametrů a další věci. Její strukturu prezentuje tabulka 31-1.

Tabulka 31-1. Tabulka proc databáze mysql

Sloupec	Datový typ	Null	Výchozí hodnota
db	char(64)	Ano	Není
name	char(64)	Ne	Není
type	enum('FUNCTION','PROCEDURE')	Ne	Není

Sloupec	Datový typ	Null	Výchozí hodnota
specific_name	char(64)	Ne	Není
language	enum('SQL')	Ne	SQL
sql_data_access	*enum přístupu k datům*	Ne	CONTAINS_SQL
is_deterministic	enum('YES', 'NO')	Ne	NO
security_type	enum('INVOKER','DEFINER')	Ne	DEFINER
param_list	blob	Ne	Není
returns	char(64)	Ne	Není
body	longblob	Ne	Není
definer	char(77)	Ne	Není
created	timestamp	Ano	CURRENT_TIMESTAMP
modified	timestamp	Ano	0000-00-00 00:00:00
sql_mode	*mmožina módů sql*	Ne	Není
comment	char(64)	Ne	Není

Aby nebyl sloupec tabulky pro datový typ zoufale široký, je termín *enum přístupu k datům* zástupce skutečného výčtu

```
enum('CONTAINS_SQL', 'NO_SQL', 'READS_SQL_DATA', 'MODIFIES_SQL_DATA')
```

a termín *množina módů sql* zastupuje množinu

```
set('REAL_AS_FLOAT', 'PIPES_AS_CONCAT', 'ANSI_QUOTES', 'IGNORE_SPACE', 'NOT_USED',
'ONLY_FULL_GROUP_BY', 'NO_UNSIGNED_SUBTRACTION', 'NO_DIR_IN_CREATE', 'POSTGRESQL',
'ORACLE', 'MSSQL', 'DB2', 'MAXDB', 'NO_KEY_OPTIONS', 'NO_TABLE_OPTIONS',
'NO_FIELD_OPTIONS', 'MYSQL323', 'MYSQL40', 'ANSI', 'NO_AUTO_VALUE_ON_ZERO',
'NO_BACKSLASH_ESCAPES', 'STRICT_TRANS_TABLES', 'STRICT_ALL_TABLES',
'NO_ZERO_IN_DATE', 'NO_ZERO_DATE', 'INVALID_DATES', 'ERROR_FOR_DIVISION_BY_ZERO',
'TRADITIONAL', 'NO_AUTO_CREATE_USER', 'HIGH_NOT_PRECEDENCE').
```

Všechny sloupce se podrobněji probírají v oddílu „Jak se vytvoří uložená rutina“.

procs_priv

V tabulce `procs_priv` jsou uloženy informace o přístupových oprávněních vyjadřující, kteří uživatelé mohou komunikovat s rutinami definovanými v tabulce proc. Struktura tabulky `procs_priv` je uvedena v tabulce 31-2.

Tabulka 31-2. Tabulka procs_priv databáze mysql

Sloupec	Datový typ	Null	Výchozí hodnota
Host	char(60)	Ne	Není
Db	char(64)	Ne	Není
User	char(16)	Ne	Není
Routine_name	char(64)	Ne	Není
Routine_type	enum('FUNCTION', 'PROCEDURE')	Ne	Není