Bazy danych

Haskell

Maria Budziło, Marcin Kubański, Hiacynta Stypuła

18.01.2022

Politechnika Krakowska
Wydział Inżynierii
Materiałowej i Fizyki

Plan prezentacji

- Wstęp
- Przegląd HDBC
- Instalacja HDBC oraz Sterowników
- Podłączanie się do baz danych
- Zagadnienie transakcji
- Proste zapytania
- SqlValue
- Parametry zapytania
- Prepared Statement - sparametryzowana instrukcja
- Odczytywanie wyników
- Czytanie danych za pomocą instrukcji
- Lazy Reading
- Metadane baz danych
- Zgłaszanie błędów
- Bibliografia

Wstęp

Ponieważ bazy danych są tak ważne, wsparcie Haskella dla nich jest również ważne. Przedstawimy teraz jeden z frameworków Haskella do pracy z bazami danych - **HDBC** *(Haskell Database Connectivity)*.

Przegląd HDBC

Silnik bazy danych odpowiada za przechowywanie danych na dysku.

Standardowym sposobem pobierania danych do i z relacyjnych baz danych jest **SQL** *(Structured Query Language)*.

Możliwe jest stworzenie ogólnego interfejsu, który używa sterowników dla każdego indywidualnego protokołu.

Haskell udostępnia kilka różnych frameworków bazodanowych.

Przegląd HDBC

HDBC *(Haskell DataBase Connectivity)* to biblioteka abstrakcji bazy danych. System sterowników HDBC daje dostęp do wielu opcji za pomocą jednego interfejsu.

Instalacja HDBC oraz Sterowników

Potrzebujemy co najmniej dwóch pakietów: ogólnego interfejsu i sterownika dla konkretnej bazy danych.

Będziemy także potrzebowali backendu bazy danych i sterownika backendu. Dlatego też użyjemy Sqlite w wersji 3. Konkretny sterownik dla Sqlite w wersji 3 można uzyskać z Hackage.

Jeśli chcemy używać HDBC z innymi bazami danych, sprawdzamy stronę znanych sterowników HDBC pod adresem `http://software.complete.org/hdbc/wiki/KnownDrivers`.

Podłączanie się do baz danych

Aby połączyć się z bazą danych, używamy funkcji połączenia ze sterownika backendowego bazy danych. Każda baza danych ma swoją unikalną metodę łączenia.

```
marcinstudia@MarcinPC:~/Desktop$ ghci
GHCi, version 8.6.5: http://www.haskell.org/ghc/  :? for help
Prelude> :module Database.HDBC Database.HDBC.Sqlite3
Prelude Database.HDBC Database.HDBC.Sqlite3> conn <- connectSqlite3 "test1.db"
Prelude Database.HDBC Database.HDBC.Sqlite3> :type conn
conn :: Connection
Prelude Database.HDBC Database.HDBC.Sqlite3> disconnect conn
Prelude Database.HDBC Database.HDBC.Sqlite3>
```

Rysunek: Przykład nawiązania połączenia z bazą danych.

Zagadnienie transakcji

Transakcja ma na celu zapewnienie, że wszystkie składniki modyfikacji zostaną zastosowane lub że żaden z nich nie zostanie zastosowany.
Transakcje pomagają także zapobiegać wyświetlaniu części danych pochodzących z trwających modyfikacji innym procesom uzyskującym dostęp do tej samej bazy danych.

Zagadnienie transakcji

HDBC celowo nie obsługuje trybu automatycznego zatwierdzania. Należy jawnie doprowadzić do ich zatwierdzenia na dysku.

W HDBC można to zrobić poprzez użycie funkcji ***withTransaction***, aby otoczyć kod modyfikacji. Funkcja ta spowoduje przekazanie danych po pomyślnym zakończeniu funkcji.

Zagadnienie transakcji

W HDBC w celu zatwierdzenia lub wycofania zmian używamy także funkcji *withTransaction*. Każdy nieprzechwycony wyjątek spowoduje wycofanie.

Proste zapytania

```
Prelude Database.HDBC Database.HDBC.Sqlite3> conn <- connectSqlite3 "test1.db"
Prelude Database.HDBC Database.HDBC.Sqlite3> run conn "CREATE TABLE test (id INTEGER NOT NULL, desc VARCHAR(80))" []
0
Prelude Database.HDBC Database.HDBC.Sqlite3> run conn "INSERT INTO test (id) VALUES (0)" []
1
Prelude Database.HDBC Database.HDBC.Sqlite3> commit conn
Prelude Database.HDBC Database.HDBC.Sqlite3> disconnect conn
```

Rysunek: Najbardziej podstawowa funkcja wysyłania zapytań do bazy danych, używana do ustawienia niektórych rzeczy w bazie danych.

SqlValue

SqlValue to typ danych, mający wiele konstruktorów, takich jak SqlString, SqlBool, SqlNull, SqlInteger...

Pozwala on reprezentować różne typy danych na listach argumentów w bazie danych oraz wyświetlać różne typy danych w przychodzących wynikach, przy jednoczesnym przechowywaniu ich wszystkich na liście.

Parametry zapytania

HDBC obsługuje w zapytaniach pojęcie parametrów zastępowalnych, które mają kilka ważnych zalet.

```
Prelude Database.HDBC Database.HDBC.Sqlite3> conn <- connectSqlite3 "test1.db"
Prelude Database.HDBC Database.HDBC.Sqlite3> run conn "INSERT INTO test VALUES (?, ?)" [toSql 0, toSql "zero"]
1
Prelude Database.HDBC Database.HDBC.Sqlite3> commit conn
Prelude Database.HDBC Database.HDBC.Sqlite3> disconnect conn
```

Rysunek: Przykład nawiązania połączenia z bazą danych.

Prepared Statement – sparametryzowana instrukcja

HDBC definiuje funkcję przygotowania,przygotowującą zapytanie SQL, ale nie wiąże jeszcze parametrów z zapytaniem. Przygotowanie zwraca instrukcję reprezentującą skompilowane zapytanie.

```
Prelude Database.HDBC Database.HDBC.Sqlite3> conn <- connectSqlite3 "test1.db"
Prelude Database.HDBC Database.HDBC.Sqlite3> stmt <- prepare conn "INSERT INTO test VALUES (?, ?)"
Prelude Database.HDBC Database.HDBC.Sqlite3> execute stmt [toSql 1, toSql "one"]
1
Prelude Database.HDBC Database.HDBC.Sqlite3> execute stmt [toSql 2, toSql "two"]
1
Prelude Database.HDBC Database.HDBC.Sqlite3> execute stmt [toSql 3, toSql "three"]
1
Prelude Database.HDBC Database.HDBC.Sqlite3> execute stmt [toSql 4, SqlNull]
1
Prelude Database.HDBC Database.HDBC.Sqlite3> commit conn
Prelude Database.HDBC Database.HDBC.Sqlite3> disconnect conn
```

```
Prelude Database.HDBC Database.HDBC.Sqlite3> conn <- connectSqlite3 "test1.db"
Prelude Database.HDBC Database.HDBC.Sqlite3> stmt <- prepare conn "INSERT INTO test VALUES (?, ?)"
Prelude Database.HDBC Database.HDBC.Sqlite3> executeMany stmt [[toSql 5, toSql "five's nice"], [toSql 6, SqlNull]]
Prelude Database.HDBC Database.HDBC.Sqlite3> commit conn
Prelude Database.HDBC Database.HDBC.Sqlite3> disconnect conn
```

Odczytywanie wyników

Typ funkcji ***quickQuery'***, używany zwykle z instrukcjami ***SELECT***, wygląda bardzo podobnie do *run*, ale zwraca listę wyników zamiast liczby zmienionych wierszy.

```
Prelude Database.HDBC Database.HDBC.Sqlite3> conn <- connectSqlite3 "test1.db"
Prelude Database.HDBC Database.HDBC.Sqlite3> quickQuery' conn "SELECT * from test where id < 2" []
[[SqlInt64 0,SqlNull],[SqlInt64 0,SqlByteString "zero"],[SqlInt64 1,SqlByteString "one"]]
```

Rysunek: Przykład odczytania wyników z bazy danych.

Odczytywanie wyników

Niestety, odczytywane wyników wygląda niezbyt dobrze więc aby to uprościć użyjemy tego kodu:

```haskell
import Database.HDBC.Sqlite3 (connectSqlite3)
import Database.HDBC

query :: Int -> IO ()
query maxId =
    do
       conn <- connectSqlite3 "test1.db"

       r <- quickQuery' conn
            "SELECT id, desc from test where id <= ? ORDER BY id, desc"
            [toSql maxId]

       let stringRows = map convRow r

       mapM_ putStrLn stringRows

       disconnect conn

    where convRow :: [SqlValue] -> String
          convRow [sqlId, sqlDesc] =
              show intid ++ ": " ++ desc
              where intid = (fromSql sqlId)::Integer
                    desc = case fromSql sqlDesc of
                             Just x -> x
                             Nothing -> "NULL"
          convRow x = fail $ "Unexpected result: " ++ show x
```

Rysunek: Przykład kodu do łatwiejszego odczytywania wyników.

Odczytywanie wyników

Następnie kompilujemy plik query.hs:

```
marcinstudia@MarcinPC:~/Desktop$ ghci
GHCi, version 8.6.5: http://www.haskell.org/ghc/  :? for help
Prelude> :load query.hs
[1 of 1] Compiling Main             ( query.hs, interpreted )
Ok, one module loaded.
*Main> query 2
0: NULL
0: zero
1: one
2: two
*Main>
[2]+  Stopped                 ghci
```

Rysunek: Wyświetlenie wyników za pomocą pliku query.hs w GHCi.

Wyniki z bazy danych są teraz bardziej przejrzyste.

Czytanie danych za pomocą instrukcji

- Funkcja ***fetchAll Rows*** zwraca ***[[SqlValue]]***, podobnie jak *quickQuery'*.
- ***sFetchAllRows'*** konwertuje dane każdej kolumny przed ich zwróceniem.
- ***fetchAllRowsAL'*** zwraca pary *(String, SqlValue)* dla każdej kolumny.

Lazy Reading

Lazy Reading jest szczególnie przydatne w przypadku zapytań zwracających stosunkowo dużą ilość danych. Jednak jest ono bardziej złożone niż czytanie z pliku.

Niektóre bazy danych mogą nawet obsługiwać wiele jednoczesnych zapytań, więc HDBC nie może po prostu zamknąć połączenia, gdy skończymy zadanie.

Lazy Reading

W przypadku korzystania z Lazy Reading niezwykle ważne jest, aby zakończyć odczytywanie całego zestawu danych przed próbą zamknięcia połączenia lub wykonania nowego zapytania.

```
Prelude Database.HDBC Database.HDBC.Sqlite3> conn <- connectSqlite3 "test1.db"
Prelude Database.HDBC Database.HDBC.Sqlite3> stmt <- prepare conn "SELECT * from test where id < 2"
Prelude Database.HDBC Database.HDBC.Sqlite3> execute stmt []
0
Prelude Database.HDBC Database.HDBC.Sqlite3> results <- fetchAllRowsAL stmt
Prelude Database.HDBC Database.HDBC.Sqlite3> mapM_ print results
[("id",SqlInt64 0),("desc",SqlNull)]
[("id",SqlInt64 0),("desc",SqlByteString "zero")]
[("id",SqlInt64 1),("desc",SqlByteString "one")]
Prelude Database.HDBC Database.HDBC.Sqlite3> disconnect conn
```

Rysunek: Przykład Lazy Reading.

Metadane baz danych

- Funkcja ***getTables*** uzyskuje listę zdefiniowanych tabel w bazie danych. Funkcja ***defineTable*** dostarcza informacji o zdefiniowanych kolumnach w danej tabeli.
- Można wywołać **dbServerVer** i **proxiedClientName**, by dowiedzieć się czegoś o aktualnie używanym serwerze bazy danych.
- Funkcja ***dbTransactionSupport*** pozwala określić, czy dana baza danych obsługuje transakcje.

```
Prelude Database.HDBC Database.HDBC.Sqlite3> conn <- connectSqlite3 "test1.db"
Prelude Database.HDBC Database.HDBC.Sqlite3> getTables conn
["test"]
Prelude Database.HDBC Database.HDBC.Sqlite3> proxiedClientName conn
"sqlite3"
Prelude Database.HDBC Database.HDBC.Sqlite3> dbServerVer conn
"3.31.1"
Prelude Database.HDBC Database.HDBC.Sqlite3> dbTransactionSupport conn
True
```

Rysunek: Funkcji *dbTransactionSupport*, za pomocą której można określić, czy dana baza danych obsługuje transakcje.

Zgłaszanie błędów

```
Prelude Database.HDBC Database.HDBC.Sqlite3> conn <- connectSqlite3 "test1.db"
Prelude Database.HDBC Database.HDBC.Sqlite3> quickQuery' conn "SELECT * from test2" []
*** Exception: SqlError {seState = "", seNativeError = 1, seErrorMsg = "prepare 20: SELECT * from test2: no such table: test2"}
```

Rysunek: Informacja o błędzie w bazie danych

Zgłaszanie błędów

HDBC zgłosi wyjątki, gdy wystąpią błędy.

SqlError przekazuje informacje z bazowego silnika SQL o stanie bazy danych, komunikat o błędzie i numeryczny kod błędu bazy danych.

```
Prelude Database.HDBC Database.HDBC.Sqlite3> handleSqlError $ quickQuery' conn "SELECT * from test2" []
*** Exception: user error (SQL error: SqlError {seState = "", seNativeError = 1, seErrorMsg = "prepare 20: SELECT * from test2: no such table: test2"})
Prelude Database.HDBC Database.HDBC.Sqlite3>
```

Rysunek: Informacja o błędzie w bazie danych

Bibliografia

- Materiały udostępnione w pliku „3Database_Haskell.pdf”
- https://hackage.haskell.org/
- http://www.sqlite.org/

Dziękujemy za uwagę!