PROTODEMOS

PD/DBL

Introducing Protodemos PD/DBL - The first database programming and stored procedure language designed to support Map-Reduce in a Massively Parallel Processing environment.

Taking advantage of the advanced APIs provided by Teradata Aster* database, PD/DBL programs run in parallel on hundreds of commodity servers. PD/DBL embraces the entire Map-Reduce flow of data, allowing separate program elements for setup, data ingest, map, reduce operations, and post processing operations.

Features and Functionality

- **Script entry and execution at the command line:**
  No java programming needed. PD/DBL programs can be executed directly from any JDBC or ODBC query tool or ACT.
- **Stored procedure-like functionality:**
  PD/DBL programs can be stored as schema objects in the Teradata Aster database and executed in-place.
- **Enhanced security:**
  Ad-hoc database programming without using an interface that requires OS-level or other non-secure access. PD/DBL programs and stored procedures do not allow outside network access or arbitrary method execution, exposing only the interfaces required for database interaction and program execution. Database-level access control of stored procedures is supported via GRANT USAGE ON statements.
- **Large function library:**
  A full complement of built-in functions for string handling, math, regular expressions, data manipulation, and more is included.
- **Full support for Teradata Aster Sql-MR:**
  PD/DBL is built on top of the Teradata Aster Sql-MR Application Programming Interface and supports:
  - Running as map, reduce or multi-input functions and emitting output records to the database
  - Controlling node execution- stored procedures may be executed on one, several, or a "modulus" of worker nodes
  - Passing options as input parameters to programs and stored procedures
  - Using text and XML files stored in the nCluster database as data for PD/DBL stored procedures
- **JDBC support:**
  Connections and Cursors: Connect to a JDBC database and execute arbitrary SQL- or create a cursor and iterate through it, reading, updating, inserting and deleting records with full support for transaction integrity including rollback and commit operations.
- **High-level language constructs:**
  PD/DBL is very similar to typical current programming languages, with familiar elements such as:
  - Flow control constructs like if/then/elseif/else, branch-on (similar to a java "switch"), for, foreach and do/while/for looping
  - User-defined functions with return values, by-reference and by-value arguments, and recursion
  - Simple data primitives (boolean, date, integer, numeric, string and timestamp)
  - Complex variables like records, arrays, maps and stack
  - Database objects like Connections and Cursors
  - Exception handling in try/catch/finally blocks

PD/DBL Use Cases

- **Test Data Generation:**

  PD/DBL's powerful randomizing library supports functions such as:

  - pickone() - accepts an array of objects and a probability array that specifies the relative likelihood of being picked for each element. Allows picking based on existing or hypothetical probability distributions
  - randstr() - generates a random string based on a template specification

  Use these functions and others in the library to generate random names, transactions, places, products, etc.. Imagine having an entire data mart's worth of test data at your disposal without having to actually store ANY of it in the database.

- **Extract, Transform, Load (ETL) and Extract, Load, Transform ELT:**

  Use PD/DBL to easily and rapidly get data into and out of your database

  - Extract: PD/DBL supports JDBC compliant databases, allowing you to open multiple connections to external databases, declare cursors, and iterate over result sets
  - Transform: Manipulate ingested data using high-level language constructs like functions, if/then/else, case statements, etc.
  - Load: Return records the to the database using the "emitrow()" function. Use the resulting data in "CREATE TABLE AS…," "INSERT INTO…," "MERGE…," and "UPDATE…" data manipulation language (DML) statements

- **Powerful Ad-hoc Data Analysis:**

  PD/DBL does not require users to write, compile, upload and execute java code. All that is needed is a JDBC compliant database user interface. PD/DBL functions such as:

  - setsaverows() - directs a cursor (a wrapper around database retrieval and output functionality like a Sql-MR function's RowIterator or OutputEmitter or a JDBC ResultSet) to maintain a specified number of historical versions of records read from or written to a database
  - getlagrow() gets a saved historical version of a record read from or written to a database

  And features such as:

  - Record, Stack, Array, and Map variables - built-in object types that allow non-traditional analysis (iterate over an array, push/pop stacks, etc.)
  - High-level language syntax elements - if/then/else, case/when, branch-on, and the ability to define and call functions (even recursively) in a stored procedure or in a program directly entered at the user interface.

  All these features provide the Data Analyst and other users with unprecedented ad-hoc querying capability.

PD/DBL Example

Script Text

```
1   SELECT * FROM pddbl (
2
3   ON (SELECT * FROM persons) PARTITION BY ssn
4
5   SCRIPT('
6
7   #Trivial script - read in person data, proper-case names
8   #  and provide a concatenated "last, first" fullname output
9   script fullname {
10      function setup() {
11          addoutput("input0.*")
12          addoutput("fullname character varying")
13      }
14
15      function main() {
16          while(consume(input0)) {
17              output.lastname=strproper(input0.lastname)
18              output.firstname=strproper(input0.firstname)
19              output.ssn=input0.ssn
20              output.fullname=output.lastname + ", "
21              output.fullname=output.fullname + output.firstname
22              emitrow()
23          }
24      }
25  }'))
```

Script Output

lastname	firstname	ssn	fullname
Strickland	Mitchell	100000003	Strickland, Mitchell
Townsend	Lorenzo	100000004	Townsend, Lorenzo
Cain	Scott	100000006	Cain, Scott
Noel	Nick	100000007	Noel, Nick
Sharp	Willian	100000009	Sharp, Willian
Evans	Landon	100000010	Evans, Landon
Cherry	Waldo	100000012	Cherry, Waldo
Blanchard	Barney	100000013	Blanchard, Barney
Alford	Sergio	100000015	Alford, Sergio
Mcguire	Boyd	100000016	Mcguire, Boyd

Note: The above "person" records were randomly generated by a PD/DBL stored procedure using the pickone() function, which was provided with an array from files containing common last and first names. The files themselves were stored in the database directly and accessed using the fileopen(), readline() and filesplit() file-library functions.

For more information or to schedule a demonstration, please contact info@protodemos.com, or call us at 814-574-6777

http://www.protodemos.com

P.O. Box 535, State College, PA, 16804