Chapter 1 : Object-Oriented Programming in C# .NET - Part 5 - CodeProject

SQL Programming Part 18 - Dynamic Pivot Tables Posted by Andrew Gould on 03 December IF statements in SQL allow you to check if a condition has been met and, if so, to perform a sequence of actions.

PLINQ always uses hash partitioning. Hash partitioning is relatively inefficient in that it must precalculate the hashcode of every element so that elements with identical hashcodes can be processed on the same thread. If you find this too slow, your only option is to call AsSequential to disable parallelization. For all other query operators, you have a choice as to whether to use range or chunk partitioning. In a nutshell, range partitioning is faster with long sequences for which every element takes a similar amount of CPU time to process. Otherwise, chunk partitioning is usually faster. To force range partitioning: If the query starts with Enumerable. Range, replace the latter with ParallelEnumerable. Otherwise, simply call ToList or ToArray on the input sequence obviously, this incurs a performance cost in itself which you should take into account. Range is not simply a shortcut for calling Enumerable. It changes the performance of the query by activating range partitioning. To force chunk partitioning, wrap the input sequence in a call to Partitioner. The second argument to Partitioner. Create indicates that you want to load-balance the query, which is another way of saying that you want chunk partitioning. PLINQ starts by allocating very small chunks one or two elements at a time , then increases the chunk size as the query progresses: Range partitioning bypasses the normal input-side enumeration and preallocates an equal number of elements to each worker, avoiding contention on the input sequence. But if some threads happen to get easy elements and finish early, they sit idle while the remaining threads continue working. Our earlier prime number calculator might perform poorly with range partitioning. An example of when range partitioning would do well is in calculating the sum of the square roots of the first 10 million integers: For instance, if there are two workers, one worker might process odd-numbered elements while the other processes even-numbered elements. The TakeWhile operator is almost certain to trigger a striping strategy to avoid unnecessarily processing elements later in the sequence. The following demonstrates how Aggregate can do the work of Sum: The second argument is an expression to update the accumulated value, given a fresh element. You can optionally supply a third argument to project the final result value from the accumulated value. Most problems for which Aggregate has been designed can be solved as easily with a foreach loop â€” and with more familiar syntax. The advantage of Aggregate is precisely that large or complex aggregations can be parallelized declaratively with PLINQ. Unseeded aggregations You can omit the seed value when calling Aggregate, in which case the first element becomes the implicit seed, and aggregation proceeds from the second element. We can better illustrate the difference by multiplying instead of adding: However, there is a trap with unseeded aggregations: If used otherwise, the result is either unintuitive with ordinary queries or nondeterministic in the case that you parallelize the query with PLINQ. For example, consider the following function: First, we could include 0 as the first element: To illustrate, if we denote our aggregation function as follows: The first is to turn this into a seeded aggregation â€” with zero as the seed. The second solution is to restructure the query such that the aggregation function is commutative and associative: For instance, you can use Average to calculate a root-mean-square: Parallelizing Aggregate We just saw that for unseeded aggregations, the supplied delegate must be associative and commutative. PLINQ will give incorrect results if this rule is violated, because it draws multiple seeds from the input sequence in order to aggregate several partitions of the sequence simultaneously. Explicitly seeded aggregations might seem like a safe option with PLINQ, but unfortunately these ordinarily execute sequentially because of the reliance on a single seed. To mitigate this, PLINQ provides another overload of Aggregate that lets you specify multiple seeds â€” or rather, a seed factory function. For each thread, it executes this function to generate a separate seed, which becomes a thread-local accumulator into which it locally aggregates elements. You must also supply a function to indicate how to combine the local and main accumulators. Finally, this Aggregate overload somewhat gratuitously expects a delegate to perform any final transformation on the result you can achieve this as easily by running some function on the result yourself afterward. So, here are the four delegates, in the order they are passed: This tactic fails when the seed is a reference type that you wish to mutate, because the same instance will then be shared by each thread. To give a very simple example, the following sums the values in a numbers array: To give a more realistic example, suppose we wanted to calculate the frequency of each letter in the English alphabet in a given string. A simple sequential solution might look like this: To parallelize this, we could replace the foreach statement with a call to Parallel. And locking around accessing that array would all but kill the potential for parallelization. Aggregate offers a tidy solution. The accumulator, in this case, is an array just like the letterFrequencies array in our preceding example. Notice that the local accumulation function mutates the localFrequencies array. This ability to perform this optimization is important â€" and is legitimate because localFrequencies is local to each thread. PFX provides a basic form of structured parallelism via three static methods in the Parallel class: Invoke Executes an array of delegates in parallel Parallel. For Performs the parallel equivalent of a C for loop Parallel. ForEach Performs the parallel equivalent of a C foreach loop All three methods block until all work is complete. As with PLINQ , after an unhandled exception, remaining workers are stopped after their current iteration and the exception or exceptions are thrown back to the caller â€" wrapped in an AggregateException. Invoke executes an array of Action delegates in parallel, and then waits for them to complete. The simplest version of the method is defined as follows: Invoke to download two web pages at once: Invoke still works efficiently if you pass in an array of a million delegates. This is because it partitions large numbers of elements into batches which it assigns to a handful of underlying Tasks â€" rather than creating a separate Task for each delegate. This means you need to keep thread safety in mind. The following, for instance, is thread-unsafe: A better solution is to use a thread-safe collection such as ConcurrentBag would be ideal in this case. Invoke is also overloaded to accept a ParallelOptions object: Any already-executing delegates will, however, continue to completion. See Cancellation for an example of how to use cancellation tokens. ForEach perform the equivalent of a C for and foreach loop, but with each iteration executing in parallel instead of sequentially. Here are their simplest signatures: And the following sequential foreach: ToXmlString true ; As with Parallel. Invoke , we can feed Parallel. ToArray ; Outer versus inner loops Parallel. ForEach usually work best on outer rather than inner loops. Parallelizing both inner and outer loops is usually unnecessary. You must instead use the following version of ForEach: The following code loads up a dictionary along with an array of a million words to test: We can perform the spellcheck on our wordsToTest array using the indexed version of Parallel. So, to parallelize this: Write c ; do this: Aside from this difference, calling Break yields at least the same elements as executing the loop sequentially: In contrast, calling Stop instead of Break forces all threads to finish right after their current iteration. In our example, calling Stop could give us a subset of the letters H, e, l, l, and o if another thread was lagging behind. These tell you whether the loop ran to completion, and if not, at what cycle the loop was broken. If your loop body is long, you might want other threads to break partway through the method body in case of an early Break or Stop. You can do this by polling the ShouldExitCurrentIteration property at various places in your code; this property becomes true immediately after a Stop â€" or soon after a Break. ShouldExitCurrentIteration also becomes true after a cancellation request â€" or if an exception is thrown in the loop. IsExceptional lets you know whether an exception has occurred on another thread. Optimization with local values Parallel. ForEach each offer a set of overloads that feature a generic type argument called TLocal. These overloads are designed to help you optimize the collation of data with iteration-intensive loops. The simplest is this: Essentially, the problem is this: Calculating 10 million square roots is easily parallelizable, but summing their values is troublesome because we must lock around updating the total: Imagine a team of volunteers picking up a large volume of litter.

Chapter 2 : Big Data Processing with Apache Spark - Part 2: Spark SQL

A B-Tree structure can be visualized as an inverted tree with roots right at the top splitting into branches and then into leaves right at the bottom. In a B-Tree structure, there is a single root node at the top. This node then branches out into the next level, known as the first intermediate level. The nodes at the first intermediate level can branch out further. This branching can continue into multiple intermediate levels and then finally the leaf level. The nodes at the leaf level are known as the leaf nodes. This index page contains pointers that point to the index pages present in the first intermediate level. These index pages in turn point to the index pages present in the next intermediate level. There can be multiple intermediate levels in an index B-Tree. The leaf nodes of the index B-Tree have either data pages containing data rows or index pages containing index rows that point to data rows. Contains an index page with pointers pointing to index pages at the first intermediate level. Intermediate Nodes Contain index pages with pointers pointing either to index pages at the next intermediate level or to index or data pages at the leaf level. Contain either data pages or index pages that point to data pages. A data page containing index entries is called an index page. If data is stored in a sorted manner, the data is said to be present in a clustered structure. If it is stored at random, it is said to be present in a heap structure. Heap Structures In a heap structure, the data pages and records are not arranged in sorted order. IAM pages map extents that are used by an allocation unit in a part of a database file. You can read a heap by scanning the IAM pages to look for the extents that contain the pages for that heap. If an allocation unit contains extents from more than one file, there will be multiple IAM pages linked together in an IAM chain to map these extents. Partitioning of Heap Structures A table can be logically divided into smaller groups of rows. This division is referred to as partitioning. Tables are partitioned in order to carry out maintenance operations more efficiently. By default, a table has a single partition. When partitions are created in a table with a heap structure, each partition will contain data in an individual heap structure. For example, if a heap has three partitions, then there are three heap structures present, one in each partition. Clustered Indexes A clustered index causes records to be physically stored in a sorted or sequential order. Thus, a clustered index determines the actual order in which data is stored in the database. Hence, you can create only one clustered index in a table. Uniqueness of a value in a clustered index is maintained explicitly using the UNIQUE keyword or implicitly using an internal unique identifier. Before you create a clustered index, you need to make sure the free space in your system is at least 1. The root node of the B-tree contains an index page with pointers that point to the index pages at the first intermediate level. These index pages in turn have pointers that point to the index pages at the next intermediate level. The index pages at the last intermediate level have pointers that point to actual data rows in the data pages present at the leaf level of the index. The size of a clustered index is about five percent of the table size. However, it varies depending on the size of the indexed column. Accessing Data with a Clustered Index A clustered index can be created on a table using a column without duplicate values. This index reorganizes the records in the sequential order of the values in the index column. Clustered indexes are used to locate a single row or a range of rows. Starting from the first page of the index, the search value is checked against each key value on the page. When the matching key value is found, the database engine moves to the page indicated by that value. The desired row or range of rows is then accessed. Clustered indexes are useful for columns that are searched frequently for key values or are accessed in sorted order. Guidelines A clustered index is automatically created on a table when a primary key is defined on the table. In a table without a primary key column, a clustered index should ideally be defined on: Key columns that are searched on extensively. Columns used in queries that return large result sets. Columns having unique data. Columns used in table joins. Two or more tables can be logically joined through columns that are common to the tables. Data can then be retrieved from these tables as if they were a single table. Nonclustered Indexes A nonclustered index is defined on a table that has data either in a clustered structure or a heap. Nonclustered index will be the default type if an index is not defined on a table. Each index row in the nonclustered index contains a nonclustered key value and a row locator. This row locator points to the data row corresponding to the key value in the table. Nonclustered indexes have a similar B-tree structure as clustered indexes but with the following differences: The data rows of the table are not physically stored in the order defined by their nonclustered keys. In a nonclustered index structure, the leaf level contains index rows. Guidelines Nonclustered indexes are useful when you require multiple ways to search data. There are some facts and guidelines to be considered before creating a nonclustered index. A table can have up to nonclustered indexes. Create clustered index before creating a nonclustered index. Differences Clustered and nonclustered indexes are different in terms of their architecture and their usefulness in query executions. The table highlights the differences between clustered and nonclustered indexes: This data type is used to store XML documents and fragments. Due to the large size of XML columns, queries that search within these columns can be slow. You can speed up these queries by creating an XML index on each column. An XML index can be a clustered or a nonclustered index. The size of all the data stored in an xml type column cannot exceed 2 Gigabytes GB. This primary key cannot exceed 15 columns. There are two types of XML indexes: It is a special index that shreds the XML data to store information. Searching for values anywhere in the XML document. Retrieving particular object properties from within an XML document.

Chapter 3 : SQL Standards - JCC Consulting

This series has covered many of the SQL statements that are used in today's programming environment. In addition to covering the Select, Insert, and Delete statements, we discussed how to code different SQL join statements.

So is there any possible benefit to this new feature? Make Software Delivery Easier? Someone with sysadmin rights would execute that script. Then, a non-sysadmin can install the Assemblies whenever they are able to. During installation, the script can verify the existence of the hashes and abort cleanly if they are not present. The only problem is that it has been possible to do this, and in a cleaner, simpler, and more secure way, starting with SQL Server Certificates are at least just as easy if not easier than registering a Trusted Assembly: This is provided by Microsoft. AND, permissions can be granted per each Certificate, so Logins only see what they are allowed to see i. Even better, a Login can automatically see Certificates that they create, so no additional permissions needed for verification i. Creating the Certificate and the install script for it is a one-time operation. This means a one Certificate and one Login is all that ever get creatd, and b no additional sysadmin work for software updates, and no growing pile of obsolete meta-data i. Logins can only drop Certificates that either they created or were granted permission to i. And then, they can see all Trusted hashes, no fine-grained access i. Calculating and registering the hashes to trust needs to be done for every compilation of the DLL s. This means that software updates require a sysadmin every time, and might be adding to the trusted hash list every time if not being cleaned up i. Unregistering an obsolete hash needs to be done by someone with sysadmin rights, who can then unregister any number of or all trusted hashes. Hence, if anyone wanted to use a non-sysadmin account for installation, Certificates make that rather easy. It is only 64 bytes and contains no meta-data to indicate origin, etc. On the other hand, a certificate is approximately â€” bytes and does contain additional information to help determine authenticity. Also, the certificate value is calculated from both the item being signed and the private key. Well, this just happens to be the one area where there might possibly be, in the worst-case scenario, some argument made for keeping this feature. That is definitely a nail-in-the-coffin for the Certificate idea. As detailed in the previous post i. Below are a few other options that should be considered before proceeding to use an inferior simple hash. This requires no additional documentation or training, and simplifies deployment scripts by not complicating them. Scan for matching Certificate in [master], without requiring the Login: Now, assuming that Certificates can be created in [master], then check to see if any of those Certificates matches the Certificate of any Assembly being loaded. But there is no valid reason for hurting the main SQL Server product with this proven anti-pattern. So, please support i.

Chapter 4 : SQLCLR vs. SQL Server , Part 5: â€œTrusted Assembliesâ€• â€“ Valid Use Cases? â€“ Sql Q

The CASE function is a very useful T-SQL function. With this function you can replace a column value with a different value based on the original column value. An example of where this function might come in handy is where you have a table that contains a column named SexCode, where 0 stands for.

I am curious about the computation aspect of Hadoop and want to know what it is all about, how it works, and any other relevant information. Solution In this tip we will take a look at the 2nd core component of Hadoop framework called MapReduce. Key Concepts Here are some of the key concepts related to MapReduce. These tasks can be run independent of each other on various nodes across the cluster. There is only one JobTracker node per Hadoop Cluster. There is no restriction on the number of TaskTracker nodes that can exist in a Hadoop Cluster. This part of the MapReduce is responsible for processing one or more chunks of data and producing the output results. Data Locality MapReduce tries to place the data and the compute as close as possible. First, it tries to put the compute on the same node where data resides, if that cannot be done due to reasons like compute on that node is down, compute on that node is performing some other computation, etc. This feature of MapReduce is "Data Locality". Let us understand each of the stages depicted in the above diagram. Hadoop splits the incoming data into smaller pieces called "splits". In this step, MapReduce processes each split according to the logic defined in map function. Each mapper works on each split at a time. Each mapper is treated as a task and multiple tasks are executed across different TaskTrackers and coordinated by the JobTracker. This is an optional step and is used to improve the performance by reducing the amount of data transferred across the network. Combiner is the same as the reduce step and is used for aggregating the output of the map function before it is passed to the subsequent steps. In this step, outputs from all the mappers is shuffled, sorted to put them in order, and grouped before sending them to the next step. This step is used to aggregate the outputs of mappers using the reduce function. Output of reducer is sent to the next and final step. Each reducer is treated as a task and multiple tasks are executed across different TaskTrackers and coordinated by the JobTracker. Finally the output of reduce step is written to a file in HDFS. Let us assume that we have a file which contains the following four lines of text. Let us see how this counting operation is performed when this file is input to MapReduce. Below is a simplified representation of the data flow for Word Count Example. In this step, the sample file is input to MapReduce. Note that, for the purpose of this example, we are considering one line as each split. However, this is not necessarily true in a real-time scenario. In this step, each split is fed to a mapper which is the map function containing the logic on how to process the input data, which in our case is the line of text present in the split. This is an optional step and is often used to improve the performance by reducing the amount of data transferred across the network. This is essentially the same as the reducer reduce function and acts on output from each mapper. In this step, output of all the mappers is collected, shuffled, and sorted and arranged to be sent to reducer. In this step, the output of the reducer is written to a file on HDFS. The following image is the output of our word count example. JobTracker acts as the master and TaskTrackers act as the slaves. MapReduce has two major phases - A Map phase and a Reduce phase. Map phase processes parts of input data using mappers based on the logic defined in the map function. The Reduce phase aggregates the data using a reducer based on the logic defined in the reduce function. MapReduce has built-in fault tolerance and hence can run on commodity hardware. MapReduce takes care of distributing the data across various nodes, assigning the tasks to each of the nodes, getting the results back from each node, re-running the task in case of any node failures, consolidation of results, etc. MapReduce processes the data in the form of Key, Value pairs. Hence, we need to fit out business problem in this Key-Value arrangement. References For latest and up to date information, visit http:

DOWNLOAD PDF PART 5 : PROGRAMMING WITH SQL.

Chapter 5 : English Query - Part 5 | SQL Server Reference Guide | InformIT

SQL Server Graph Databases - Part 4: Working with hierarchical data in a graph database SQL Server Graph Databases - Part 5: Importing Relational Data into a Graph Database With the release of SQL Server , Microsoft introduced graph database features to support data sets that contain complex relationships between entities.

Stored Procedures, Programming Python by Przemyslaw Piotrowski Calling database stored procedures and other interesting aspects of advanced Python programming. Published March See series TOC There are two mainstream approaches when it comes to software development engaging the database: Bringing on pros and cons for both of these solutions is not the subject of this tutorial; nevertheless, going the Oracle Database path brings certain advantages to a database-oriented application. At the end we are going to touch on Oracle Berkeley DB, which comes built-in into Python out-of-the-box. Arguments of procedures also referring to functions from now on can be one of the three types: For illustrating the interaction between Python and Oracle procedures consider the package below to be installed in the HR schema of Oracle Database XE instance. Each one requires a different way of calling as shown below. As an example of using arrayvar objects make sure the DDL below finds its way into the database. Yet some programming aspects just cannot be expressed or used with its in-database nature. Multiprocessing Parallel processing in Python, starting with version 2. The threading module shipping with the standard library is limited to running one operation at a time. By substituting threads with operating system processes, it is now possible to leverage all the CPUs available to the application and therefore perform truly parallel computations. See below for an example of a simple database benchmark utility. It comprises a number of traversing functions that yield custom, optimized iterators. One can spot the difference in using iterators vs lists or tuples by looping through large data sets, because they basically avoid rendering the whole collection in memory. Looping through results of itertools closely resembles the one you use for lists, tuples and dictionaries. Serializing Data In Python, data serialization and de-serialization is handled by the pickle module and its C counterpart cPickle up to x faster than the native Python implementation of pickle. There are only few limits when it comes to types supported by pickle, since it can handle everything from dictionaries and tuples, through sets and functions, up to classes and instances. Connection objects, for obvious reasons. Sleepy Cache Oracle Berkeley DB is a transactional key-value storage solution with fine-grained locking, high availability, and replication. It fits in all those problems where extreme efficiency is required and the overhead of a full size relational database is too high. Up to version 2. New versions of Python, starting with 3. Next, we are going to make use of the built-in driver that comes with Python 2. When it comes to Python, you should now be familiar with the essentials of the multiprocessing module which is one of the most important recent additions to the language. Finally, Oracle Berkeley DB was used in a proof-of-concept in-memory cache scenario. Przemyslaw Piotrowski is an information technology specialist working with emerging technologies and dynamic, agile development environments. Having a strong IT background that includes administration, development and design, he finds many paths of software interoperability.

Chapter 6 : Functions in C Programming - Part 5 - The Crazy Programmer

In the previous post in this series on SQLCLR in SQL Server â€” Part 4: "Trusted Assemblies" â€" The Disappointment â€" we looked at what the "Trusted Assemblies" feature is, what it meant to do, the problems with it, and what the better and more appropriate approach is.

Virtual and Override members methods, properties are another kind of polymorphism. When you derive from a base class, that base class may have some virtual members and allow you to override them. That is, change their implementation so that when called by an object of your class does a different thing. The best way of understanding it is through an example. Suppose you derive from a base class named Car. This Car class possesses a method named Accelerate: Now, you want to create a much faster Car class that when its Accelerate method is called, its speed increases by And, of course, this new car derives from the base car class: The solution to this warning is that the Accelerate method in Car class should be declared as virtual which allows you to override it within the MuchFasterClass: It also happens on properties: What are Overriding Operators? When used on two integers, it simply works like an Add method in a Math class. WriteLine a ; Console. WriteLine s ; Console. Suppose you have a Point class which has an X and Y coordinates. Now, look at the following code: Interfaces are like contracts. If your class is to have a certain capability, it must override a certain interface. Another use of interfaces is providing multiple inheritance, which is impossible in C using class derivation. That means that you can derive from only and only one class but you can implement as many interfaces as you desire. Suppose you want to make an Animal hierarchy. You create a base abstract class named Animal that every class in the hierarchy derives from. The reason to create the Animal class as abstract is that it should not be instantiated an abstract class cannot be instantiated. Now some animals are carnivore, some are herbivore, some are reptiles. You cannot make a class, say, Crocodile derive from Animal, Carnivore, Reptile. Instead, you declare everything except the base Animal class as interfaces. Therefore, the Crocodile class is an Animal which implements the Carnivore, Reptile interfaces which are also like contracts. If the Crocodile is to be thought as a carnivore and a reptile, it must implement a certain, in this case Carnivore and Reptile, interfaces. For example, the code of the Lion class would be as follows: For instance, if you want your class to be sortable if you have an array containing objects of your class and you want to sort that array , you must add IComparable interface to your class and implement its CompareTo method. One interesting thing about interfaces is that you can assign an object of a particular type to an interface provided that class implements that interface as you can do the same with an abstract class: You can declare a class within another class. The inner class in called the nested class and the outer class is called the nesting class. This is done when the nested class is meaningless out of the scope of the nesting class. In UML, this is called composition relationship. Suppose we want to declare a class named Human. Now, Heart is the other class we want to create. Therefore, we create the Heart class within our Human class: Heart human ; Console. Thank you for spending time studying this series of articles. Hope you have learnt helpful and interesting things. History 5th July,

Chapter 7 : TSQL Interview Questions â€" Part 5 | SQL with Manoj

theinnatdunvilla.com - If you ever find yourself writing the same SQL query over and over again, a Stored Procedure could be just the time-saving tool you're looking for. This video.

Chapter 8 : Threading in C# - Part 5 - Parallel Programming

Here you can learn C, C++, Java, Python, Android Development, PHP, SQL, JavaScript,.Net, etc. Functions in C Programming - Part 5 - The Crazy Programmer I hope till now you must be having a good grip on the basic concepts of functions.

DOWNLOAD PDF PART 5 : PROGRAMMING WITH SQL.

Chapter 9 : Free Programming Books - Part 5 : PDF Download

Oracle for Absolute Beginners: Part 5 - PL/SQL A wise man once said, all software programming is about hoops and loops. You take some variables, give them hoops to jump through and, depending on their success or failure, you give them some actions to loop through a number of times.*