# Partitioned Query
for MySQL

a managed .NET component

Developer's Guide

Version 1.0

Inspireon
Software

www.InspireonSoftware.com

Table of Contents

Introduction........1

Technical Support........1

Building a Shared-Nothing MySQL Cluster........2

Using the Partitioned Query for MySQL Component........4

Important Notes Regarding MySQL Queries........10

Known Issues with Supported Data Providers........12

Working with Popular .NET Reporting Components and Tools........13

License Agreement........Appendix A

Introduction

Partitioned Query for MySQL is a managed .NET component that brings the high-performance and scalability of distributed partitioned queries to the MySQL database server. With this powerful, innovative component, .NET developers can bring a new level of performance and scalability to their data warehousing, reporting, business analytics, and business intelligence applications at significantly reduced costs when compared to developing similar applications using the Enterprise versions of the Oracle, IBM DB2, or Microsoft SQL Server databases.

Utilizing a shared-nothing approach to database clustering, developers can easily build a cluster of MySQL databases to dramatically improve the performance and response time of user queries, and can easily and affordably scale-out their applications as data and user volumes grow.

The combination of the robust .NET development environment, the powerful, fast, low-cost MySQL database server, and the Partitioned Query for MySQL component brings a new level of price-performance to the data warehousing arena.

This Developer's Guide will take you through the basics of how to build a shared-nothing cluster of MySQL database servers to house your partitioned data, and how to utilize the Partitioned Query for MySQL component in your .NET data warehousing/reporting applications.

Important: *Please read the section entitled "Important Notes Regarding MySQL Queries" on page 10 before you use the Partitioned Query for MySQL component in your projects, as it contains important information on some of the restrictions on SQL commands that can be used in a partitioned query.*

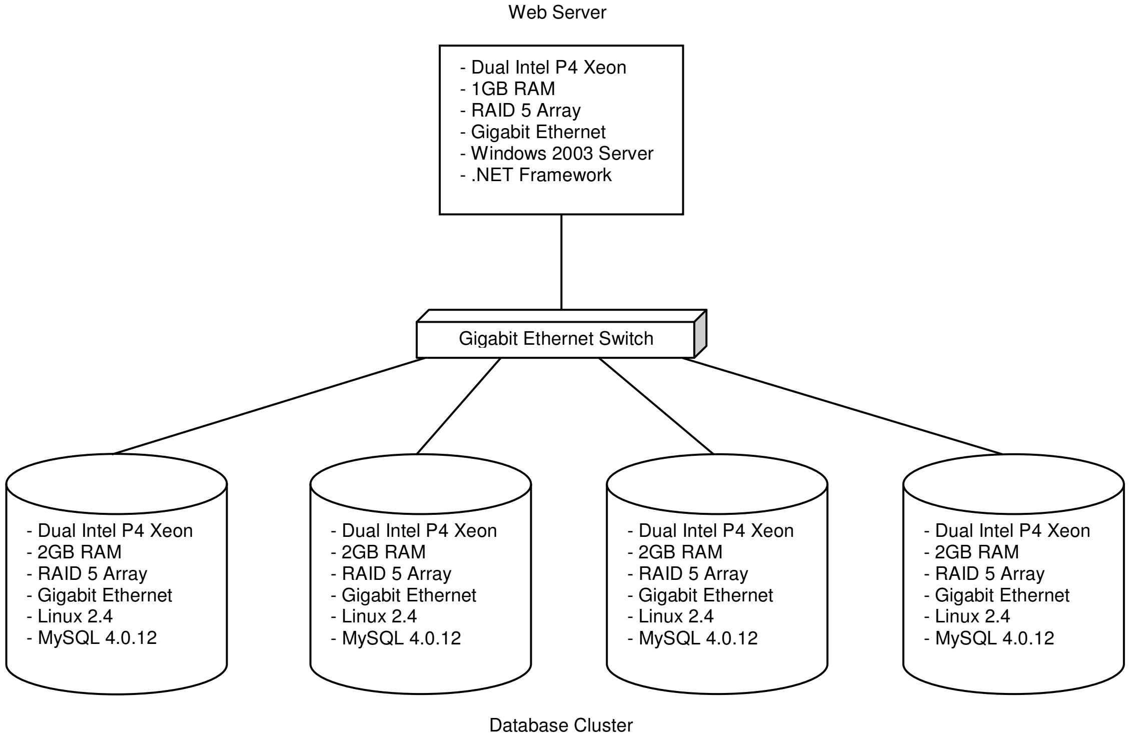
Technical Support

Technical support for the Partitioned Query for MySQL component is available through our online Knowledge Base at **www.InspireonSoftware.com** - where you can find answers to common questions and problems, and also submit technical support requests. You can also send a technical support request via e-mail to **Support@InspireonSoftware.com**.

Building a Shared-Nothing MySQL Cluster

A shared-nothing MySQL database cluster is simply a collection of independent servers running the MySQL database server, and connected by a high-speed switched Ethernet network. Each node of the cluster runs the MySQL database server as a standalone application, and the MySQL database server on each node is configured with the same version of MySQL, and the same user logins and security. For best performance, the nodes of the cluster are connected to the same Ethernet switch, and Gigabit Ethernet is recommended to minimize the time required to move data from one node to another. The Partitioned Query for MySQL component is optimized to minimize the amount of data transferred between nodes for summary-type queries, so that 100-Megabit Ethernet switches can be used and still deliver good performance if Gigabit Ethernet is not a viable option (depending on your data model and the type of query).

An example of a shared-nothing MySQL database cluster would look like the following:



Because the MySQL database server is available for a variety of operating systems, it is possible to configure a cluster using different operating systems on each node. However, for ease of manageability, it is usually recommended that each node of the database cluster run the same operating system – but it is not a requirement. The only requirement is that each node be running the same version of MySQL, and be configured with identical security and user logins. One of the most cost-effective hardware and operating system combinations for running MySQL is to utilize industry-standard Intel or AMD-based servers running Linux. The Partitioned Query for MySQL component currently supports the MySQL database server versions 3.23.51 through 3.23.57, and 4.0.12 through 4.0.13. *For the most up-to-date MySQL version compatibility information, please visit our website at www.InspireonSoftware.com.*

Once you have configured the hardware, operating systems, and MySQL database servers, and connected the servers to the Ethernet switch, you then need to partition your database across the nodes of the cluster so that the amount of data on each node is roughly the same. The data can be partitioned by date, geography, company organization, or any other method that makes logical sense based on your data model. When partitioning your database, you need to make sure that all related/child table(s) are partitioned along the same boundaries that you select to partition your main table(s), so that all related data exists on the same server where the partitioned main table(s) are going to be located. If you use static lookup tables where all values are relevant to any query regardless of partition boundary, then those tables should be copied or replicated to each partitioned database. The use of the SELECT ... INTO OUTFILE and LOAD DATA INFILE statements are very useful for partitioning an existing database, as they can quickly and efficiently extract the data from the existing database, and load it into the new partition.

The partitioning process will be different for each data warehousing scenario depending on your specific data model, but once you have invested the time to develop a partitioning strategy that effectively balances the data load across your cluster servers, the payoff in improved performance and scalability will be worth it.

In most data warehousing scenarios, one node would be designated to house the most recent data, and that node is where “new” data is appended on a periodic (usually daily or weekly) basis using your data transformation/load process. The remaining nodes would house historical “static” data that does not change once it has been loaded. This partitioning approach saves valuable time in loading new data, as the database indexes only need to be updated on the partition containing the most recent data, not on the entire database on all partitions – significantly reducing your data load window. This approach also saves time in backing up your database, as only the most recent partition needs to be regularly backed up. The historical partitions only need to be backed up one time when they are first loaded. As your data and user volumes grow, you can add additional servers to your cluster to improve the performance of your application.

Once the database is partitioned across the nodes of the cluster, the Partitioned Query for MySQL component is used to execute queries against all cluster nodes in parallel and return the aggregated results as either a DataReader or DataSet object. Because only a portion of the database exists on each node, the total time required to run most queries is significantly reduced when compared to running the same query against a single, large database on a single server. The Partitioned Query for MySQL component takes care of all of the parallel processing details for you. All you need to do is pass to the component the SQL statement that contains the query you need to execute, and the results are automatically returned in either a DataReader or a DataSet object. You can then use the returned object to format the resulting data for a web page, to pass the object to a reporting tool like ActiveReports for .NET or Crystal Reports for .NET, or to pass the object to another .NET component for further processing and/or display purposes.

If you want to improve the reliability of a shared-nothing cluster, you can setup one or more stand-by servers configured identically to your cluster nodes that you can bring online in case one of your cluster nodes should fail for some reason. For maximum failover capabilities, you could create a stand-by node for each cluster node, with each stand-by node having an identical copy of the corresponding cluster node’s database.

Using the Partitioned Query for MySQL Component

The Partitioned Query for MySQL component is a 100% managed-code component for the .NET framework. It is designed to be fast, efficient, and easy to use, so that you can quickly and easily start using it in your projects.

Installing the Component

To install the component on your .NET development workstation, simply place the **MyPQuery.dll** assembly in the **bin** folder of your application, and add a reference to the assembly to your project. If you are using the Evaluation version of the component that is available as a free download, you are allowed to use this fully-functional Evaluation version of the component for up to 30 days. After the 30-day Evaluation period, you must either purchase a License to continue using the product, or remove it from your computer(s). To purchase a License for the product, please visit our website at **www.InspireonSoftware.com** (the product is licensed on a per-developer basis).

To install the Partitioned Query for MySQL component on your web or application server, simply copy the **MyPQuery.dll** assembly to the **bin** folder where you have installed your application on the server.

Please see Appendix A for a copy of the License Agreement.

Developing with the Partitioned Query for MySQL Component

Developing applications with the Partitioned Query for MySQL component is straight-forward and easy.

Once you have installed/configured your MySQL database cluster and partitioned your database, you need to add a table to the database on each cluster node that configures the nodes of your cluster for the Partitioned Query for MySQL component. This table must be named **partitionedqueryconfig** (all lowercase), and must contain the following columns:

Column Name	Column Type
server	varchar(64)
database	varchar(64)

A MySQL script to create this table (named partitionedqueryconfig.sql) can be found in the ZIP file you downloaded for this component. You can use this script to create the table on each server node of your cluster.

Once you have created the **partitionedqueryconfig** table, you need to add a row to the table for each server node in your MySQL database cluster. For example, your table might look like the following (see next page):

server	database
mysql001	mydatabase
mysql002	mydatabase
mysql003	mydatabase
mysql004	mydatabase

The entries in this table should be the same on each server.

Note: see page 9 in this manual for information on temporary tables used by the Partitioned Query for MySQL component, and an additional configuration step required to support these temporary tables.

Now that you have created and populated this configuration table on each cluster node, you can begin using the component in your project by adding a reference to the component to the project in Visual Studio .NET.

Once you have added a reference to the component to your project, you need to instantiate an instance of the **MyPQuery.Query** class as follows:

VB.NET

```
Dim oMyPQuery As New MyPQuery.Query()
```

C#.NET

```
MyPQuery.Query oMyPQuery = new MyPQuery.Query();
```

The Partitioned Query for MySQL component has just one method named **RunQuery**. The RunQuery method is used to execute the SQL SELECT statement for you query, and return the result set as either a DataReader or a DataSet object. The RunQuery method has six parameters, and returns an empty string to indicate a successful query execution, or a string containing the error code and error message if an error occurs.

RunQuery (Provider, Connection, Command, ResultSet, ResultSetType, SQLStatement)

The parameters to the **RunQuery** method are as follows:

Parameter	Description	Passed By
Provider	The type of data provider being used to connect to the MySQL database server (see next page)	Value
Connection	The Connection object for the MySQL database	Reference
Command	The Command object	Reference
ResultSet	The DataReader or DataSet object used to return the results of the query	Reference
ResultSetType	The type of object passed as ResultSet – either ResultSetType.DataReader or ResultSetType.DataSet	Value
SQLStatement	The SQL statement for the query to execute	Value

The Partitioned Query for MySQL component currently supports the following data providers to access the MySQL database server:

- MyODBC v3.51 with ODBC.NET (using Microsoft.Data.Odbc.dll)
- dbProvider v1.6 by eInfoDesigns (dbProvider.dll) *www.eInfoDesigns.com*
- MySQLDirect.NET v1.60 by Core Lab (CoreLab.MySql.dll) *www.crlab.com*

*The assembly file for the data provider you use must be located in your project's **bin** folder in order to work correctly with the MyPQuery.dll assembly. The use of any other data provider will result in an error being returned from the RunQuery method.*

The type of data provider you are using to access MySQL is passed as the **Provider** parameter of the **RunQuery** method (see above). Valid values for the Provider parameter are:

```
ProviderType.ODBCNET
ProviderType.dbProvider
ProviderType.MySQLDirect
```

Before you can call the RunQuery method of the MyPQuery.Query class, you need to instantiate your Connection and Command objects for the provider you will be using.

For example, if you are using the MyODBC 3.51 driver with ODBC.NET, you would instantiate your objects as follows:

VB.NET

```
Dim oConnection As New OdbcConnection()
Dim oCommand As New OdbcCommand()
```

C#.NET

```
OdbcConnection oConnection = new OdbcConnection();
OdbcCommand oCommand = new OdbcCommand();
```

To instantiate the Connection and Command objects for a different supported data provider, refer to the documentation provided with that data provider.

If you want to have the query results returned in a DataReader object, then you also need to declare a variable for your DataReader object as follows:

VB.NET

```
Dim oDataReader As OdbcDataReader
```

C#.NET

```
OdbcDataReader oDataReader;
```

To declare a DataReader object for another data provider, refer to the documentation provided with that provider.

If you want to have the query results returned in a DataSet object, then you need to instantiate a new instance of a .NET DataSet object as follows:

VB.NET

```
Dim oDataSet As New DataSet()
```

C#.NET

```
DataSet oDataSet = new DataSet();
```

Once you have declared these objects, all you need to do before calling the **RunQuery** method is to set the ConnectionString property of your Connection object, and set the Connection property of your Command object.

For example:

VB.NET

```
oConnection.ConnectionString = "Driver={MySQL ODBC 3.51 Driver};" & _
             "Server=mysql001;Database=MyDatabase;UID=uuuu;PWD=pppp"

oCommand.Connection = oConnection
```

C#.NET

```
oConnection.ConnectionString = "Driver={MySQL ODBC 3.51 Driver};" +
             "Server=mysql001;Database=MyDatabase;UID=uuuu;PWD=pppp";

oCommand.Connection = oConnection;
```

The Server that you use in your ConnectionString can be any one of the server nodes in your MySQL database cluster. You can choose to always connect to the same server, or you can choose to randomly connect to any one of the nodes of your cluster to distribute the connection load among all nodes of your cluster to improve the overall performance of your application.

Important Notes:

The User ID and Password that you use in your ConnectionString must also exist in the security configuration on each MySQL database node in your cluster. The Partitioned Query for MySQL component will use this same User ID and Password to connect to each node in the cluster. For the ODBC.NET driver, you cannot use a DSN to connect to the database. You must use a DSN-less connection, and provide the Server, Database, UID, and PWD parameters in the connection string.

Once you have set the ConnectionString property of your Connection object, do not call the Open() method of the Connection object. In order for the Partitioned Query for MySQL component to be able to connect to all nodes of your MySQL database cluster, it needs to be able to read the ConnectionString property and call the Connection.Open() method. If you call the Open() method of your Connection before you call the MyPQuery.RunQuery method, an error will be returned.

Now that you have set the ConnectionString property of your Connection object, you can call the **RunQuery** method of the MyPQuery.Query class as follows:

VB.NET

```
Dim sSQLStatement As String
Dim sReturn As String

sSQLStatement = “SELECT Name,City,SUM(SalesAmount) “ & _
                  “FROM Sales “ & _
                  “GROUP BY Name,City “ & _
                  “ORDER BY Name,City “

sReturn = oMyPQuery.RunQuery(ProviderType.ODBCNET,
                              oConnection,
                              oCommand,
                              oDataReader,
                              ResultSetType.DataReader,
                              sSQLStatement)
```

C#.NET

```
String sSQLStatement;
String sReturn;

sSQLStatement = “SELECT Name,City,SUM(SalesAmount) “ +
                  “FROM Sales “ +
                  “GROUP BY Name,City “ +
                  “ORDER BY Name,City “;

sReturn = oMyPQuery.RunQuery(ProviderType.ODBCNET,
                              ref oConnection,
                              ref oCommand,
                              ref oDataReader,
                              ResultSetType.DataReader,
                              sSQLStatement);
```

When the query has completed, you should check the return string to see if any errors occurred in running the query. If no errors occurred, the return string will be an empty string (“”). If an error occurred, the error number and error text will be returned in the return string, and the DataReader or DataSet objects will have no data.

If no errors were returned, you can then use the results in the DataReader or DataSet to format the data in a web page, or to pass the results to a reporting component such as ActiveReports for .NET or Crystal Reports for .NET, or to another .NET component to format/display the resulting data.

If you use a **DataReader** object for your results, you will need to call the Close() method of the DataReader object to close the DataReader when you have finished with it, and you will also need to call the Close() method of your Connection object when you have finished with it (the

Partitioned Query for MySQL component cannot close these objects, because they must remain open in order for you to be able to access the rows of the DataReader).

If you use a **DataSet** object for your results, the Connection object is automatically closed when the query has completed. There is no need for you to call the Close() method of your Connection object (since a DataSet is a disconnected recordset type of object).

Temporary Tables

The Partitioned Query for MySQL component uses a temporary table when building the aggregate results of a partitioned query. The component will randomly select a node of your cluster to store this temporary table, so as to improve the overall performance of multiple users running your application.

These temporary tables are created in a special database named **partitionedqueryresults** (all lowercase) on each MySQL server node in your cluster. When you first configure the MySQL database server on each node of your cluster, you need to create this special database named **partitionedqueryresults**, and grant privileges to the User ID used in your Connection string to grant the user CREATE, DROP, INSERT, and SELECT privileges to this database. For additional performance, you can choose to locate this database on a different disk or volume than your main database (please see to the MySQL documentation for instructions on how to create or locate a database on a particular disk or volume).

If you use a **DataSet** object for your results, the temporary table is automatically removed after the RunQuery method returns. However, if you use a **DataReader** object for your results, this temporary table is not automatically removed, as it remains in use until you explicitly close your DataReader object. Because of this, periodic maintenance is required to clean up any temporary tables left behind by DataReader objects. If you use DataReader objects to return the results of your queries, you should periodically delete any tables in the **partitionedqueryresults** database on each node of your cluster to reclaim the disk space used by these temporary tables. Alternatively, you could periodically drop and re-create the database to quickly delete all existing tables (make sure to only perform this operation when no users are accessing your application).

Important Notes Regarding MySQL Queries

Due to the complex nature of running queries in parallel across a partitioned database, the Partitioned Query for MySQL component has a few restrictions on the SQL commands that can be used in a distributed partitioned query as follows:

- Only SELECT statements are supported, as the component is designed for running queries only. Any other data definition or data manipulation statements are not supported (you should use your own code/components or other 3rd-party utilities to add, update, and delete data from the databases).
- All columns in the SELECT statement must be explicitly defined. You cannot use the wildcard * character to specify all columns from a table, as in:

  ```
  SELECT * FROM Employees
  ```

  You must specify the specific columns you wish to retrieve, as in:

  ```
  SELECT EmployeeName,SSNum,HireDate FROM Employees
  ```

  *It is worth noting that it is good practice anyway to specify the specific column names required by your query, rather than using the wildcard * character - as it can improve the performance of the query by avoiding a lookup in the data dictionary, and by minimizing the return of unused/unneeded columns.*

- All computed columns (i.e. columns that use formulas, functions, or other calculations) must be aliased. For example, you must use the following:

  ```
  SELECT IF(HireDate<'2003-01-01','Old Plan','New Plan') AS PlanName
  FROM Employees
  ```

  instead of:

  ```
  SELECT IF(HireDate<'2003-01-01','Old Plan','New Plan')
  FROM Employees
  ```

  It is good practice anyway to alias all computed columns so that you can refer to them by their aliased column names in your code and reports.

- Only the COUNT(*), SUM(expr), MIN(expr), and MAX(expr) aggregate functions are supported. The COUNT(DISTINCT expr), AVG(expr), STD(expr), STD_DEV(expr), BIT_OR(expr), and BIT_AND(expr) aggregate functions are not supported at this time.
- The UNION and UNION ALL statements introduced in MySQL version 4.0.x are not supported, as only a single SELECT statement can be issued in the query at this time.
- Embedded comments in a SELECT statement are not supported at this time.

In addition, running MySQL in ANSI mode (by starting mysqld with the `--ansi` option) is not supported at this time

Important Notes Regarding MySQL Queries (continued)

Note that you can get around the AVG(expr) limitation by including both the COUNT(*) and SUM(expr) functions in your query, and then perform the calculation of the average in your application code as in the following example:

SQL Statement

```
SELECT Region, SUM(SalesAmount) AS SumOfSales, COUNT(*) AS CountAll
FROM Sales
GROUP BY Region
ORDER BY Region
```

VB.NET

```
Dim Avg As Double
Avg = MyDataReader(“SumOfSales”) / MyDataReader(“CountAll”)
```

C#.NET

```
double Avg;
Avg = MyDataReader(“SumOfSales”) / MyDataReader(“CountAll”);
```

It is also important to note that not all queries will run faster as distributed partitioned queries across a cluster of databases. Some quick queries that can access a single or a few rows via an index will actually take slightly longer to run as a distributed partitioned query – due to the overhead involved in running the query in parallel across all of the cluster nodes. But on average, many queries can run much faster as distributed partitioned queries, and you should see significant overall improvement in query response times (depending on how many servers you have in your cluster, how you have partitioned your database, and the nature of the queries involved).

The Partitioned Query for MySQL component currently supports the MySQL database server versions 3.23.51 through 3.23.57, and 4.0.12 through 4.0.13. *For the most up-to-date MySQL version compatibility information, please visit our website at www.InspireonSoftware.com.*

Note: although we have attempted to accommodate all other combinations of MySQL keywords, functions, literals, etc. that can occur in a SELECT statement, it is possible that some unique combinations may not work correctly when executed as a partitioned query using the Partitioned Query for MySQL component. In the case that you are unable to get a particular SELECT statement to work correctly, or if the returned aggregated results are not accurate, please send us an e-mail at Support@InspireonSoftware.com and we will attempt to help resolve the problem as quickly as possible.

Known Issues with Supported Data Providers

The Partitioned Query for MySQL component currently supports the following data providers to access the MySQL database server:

- MyODBC v3.51 with ODBC.NET (using Microsoft.Data.Odbc.dll)
- dbProvider v1.6 by eInfoDesigns (dbProvider.dll) *www.eInfoDesigns.com*
- MySQLDirect.NET v1.60 by CoreLab (CoreLab.MySql.dll) *www.crlab.com*

The assembly file for the data provider you use must be located in your project's ***bin*** *folder in order to work correctly with the MyPQuery.dll assembly. The use of any other data provider will result in an error being returned from the RunQuery method.*

The following is a list of known issues with the supported data providers:

MyODBC v3.51 with ODBC.NET

With the ODBC.NET provider, there is a known problem with the provider not being able to handle the MySQL Decimal data type correctly. This known problem may raise an error if one of the columns in your query is a MySQL Decimal data type (for further information on this problem, please see the Microsoft Knowledge Base article #811239).

If you receive an error similar to the following:

```
Error 5, ERROR [07006] [MySQL][ODBC 3.51 Driver][mysqld-4.0.12]Restricted
data type attribute violation(SQL_C_NUMERIC)
```

then you should use the Double data type instead of the Decimal data type to avoid receiving an error when using the ODBC.NET provider.

dbProvider

There are no known issues with this provider at this time.

MySQLDirect.NET

There are no known issues with this provider at this time.

Working with Popular .NET Reporting Components and Tools

The Partitioned Query for MySQL component can easily be used with popular .NET reporting components and tools to increase the performance and scalability of solutions that utilize these products when accessing MySQL databases. The following describes how to use some of these popular tools with the Partitioned Query for MySQL component:

ActiveReports for .NET

by Data Dynamics

ActiveReports for .NET by Data Dynamics supports binding a DataSet object as the data source for a report at runtime. To use the Partitioned Query for MySQL component with an ActiveReports for .NET report, all you need to do is set the **DataSource** property of the report to the DataSet object that is returned from the call to the **RunQuery** method of the Partitioned Query for MySQL component.

An example of binding a DataSet object as the data source for an ActiveReports for .NET report can be found in the ActiveReports for .NET User Guide in the section titled “Binding Reports to a Data Source”.

Additional information on ActiveReports for .NET can be found at www.datadynamics.com.

Crystal Reports for Visual Studio .NET / Crystal Reports v9

by Crystal Decisions

Crystal Reports for Visual Studio .NET and Crystal Reports v9 support reports that access DataSet objects by passing the populated DataSet object to the Report Engine using the Report Engine Object Model. You can populate the DataSet object by calling the **RunQuery** method of the Partitioned Query for MySQL component, and then pass the resulting DataSet object to Crystal Reports.

Documentation on using DataSet objects with Crystal Reports for Visual Studio .NET and Crystal Reports v9 can be found in the Crystal Decisions Technical Briefs database at http://support.crystaldecisions.com/docs.

Search the Technical Briefs database for documents with the file names **crnet_adonet.pdf** and **rtm_ReportingOffADONETDataSets.pdf**.

Additional information on Crystal Reports can be found at www.crystaldecisions.com.

Appendix A – License Agreement

LICENSE AGREEMENT

You should carefully read the following terms and conditions before using the Partitioned Query for MySQL software (the "Product"). Your use of the Product indicates your acceptance of this License Agreement. Do not use the Product if you do not agree with the License Agreement.

The Product is proprietary to Inspireon Software, LLC and is protected by United States Federal Copyright Law, international treaty provisions, and applicable laws of the country in which it is used. Inspireon Software, LLC retains the title to and ownership of the Product. You are licensed to use this Product under the following terms and conditions:

LICENSE:

The Licensee is defined as the individual software developer utilizing the Product. This License is not for an entire company, but for a single developer. Each developer within your company that develops applications using the Product must have a License in order to legally use the Product.

Evaluation Period: you are allowed up to 30 days from the date of first installation to evaluate the Product at no charge. During this 30-day Evaluation Period, the Product has full functionality, with no limits on features or capabilities. After 30 days from the date of first installation on any of your computers, you must either purchase a valid License to continue using the Product, or you must remove the Product from your computer(s) and cease using the Product. You are allowed to distribute the Product to other developers so that they can evaluate the Product. When distributing the Product for evaluation, you must distribute the entire ZIP file that you originally downloaded or received – which includes the component assembly DLL, the Developer's Guide, and the partitionedqueryresults.sql, License.txt, and Readme.txt files. If the developers you send the Product to for evaluation continue to use the Product after their 30-day Evaluation Period as defined above, they must either purchase a valid License, or remove the Product from their computer(s) and cease using the Product.

Upon purchase of a valid License, Inspireon Software, LLC hereby grants the Licensee a non-exclusive License authorizing the Licensee to use the Product on one or more computers at a time for development purposes only. For applications that are developed for use within your company, the distribution of those applications using the Product in a run-time environment is royalty-free requiring no additional license fees for the Product. If you use the Product to develop applications for use outside of your company, then a runtime License is required for each web or application server on which the application is deployed outside your company.

RESTRICTIONS:

You may not distribute, rent, sub-license or otherwise make available to others the software or documentation or copies thereof except as expressly permitted in this License without prior written consent from Inspireon Software, LLC. In the case of an authorized transfer, the transferee must agree to be bound by the terms and conditions of this License Agreement.

You may not remove any proprietary notices, labels, copyrights, and trademarks that appear on the software or documentation.

You may not modify, de-compile, disassemble, reverse engineer or translate the software.

(continued on next page…)

TERM:

You may terminate your License and this Agreement at anytime by destroying all copies of the Product and Product Documentation. This License will also terminate automatically if you fail to comply with any term or condition in this Agreement.

LIMITED WARRANTY:

This software and documentation are sold "as is" without any warranty as to their performance, merchantability or fitness for any particular purpose. The Licensee assumes the entire risk as to the quality and performance of the software. In no event shall Inspireon Software, LLC or anyone else who has been involved in the creation, development, production, or delivery of this software be liable for any direct, incidental or consequential damages, such as, but not limited to, loss of anticipated profits, benefits, use, or data resulting from the use of this software, or arising out of any breach of warranty.