1 Firebird Replication for High Availability

Firebird is an open source SQL relational database management system that is ACID compliant and supports many features including SQL compatibility, triggers, stored procedures and much more. This is delivered with a very small footprint and is scalable for both small and large databases with a very low cost base. This makes it an ideal database for both small and large businesses.

Contents

2 Introduction

Creating high availability architecture for 24/7 real world business operations introduces many challenges. Balancing availability and maintenance encompasses a number of important aspects:

- Data availability. Prevent interruption to business processes by ensuring access to data.
- Performance. Providing adequate response times for efficient business operations.
- Cost. Reducing costs as part of overall business strategy.

3 Replication

Replication allows data from a master database to be replicated to one or more child databases (also known as nodes) and vice versa. SieraDelta's replication engine is easy to setup and configure and allows workloads to be scaled between instances of Firebird on multiple servers.

3.1 Advantages of replication

There are many advantages to replication, including:

- Analytics. Analysis of information can take place on a child database, without affecting performance of the master database.
- Data distribution. Remote sites can work on local copies of data without permanent access to the master database.
- Increased availability. Should a node become unavailable due to hardware or other failure, clients can access an alternative node.
- Increased speed. Accessing data on a local network is generally faster than public network access.

3.2 Disadvantages of replication

There are several disadvantages to replication, including:

- Increased disk space. Storing copies of data on different sites consumes more disk space.
- Maintaining data integrity is more complex.

3.3 Replication Schema

The replication engine stores DML changes within a series of tables in each database that is replicated at user defined intervals between master and child databases. The interval specified would depend on the necessity of available data within the location. For instance, a support department or help desk facility linked to an online portal would require replication more regularly than the human resources department.

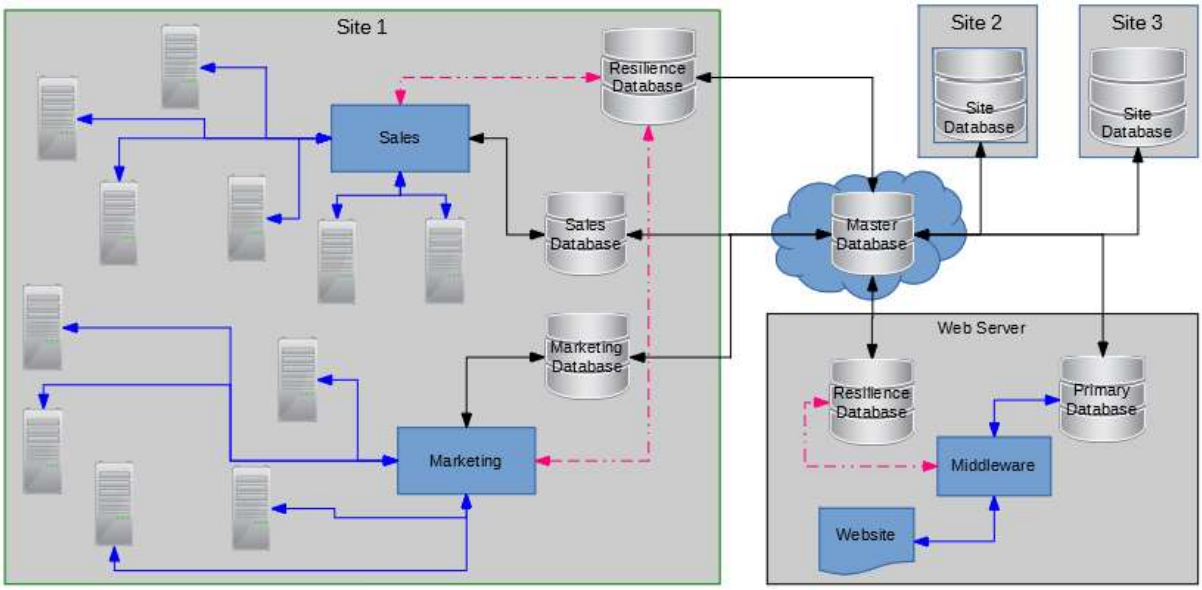
3.4 Asynchronous Replication

The replication engine is asynchronous by default; data is stored within a replication schema which is requested by child databases on request. This has the advantage that:

- The master database does not rely on or wait for child databases.
- Child databases determine which data is read from the master database.
- Child databases synchronise at pre-determined intervals.

4 Resilience

Providing a database for resilience, would allow business operations to continue should the primary database become unavailable through failure or maintenance. Replication is used to ensure the resilience and uptime within a site or location.



This image shows a configuration for departments on a site, each department would use their database instance, optimised for their needs, replication would synchronise their data and a further "resilience" database would be available should their primary database become unavailable. This model can be easily replicated across multiple sites and departments.

5 Remote Updates

As applications and websites evolve the need to automatically update remote databases will be necessary within a distributed model using replication. The replication engine provides a mechanism to update the database remotely. Remote update is configurable for each database to allow maximum flexibility and different database optimisations.

6 One Size Fits All

This is very rarely true; more often than not different business operations would require different database optimisations. Using replication, applications can connect to the most appropriate database for their needs. For instance, a website collecting SEO data would need to optimise the writing of data in order to optimise the user experience. This could be achieved by removing all indexes used in search operations. Using replication, a specific child database could be designated for SEO reports and include the required indexes to optimise querying the data. *Primary and foreign keys should never be disabled in node databases.*

7 Database Maintenance

At some point in the life cycle of a database it will require maintenance, whether it be backup and restore, optimizations or upgrading system versions. Using resilience and the Replication Engine a database, even the master database, can be taken off line for a pre-determined period for maintenance purposes.

If the master database is taken off line then no replication can occur until it becomes available again.

8 Automatic Backups

The replication engine has built in support for backing up databases and optionally sending copies of the backup, via FTP to a remote offsite location for safety.

9 Automatic Data Correction Rules

The replication engine contains a rules engine for automatically correcting data should it fail to replicate. In general the primary reasons for records to fail in replication are:

- Inconsistency in database structure between child and master databases.
- Index validation.

System managers can configure the replication engine to automatically adjust data in either the master or child database when records fail to replicate.

Should records fail to replicate due to validation errors etc, then an error log will be produced with complete details about the record that has failed to replicate.

10 Forced Verification

The replication engine contains a process of validating all records between master and child databases.

The process can be initiated in several ways:

- After a predetermined time each day.
- After a specific number of replications have taken place.
- Manually using the replication console.

Forced verification allows the replication engine to scan all records within replicated tables and verify that records match in both master and child databases. Missing records will be automatically inserted and subject to Automatic Data Correction Rules.

11 Connecting To Databases

The application code required to access multiple databases requires little overhead, a single method can be used to connect to a specific database, a further method would be required to specify the connection string.

The following code is in C# but can easily be ported to other languages.

11.1 Database Type

An enum is used to specify which database to connect to. The Standard database is generic for most connections and the FailOver is used should the connection to the Standard database fail. The enum can be expanded to suit business specific requirements.

```
public enum DatabaseType
{
    Standard = 0,
    FailOver = 1,
    Reports = 2,
    SeoData = 3
}
```

11.2 Connection Strings

A string array is used to hold connection strings to specific database types, this would be one for each enum in DatabaseType.

```
protected string[] _connectionString;

// If true, then the connection string provided will be used,
// otherwise the connection string provided will be substituted for a
// standard build connection string
protected bool StandardConnection = false;
```

The array can be initialised thus:

```
_connectionString = new string[Enum.GetNames(typeof(DatabaseType)).Length];
```

11.3 GetConnectionString Method

The GetConnectionString method will return the connection string required to connect to the database type specified.

```
private string GetConnectionString(DatabaseType databaseType)
{
    string connString = _connectionString[(int)databaseType];

    if (databaseType != DatabaseType.Standard && String.IsNullOrEmpty(connString))
    {
        connString = _connectionString[(int)DatabaseType.Standard];
    }

    FbConnectionStringBuilder csb = new FbConnectionStringBuilder(connString);

    if (csb.ServerType == FbServerType.Embedded)
        csb.ClientLibrary = String.Format("{0}\\fbembed\\fbembed.dll",
            Utilities.CurrentPath());

    if (StandardConnection)
    {
        csb.Pooling = true;
        csb.MinPoolSize = 0;
        csb.MaxPoolSize = 250;
        csb.ConnectionTimeout = 15;
        csb.ConnectionLifeTime = 60 * 20;
    }

    return (csb.ToString());
}
```

11.4 ConnectToDatabase Method

The connect to database method makes a physical connection to the required database and starts a transaction, this method requires several private fields in order to automatically attempt to switch back from a FailOver database to the Standard database.

Please note it is the responsibility of the calling method to dispose of the FbConnection and FbTransaction objects.

The method will attempt to make a connection to the specified database type, if an error occurs more than 3 times whilst making the connection then the FailOver connection will be used up to MaxReconnectAttempts, after this the method will throw an exception.

This method will also work for pooled connections, if for any reason the pooled connection is lost then further connection attempts will be made.

```
// Maximum database connection attempts
private const int MaxReconnectAttempts = 6;

// Maximum number of minutes to use fail over database
private const double MaxFailOverUsage = 5.0;

// Determines whether the fail over database is in use
private bool FailOverInUse = false;

// Date/Time Fail over database started to be used
private DateTime FailOverStart = DateTime.MaxValue;
```

```
private FbConnection ConnectToDatabase(ref FbTransaction tran,
    DatabaseType databaseType,
    System.Data.IsolationLevel isolationLevel = IsolationLevel.Snapshot,
    int attempt = 0)
{
    FbConnection Result = null;
    try
    {
        if (FailOverInUse)
        {
            TimeSpan span = DateTime.Now - FailOverStart;

            if (span.TotalMinutes > MaxFailOverUsage)
                FailOverInUse = false;
            else
                databaseType = DatabaseType.FailOver;
        }

        Result = new FbConnection(GetConnectionString(databaseType));
        Result.Open();
        tran = Result.BeginTransaction(isolationLevel);

        return (Result);
    }
    catch (Exception err)
    {
        if (err.Message.Contains("Error reading data from the connection") ||
            err.Message.Contains("connection shutdown") ||
            err.Message.Contains("Unable to complete network request"))
        {
            if (attempt < MaxReconnectAttempts)
            {
                if (attempt > 3 ||
                    err.Message.Contains("Unable to complete network request"))
                {
                    FbConnection failedConn = ConnectToDatabase(ref tran,
                        DatabaseType.FailOver, isolationLevel, attempt + 1);

                    if (!FailOverInUse && failedConn != null)
                    {
                        FailOverInUse = true;
                        FailOverStart = DateTime.Now;
                    }

                    return (failedConn);
                }
                else
                {
                    return (ConnectToDatabase(ref tran, databaseType, isolationLevel,
                        attempt + 1));
                }
            }
            else
            {
                throw new Exception("Unable to connect to database");
            }
        }
        else
            throw;
    }

}
```

11.5 Example Usage

The following code shows example usage for connecting to a database using the above methods.

```
internal override void AutoUpdateDelete(AutoUpdate autoUpdate)
{
    FbTransaction tran = null;
    FbConnection db = ConnectToDatabase(ref tran, DatabaseType.Standard);
    try
    {
        string SQL = "DELETE FROM AUTO_UPDATE_DATA WHERE ID = @ID";

        FbCommand cmd = new FbCommand(SQL, db, tran);
        try
        {
            cmd.Parameters.Add("@ID", FbDbType.BigInt);
            cmd.Parameters["@ID"].Direction = ParameterDirection.Input;
            cmd.Parameters["@ID"].Value = autoUpdate.ID;
            cmd.ExecuteNonQuery();
        }
        finally
        {
            cmd.Dispose();
            cmd = null;
            tran.Commit();
        }
    }
    finally
    {
        tran.Dispose();
        tran = null;
        db.Dispose();
        db = null;
    }
}
```

12 Summary

This paper talks about Firebird and SieraDelta's replication engine, there are many replication engines available and users should choose the one that suits their business requirements. Likewise, there are many RDBMS Servers available and the methodology discussed here is easily transportable to these.

Author: Simon Carter