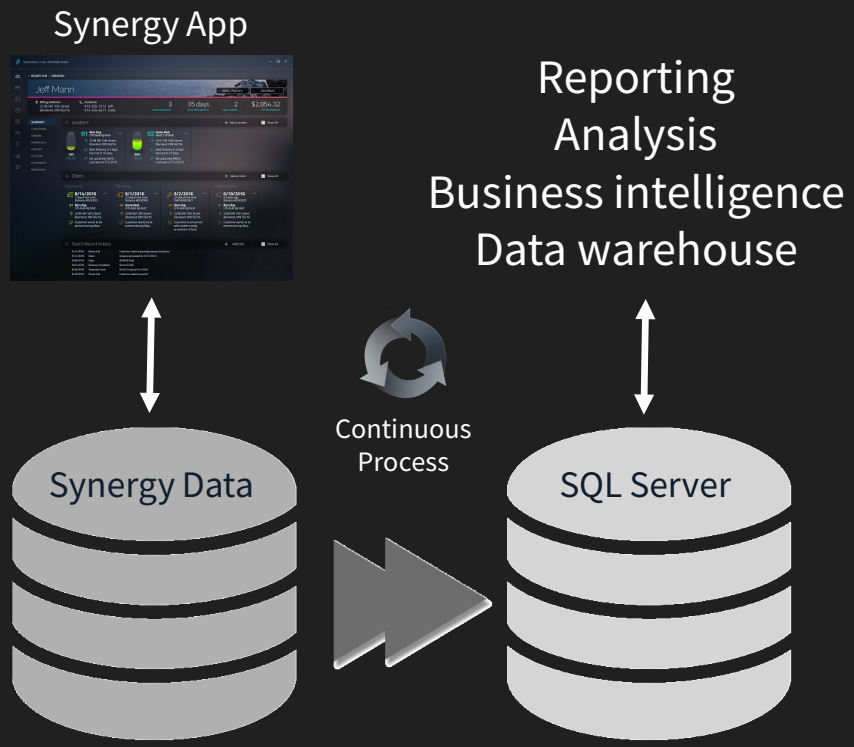
SYNERGY
DEVPARTNER
2018
CONFERENCE
OCT 8-12 NEW ORLEANS, LA

SQL Replication Project Update

Presented by Steve Ives

SQL Replication Project Update

- Basic principles
  - What, why, and how
- Project update
  - What's new since the last conference
- Getting started
  - Sample environment
  - Your own environment

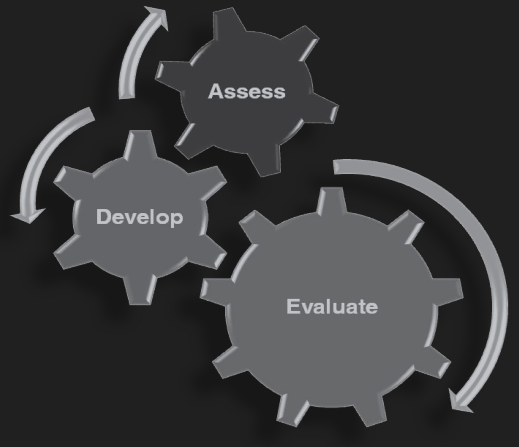


Basic Principles & Goals

- Data from one or more files is replicated to matching tables in a relational database
- Close to real-time in most cases
- One-way replication
- Easy to implement (days not weeks)
- Minimal changes to original app
- Minimal performance impact on original app
- Minimal cost

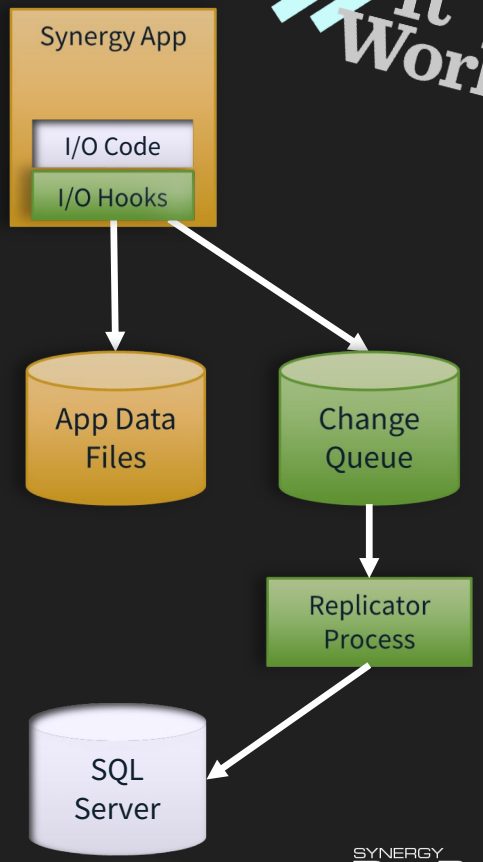
Approach

- Core environment
  - Pre-written generic code providing a replication framework
  - Replicator process and supporting utilities and APIs
- Code generation
  - Database interaction code generated for each file
- “Plug-in” to original app code
  - Add I/O hooks object to channels opened for update (frequently a 1-line change)
- I/O hooks code (supplied)
  - Records changes (creates, updates, and deletes) to “change queue” file



How Replicator Works

- Detached process or Windows service
- Processes change queue and updates database
  - Sequence of events from original app
    - Insert, update, delete
  - Management features
    - Create table, bulk load data, etc.
- Naming conventions allow generic code to call generated table-specific routines via XSUBR
- Processing loop
  - Wake up and check change queue for instructions
  - Process instructions
  - Sleep a while
  - Repeat



How It Works

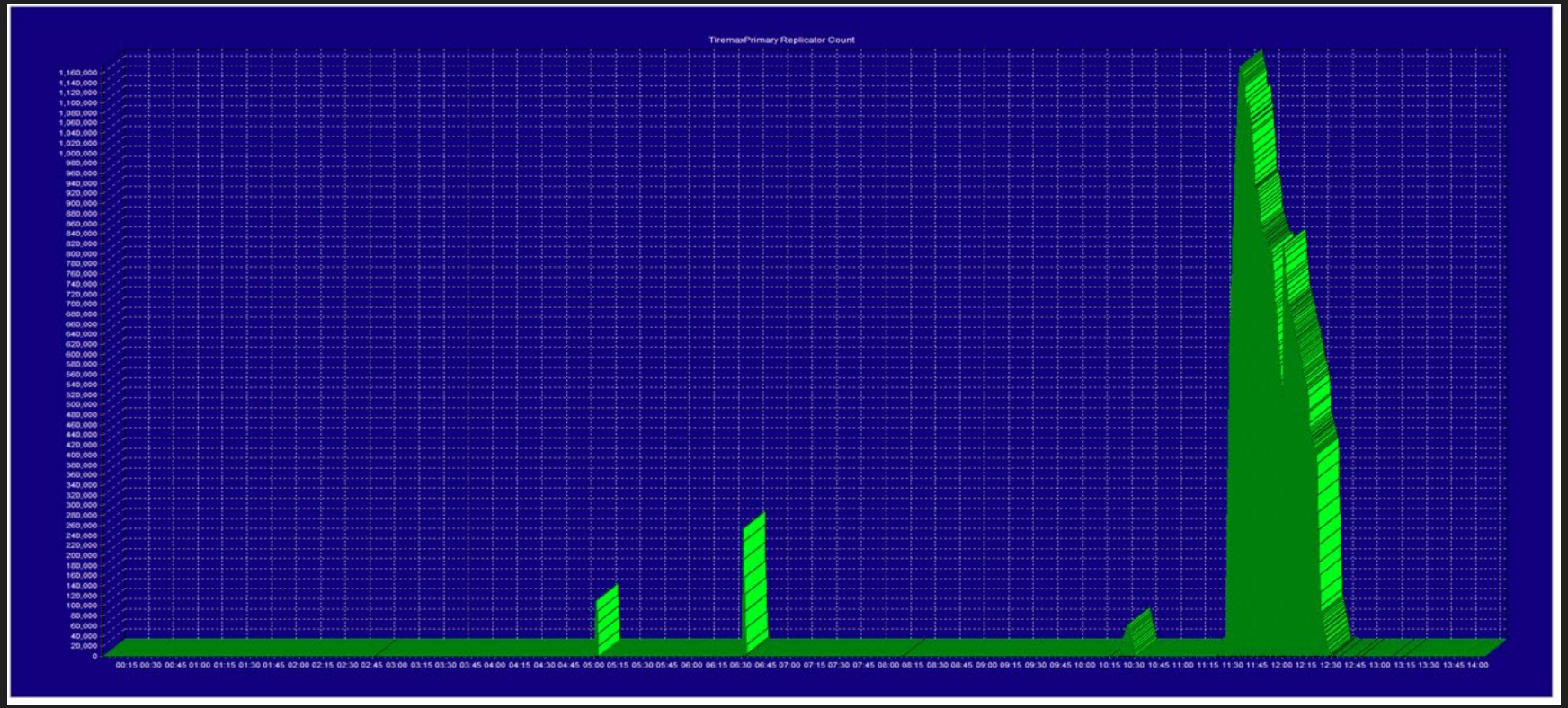
Replicator Capabilities

- Table & index management
  - Create table
  - Add indexes
  - Delete indexes
  - Delete table
- Initial data management
  - Load table
  - Create and load table
  - Bulk load table
  - Delete all rows
- Data export
  - Export to delimited file
- Individual changes
  - Insert row
  - Update row
  - Delete row
- Replicator management
  - Change interval
  - Cycle log file
  - Delete all instructions
  - Close file
  - Shutdown

Demonstration

- Exploring the sample environment
  - Download the code from GitHub
  - Create a new database
  - Configure the environment
  - Build the code
  - Start the replicator
  - Use the sample app to make changes to the data
  - See the changes replicated to the database

Sample Throughput – Mavis Tire Day End Posting



Adding Files to the Replicated Environment

- Describe the files in the repository
  - STRUCTURES, KEYS, TAGS, FILES
- Generate code for each STRUCTURE and add it to your ELB

```
rem Define the structures to generate code for
set STRUCTURES=CUSTOMER INVENTORY ORDER VENDOR

rem Generate SQL I/O code
codegen -s %STRUCTURES% -t SqlIO -define CLEAN_DATA -r
```

- In ConfigureReplication.dbl, declare the new replicated file

```
("EMPLOYEE.ISM"),
    new IoHooksISAM(channel,"EMPLOYEE")
```

- Call ConfigureReplication when you open the file for update

```
open(ch=0,u:i,"DAT:customer.ism")
xcall ConfigureReplication(ch)
```

Change Queue File

Previous layout

```
RECORD REPLICATION
    TRANSACTION_ID,
            A20
    ACTION,
            D2
    STRUCTURE_NAME,         A32
    REPLICATION_KEY,        A20
ENDRECORD
```

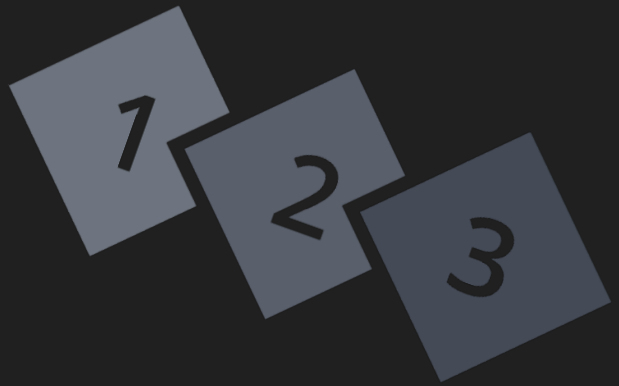
New layout

```
RECORD REPLICATION
    TRANSACTION_ID,                 I8
    ACTION,
            D2
    STRUCTURE_NAME,         A32
    RECORD,                 A65000 *
ENDRECORD
```

Note: Entire record now in queue file, so replicator no longer needs to access underlying files. Synergy I/O reduced by 50%

* RECORD is an A32000 on OpenVMS where the RMS maximum record length is 32234

Change Queue Auto Sequence Key



- Change queue had an A20 key with a unique %DATETIME value
- Not granular enough
  - Up to hundreds of duplicate keys in busy environments, each with a short SLEEP!
- Significant bottleneck putting instructions INTO the queue!
- Solution: I8 auto sequence key
- DON’T use ISLOAD, and ONLY use FCONVERT with <u>COUNTED</u> files!
- Replicator now creates change queue file if not found

ISAM File Key Requirements

- Previous behavior
  - ISAM files must have a unique PRIMARY key
  - Otherwise a CONSISTENT unique key needed to be added
    - REPLICATION_KEY, A20, %DATETIME
  - Required a record layout change and file conversion

- New behavior
  - Replicator detects the first unique key and uses that
  - If no unique key, one must still be added
    - Any unique key!
  - Only requires a file conversion

Relative File Support

- Added support for replication of relative files
- Record number and a colon prepended to record in queue file
- RowNumber column added to database table for synchronization
- New IoHooksRELATIVE class
- Appropriate hooks object (ISAM or Relative) applied by ConfigureReplication()

“FileService” on Windows Database Server

- Synergy .NET Windows Service
  - Hosts an ASP.NET Web API 2 RESTful web service

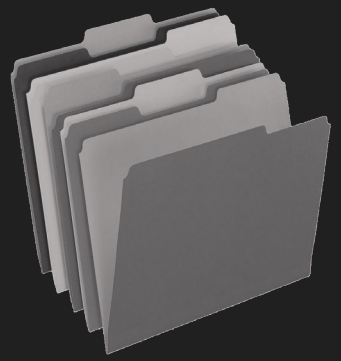
- Purpose
  - Manage server-based files via HTTP
  - Upload, list, download, delete, etc.
  - Chunked upload allows very large files to be uploaded

- Used by replicator if database server is remote
  - Upload files to database server
  - Facilitates local BULK LOAD on remote server

Bulk Load

- Previous “table load” mechanism
  - Create table and add indexes
  - Read 1000 records into dynamic memory
  - Insert them individually, but in a single transaction
  - Repeat until no more records
  - Still available if needed
- New “bulk load” mechanism
  - Create table
  - Export data to delimited text file
  - If remote, transfer to database server via “FileService”
    - Reads file in 32K chunks, uploads in 10MB chunks
  - BULK INSERT local file to database, VERY FAST!
  - Create indexes
- Bulk load is orders of magnitude faster than table load

Index Creation & Removal



- Database indexes created to match ISAM alternate keys
- Previous mechanism
  - Indexes created immediately after CREATE TABLE
  - All indexes must be updated during initial data load
- New mechanism
  - Indexes created AFTER initial data load
  - Improved performance during load
  - Much faster to add indexes once data present
- Indexes may now be removed and re-added as necessary

Clustered Indexes

- A “clustered index” defines how the physical data is organized and stored in a database table
  - One clustered index per table
    - Usually the primary key
  - Having a clustered index generally improves performance when accessing the table
- SQL Server primary keys MUST be unique
- For ISAM files with a unique primary key (most), a matching clustered primary key is created
- Previous behavior when primary key not unique
  - No clustered index created
- New behavior
  - Index associated with first unique key is clustered

Database Commit Modes

- Manual commit
  - Generated code explicitly wraps each change in a transaction
  - Each change explicitly committed
  - Expensive: three network round-trips
  - Original mechanism
- New autocommit
  - SQL Server connection placed into “autocommit” mode
  - Generated code performs no transaction processing
  - Significantly reduces database operations
- New batch commit
  - Replicator “batches” changes together
  - Three events cause a commit
    - Replicator queue empty
    - Batch full (configurable, default 1000)
    - Major event (e.g., create table)
  - Significantly improves performance
  - Default behavior

Improved Shutdown Mechanism

- Previous behavior
  - Stop command queued in-line with all others
  - Had to wait for replicator to get to the stop command

- New behavior
  - Stop command is encoded into the first instruction in the queue
  - Replicator stops immediately after completing current operation
  - Could still be a while if the current operation is a table load

Multiple Replicator Instances

- Now possible to run multiple replicator instances
  - Configurable instance name

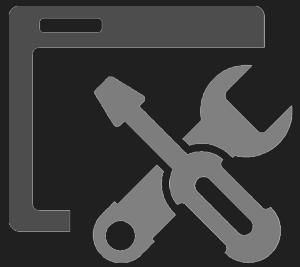
- Not a mechanism to further improve scalability
  - Single replicator per data set to ensure correct sequencing & data integrity

- Intended use case
  - One replicator per data set
  - One queue file per instance
  - One database per instance
    - Wouldn't be hard to append instance name to table names

Optional Data Cleanup

- Turns out bad data is surprisingly common
  - Alpha data in numeric fields
  - Required fields with no value
  - Date and time fields with invalid data
- Common causes
  - Re-purpose previously used space in record, file not “cleansed”
  - Data updated by external sources without appropriate validation
    - *xf*ODBC, web services, etc.
  - Programming errors
- Data “verification” on by default, but at a cost
  - Null terminate alpha fields
  - Verify numeric fields contain numeric values
  - Verify date and time values, etc.
- New option to disable verification if you KNOW data is clean

Configuring Replicator



- Two mechanisms for configuring replicator options
  - Environment variables
  - Command-line options (new)

- Command-line options
  - Always override environment variables
  - Particularly useful when configuring & starting “services”

- More information
  - https://github.com/SteveIves/SqlReplication/wiki/Configuring-the-Replicator-Environment

Replicator Logging

- Previous behavior
  - Opened and closed log file for each operation
  - Great with low to moderate throughput
  - Bottleneck with high throughput

- New behavior
  - Log file remains open
  - New replicator instruction to “cycle” the log
- Added support for additional logging to system log (SYN_REPORTEVENT)
- Detailed logging is for <u>development & debugging</u> only
  - Significant performance overhead

Email Notification

- Send email when significant events occur
  - Requires access to a relaying SMTP server

- Events
  - Failed to connect to database
  - Replicator started
  - New table created
  - Indexes added to a table
  - Indexes removed from a table
  - Full table load completed
  - Table truncated
  - Table deleted
  - Replicator stopped
  - An unexpected error occurred

- Configuration
  - Mail server
    - REPLICATOR_SMTP_SERVER = server
    - -mailserver server
  - Sender address
    - REPLICATOR_EMAIL_SENDER = email
    - -mailfrom email
  - Recipient address
    - REPLICATOR_ERROR_EMAIL = email
    - -erroremail email

Stop on Error

- New option to stop replicator if error occurs
  - Most common cause of errors is bad data
  - Instruction causing error will be at head of queue
  - Fix issue and restart replicator to retry and continue
- If not enabled, data can get out of sync
  - Tough to diagnose
  - Some may prefer this
- Configuration
  - REPLICATOR_ERROR_STOP = YES
  - -stoponerror

Database Cursor Use

- Previously, cursors opened & closed for every operation

- Now, frequent operations (INSERT and UPDATE)
  - Opened on first use, maintained and reused
- Less frequent operations
  - Cursors still opened & closed

Database Connection Configuration

- Configure SQL Connection “Max Columns”
  - REPLICATOR_MAX_COLS = n
  - –maxcolumns n
  - Default is 254

- Configure SQL Connection “Max Cursors”
  - REPLICATOR_MAX_CURSORS = n
  - –maxcursors n
  - Default is 128
  - Allow 4 per file / table

Replicator Menu Utility

- Instruction queue is the primary replicator control mechanism
- Management utility queues common commands
  - Start & stop
  - Manage queue
  - Change processing interval
  - Cycle log file
- Also manage tables
  - Create, load, purge, delete, etc.
- Primarily a development & testing tool
- Simple queue monitor tool

```
Replicator Control      Instance: TESTINSTANCE   Table: Employee

 SR. Start Replicator (Normal)        C.  Create table
 SD. Start Replicator (Debug)         L.  Load table
 SN. Stop Replicator  (Now)           CL. Create and load table
 SQ. Stop Replicator  (Queued)        BL. Bulk load table
                                      AI. Add indexes
 V.  View instructions                DI. Delete indexes
 P.  Purge instructions               TR. Truncate table
 Q.  Queue length                     D.  Delete table
 QM. Start Queue Monitor              X.  Export to CSV

 I.  Change interval
 N.  Change instance name
 R.  Restart log file

 A.  Available tables
 T.  Set current table
 M.  Start maintenance program

 E.  Exit


 Select an option:
```

```
Replicator Queue Monitor

Queue length: 0
```

Reduced Code Generation

- More code now “supplied and ready to go”
- Two important things code generated
  - Structure specific SQL code (18 routines per table)
    - Create table
    - Load table
    - Insert row
    - Update row
    - Delete row, etc.
  - Routine providing a list of replicated tables

```
                ,boolean    ;;OK to co
            nd  ,boolean    ;;Should we o
                ,int        ;;Return status
        ror     ,int        ;;Database error n
    ansaction   ,int        ;;Transaction in pro
    ength       ,int        ;;Length of a string
    errtxt      ,a512       ;;Error message text
  irecord

iteral
    sql         ,a*, "INSERT INTO Employee ("
    & +                '"EmpId",'
    & +                '"EmpFirstName",'
    & +                '"EmpLastName",'
    & +                '"EmpDept",'
    & +                '"EmpHireDate",'
    & +                '"EmpPhoneWork",'
    & +                '"EmpPhoneHome",'
    & +                '"EmpPhoneCell",'
    & +                '"EmpPaid",'
    & +                '"EmpHomeOk",'
    & +                '"EmpDateOfBirth",'
    & +                '"EmpHireTime",'
    & +                '"EmpEmail",'
      +                '"EmpAddressStreet",'
                       '"EmpAddressCity",'
                       '"EmpAddressState",'
                       '"EmpAddressZip"'
                       ") VALUES(:1,:2,:3
```

Availability and Documentation

- Sample environment fully tested on
  - Windows
  - Linux
  - OpenVMS
- Should work on all currently supported Synergy platforms
- Improved documentation
  - https://github.com/SteveIves/SqlReplication/wiki

Getting Started with the Sample Environment

- Source code
  - https://github.com/SteveIves/SqlReplication
  - Name recently changed from SqlReplicationIoHooks
  - Redirect in place, but better to change local copies
    - `git remote set-url origin new_url`

- Development environments
  - Visual Studio solution
  - Workbench workspace
  - Linux shell scripts (bash)
  - OpenVMS command procedures

- Follow the “Getting Started” guide

Getting Started in Your Own Environment

- Pick a small number of files to start with
- Fully and accurately configure repository for those files
  - Structures, fields, keys, tags, files
  - Multi record type files require one structure per record type
- Create a new ELB project
  - Add all the supplied subroutines, functions and classes
  - Generate SQL connection code for each structure (follow regen.bat)
  - ELB must be prototyped because it contains OO code
- Manually maintain
  - ConfigureReplication to know about replicated files
  - IoHooksISAM to know about multi-record type files / tables
- Build replicator and other utility programs, link against ELB
- Modify your code to call ConfigureReplication when channels are opened

Requirements & Restrictions

- ISAM files must have at least one unique key
- Maximum record lengths
  - 65,000 with SDMS
  - 32,000 with RMS (OpenVMS)
- Repository to support code generation
  - Structures
  - Keys
  - Tags
  - Files
- Synergy V10 or higher
  - Auto-sequence keys
- CodeGen 5.3.3 or later
  - Always try to stay up to date
- Windows licenses
  - 1 SQL Connection for SQL Server
  - 1 runtime for FileService (if used)
  - 1 runtime for CodeGen (development)
- Replicator platform licenses
  - 1 runtime for replicator

SQL Replication Project Update

Who has the first question?