# Continuous Availability and Disaster Recovery using IBM Db2 Q Replication – Best Practices for Application Upgrades with DDL changes

Keywords: IBM Data Replication for Availability; Db2 Disaster Recovery; Q Replication; Db2 LUW; DDL replication; schema subscriptions

Author: Serge Bourbonnais - IBM Silicon Valley laboratory – 01/30/2020

## Introduction

IBM Q Replication is a log capture, transaction replay software-based replication technology that enables the deployment of Active-Active configurations for *Continuous Availability* -- which includes *Disaster Recovery*. This paper discusses the considerations and best practices for handling application upgrades that may include database schema changes in such configurations.

## What is an Active-Active Configuration?

An Active-Active configuration consists of *independent* and *active* database servers that can be geographically dispersed and kept synchronized using replication technology. Each server is active in that it can process database workloads; neither is an idle standby that needs to be activated before an application can run on it.

## Using Q Replication Technology for Continuous Availability

With Q Replication, Continuous Availability is provided at the application-level by replicating database transactions against selected sets of tables. Replication is established in both directions between pairs of Db2 servers that can be located very far apart, and applications can freely switch back and forth between them. The replication process can be suspended and restarted in any of the directions for any period. Applications that are switched over to another server during an outage can be redirected to the initial Db2 server once this server is restored and has been re-synchronized by restarting replication.

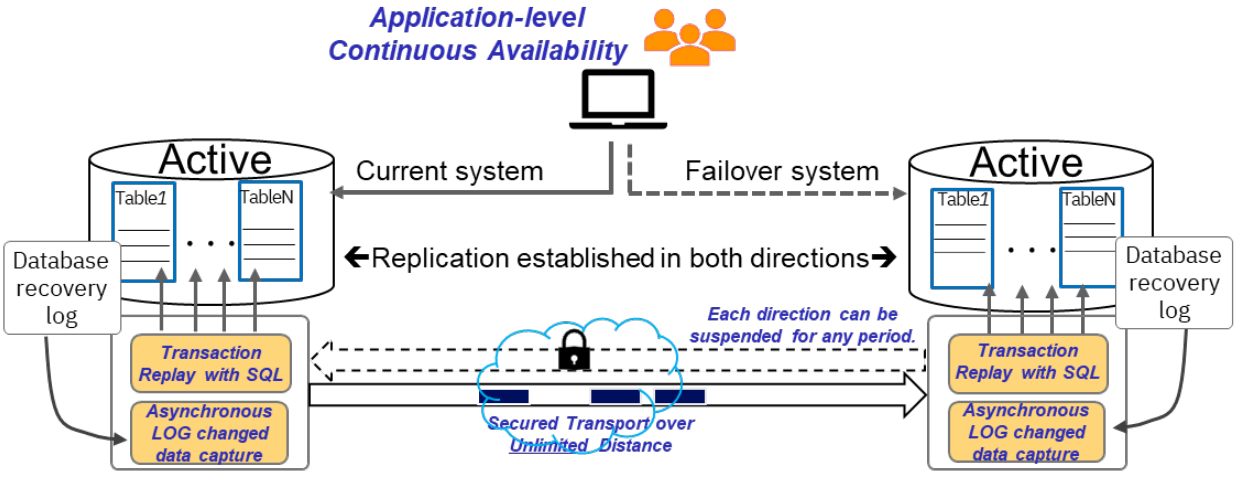


*Figure 1 - Using Q Replication Technology for Continuous Availability*

## Upgrades are Increasingly Frequent -- Maintenance Windows are Not

Modern businesses cannot tolerate service interruptions, yet at the same time aspire to constantly improve the customer experience in terms of performance, usability, and functionality by pushing product enhancements as frequently as possible through an agile development cycle.

Change always introduces risk. As most software users have experienced, upgrades sometimes introduce unpredictable and undesirable effects that in the worst case make the upgraded system unusable. Moreover, some upgrades are not reversible and may cause repeated system outages until the problems caused by the upgrade are resolved and stability is restored.

## Continuous Availability and Disaster Recovery – Requirements' Paradox

Continuous Availability means that you can avoid all outages, both unplanned as the result of a failure or disaster; and planned for any kind of upgrades. Eliminating outages for upgrades has different requirements on the replication process than providing disaster protection.

**Disaster Recovery (DR)** requires maintaining, ideally in real-time, a redundant, identical system situated at enough geographic distance to be in a different risk zone and where the applications can be switched over if a disaster occurs. The DR system must have all objects and data that are needed for the application to be restarted within the Recovery Time Objective (RTO): the time it takes for the application to be restarted, and within the Recovery Point Objective (RPO): the amount of tolerable data loss following a disaster.

**Continuous Availability (CA)** goes beyond DR. The system must not only be recoverable should a disaster occur, but must also provide business continuity throughout upgrades, migrations, application deployments and configuration changes. Quality of Service must be guaranteed to the end users during and after changes are made.

Keeping a down-level system available for fallback in case of unexpected problems with the upgrade is necessary to achieve Continuous Availability, whereas Disaster Recovery readiness requires a current-level system that is ready for failover when needed.

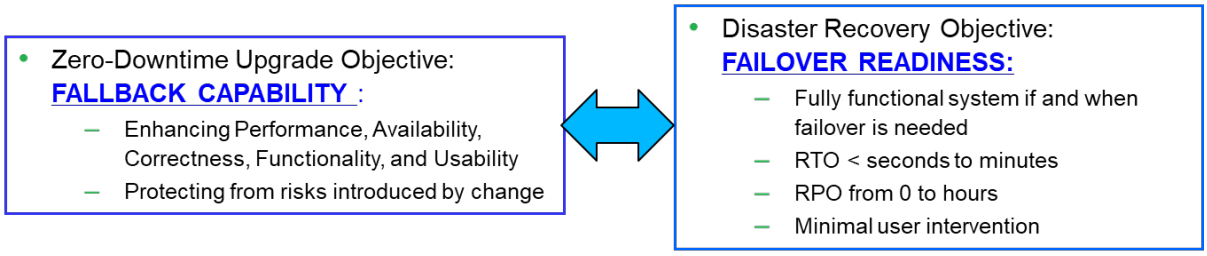


*Figure 2- Balancing No-Outage upgrades with Readiness for Disaster Recovery*

## Zero-downtime Upgrades with Fallback Capability

Because each Db2 system in an Active-Active configuration can have different capacity, hardware, and software versions, you can keep the old version operational and ready for fallback until the upgraded system is fully validated and stable.

The trade-off in postponing the upgrade of the failover system is an exposure for Disaster Recovery Readiness. The changes required for the upgrade need to be applied at the failover system, which may elongate the RTO should a disaster occur once the upgraded application is operational.

## How to Meet your Disaster Recovery Requirements

For disaster recovery, the standby system must have all objects required by the application **before** this application can be switched over to that system. For example, if a new user is added with database

access credentials, these credentials must exist at the alternate system for the application to be successfully redirected. Other examples include the creation of stored procedures and views, or the installation of dependent software. The changes can be applied at any time before a failover is needed, not necessarily simultaneously on all systems.

Data Definition Language (DDL) statements that do not affect the transaction replay replication process, such as adding credentials for a new user, can be applied simultaneously on all systems. Replication runs under its own user id and credentials and is not affected by such environment changes.

Note that even though the creation of a stored procedure is not replicated, the effect of invoking a stored procedure is replicated. For example, if a stored procedure's execution at the source system changes rows in a table that is subscribed for replication, then these rows are replicated to the target. Deploying any new stored procedure on all systems where it might be eventually needed will not impact the replication process and is a recommended practice.

Changes that affect the replication process can be automatically handled by Q Replication.

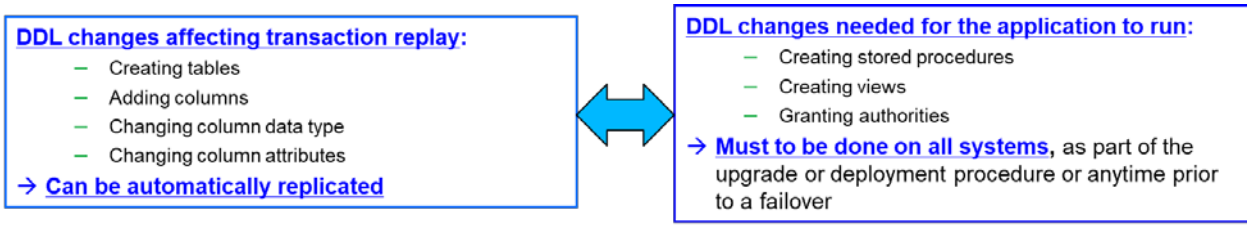


*Figure 3 - Applying DDL changes during Maintenance*

## How does Q Replication handle DDL changes?

Q Replication replicates database transactions by replaying insert, update, delete, and truncate statements. It also deals with most table changes that affect these operations. If you elect not to replicate an alter operation, in order to keep the target system down-level, but keep Q Replication running, Q Replication will still update its control information to remember the change and deal with the mismatch.

For example, if a column of a table is altered from integer to decimal type, Q Replication will detect the change to the source table and (optionally) perform the alter on the table at the target system. If a new partition is attached to a source table, Q Replication will replicate any rows inserted and modified into this new partition, but it will not create and attach an identical partition at the target system; the rows will fall into an existing partition at the target. If the new partition is needed at the target, it must be created outside of the replication process.

## DDL Changes made by Applications at Run-time

Some applications may dynamically create tables, either temporary or long-lived, a part of their run-time. Q Replication can replicate create/drop tables made by such applications. If the application is switched to the other site, it will have the tables needed to continue running from where it was interrupted on the primary site.

On Db2 distributed platforms (not z/OS), a Q Replication ***schema subscription*** identifies the schema names for which create and drop table operations are replicated. This is explained in more detail in a subsequent section.

## DDL Changes made for Application Upgrades

Changes required for deploying or upgrading an application are generally best carried out on all servers where the application may eventually run, even if some of these changes can be replicated automatically by Q Replication. Creating new tables for an upgrade is a maintenance activity that is usually under strict control in order to minimize risk. A reason often cited for controls and checks on database changes is that they may affect several applications and users that are sharing the system.

## How and When to Upgrade the Server that is used for DR Failover

How do you manage database schema changes introduced for an application upgrade when Q Replication is used to maintain a DR site for this application? If your upgrade introduces new tables, should you create the new tables on both systems as part of the maintenance activity or should you let replication automatically detect and replicate the creation of these new tables to the DR site?

The creation of new tables is often only a fraction of the changes needed for an application upgrade. The new application may also require upgrading the database to a new version, modifying disk configuration, adding capacity, changing database parameters, and so on. All changes are generally bundled within one maintenance intervention for risk mitigation and delivery time objectives. The decision on how to upgrade the DR site often depends on the complexity of the changes needed:

- **If all changes are limited to what can be replicated, rely on Q replication to automatically upgrade the failover site.** Some enterprises have corporate rules restricting application changes to backward compatible changes, for example allowing columns to be added to existing tables, but not dropped.
- **If the required changes go beyond what can be replicated, run the upgrade script on all sites.** You stop replication and upgrade a first site, keeping the other site for fallback in case of a problem with the upgrade. You upgrade the second site later, perhaps even a few days later.

With either approach, you can keep the down-level site available for fallback, by suspending replication before starting the first upgrade; and keeping the application live on the down-level site.

## What Schema Changes Can Q Replication Handle?

Replication can replicate between dissimilar databases by converting the data if necessary, allowing for *concurrent versioning:* simultaneously running different versions of an application operating on common business data, for example, the same set of customers. Supporting concurrent versions with *incompatible* schema changes requires writing stored procedures or expressions to transform the data, which can be justified when no outage is tolerable or when both old and new systems might be required to concurrently run for extended periods of time. For example, in a business consolidation scenario following an acquisition, an enterprise concurrently ran two versions of a system for months, keeping them synchronized with Q Replication, before the enterprise was finally ready to migrate all its clients onto the preferred system. Figure 4 illustrates how to deal with dissimilar schemas.

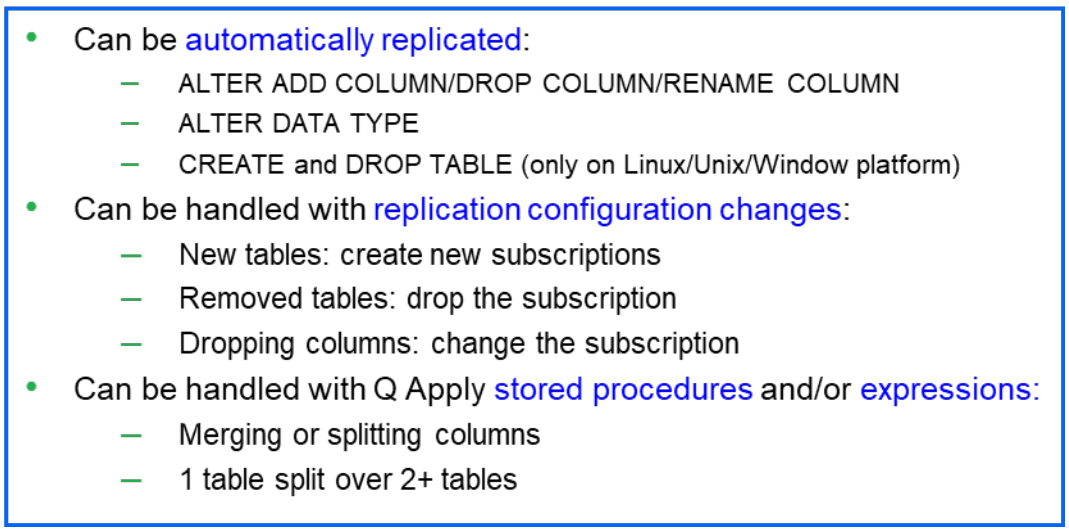


*Figure 4 – Allowing Concurrent Versioning: Dealing with disruptive schema changes introduced by an application upgrade*

## Adapting your Application Upgrade Process to the Active-Active Configuration

Increasingly, modern enterprises adopt agile development policies, giving power to the application developer to push changes onto the servers. The operational procedure must account for pushing the changes to each server as part of the deployment process.

The Q Replication process tolerates schema changes that have already been made at a target site. For example, if you change a data type at both servers when pushing an upgrade, when Q Replication is restarted, it will detect the change in the Db2 log of the source database, update its meta-data so it can correctly replicate the data, and either make the same change or simply verify that the change has been correctly made by the user at the target.

## Database Maintenance

In an Active-Active configuration, each database must be individually managed. For example, changing Db2 database configuration parameters, taking backups, running database utilities such as REORG, RUNSTATS, etc., need to be done for each site. However, you only need to do it after you are satisfied that the change has the desired impact on a first system.

## How to Replicate create and drop table – the *Schema Subscription*

This function is currently not available for Db2 on z/OS, only for Db2 on distributed platforms. To replicate create and drop table operations, you need to create ***schema subscriptions***.

A schema subscription provides a pattern for the schema name, i.e., owner name, under which tables might be created. For example, assume the database contains the schema **SALESREPORT.** You can create a schema subscription for 'SALESREPORT', by using the Q Replication **ASNCLP** scripting language:

```
ASNCLP SESSION SET TO Q REPLICATION;
SET RUN SCRIPT LATER;
SET SERVER CAPTURE TO DB A;
SET SERVER TARGET  TO DB B;
SET CAPTURE SCHEMA SOURCE ASN;
SET APPLY SCHEMA ASN;
CREATE SCHEMASUB schemasub1 SUBTYPE U REPLQMAP A2B FOR TABLES OWNER
LIKE SALESREPORT;
```

Creating a schema subscription updates Q Replication control tables in Db2. Using schema subscriptions requires the Db2 database configuration parameter LOG_DDL_STMTS=Y, which instructs Db2 to write the SQL text in the recovery log for DDL statements. The schema subscription 'schemasub1' created with the CREATE SCHEMASUB command above is *activated* when you recycle the Q Capture program or when you issue a start command from ASNCLP:

```
START SCHEMASUB schemasub1 ALL;
```

When you start a schema subscription with the keyword **ALL**, it automatically starts replicating all existing tables for the schema that are not already subscribed for replication. This is a very powerful feature of schema subscriptions that can be leveraged to ensure no table is missing from your replication configuration when you start replication for the first time.

Specifying '**NEW ONLY** will not replicate tables for this schema that are not already replicated. It will only check for tables created and dropped in the future under the schema. Tables will be dropped at the target, as long they match the name pattern, even if they are not replicated.

```
START SCHEMASUB schemasub1 NEW ONLY;
```

When a new table is created under the schema, Q Replication will create the same table at the target with the same attribute as the source. For example, given the following script at the source system:

```
CREATE TABLE SALESREPORT.T1 (col2 char(5));
ALTER TABLE SALESREPORT.T1 ADD COLUMN col1 int NOT NULL default 0);
ALTER TABLE SALESREPORT.T1 ADD CONSTRAINT t1key primary key(col1);
INSERT INTO SALESREPORT.T1 (col1, col2) values (1, 'FIRST');
```

Q Replication will create the same table T1(col1 int, col2 char(5)), by replaying all DDL operations up to the first insert. After the table is first used, the replication subscription is finalized, and changes such as create index on this table are no longer replicated.

## Caveats with Using *Schema Subscriptions*

### Tablespaces

Q Replication will replay the create table operation using the tablespace name specified by the user (If a tablespace name is not specified on the create table statement, the table is created into the default Db2 tablespace). Currently, Q Replication cannot create the tablespace automatically when it does not exist at the target, but this is expected to be supported in the future. When creating new tablespaces, you should do it on all systems where your application might run.

### Identity Columns

The values for an identity column are automatically generated by Db2 when a row is inserted. For example,

```
CREATE TABLE SALESREPORT.T2 (col1 bigint not null generated by default
as identity (start with 1, increment by 2), col2 int) "
```

The values dispensed by Db2 are specific to each server, with each Db2 instance internally maintaining a sequence object to generate those values. Q Replication will replicate rows that include the identity column into the target system (even if declared as generated always on Db2 for distributed platforms) and keep the tables identical. However, when you switch your application to the remote site, the Db2 at

that site will start generating values from 1, potentially causing replication conflicts. A precautionary action is required before the application can be switched to the remote site.

Even if the table SALESREPORT.T2 is automatically created at the DR site by an active schema subscription, the table cannot be used without resetting the START value of the identity column to ensure that Db2 generates non-overlapping values with those generated by the Db2 at the source.

The simplest way to ensure that values do not overlap is to declare the identity columns to generate **odd** values at one site and **even** values at the other site. In our example, because the increment is by 2, the first site is already generating odd values. By changing START from 1 to 2 at the DR site, Db2 will start generating even values. This only needs to be done once for the lifetime of that table. before a first site switch.

```
ALTER TABLE SALESREPORT.T2 alter column col1 RESTART with 2;
```

An alternative is creating and maintaining a script that does 'select max(col1)' from the table and alter the table identity column to restart past that number, prior to switching over the application.

## Initial Load of the Target Table when Activating a Schema Subscription

A schema subscription creates table-level subscriptions with the options defined in the Q Replication configuration control table IBMQREP_SUBS_PROF. One important attribute is HAS_LOADPHASE, which determines if the newly created subscription will load the table onto the target. It should be defined as N in an Active-Active environment, because schema subscriptions in both directions will ensure that all data inserted into a new table is replicated.

## Using Schema Subscriptions in the Reverse Direction

Assume that your application currently runs on site A, changes are replicated to site B, and you had a schema subscription replicating create/drop tables from A to B. Each time a table was created at site A, it was also created at site B by Q Replication.

You now want to switch the application from A to B. Site A will become the failover site, for which you will start replication in the reverse direction from B to A.

When a table was created at site B by Q Replication under a schema subscription, the newly created table at the target was not automatically subscribed for replication in the reverse direction. Therefore, before you start replication from B to A, you must ensure that all tables that exist at site B are replicated back to A.

Before switching the application to site B, start the SCHEMA subscription from B to A with ALL.

```
START SCHEMASUB schemasub1 ALL;
```

This will ensure that any table that Q Replication might have created at site B will be replicated back to A. You must do this before you start the application to ensure that all changes are captured. The subscriptions for the reverse direction should be defined with HAS_LOADPHASE=N to ensure there is no load of data from B to A, since both sites are already loaded. (It would be harmless to reload the data, but potentially take a long time.) This option is in the IBMQREP_SUBS_PROF table.

## Best Practices -- Summary

Changes made by an application at run-time need to be handled differently from more controlled changes made during maintenance for upgrades. The reason is that changes made for an upgrade can affect the stability of the system and often impact several users and applications. Schema subscriptions must be used for applications that dynamically create/drop tables as part of their business logic.

Any change introduces risk, but risk can be mitigated by keeping an unchanged system available for fallback. Upgrading a second site can be postponed until the upgrade proves stable, by:

1. Suspending replication to the DR site until the upgraded system proves stable
2. Making any necessary replication configuration changes at the primary site
3. Restarting replication toward the DR site

In general, changes that affect the transaction replay replication process, such as altering columns in a table can be replicated, and changes that do not affect the transaction replay process are not replicated.

Schema changes introduced by application upgrades can be handled automatically by Q Replication with the following restrictions:

- New tables are created into existing tablespaces
- New tables do not have identity columns
- Table alters are limited to changing data types and adding columns

Ensuring the DR site is ready for failover after an upgrade may require the following actions, which might need to be accounted in the recovery time:

- Creating any object needed by the application that is not replicated, e.g., stored procedures
- Running a failover script to adjust sequence and identify columns RESTART values, if needed

Re-starting replication in the reverse direction after a failover requires that all newly created tables are replicated back, either by subscribing them explicitly or by starting and activating a schema subscription prior to the failover.

## References

1. Uni-directional schema subscriptions in DB2 for Linux, UNIX, and Windows 10.1
Setting up Q Replication schema subscriptions in an active/active scenario
Olaf Stephan and Christian Lenke. Published on May 16, 2013

2. The Value of Active-Active Sites with Q Replication for IBM DB2 for z/OS An Innovative IBM Client's Experience. An IBM Redpaper publication. Published 23 January 2015

3. Combining IBM DB2 pureScale with Q Replication for scalability and business continuity.
An IBM Best Practices paper. Updated April 17, 2013

4. How Q Capture handles DDL operations at the source database. IBM Knowledge Center.
https://www.ibm.com/support/knowledgecenter/SSTRGZ_11.4.0/com.ibm.swg.im.iis.repl.qrepl.doc/topics/iiyrqcapddlhandle.html