**Addressing SQL Server Replication Violation of Primary Key Errors**

By: [Tim Smith](https://www.mssqltips.com/sqlserverauthor/92/tim-smith/)

FROM: <https://www.mssqltips.com/sqlservertip/4509/addressing-sql-server-replication-violation-of-primary-key-errors/>

**Problem**

We're seeing the SQL Server error, "Violation of PRIMARY KEY constraint 'xxx'. Cannot insert duplicate key in object 'xxx'. The duplicate key value is (xxx)" in our transactional replication error log table (note - actual object names and values replaced with xxx) and are wondering how to troubleshoot this and what this means for SQL Server replication.

**Solution**

The above error indicates that the SQL Server primary key is being violated and we can simulate this by adding a primary key value to the subscriber table, then adding the same primary key value to the publisher table and then having replication replicate this row. Transactional replication requires a primary key field for the rows to be replicated. One of the main functions of transactional replication is high availability, so in most cases we want the subscriber data to mirror the publisher.

Fundamentally, this error generally indicates that the use of the subscriber table is incorrect, though there may be situations where we want the subscriber-level data to differ.

**Options t**o Address Violation of PRIMARY KEY Errors

**1. Union two tables**

One way that we can work around this issue, if we need a data set on the subscriber to have data that the publisher isn't supposed to have is to use a UNION ALL of the subscribed table and the additional table. For an example:

SELECT

OurPKIDField

, OurExampleDateField

, OurExampleAnalysisField

FROM tblOurSubscriberTable

UNION ALL

SELECT

OurPKIDField

, OurExampleDateField

, OurExampleAnalysisField

FROM tblOurAdditionalTable

With this query, we could have additional values that the subscriber table does not have (because the publisher does not) and any additions that we want to maintain in this manner would be added to the *tblOurAdditionalTable* table. However, these cases should be incredibly rare, if ever used. Even with ETL, we may combine data sets, but these combinations come from matched sources. An example use case for this design would be compliance with healthcare regulations at a Federal level with the additional tables being regulations on the State or Province level, where Federal regulations are replicated, while State or Province level regulations are on the database level for scale.

**2. Limit subscriber permissions**

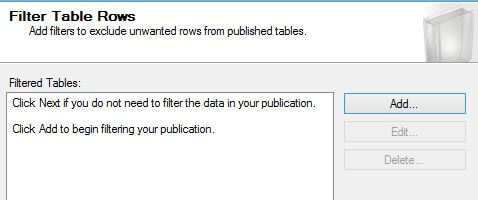
Since replication is primarily for high availability architecture (outside the scope of this tip, replication makes a poor quality ETL solution), we should consider restricting the subscriber tables to read access for developers if we want the publisher and subscriber tables to match. The replication agent user should have permission to write, with all other users in this case with read permissions. This eliminates the possibility that developers update the wrong table by forcing them to only have write access to the publisher tables. In these cases, I would recommend considering the high availability architecture of read-only replicas with availability groups, since these inherently prevent writing to the replicas.

**3. Use merge replication**

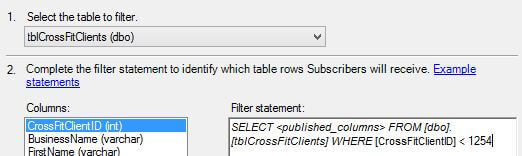
If we're in a situation where the subscriber will receive new data or updated data and we want the publisher to sync these changes too, instead of using transactional replication, we should use merge replication. Unlike transactional replication's log reader, merge replication uses a comparison between the tables on separate servers (while the term publisher and subscriber are used, functionally it's better to think of these as equals) and an update or insert to what someone might call the subscriber will be passed to the publisher and vice versa. By contrast with transactional replication, the log reader in simply passes new data to the subscriber, so if the subscriber already has it and carries the same constraints, we'll see constraint failures when the publisher tries to insert data that already exists. The best design for an either/or architecture where the subscriber or the publisher can have additions, changes or removals is merge replication, if replication is a design that is considered.

**4. Use filters**

We can filter data with transactional replication by selecting a subset of a table's data. If we're in a situation where we only want to replicate some primary key values, but have another set after a certain record in the publisher for the subscriber table, we can filter the primary key records that we want in the publisher table and add new values in the subscriber table without the publisher being affected. When we set up the transactional replication publication, we can select this filter in the set up; in the below images, I use a contrived example where I only replicate some CrossFitClientIDs to the subscriber table, and this CrossFitClientID field is the primary key. This means that replication would not receive this failure if the CrossFitClientID is above the IDs being replicated.



We will add a filter to our transactional replication when setting up the publication.



We then select our filter; in this case, it's the primary key column CrossFitClientID.

**Next Steps**

* Each of the above methods carry costs and benefits to their design and will definitely be favorable in some environments.
* In cases where the publisher should be the absolute source of truth, I would highly recommend restricting write access as that would be the simplest solution to this problem.
* In other situations where the subscriber tables should receive writes, we have alternative ways to handle these exceptional cases.
* Read more [SQL Server Replication tips](https://www.mssqltips.com/sql-server-tip-category/7/replication/)

Last Update: 10/28/2016