Migrating an Oracle database with a C# application attached to it: How to manage database migration?

Asked 15 years, 10 months ago Modified 11 years, 10 months ago

Viewed 2k times

3

I have a C# application which works with an Oracle database and has been already shipped. Now it's time to ship out a new release. The C# object model have been revised and had an impact on the table structure.

If I ship out the new release, I need to take care of existing data. Just dropping tables and recreate these tables wouldn’t make any customers happy.

To counter this problem I have collected SQL scripts, which alters the previously released database structure to the new database structure. In the course of this, the data are migrated too. The SQL scripts are committed to a repository like C# source code. The patching of the database is tested on regular basis with the help of CruiseControl.NET. NUnit tests are run against the patched database to uncover mismatches between database tables and C# object model.

The whole procedure does work, but I have the feeling that this could be done better. I regard database migration as very critical. A shipped application, which

doesn't work with a wrongly patched database, has no value. Losing data is inacceptable. These horror scenarios might make me think not to change the database at all. So it's very important for me to have full confidence in the tools and practices I use.

Last week I stumbled over LiquiBase and I asked myself -- and now in SO:

What tools or practices can help to do database migration with lesser risks and more confidence? Are there any good books or internet resources out there?

I am especially interested in specific solutions to C# and Oracle, which might fit in the development procedure I have outlined above.

c# database oracle-database migration

Share

Improve this question

Follow

edited Jan 31, 2009 at 17:05

Jon Skeet

1.5m ● 889 ● 9.3k ● 9.3k

asked Jan 31, 2009 at 17:01

Theo Lenndorff

4,596 ● 3 ● 29 ● 43

4 Answers

Sorted by: Highest score (default)

Database upgrade scripts must be part of development process. Here is one way of keeping track about database schema upgrades:

- create VERSION table in database that contains one record with version number
- each time you make change to database schema of your application you should:
    - create SQL script for creating, altering or dropping database objects
    - create SQL script for managing data changes that must be done with new data schema (e.g. insert defaults in new fields, insert default records in new tables, create script for splitting or merging tables, ...)
    - increment database version number
        - For each change I usually create one script named DbVerXXXX.SQL that contains all necessary upgrades (XXXX is version number). Also, I do changes in small steps - change DB schema only for next change you will do in your application. Don't create database upgrade that will take weeks or months of work to upgrade your application.
- create script that will upgrade your user's database to new version:
    - script should check current version of database and then execute database upgrade scripts that

will convert schema to required level

- change version number in VERSION table

This process enables you to:

- put all database schema changes under source control, so you have complete history of changes
- try and test your upgrade scripts on test databases, before you ship it to customer
- automatically upgrade user databases with confidence

Share Improve this answer Follow

edited Feb 1, 2009 at 17:15

answered Feb 1, 2009 at 16:53

zendar

13.6k ● 15 ● 61 ● 76

0

You may want to look into some of the database redundancy technologies available out there, such as Oracle Dataguard. I believe it in particular has some features that could help with this type of scenario.

Regardless of your technology however, anytime you have a schema change in the database you will obviously run some risk. One simple practice that you can always perform is to create a copy of the database, put it on a different server, and run your upgrade procedure there first to work through your bugs. When working with

customers in similar scenarios we've typically done that with them, just to alleviate their concerns and iron out any potential issues before performing the operation on the live environment.

Share Improve this answer

Follow

answered Jan 31, 2009 at 17:10

Matt Jordan

2,884 ● 1 ● 27 ● 28

0

To make very sure that you are not losing data when you alter a database, you can create scripts to confim the new structure and the old one contains the same logical data. For example, say version 1 of database looks like this (pseudo code)

```
CREATE TABLE Customer
CustomerID INT,
FirstName string,
Surname string,
AddressLine1 string,
AddressLine2 string,
AddressLine3 string,
AddressLine4 string
```

In version 2, you want to be able to allow customers to have more than one adress so you move the address fields into a new table:

```
CREATE TABLE Address
AddressID INT,
CustomerID INT,
AddressLine1 string,
AddressLine2 string,
```

```
AddressLine3 string,
AddressLine4 string
```

You move the addresses from Customer table into the new address table like this:

```
INSERT Address
CustomerID ,
AddressLine1 ,
AddressLine2 ,
AddressLine3 ,
AddressLine4

SELECT
*
FROM Customer
```

Then you remove the redundant address fields from Customer:

```
ALTER TABLE Customer
DROP COLUMNS
AddressLine1 ,
AddressLine2 ,
AddressLine3 ,
AddressLine4
```

So far so good. But how do I know that the new Address table contains the exact same addresses as the old Customer table. It would be very easy for the whole process to run and somehow scramble the address so that Customers effectively changed adresses with each other. The code could pass all tests, but we will have destroyed our clients data as they no longer know where their customers live.

We can confirm the move of address fields works by running

If this returns any records, the upgrade failed because some customers didn't get their address moved:

```
SELECT
 *
FROM

  OldCustomerTable  OCT LEFT JOIN Address A
  ON OCT.CustomerID = A.CustomerID
WHERE
  A.CustomerID IS NULL
```

If this returns any record, the upgrade failed because addresses were scrambled

```
SELECT
  *
FROM
 OldCustomerTable  OCT INNER JOIN Address A
  ON OCT.CustomerID = A.CustomerID
WHERE
  OCT.Address1 != A.Address1
  OR OCT.Address2 != A.Address2
  OR OCT.Address3 != A.Address3
```

OR OCT.Address4 != A.Address4

You can additionally check that the new address table only contains 1 address for each customer

```
SELECT
 CustomerID
 , COUNT(AddressID)
```

```
FROM
 Address
GROUP BY
 CustomerID
HAVING
 COUNT(AddressID) >1
```

Share Improve this answer

Follow

answered Jan 31, 2009 at 18:26

Noel Kennedy

12.3k ● 3 ● 43 ● 57

0

I fully agree with @Zendar that you must have proper versioning of your migration scripts, and that requires versioning metadata included in your database. Your scripts will modify your schema while updating the versioning information. This is how most applications update their database schemas.

That said, his process is a little too manual, and you might be interested in more automated tools to this end.

Check out:

- Autopatch
- The answers to this question
- The answers to this question

You could also consider the ideas in these tools and develop your own.

Share Improve this answer

edited May 23, 2017 at 12:19

Follow

Community Bot

1 ● 1

answered Feb 13, 2013 at 2:00

Alvaro Rodriguez

2,848 ● 3 ● 33 ● 40