The Role of a DBA on an Agile Project

ccpace

While the role of Application DBA ("App DBA") is not new for development teams, how the App DBA interacts with the rest of the team, and how and when the App DBA performs their tasks has been challenged by the advent of Agile development methodologies.

This white paper discusses the role of the App DBA on Agile projects and demonstrates how we at CC Pace incorporate App DBA tasks into the Agile development process. The scope is limited to a project using a relational database. And while some of the terminology used might be more associated with Oracle, the concepts apply to relational databases in general.

Intended Audience

This document is aimed at:

- App DBAs who may be starting their first Agile project, or who may want to contrast and compare their existing Agile techniques with alternate methods
- Data team managers who are responsible for providing development teams with database staff
- Team leads, Agile coaches, ScrumMasters and Project managers who are responsible for a smooth-running development team

What is an Application DBA?

The role of the App DBA on an Agile project is not that much different from that required by other development methodologies in terms of tasks and responsibilities. What is different is how and when these tasks and responsibilities are applied in the development process.

An Agile project is likely to require some or all of the following from an App DBA:

- Data modeling
- Creating the other database objects, such as tables, and maintaining the database build script
- Developing stored procedures, stored functions, triggers and views
- Writing and tuning queries
- Establishing database development standards for the other developers
- Reviewing database code and queries to make sure standards are met
- Building sandbox, integration and test database environments for developers and testers

- Creating database user accounts and granting appropriate privileges to those accounts
- Supporting the developers and testers and making sure that there are no impediments to the development team from the perspective of the database

How has Agile changed the role of the Application DBA?

- The App DBA has to be fully integrated into the development team. A core of Agile development is communication, and preferably co-location. No longer can the App DBA do their work from the "DBA Cave". With a smooth-running Agile team, a development task that involves the database may turn a programming pair into a programming trio. In addition, the App DBA needs to be available to ensure database problems do not stand in the way of progress.

- Data modeling occurs throughout the development process. While data modeling is not necessarily the responsibility of App DBAs in some organizations, the App DBA on an Agile project will be hard-pressed to avoid it. Data models not only change from release-to-release, but from sprint-to-sprint and story-to-story. The App DBA will have to be able to make different structural changes to different developer databases at the same time, and at any time during the development process.

- Additional development and testing databases are required. Automated unit testing and continuous integration, key engineering components of the Agile development process, work best with individual developer/tester databases, or sandboxes. Ironically, multiple sandboxes with a well-written database build script are significantly easier for an App DBA to maintain than is a database shared by developers.

- Database code is application code. Every component of the application's database – DDL, stored procedures, reference data, etc. should be stored in the same source code repository as the rest of the application code. A stored procedure being checked-in should cause the same continuous integration process to run as would the check-in of any of the application code.

  As an aside, many database development tasks are done in tandem with a developer or developer pair. For example, a new stored procedure might be required in order for the pair to complete their task. In these cases, it is best if all related files are checked-in together by the developer.

Continuous Integration and Building the Database

If your project uses continuous integration, the integration server is likely tearing down, rebuilding and redeploying the application with each source code check-in. The database objects should be included in these steps.

With each check-in, tear down and rebuild as many of the application's database objects as is feasible. In many cases, this could be all of the objects including tables, indexes, stored procedures, grants, synonyms, etc. It can, but does not have to include some or all of the database roles and users, but it should not include data files and tablespaces and other infrastructure-type items.

It is typically important to drop all of the application schemas before rebuilding any of them. If schema A is dropped and rebuilt before dropping schema B, and A has dependencies with B, then the build may appear clean, but have underlying problems.

Dropping tables with large amounts of data is not always practical, especially if you want your build to complete in a reasonable amount of time. However, it might be worth reviewing why such a large amount of data exists in the integration environment or developer sandbox.
Ask yourself,

- How much data is really required to test the functionality of the application?
- Can the automated tests setup and teardown only that data which is required by the tests, eliminating the need to have a pre-populated table?

After dropping all of the objects, but before running tests, rebuild the objects. This could include loading reference and common test data.

Developer and Tester Sandboxes

Each developer (or developer pair) and tester should have their own database environment, or sandbox. In addition, the continuous integration ("CI") server should have its own environment. Having a personal environment allows a developer to make database changes without affecting anyone else on the team, until they are ready to check-in and make the changes known. These personal sandboxes are an essential part of automated unit testing in a continuous integration environment.

Individual database components have the following advantages:

- Reduced dependency between development team members
- Less maintenance for the development DBA
- Ease of switching from one version of the database to another

The individual sandboxes can be set up as separate schemas within the same database or separate database instances. Each approach has its advantages and disadvantages.

	Advantage	Disadvantage
Separate Schemas	Most if not all database tasks can be handled exclusively by the App DBA or development team.	Have to account for schema name variations in build scripts and during other database operations. For example, instead of having an app_owner schema in each developer instance, there might be schemas app_owner01, app_owner02, etc. in a shared instance.
Separate Instances	Sandboxes more closely resemble the production environment in terms of schema and user names.	Typically requires the help of an operational DBA to setup environments and perform some day-to-day tasks. Likely requires more powerful (or just more) hardware.

Each sandbox should contain the minimum amount of data required for development or testing. Ideally, the data is loaded (and removed) as part of unit test setup and teardown. In some cases, it may be practical to load some of the data as part of a database build process.

Application Schemas

Every application is different, and different organizations have varying standards and policies regarding database access, but the following schemas are a common starting point for most Agile projects:

- Schema owner – owner of the application's objects (ex. tables, views, stored procedures)
- Application user – the application logs into the database as this user who has been granted minimal privileges to execute stored procedures or run queries. This user owns no objects other than synonyms.
- Test user – Used by the automated unit test framework to setup and teardown test data. This user may have more privileges than the application user (ex. delete privilege), but less privileges than the schema owner. This user owns no database objects other than synonyms.

And in conclusion…

Adopting some of the practices outlined in this paper may mean a shift in deep-seated habits and way of thinking, but the demands of an Agile development team and an Agile-inspired product owner require such a shift. The development DBA can bring unique skills to many project teams, and indispensible skills to some project teams, and the goal should be to help move the team forward by applying those skills and clearing database-related obstacles.