

## Agility and the Database

Peter Schuh  
Quoin Inc.

*peter.schuh@quoininc.com*

## Agility and the Database

### Part One

1. The Agile Data Movement
2. The Agile DBA
3. Complexity and the Database
4. Creating and Managing Test Data
5. Tools and Automation

2

## Agility and the Database

### The Agile Data Movement

3

*Agile.Data.Movement://*

What does it relate to?

- All data sources, regardless of type:
  - Relational databases, flat files, XML, and object databases
- Applications that:
  - Are in development
  - Store data
  - Are not simple

4

*Agile.Data.Movement://*

What is it?

- A growing reluctance to accept that:
  1. Managing data has to be a chore
  2. Conflict must exist between DBAs and developers
  3. The database (or data source) can and should be treated as a black box by the development team

5

*Agile.Data.Movement://*

What is it?

- A gradual acceptance that:
  1. Data is at the heart of most mission-critical enterprise systems
  2. Data generally is not owned by any single application
  3. Regardless of whether it is relational, serialized objects, XML or flat files, it's all still data
  4. Teams that learn to leverage their data infrastructures will outperform those that do not

6

**Agile.Data.Movement://****What are its goals?**

1. Make managing the database and data sources as straight-forward as managing code
2. Make database development an activity that is owned by the development team instead of the responsibility of some guy who works on another floor or in another building

7

**The Setting://****More goals**

3. Allow team members to code and test against real data without impacting the work of others
4. Make the database a platform that can be leveraged to encourage agile processes such as:
  - Incremental design
  - Continuous integration
  - Small releases
  - Unit and functional testing
  - Refactoring

8

**The Setting://****Even more goals**

5. Advocate a framework for database development and management that may be adapted and applied to most situations.

9

**Agile.Data.Movement://****Who is behind it?**

Pramod Sadalage & Martin Fowler  
*ThoughtWorks Inc.*  
[agileDatabases@yahoogroups.com](mailto:agileDatabases@yahoogroups.com)

Scott Ambler  
*Ronin International*  
[www.agiledata.org](http://www.agiledata.org)

Peter Schuh  
*Quoin Inc.*  
[www.peterschuh.com/it](http://www.peterschuh.com/it)

10

**Agility and the Database****The Agile DBA**

11

**The.Agile.DBA://****Database-related Roles**

- Production DBA
- Database Developer
- Development DBA
- Agile DBA

12

The.Agile.DBA://  
**Conventional Roles**

- Production DBA
  - Tunes SQL and adds indexes
  - Writes procedures and triggers
  - Very concerned with uptime, data integrity and security
  - Often does not see things from the developer's point of view

13

The.Agile.DBA://  
**Conventional Roles**

- Database Developer
  - Writes SQL, procedures, triggers, reports, data conversions, etc.
  - Thinks from more of a development mindset
  - Tends not to appreciate the complexities of multi-tier development

14

The.Agile.DBA://  
**Conventional Roles**

- Development DBA
  - Builds and maintains a development database for one or more teams
  - Creates tables, constraints and other metadata
  - Assists with queries, procedures, etc.
  - May be spread across multiple teams
  - May be everything a development team requires

15

The.Agile.DBA://  
**New Role**

- Agile DBA
  - Does everything required of a development DBA
  - Is a full-time member of the team, and, therefore:
    - Is forward-thinking about the needs of the team
    - Speeds development time by automating processes
    - Empowers team members by pushing out tools

16

The.Agile.DBA://  
**Ease of Development**

■ Production DBA	Torturous
■ Database Developer	Difficult
■ Development DBA	Adequate
■ Agile DBA	Asset

17

The.Agile.DBA://  
**Who is the Agile DBA?**

- Must be a full-fledged member of the team
- Could be a developer or analyst
- Needs to know about production database issues
- Must have a good understanding of the software development process in order to quickly respond to the changing needs of the team

18

The.Agile.DBA://  
**What the Agile DBA Does**

- Responds quickly to the needs of the team by:
  - Defining, simplifying and automating processes
  - Empowering team members by developing and pushing out tools
- Makes the database more agile by:
  - Decreasing response times
  - Making changes easier to perform

19

The.Agile.DBA://  
**Agile DBA - Buildmaster**

- The DBA on an agile project could be:
  1. A designated team member
  2. A "hat" that is passed around between team members
  3. A toolset

20

**Agility and the Database**

Complexity and the Database

(The Two-Dimensional Database)

21

The.Two.Dimensional.Database://  
**The Development Timeline**

- Application development follows a one-dimensional timeline

22

The.Two.Dimensional.Database://  
**The Development Timeline**

- Database schema and setup data progress on a similar path

23

The.Two.Dimensional.Database://  
**The Development Timeline**

- For both code and the database, a build may exist at many points ...

... on this timeline at once

24

The.Two.Dimensional.Database://  
The Development Timeline

- Other complexities, like branching, can also be relatively easily understood ...

... and maintained with tools and processes that are currently available

25

The.Two.Dimensional.Database://  
What Makes it Complicated

- Other data (such as test and deployment datasets that sit on top of the schema and setup data) add another dimension to the database

26

The.Two.Dimensional.Database://  
What Makes it Complicated

- In this representation a point in time (with no context) can represent no less than a line

- Knowing a point in time is no longer enough information to locate a database

27

The.Two.Dimensional.Database://  
What Makes it Complicated

- By adding context, we are able to capture the intricacies inherent in the development of application databases

28

The.Two.Dimensional.Database://  
What Makes it Complicated

- We use the term **lineage** to refer to a specific dataset as it evolves over time
- The concept is similar to (but not the same as) a code branch

29

The.Two.Dimensional.Database://  
What Makes it Complicated

- Introducing a code branch shows how database management can become more complex than managing builds

30

**The.Two.Dimensional.Database:// Complexity**

- And who says the fun has to stop there?

31

**The.Two.Dimensional.Database:// Solutions**  
**How to reign in complexity**

Track the database by lineage

- Every lineage represents a unique context of the database instance

You need to know where you are in order to know where you want to go

32

**The.Two.Dimensional.Database:// Solutions**  
**New lineages should ...**

be created when

- A portion of the team (QA, reporting, etc) requires a unique and maintainable dataset
- The application is branched not be created to for non-essential reasons
- example: to maintain a unit-test suite that fails to work on the latest load of converted data

33

**The.Two.Dimensional.Database:// Solutions**  
**Managing lineages**

Changes to each lineage must be tracked in the same manner as changes to the application code

```
--story card 12
--2002.08.01
CREATE TABLE customer
(
  code          VARCHAR2(10) NOT NULL,
  name         VARCHAR2(40) NOT NULL,
  customer     VARCHAR2(40) NOT NULL,
  lastorder    DATE NOT NULL,
  website     VARCHAR2(100) NOT NULL,
  status      VARCHAR2(5) NOT NULL,
  CONSTRAINT pk_customer PRIMARY KEY (code))
TABLESPACE userables;

--story card 13
--2002.08.01
ALTER TABLE address ADD countrycode VARCHAR2(3);
UPDATE address SET countrycode = 'USA';
```

34

**The.Two.Dimensional.Database:// Solutions**  
**Tracking database changes**

Allows the team to:

- Pull up a previous version of any database lineage at any time
  - Like rolling back to a previous build
  - Very useful for debugging old releases
- Roll a database forward
  - To migrate old data forward for regression testing
  - To pull data out of production so that developers can code against the freshest data

*To be discussed further later in the presentation*

35

**Agility and the Database**

Complexity and the Database

(The Tragedy of the Commons)

36

Tragedy of the Commons://  
**A Commons is a resource that ...**

- Is accessible to all the members of a particular group
- May benefit all members of the group
- If not managed, may degrade from overuse or misuse
- Can, through overuse or misuse, be spoilt and become no benefit to anyone

*Conceived by Garrett Hardin (1968)*

37

Tragedy of the Commons://  
**Commons currently under threat**

- International waters (for fishing)
- Equity markets (for investing)
- Enterprise databases (for developing, testing and running business applications)

38

Tragedy of the Commons://  
**THE TOTC Scenario**

What Makes it Possible?

- One data source, many applications
- One application, many data sources
- Many applications, many data sources

*In Other Words: Complexity*

39

Tragedy of the Commons://  
**Symptoms of a TOTC scenario**

1. A new application is to be released into the system, and no one can say if or how it will effect existing apps
2. Developers of one app learn about another team's changes to the database only after their app breaks
3. The process of performing an impact assessment becomes a project in and of itself

40

Tragedy of the Commons://  
**TOTC on an enterprise database**

What are the causes?

1. No system integration environment
2. No automated regression testing
3. Poor test data management
4. Failure to regulate and manage access and changes to the database

**NOT A CAUSE: Too many applications accessing the database**

41

Tragedy of the Commons:// **Solutions**  
**How to avoid a TOTC scenario**

- Provide separate development environments at the team- or developer-level \*
- Rationalize test data management \*
- Mandate automated testing suites that can be reused for regression and integration testing

*\*Will be discussed later in the presentation*

42

**Tragedy of the Commons:// Solutions****How to address a TOTC scenario**

Define and enforce a System Integration "tollgate" that:

1. Tests whether new releases adversely affect existing apps
2. Is separate from any UAT process
3. Requires that new functional releases contribute tests to the SI environment

*This process MUST be automated*

43

**Agility and the Database****Creating and Managing Test Data**

44

**Test Data://****Standard Data Creation Methods**

1. Using whatever data is at hand
  - Grab it from the database
  - Pros
    - It's easy
  - Cons
    - Anyone can alter or delete your data
    - Tests are dependent upon a specific dataset
    - Difficult to alter the dataset to meet changes made to the application

45

**Test Data://****Standard Data Creation Methods**

2. Ad hoc test data instantiation
  - Build it and insert it into the database
  - Pros
    - Completely independent of the database and any other testing activity
  - Cons
    - Instantiation logic is spread throughout the tests, making them difficult to maintain
    - Can become a serious burden to build complex objects

46

**Test Data://****Standard Data Creation Methods**

3. Restorable data sets
  - Load, test, reload
  - Pros
    - Objects do not need to be instantiated or maintained in the code
    - Easy to recreate a clean test environment
  - Cons
    - Tests may fight over the same data
    - Test suite is not portable
    - Data set may be cumbersome to maintain

47

**Test Data://****Making Test Data Agile**

1. Test data should be maintained in a way that forces it to be kept in sync with the database
2. Test data should be easy to obtain, so that developers have one less reason not to test
3. Test data should not be accessible by multiple tests, ensuring that one test's operation will not interfere with another

48

Test Data://

## Recommendations

1. ObjectMother
2. Mock Objects

49

Test Data:// ObjectMother

## What is ObjectMother?

A utility class or framework of classes that:

1. Provides a fully-formed business object along with all of its required attribute objects
2. Returns the requested object at any point in its lifecycle
3. Facilitates customization of the delivered object

50

Test Data:// ObjectMother

## What is ObjectMother?

A utility class or framework of classes that:

4. Allows for updating of the object during the testing process
5. When required, terminates the object and all its related objects at the end of the test process

51

Test Data:// ObjectMother

## Example Test

```
public void testInvoiceTotal() throws Exception {
    Invoice invoice =
        ObjectMother.createNewInvoice();
    ObjectMother.attachInvoiceLineAsCharge(
        invoice,
        new Money("199.95", "USD"));
    ObjectMother.attachInvoiceLineAsCharge(
        invoice,
        new Money("100", "USD"));
    assertEquals("Invoice did not total correctly",
        invoice.getTotal(),
        new Money("299.95", "USD"));

    ObjectMother.purge(); //Method call is optional
}
```

52

Test Data:// Mock Objects

## What are Mock Objects?

A class or set of classes that:

- Mimic application interfaces or objects
- Are passed in as parameters and test by monitoring the behavior (method calls) of other objects
- Removes the need to create test data (or delete it)

53

Test Data:// ObjectMother

## Example Test

```
public void testFileSystemFailure() {
    myMockServer.setFailure(FILE_SYSTEM_FAILURE);
    myApplication.connectTo(myMockServer);
    try {
        myApplication.doSomething();
        fail("App server should have failed");
    } catch (ServerFailedException e) {
        assert(true);
    }
    myMockServer.verify();
}
```

Source: *Endo-Testing: Unit Testing with Mock Objects (2000)*

54

## Agility and the Database

### Tools and Automation

55

Tools.and.Automation://  
What Can be Automated?

- Building application databases
- Copying databases
- Setup data loads and maintenance
- Data migration
- Database refactoring
- Testing activities

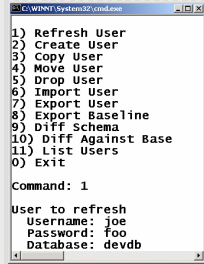
56

Tools.and.Automation://  
Process Automation

- The DBA should attempt to automate any process he finds himself doing repeatedly
- Process automation is absolutely, utterly essential if the DBA is to respond to a myriad of requests in a timely manner

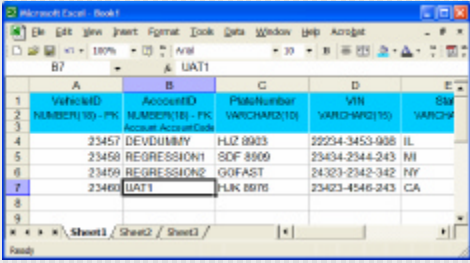
57

Tools.and.Automation://  
Example: Database Manager



58

Tools.and.Automation://  
Examples: Data Population



59

Tools.and.Automation://  
Pushing out Tools

- Whenever possible, tools used by the DBA should be simplified and pushed out to team members
- It is better to give too much access than not enough

60

## Agility and the Database

### Break

[www.otug.org](http://www.otug.org)  
[otug\\_twincities@yahoo.com](mailto:otug_twincities@yahoo.com)  
[agile\\_experience\\_group@yahoo.com](mailto:agile_experience_group@yahoo.com)  
[otug\\_java\\_sig@yahoo.com](mailto:otug_java_sig@yahoo.com)

61

## Agility and the Database

### Part Two

## The Agile Database Framework

62

Agile.Database.Framework:// Introduction

### The Traditional Database

- Continuous changes to the data model or the database are not allowed
- Production changes are rare or not allowed
- Migration is a project
- There is one Development / QA / UAT / Production database

63

Agile.Database.Framework:// Introduction

### The Agile Database

- Slowly changing database
- Refactoring is possible even when applications are in production
- Developers and the DBA collaborate to build the database
- Isolated, fully-functional copies of the database are freely available

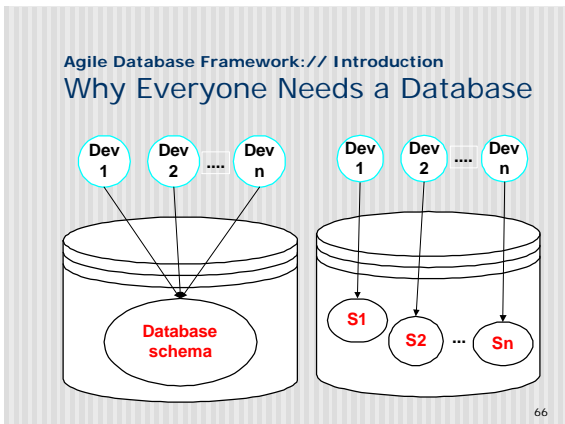
64

Agile Database Framework:// Introduction

### Everyone Gets a Database

- A database instance is the same as an application instance
- Developers may make changes to a database as if it were a scratch pad
- Any developer should be able to create a clean copy of the database, or "refresh" an existing copy

65



**Agile.Database.Framework://**  
**The Database Instance**

- The database analog to the application instance
- A data space containing, at the very least, a copy of all the schema and setup data necessary to support an application instance
- A data space that is independent—not affected by nor able to affect the space of another database instance

67

**Agile.Database.Framework:// The Database Instance Examples**

The diagram illustrates four examples of database instances. Oracle Database is shown as a cylinder containing two 'User' icons. Microsoft Access is shown as a folder containing three 'File' icons. Sybase Server is shown as a container with two 'DB' icons. XML is shown as a folder containing three 'File' icons.

68

**Agile.Database.Framework:// The Database Instance**  
**Why does the DBA need it?**

- Easier to track and maintain specialized datasets
  - for development, automated testing, demos, production copies, etc.
- Easier to communicate the specifics of a known or needed version of the database
  - Allows us to think of the database as another component in the system, with a build number and a context

69

**Agile.Database.Framework:// The Database Instance**  
**Is Statefull and Locatable**

- E.g. iteration 2, regression test database

The diagram shows a 3D perspective of a database instance. The vertical axis is labeled 'Context', the horizontal axis is 'Time', and the depth axis is 'State'. A black dot on the surface represents a specific state at a specific time and context. Arrows indicate the direction of time and context.

70

**Agile.Database.Framework:// The Database Instance**  
**Why Would the Team Want It?**

- To allow every developer the ability to code and test with his or her own, independent database
  - Would you have an entire team develop all using the same single application instance?
- To make the database portable and restorable
  - A developer can code and test against a specific set of data. If the developer breaks the dataset, no one else is affected and the developer simply loads up a fresh copy.

71

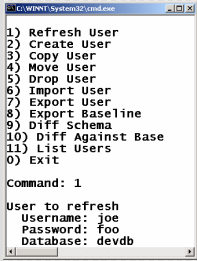
**Agile.Database.Framework://**  
**The Database as File System**

The instance analogy reduces the database to the status of an object, allowing us to think of it and treat it as though it were merely a file.

Development Server	QA Server	Demo Server
<ul style="list-style-type: none"> <li>Mike</li> <li>Jay</li> <li>Bob</li> <li>Jill</li> <li>Card 73 Test Case</li> <li>Card 75 Test Case</li> <li>Conversion</li> </ul>	<ul style="list-style-type: none"> <li>Mark</li> <li>Joe</li> <li>Iteration 1 Regression</li> <li>Iteration 2 Regression</li> <li>Card 73 Test Case</li> <li>Conversion</li> </ul>	<ul style="list-style-type: none"> <li>Acceptance Tests 1</li> <li>Executive Demo</li> <li>Third Party Demo</li> </ul>

72

**Agile.Database.Framework://**  
**The Database as File System**



```

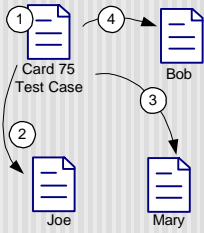
1) Refresh User
2) Create User
3) Copy User
4) Move User
5) Drop User
6) Import User
7) Export User
8) Export Baseline
9) Diff Schema
10) Diff Against Base
11) List Users
0) Exit
Command: 1
User to refresh
Username: joe
Password: foo
Database: devdb
    
```

Commands such as create, copy, move and delete can be made applicable to the database instance.

73

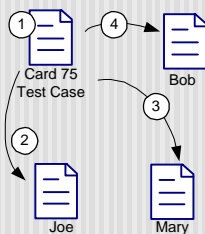
**Agile.Database.Framework:// Database File System**  
**"File Sharing" Example**

- Mary creates a new database instance and, using the application, creates the setup data for the story card 75 test script.
- Joe codes card 75, then loads the test case into his personal instance, runs the test and finds a bug. He fixes the bug, reloads the test case data and runs the test again.



74

**Agile.Database.Framework:// Database File System**  
**"File Sharing" Example**



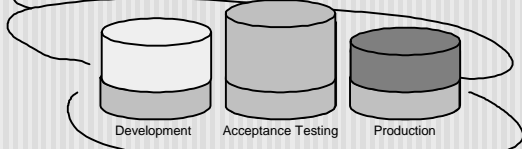
- When Joe tells her he has completed the card, Mary loads the test case into her personal instance and runs the test script. When that succeeds, she reloads the test case and does some ad hoc testing.
- Two iterations later, Bob loads a copy of the test case into his personal instance and runs the test script as part of regression testing.

75

**Agile.Database.Framework://**  
**The Bifurcated Database**

In order to manage all these database instances easily, we need to split the application database into two distinct portions.

Use-specific data is always different

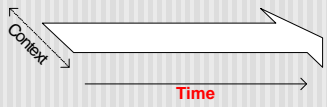


Schema and setup data is always the same

76

**Agile.Database.Framework:// Bifurcated Database**  
**Schema and Setup Data**

- This portion of the database instance changes in one dimension (time) and in tandem with application code
- Every alteration to the schema and setup data must be logged and source-controlled with the application code



77

**Agile.Database.Framework:// Bifurcated Database**  
**Schema and Setup Data**

- A fully-functional (and clean) database instance can be built at any time to any point in the development timeline (similar to the rolling back of a code base)
- Old database instances can be rolled forward by applying the logged changes made to schema and setup data

78

**Agile.Database.Framework:// Bifurcated Database Use-Specific Datasets (Lineages)**

- This portion of the database is what makes one instance different from another (context) while at the same point in development (time)
- The variety of datasets required by the team for its daily activities but not explicitly required for the application to operate

79

**Agile.Database.Framework:// Bifurcated Database Use-Specific Datasets (Lineages)**

- These must be typed and tracked
  - Examples: development, automated testing and production
- Because context is discrete
  - It is more difficult to track changes on this portion of the database instance
  - Database instances cannot generally be migrated from one lineage to another

80

**Agile.Database.Framework:// Lineage The Lineage and Its Attributes**

Every lineage has:

- A master instance
- A change log
- An update list
- A collection of child instances

81

**Agile.Database.Framework:// Lineage The Master Instance**

Every lineage has a master upon which it is based

The master instance:

- Is the gold standard for that lineage
- Is updated every time an update is made to the lineage
- Is the mold from which all new copies of that instance are made
- Should be updated only by designated members of the team

82

**Agile.Database.Framework:// Lineage The Master Instance**

You can have more than one master instance for a lineage

*Example: A QA team that takes a build only at the end of each iteration*

But a lineage will have only one master instance for any given point in time

83

**Agile.Database.Framework:// Lineage The Change Log**

As previously discussed

```

--story card 12
--2002.08.01
CREATE TABLE customer
(
  code          VARCHAR2(10) NOT NULL,
  name          VARCHAR2(40) NOT NULL,
  surname      VARCHAR2(40) NOT NULL,
  lastorder    DATE NOT NULL,
  website      VARCHAR2(255) NOT NULL,
  status       VARCHAR2(5) NOT NULL,
  CONSTRAINT pk_customer PRIMARY KEY (code)
)
TABLESPACE user_tables;

--story card 13
--2002.08.01
ALTER TABLE address ADD countrycode VARCHAR2(3);
CREATE address SET countrycode = 'USA';
    
```

84

Agile.Database.Framework:// Lineage

## The Change Log

A stored collection of all changes made to the lineage master and its child instances.

The change log:

- Will likely be written in SQL
- Should be source controlled
- Is the tool by which instances may be rolled forward

85

Agile.Database.Framework:// Lineage

## The Change Log [an exception]

A lineage with a master that cannot be updated via some scripted means will likely not have a change log.

For example:

- A development database that is regularly loaded with updated production data
- A development database that is based on converted data

86

Agile.Database.Framework:// Lineage/ Change Log

## Handling Destructive Actions

Actions such as ...

**Column renames**  
**Table deletions**  
**Data Migrations**

... must be tracked and handled in a special manner in order to keep everyone developing smoothly

87

Agile.Database.Framework:// Lineage/ Change Log

## Destructive Actions / Clean-up Log

Whenever possible, use a clean up log

- Destructive actions are entered into the clean-up log with a run date that is several days in the future
- Delays destructive actions by several days
- Destructive changes are propagated only after all team members have been given a chance to move to the latest code

88

Agile.Database.Framework:// Lineage/ Change Log

## Handling Destructive Actions

When a clean-up log will not work ...

**Such as complex data migrations**

... either

1. Run the change during an iteration switch
2. Force everyone to migrate to the latest build at a designated time

89

Agile.Database.Framework:// Lineage

## The Update List

The means by which database instances are kept up to date ...

- While the data model evolves
- As developers and other team members actively code and test against their individual database instances

90

Agile.Database.Framework:// Lineage  
**The Update List**

Communicates changes to the lineage's master and all its child instances

Each master instance should have an update list

Database instances can subscribe to the update lists that belong to their lineage

91

Agile.Database.Framework:// Lineage  
**The Update List [an example]**

```
connect devmaster/foo@devdb
run c:\dbscripts\changes.sql
connect mary/jane@devdb
run c:\dbscripts\changes.sql
connect build/machine@devdb
run c:\dbscripts\changes.sql
connect bob/rick@testdb
run c:\dbscripts\changes.sql
```

92

Agile.Database.Framework:// Lineage  
**The Child Instance**

Any instance that was, initially, derived from a particular lineage

```
connect devmaster/foo@devdb ← Master Instance
run c:\dbscripts\changes.sql
connect mary/jane@devdb ← Child Instances
run c:\dbscripts\changes.sql
connect build/machine@devdb ← Child Instances
run c:\dbscripts\changes.sql
connect bob/rick@testdb ← Child Instances
run c:\dbscripts\changes.sql
```

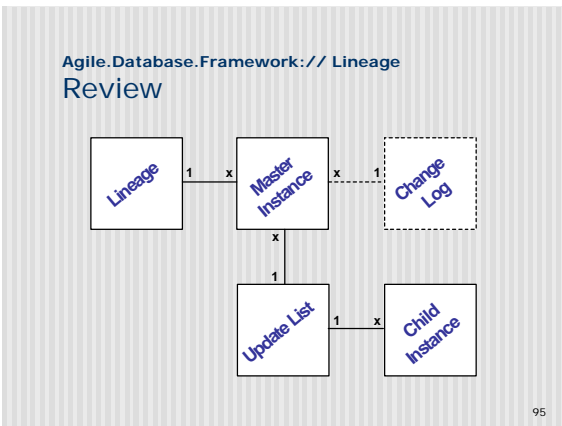
93

Agile.Database.Framework:// Lineage  
**The Child Instance**

The instance may or may not be subscribed to one of the lineage's update lists

If the instance is out of date (not subscribed to an update list) it may be brought up to date by applying the master instance's change log

94



Agile.Database.Framework://  
**The Rules and Culture**

- Allow Developers to Experiment
- Function-Driven Database Design
- Clean Tests
- People
- Infrastructure

96

Agile.Database.Framework:// Rules and Culture  
 Allow Developers To Experiment

- Developers can change the model in their local schema so that they can rollback if the change they made does not work efficiently.
- The DBA should get involved in the design sessions for the story card being developed.
- The DBA should help the developers to design the tables.

97

Agile.Database.Framework:// Rules and Culture  
 Function-Driven Database Design

Use functionality or change requests to grow the database

- How all modifications to the database should be made
- Pertains to schema, setup data and even lineage-specific data

98

Agile.Database.Framework:// Rules and Culture  
 Function-Driven Database Design

A change request will contain:

- The schema alterations requested
- Descriptions for all new tables and columns
- Default values and data migration details

99

Agile.Database.Framework:// Rules and Culture  
 Function-Driven Database Design

The change request process:

1. Developers submit change requests
  - When starting work on a new piece of functionality
2. DBA reviews all change requests
  - Speaks for other parties (e.g. conversion and reporting)
  - Delays destructive actions
  - Keep a log of all change requests

100

Agile.Database.Framework:// Rules and Culture  
 Clean Tests

Clean Tests Do Not:

- Rely on preexistent data in the database
- Leave remnants behind in the database

Why Clean Tests:

- Application and database code is tested, automatically
- Setup data can be maintained and easily altered or updated
- Production data can be used by the developer to test against ... copies off course

101

Agile.Database.Framework:// Rules and Culture  
 People

- Team collaboration is essential
  - Developers, QA, Analysts and DBAs need to be on board
- Project management must understand and think Agile
  - Encourage experimentation and tolerate failures
  - Accept that some information will not be known in advance

102

Agile.Database.Framework:// Rules and Culture  
**Infrastructure**

- Hardware to support one or more database instance per person is necessary
- It should not take more than 30 minutes to copy a database instance
- If necessary, work with slices of data

103

Agile.Database.Framework://  
**Production**

Handling the application in production is surprisingly simply because:

- The database is being developed, tracked and sourced with the same rigor as code
- The Change Logs *are* your migration scripts

104

Agile.Database.Framework:// Production  
**First Time Deployment**

Should be handled almost exactly the same as deploying code

- Archive a copy of the production instance
- Branch if necessary

The production instance will be either

- A clean copy of the application database
- Chock full of converted data

105

Agile.Database.Framework:// Production  
**Additional Releases**

All of the change logs should be applied following the last production migration

Since data migration scripts are part of change logs, data is also migrated

It is wise to test the migration on a test database

106

Agile.Database.Framework:// Production  
**Additional Releases**

107

Agile.Database.Framework:// Production  
**Thinking Forward**

Production database instances can be migrated to a future state of development

New releases can be acceptance tested against the most recent data currently in production

108

**Agile.Database.Framework:// Production**  
**Production Is Still Production**

The Agile Database Framework  
relates to production only to the  
extent that development activities  
touch upon production

109

**Agility and the Database**

Questions

110