Refactoring Databases: Evolutionary Database Design

Pramod Sadalage (沙朴木)
ThoughtWorks Inc.

Evolutionary Database Development

- Reality is “Businesses are Changing”
- Change management for database assets
- Learn when you fail, so make it easy to fail
- Functionality added in increments
- Facilitates automated testing
- Knowledge of the functionality
- Acknowledge team interaction
- DBA = Role != Person
Version Control Database Assets

- Greater control over changes
- Couples database and application
- Integrate in version control instead of database

Swap Best Practices

- Educate DBA about coding practices
- Nothing is used only once
- Educate Developers write better SQL
- How to make the DBA redundant
- Automate tasks such as
  - Physical table deployment
  - Usage statistics
  - Schema verification
  - Data migration verification
Let the DBA and Developers pair

- Better to have “Generalizing Specialists”
- Helps DBA understand the application
- DBA has a better understanding of other areas of the business data
- Write database tests
- Migration of production data is critical
- Gain knowledge of SQL Tuning etc
- Make the team aware of production data
- Understand performance implications

Give everyone a sandbox

![Diagram of Developer schema](Image)
Why?

- One Application instance = One Database instance
- Developers work independent of other developers
- Liberty of experimenting with the database
- Use ANT/Make/Rake to automate tasks
- Database could just be a Schema
- Reduces contention
- Integrate in version control not in database

Ultimate Nirvana
Demo: Sandbox

- Shows how new person joining project can startup effortlessly
- Shows reduction in waste (Lean)
- Shows how to share DBA tasks
- Shows how the DBA will get to do more
- Shows productivity gains by automation
- Shows ANT targets to Create, Clean, Initialize(populate) and Drop databases
- Shows how changes are local

Typical Integration Environment

Integrate more than once a day
http://martinfowler.com/articles/continuousIntegration.html
Details

Check in Application Code and Database Delta Scripts

Local dev environment

DB delta scripts

ANT
Maven
Make

Dev DB

Central integration environment

Source Control

Check out all and build

CRUISE Database

Apply DB delta scripts

Continuous Integration Engine

Artifacts

DB delta scripts

PROD UAT QA Environment

PROD UAT QA Environment

Copyright 2007 Pramod Sadalage. All rights reserved. Do not copy or distribute without permission.

Demo: Integration

- Shows DBA/Developer collaboration
- Allows to experiment with design options
- Shows DBA involvement in the design of the functionality being developed
- Shows developers changing the model in their local schema
- Shows how migration scripts gets tested
- Shows integration in version control

Copyright 2007 Pramod Sadalage. All rights reserved. Do not copy or distribute without permission.
Database Refactoring

- A database refactoring is a simple change to a database schema that improves its design while retaining both its *behavioral* and *informational* semantics.
- A database refactoring can include both structural aspects such as table and view definitions as well as functional aspects such as stored procedures and triggers.

[www.databaserefactoring.com](http://www.databaserefactoring.com)

---

Evolutionary Design

- Employee (Table)
  - S_Employee (Sequence)
  - FK Constraint (Manager)
- Index FirstName
- Comments
- HasAccessToCustomer
- TerminationDate
- BirthDate
- CHK Hiredate < TerminationDate

- AccountType (Table)
  - S_AccountType (Sequence)
  - Data (Production)
- Index Name

- Customer (Table)
  - S_Customer (Sequence)
- Index LastName
- Comments
- CustomerTaxId
- TaxLocation
- CreationDate
- FK CustomerType

- Account (Table)
  - S_Account (Sequence)
  - Comments
  - FK Constraint (Customer)
  - FK AccountType

- CustOpenAcct (View)
  - Comments
  - FK Constraint (Customer)
  - FK AccountType

- Contact (Table)
  - S_Contact (Sequence)

- AccountTransaction (Table)
  - S_AccountTransaction (Sequence)
Database refactoring is Hard.

Continuous Refactoring Controlled release

- Development Sandbox
  - Highly Iterative Development
  - Bug Reports

- Project Integration Sandbox
  - Project-Level Testing
  - Bug Reports

- Test/QA Sandbox
  - System Integration Testing
  - Bug Reports

- Production
  - Operations and Support
Demo: Merge Columns

- Isolated change
- Use of dbdeploy
- Developer involvement
- Migration and new development
- How application build, links to database version

Demo: Assert Database Behavior

- Like any framework test your database
- These tests are your database specification
- No inadvertent change can be made to the database.
First deployment

- Should be handled almost exactly the same as deploying code
- Database creation script is used to create the production instance of the database.
- Branch code if necessary
- The production instance will be either a clean copy of the application database schema or full of converted data

Demo: Deployment

- Shows DBA tasks
- Shows time saved by the DBA
- Shows how there are no extra step to extract the schema and create scripts for deployment
- Shows how application and database are coupled
Manage Additional Releases

- All of the change scripts should be applied following the last production release. In chronological order. (Change Logs are stored by Build/Release/Day/Sequential)
- Since data migration scripts are part of change logs, data is also migrated
- Allows to test the migration on a test database for performance impacts
Fast Forward

- Production database instances can be migrated to a future state of deployment.
- New application releases can be acceptance tested against the most recent data currently in production.
- Migrations can be tested continuously against copy of production data.

Boat Metaphor (Deploy Frequently)
More Info

Keep in touch
www.sadalage.com

Additional Resources
groups.yahoo.com/group/agileDatabases
databaserefactoring.com
martinfowler.com
dbdeploy.com