Evolving Database Design
and Architecture

Patterns and Practices

Pramod Sadalage
ThoughtWorks Inc.
@pramodsadalage
Patterns of Database Changes
Patterns of Database Changes

• Architecture
Patterns of Database Changes

- Architecture
- Structure
Patterns of Database Changes

- Architecture
- Structure
- Data Quality
Patterns of Database Changes

- Architecture
- Structure
- Data Quality
- Referential Integrity
Patterns of Database Changes

• Architecture
• Structure
• Data Quality
• Referential Integrity
• Database Code
Timeline of Change

Original
Implement the refactoring

Transition
Deploy new changes, migrate data, put in scaffolding code
Transition Period (old and new)

Result
Remove old schema, scaffolding code
Refactoring completed

Thursday, November 17, 2011
Architecture Patterns

Changes that improve the overall manner in which external programs interact with the database
Add Read Method

Original Schema

Resulting Schema
Introduce Read Only Table

Customer
- CustomerID <<PK>>
- Name
- PhoneNumber

Account
- AccountID <<PK>>
- CustomerID <<FK>>
- AccountType <<FK>>
- Balance

Insurance
- PolicyID <<PK>>
- CustomerID <<FK>>
- Payment
- Value
- PaymentPeriod
- InceptionDate

CustomerPortfolio
{access = read only}
- CustomerID <<PK>><<FK>>
- Name
- PhoneNumber
- AccountsTotalBalance
- InsuranceTotalPayment
- InsuranceTotalValue
Encapsulate Table with View

Original Schema

Resulting Schema
Migrate Method from Database

Original Schema

CRM
Policy Administration
CustomerDB
GetValidPolicies()
eService

Transition Period

CRM
Policy Administration
GetValidPolicies()
{drop date = 12/12/2011}
eService

Resulting Schema

CRM
Policy Administration
GetValidPolicies()
eService
Introduce Calculation Method

<table>
<thead>
<tr>
<th>CustomerDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;Stored Procedures&gt;&gt;</td>
</tr>
<tr>
<td>GetAccountDetails ( int AccountID ): Record</td>
</tr>
<tr>
<td>GetAccountList ( int CustomerID ): Records</td>
</tr>
</tbody>
</table>

Original Schema

<table>
<thead>
<tr>
<th>CustomerDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;Stored Procedures&gt;&gt;</td>
</tr>
<tr>
<td>GetAccountDetails ( int AccountID ): Record</td>
</tr>
<tr>
<td>GetAccountList ( int CustomerID ): Records</td>
</tr>
<tr>
<td>GetCustomerAccountTotal ( int CustomerID ): Currency</td>
</tr>
</tbody>
</table>

Resulting Schema
Migrate Method to Database

Original Schema

Policy Administration
isCustomerDefaulted()

Customer Credit
hasCustomerDefaulted()

CustomerDB

Transition Period

Policy Administration
isCustomerDefaulted()
{drop date = 12/12/2011}

Customer Credit
hasCustomerDefaulted()
{drop date = 12/12/2011}

CustomerDB

Resulting Schema

Policy Administration

Customer Credit

CustomerDB

hasCustomerDefaulted()
more at

http://databaserefactoring.com
Structural Change Patterns

Change the structure of the database schema, for better database design.
Split Column

Original Schema

Customer
- CustomerID
- Name
- PhoneNumber

Transition Period

Customer
- CustomerID
- Name {drop date = 12/12/2011}
- FirstName
- MiddleName
- LastName
- PhoneNumber

SynchronizeCustomerName
{event = update | insert, drop date = 12/12/2011}

Resulting Schema

Customer
- CustomerID
- FirstName
- MiddleName
- LastName
- PhoneNumber
Replace One-To-Many with Associative Table

Original Schema

Customer
CustomerPOID <<PK>>
Name

Policy
PolicyID
<<PK>>
CustomerPOID <<FK>>
Amount

Holds
<<<Associative Table>>>
CustomerPOID <<FK>>
PolicyID <<FK>>
UpdatePolicyCustomerPOID
{ event = insert,
drop = 12/12/2011 }

InsertHoldsRow { event = insert,
drop date = 12/12/2011 }

Transition Period

Customer
CustomerPOID <<PK>>
Name

Holds
<<<Associative Table>>>
CustomerPOID <<FK>>
PolicyID <<FK>>

Policy
PolicyID <<PK>>
CustomerPOID <<FK>>
Amount

Resulting Schema
Rename Table

Original Schema

Cust_TB_Prod
{drop date = 12/12/2011}

SynchronizeWithCustomer
{ event = update | insert | delete,
   drop date = 12/12/2011 }

Customer

Transition Period

Customer

Resulting Schema

Thursday, November 17, 2011
Merge Columns

Original Schema

<table>
<thead>
<tr>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>PhoneCountryCode</td>
</tr>
<tr>
<td>PhoneAreaCode</td>
</tr>
<tr>
<td>PhoneLocal</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>PhoneCountryCode</td>
</tr>
<tr>
<td>PhoneAreaCode</td>
</tr>
<tr>
<td>PhoneLocal</td>
</tr>
<tr>
<td>PhoneNumber</td>
</tr>
</tbody>
</table>

| SynchronizePhoneNumber |
| { event = update | insert, drop date = 12/12/2011 } |

Transition Period

Resulting Schema

<table>
<thead>
<tr>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>PhoneCountryCode</td>
</tr>
<tr>
<td>PhoneNumber</td>
</tr>
</tbody>
</table>
Replace Column

Original Schema

Customer

CustomerPOID <<PK>>
CustomerNumber: integer
FirstName
LastName

Transition Period

Customer

CustomerPOID <<PK>> CustomerNumber: integer
{ drop date = 12/12/2011 }
CustomerID: char(12)
FirstName
LastName

SynchronizeCustomerIDNumber
{ event = update | insert,
  drop date = 12/12/2011 }

Resulting Schema

Customer

CustomerPOID
<<PK>> CustomerID: char(12)
FirstName
LastName
Split Table

**Original Schema**

**Employee**
- EmployeeID <<PK>>
- Name
- Picture
- Phone
- Email

**EmployeePicture**
- EmployeeID <<PK>><<FK>>
- Picture

**Transition Period**

**Employee**
- EmployeeID <<PK>>
- Name
- Phone
- Email

**EmployeePicture**
- EmployeeID <<PK>><<FK>>
- Picture

**Resulting Schema**

**Employee**
- EmployeeID <<PK>>
- Name
- Phone
- Email

**EmployeePicture**
- EmployeeID <<PK>><<FK>>
- Picture
more at

http://databaserefactoring.com
Data Quality Patterns

Changes that improve the quality of the information within a database or ensure the consistency and usage of data.
Make Column Non-Nullable

Original Schema

Customer
- CustomerID <<PK>>
- FirstName
- Surname

Resulting Schema

Customer
- CustomerID <<PK>>
- FirstName <<Not Null>>
- Surname
Introduce Column Constraint

Original Schema

<table>
<thead>
<tr>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>CustomerID &lt;&lt;PK&gt;&gt;</td>
</tr>
<tr>
<td>FirstName</td>
</tr>
<tr>
<td>Status</td>
</tr>
<tr>
<td>CreditLimit</td>
</tr>
</tbody>
</table>

Resulting Schema

<table>
<thead>
<tr>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>CustomerID &lt;&lt;PK&gt;&gt;</td>
</tr>
<tr>
<td>FirstName</td>
</tr>
<tr>
<td>Status</td>
</tr>
<tr>
<td>CreditLimit { &lt; 50000 , name = Check_Credit_Limit }</td>
</tr>
</tbody>
</table>
Add Lookup Table

Original Schema

<table>
<thead>
<tr>
<th>Address</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Street char (40)</td>
<td>State char (2)</td>
</tr>
<tr>
<td>State char (2)</td>
<td>Name char (50)</td>
</tr>
<tr>
<td>PostCode char (15)</td>
<td></td>
</tr>
</tbody>
</table>

Resulting Schema

<table>
<thead>
<tr>
<th>Address</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Street char (40)</td>
<td>State char (2)</td>
</tr>
<tr>
<td>State char (2)</td>
<td>Name char (50)</td>
</tr>
<tr>
<td>PostCode char (15)</td>
<td></td>
</tr>
</tbody>
</table>
more at

http://databaserefactoring.com
Referential Integrity Patterns

Changes ensure Referential Data is maintained making sure Data Quality is improved
Add Foreign Key Constraint

Original Schema

Resulting Schema
Introduce Cascading Delete

Original Schema

Resulting Schema
more at

http://databaserefactoring.com
Database Code

Like code refactoring (Fowler 1999) refactor database code to improve the design of database code i.e Stored Procs and Triggers.
Without good development Practices, using and implementing the Patterns is going to be difficult.
Configuration Management
Configuration Management

- Allow for common code ownership
Configuration Management

- Allow for common code ownership
- All database artifacts belong in source control repository
Configuration Management

• Allow for common code ownership
• All database artifacts belong in source control repository
• Include setup/config data
Configuration Management

- Allow for common code ownership
- All database artifacts belong in source control repository
- Include setup/config data
- Publish database artifacts with Continuous Integration
Give everyone a database Sandbox
Give everyone a database Sandbox

• Reduce waste and waiting time
Give everyone a database Sandbox

• Reduce waste and waiting time
• Automate data tasks
Give everyone a database Sandbox

- Reduce waste and waiting time
- Automate data tasks
- Allows local changes and experimenting
Give everyone a database Sandbox

• Reduce waste and waiting time
• Automate data tasks
• Allows local changes and experimenting
• Improves productivity
Give everyone a database Sandbox

• Reduce waste and waiting time
• Automate data tasks
• Allows local changes and experimenting
• Improves productivity
• Spinning up new environments is easy
Developer schema
Point of contention during development
Behavior of Database
Behavior of Database

• Like objects, database has behavior
Behavior of Database

• Like objects, database has behavior

• Develop database objects using BDD style tests
Behavior of Database

• Like objects, database has behavior

• Develop database objects using BDD style tests

• Allows easy changes to the database
Behavior of Database

• Like objects, database has behavior

• Develop database objects using BDD style tests

• Allows easy changes to the database

• Protects against changes that affect dependent functionality
Requirements:
Store a “Vehicle” with

- Unique VIN Number
- Model Year 2005 and above
- Model Year Not Null
- Model Name Not Null
- Make Not Null
- Miles not above 10000
Requirements:
Store a “Vehicle” with
- Unique VIN Number
- Model Year 2005 and above
- Model Year Not Null
- Model Name Not Null
- Make Not Null
- Miles not above 10000
Requirements:
Store a “Vehicle” with
- Unique VIN Number
- Model Year 2005 and above
- Model Year Not Null
- Model Name Not Null
- Make Not Null
- Miles not above 10000

Behavior Tests:

- shouldNotSaveDuplicateVIN()
- shouldSaveModelYear2010()
- shouldNotSaveModelYear2004()
- shouldNotSaveNullModelName()
- shouldNotSaveNullMake()
- shouldSaveMiles5000()
- shouldNotSaveMiles12000()
CREATE TABLE vehicle(
    id NUMBER(18) NOT NULL,
    vin VARCHAR2(32) NOT NULL,
    name VARCHAR2(32) NOT NULL,
    make VARCHAR2(32) NOT NULL,
    year NUMBER(4) NOT NULL,
    miles NUMBER(10) NULL,
    CONSTRAINT chk_vehicle_year_gt_2005
    CHECK (year > 2004));
CONSTRAINT chk_vehicle_miles_lt_10001
    CHECK (miles < 10001));
CREATE UNIQUE INDEX uidx_vehicle_vin
    ON vehicle(vin);
ALTER TABLE VEHICLE ADD CONSTRAINT
    pk_vehicle PRIMARY KEY (id);
Tracking Changes
Tracking Changes

• Write each change as a delta (migration) script
Tracking Changes

- Write each change as a delta (migration) script
- Migration scripts are development time activity not deployment time project
Tracking Changes

• Write each change as a delta (migration) script

• Migration scripts are development time activity not deployment time project

• Package delta scripts for automated deployment
Tracking Changes

• Write each change as a delta (migration) script

• Migration scripts are development time activity not deployment time project

• Package delta scripts for automated deployment

• Same scripts for: developers, QA, UAT and Production
ALTER TABLE customer ADD customeridentifier VARCHAR2(12);
UPDATE customer SET customeridentifier = customernumber;
--If No Transistion Period
ALTER TABLE customer DROP COLUMN customerNumber;
--//@UNDO
ALTER TABLE customer ADD customernumber NUMBER(10);
UPDATE customer SET customernumber = customeridentifier;
ALTER TABLE customer DROP COLUMN customeridentifier;
Deployed Database Version

Andy/Joe v48
Jim/Rex v49
Jack/Ram v43

QA v48
UAT v44
QA v47
CT v45

PROD v43

Migration scripts, written once deployed multiple times
Continuous Integration
Continuous Integration

- Test application code and database at one place
Continuous Integration

• Test application code and database at one place

• Generate code and database artifacts
Continuous Integration

- Test application code and database at one place
- Generate code and database artifacts
- Integrate application and database changes in an independent environment
Check in application code and database migration scripts

Local dev environment

ANT
Maven
Make
Rake

Dev DB

Central integration environment

Continuous Integration Engine

Source Control

Update and build

Dev Database

PROD
CT
QA

PROD
UAT
QA
Environment
Check in application code and database migration scripts

Local dev environment
- Migration scripts
- ANT
- Maven
- Make
- Rake

Central integration environment
- Source Control
- Continuous Integration Engine
- Update and build

Environment:
- PROD
- CT
- QA
- PROD
- UAT
- QA
Check in application code and database migration scripts

Local dev environment

Migration scripts

ANT
Maven
Make
Rake

Dev DB

Central integration environment

Migration scripts

Source Control

Continuous Integration Engine

Update and build

Integration Database

PROD
CT
QA

PROD
UAT
QA
Environment

Thursday, November 17, 2011
Check in application code and database migration scripts

Central integration environment

Local dev environment

Migration scripts

ANT
Maven
Make
Rake

Dev DB

Source Control

Update and build

Continuous Integration Engine

Integration Database

PROD
CT
QA

PROD
UAT
QA
Environment
Check in application code and database migration scripts

Local dev environment

Migration scripts

Dev DB

ANT
Maven
Make
Rake

Central integration environment

Migration scripts

Continuous Integration Engine

Source Control

Update and build

Integration Database

Apply migration scripts

PROD

CT

QA

UAT

QA

Environment

Thursday, November 17, 2011
Check in application code and database migration scripts

Local development environment

Migration scripts

ANT
Maven
Make
Rake

Dev
DB

Central integration environment

Migration scripts

Source Control

Update and build

Integration Database

Apply migration scripts

Continuous Integration Engine

Artifacts

War
Jar

Migration scripts

PROD
CT
QA

PROD
UAT
QA
Environment

Thursday, November 17, 2011
Check in application code and database migration scripts

Local dev environment

Migration scripts

ANT
Maven
Make
Rake

Dev
DB

Central integration environment

Continuous Integration Engine

Source Control

Update and build

Migration scripts

Integration Database

Apply migration scripts

Migration scripts

Artifacts

War Jar

PROD
CT
QA

PROD
UAT
QA Environment

War Jar

Migration scripts
Deployment
Deployment

- Database migration/upgrade should be a development time task not deployment time task
Deployment

- Database migration/upgrade should be a development time task not deployment time task

- Package all the migration scripts, during Continuous Integration cycle
Deployment

- Database migration/upgrade should be a development time task not deployment time task
- Package all the migration scripts, during Continuous Integration cycle
- Apply these migration scripts
Pair with the Data Team
Pair with the Data Team

• Break down the silos
Pair with the Data Team

• Break down the silos
• Allows continuous reviews
Pair with the Data Team

- Break down the silos
- Allows continuous reviews
- Understand performance implications early
Pair with the Data Team

• Break down the silos
• Allows continuous reviews
• Understand performance implications early
• Put database code and application code in same repository
Resources

bit.ly/qconbddd
bit.ly/evolvedb
Thanks

@pramodsadalage

www.sadalage.com

www.databaserefactoring.com