
ORACLE®

ORACLE

Patterns of SaaS: Database refactoring

Leonid Igolnik, Marcin Burlinski

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Agenda

- Introductions
- Fundamentals of Database Refactoring
- “Putting it all together” - a case study
- Q&A

About Us

- Based in California
- Java Developer for over 10 years
- Over 13 years of SaaS experience
- Avid traveller and photographer

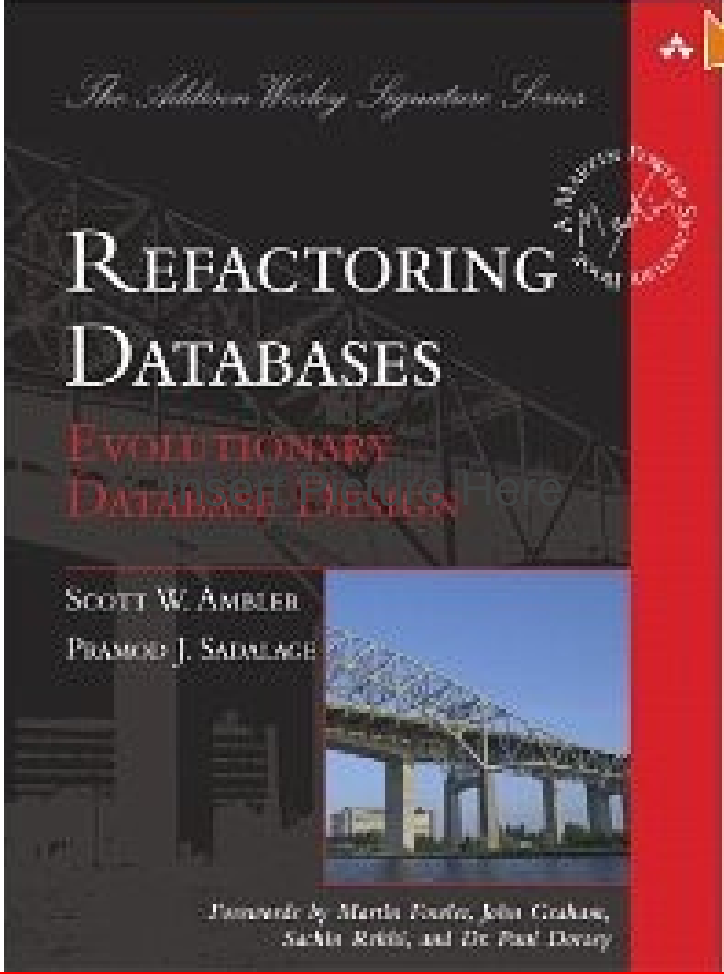
About Us - Marcin

- Based in Poland
- Java Developer for over 10 years
- Over 10 years of SaaS experience
- Not a big fan of SQL
- Avid traveller

Are you a ...

- Developer
- DBA
- Some who likes writing complex SQL
- Someone who maintains their own schema and writes change scripts
- Someone who has a tool do it for you (Play!, Spring Roo, etc..)

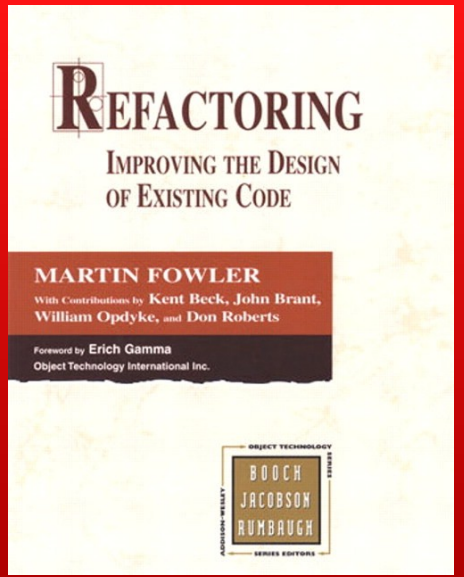
Database refactoring fundamentals



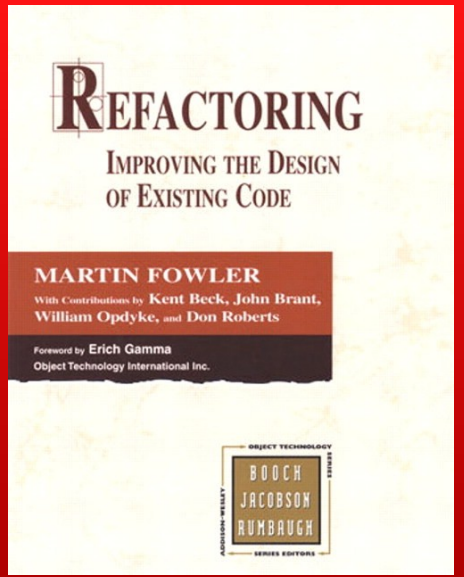
I live in the world of enterprise applications, and a big part of enterprise application development is working with databases. In my original book on refactoring, I picked out databases as a major problem area in refactoring because refactoring databases introduces a new set of problems. These problems are exacerbated by the sad division that's developed in the enterprise software world where database professionals and software developers are separated by a wall of mutual incomprehension and contempt.

Martin Fowler

Chief Scientist, ThoughtWorks.



I live in the world of enterprise applications, and a big part of enterprise application development is working with databases. In my original book on refactoring, I picked out databases as a major problem area in refactoring because refactoring databases introduces a new set of problems. **These problems are exacerbated by the sad division that's developed in the enterprise software world where database professionals and software developers are separated by a wall of mutual incomprehension and contempt.**



Martin Fowler

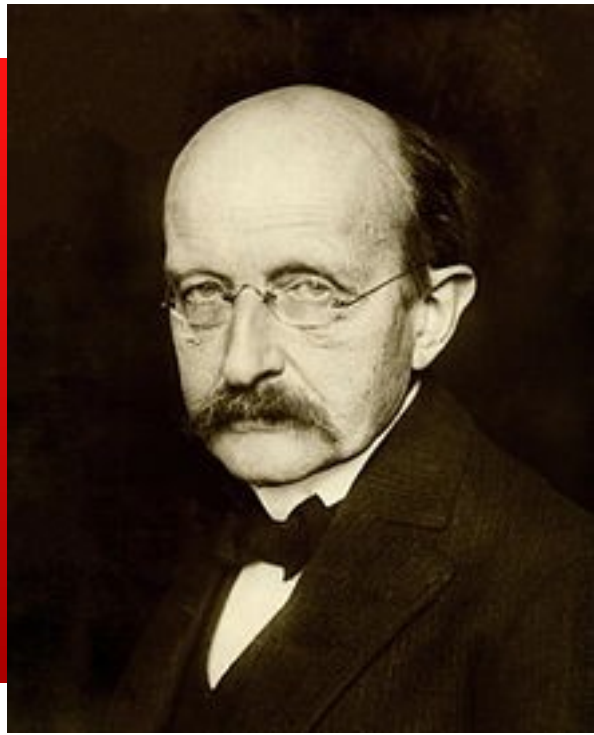
Chief Scientist, ThoughtWorks.

Database refactoring stakeholders

- Developers
- DBAs
- QA
- Business Analysts

A new scientific truth does not triumph by convincing its opponents and making them see the light, but rather because its opponents eventually die, and a new generation grows up that is familiar with it.

Max Planck

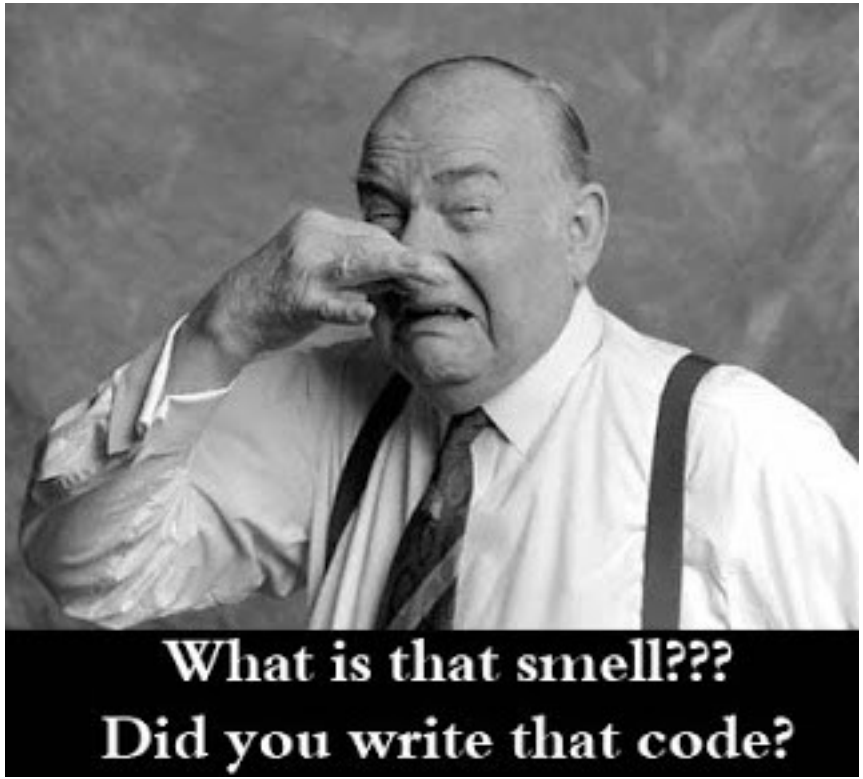


But there are great benefits to be gained

- Designing data model up front is hard
- At times upfront data modeling is impossible
- Knowing how to evolve your data model efficiently improves your system architecture
- Much like getting rid of “Code Smells” using code refactoring , you can get rid of “Database Smells”

Database smells

- Multipurpose column
- Multipurpose table
- Redundant data
- Too many columns
- Fear of change



Mechanics of database refactoring

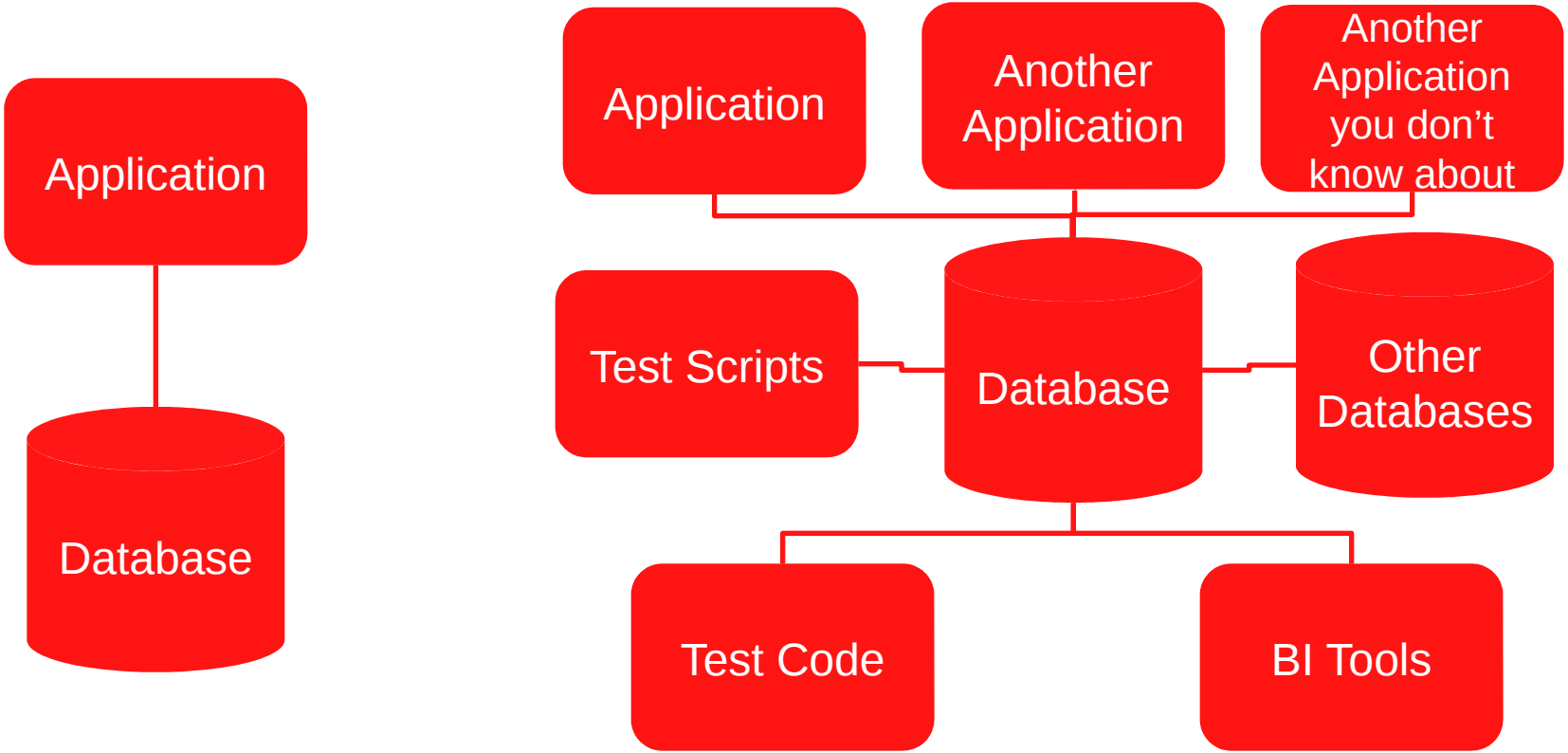
A database refactoring is a simple change to a database schema that improves its design while retaining both its behavioral and informational semantics.

Ambler, Scott W.; Sadalage, Pramod J. (2006-03-03). Refactoring Databases: Evolutionary Database Design (Addison-Wesley Signature Series) (Kindle Locations 637-638). Addison-Wesley Professional. Kindle Edition.

A database refactoring is a **simple change** to a database schema that improves its design while retaining both its behavioral and informational semantics.

Ambler, Scott W.; Sadalage, Pramod J. (2006-03-03). Refactoring Databases: Evolutionary Database Design (Addison-Wesley Signature Series) (Kindle Locations 637-638). Addison-Wesley Professional. Kindle Edition.

Application architecture types



Key steps of refactoring process

1. Agree on the changes with all stakeholders
2. Refactor the schema
3. Migrate the data as needed
4. Refactor external access programs
5. **Run your tests**
6. Version control the change
7. Announce the change to other consumers
8. Deploy to production
9. Deprecate the old schema

Refactoring types

Database refactoring categories

- Structural
- Data Quality
- Referential Integrity
- Functional
- Non-refactoring transformations

Structural refactoring types

- Replace Column
- Drop Column
- Rename Table
- Rename Column
- Introduce Calculated Column
- Introduce Surrogate Key
- Replace One-to-Many With Associative Table
- Move Column
- Merge Columns
- Merge Tables
- Drop Table
- Drop View
- Rename View
- Replace Large Object (LOB) With Table
- Replace Surrogate Key With Natural Key
- Split Column
- Split Table

Structural: Rename column

Motivation

- Readability of schema
- Database porting (e.g. to remove reserved keywords)

Tradeoffs

- Cost of application change vs. readability

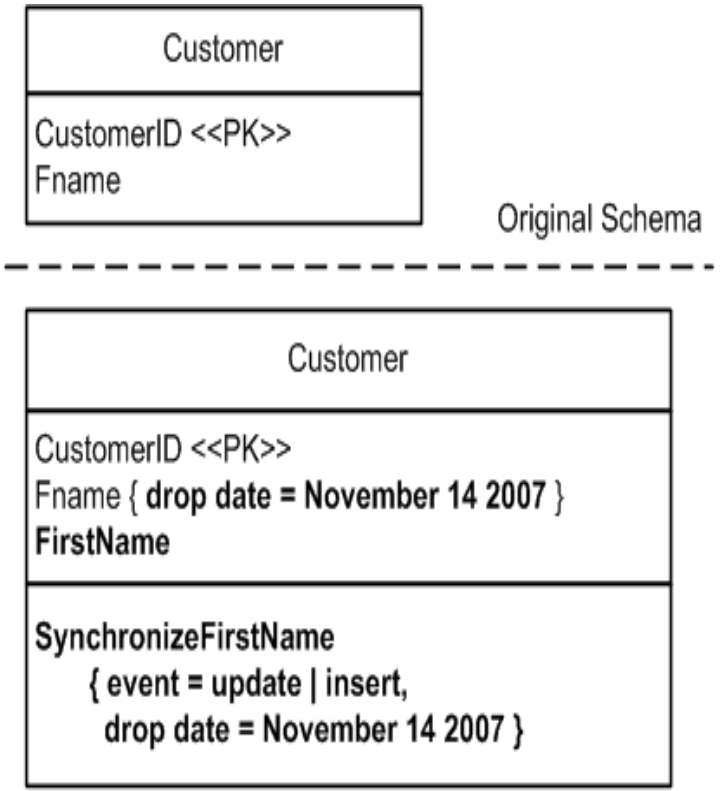
Update Mechanics

- Introduce new column
- Add sync trigger (if multi-app access)
- Consider renaming referencing columns

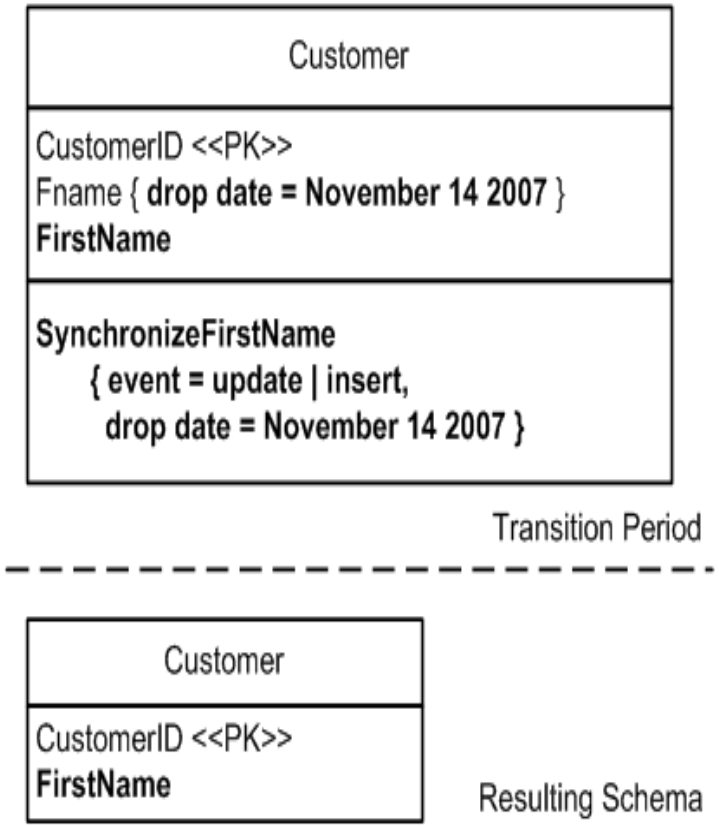
Data Migration

- Copy data

Example



Example



Data Quality refactoring types

- Make Column Non-Nullable
- Make Column Nullable
- Drop Default Value
- Add Default Value
- Introduce Column Constraint
- Drop Column Constraint
- Move Data
- Replace Type Code with Property Flags
- Introduce Column Format
- Consolidate Key Strategy
- Apply Standard Type
- Apply Standard Codes
- Add Lookup Table

Referential Quality refactoring types

- Add Foreign Key Constraint
- Introduce Soft Delete

- Add Trigger for Calculated Column
- Drop Foreign Key Constraint
- Introduce Cascading Delete
- Introduce Hard Delete
- Introduce Trigger for History

Key Considerations: Testing

- Stored procedures
- Referential integrity
- View definitions
- Default values
- Data invariants (constraints)
- Data migration testing

Putting it all together – or how to apply 650,000 changes in a few hours without loosing sleep

Approaches to Managing Multi-Tenant Data

Approaches to Managing Multi-Tenant Data

	Separate Database	Separate Schema	Shared Schema

Approaches to Managing Multi-Tenant Data

	Separate Database	Separate Schema	Shared Schema
Pros	• Database level data isolation • Out of the box per tenant recovery		
	• Can be expensive to operate • Inherit limitations of number of databases per instance • Schema sync		

Approaches to Managing Multi-Tenant Data

	Separate Database	Separate Schema	Shared Schema
Pros	• Database level data isolation • Out of the box per tenant recovery		• Guaranteed schema sync
	• Can be expensive to operate • Inherit limitations of number of databases per instance • Schema sync		• Per tenant operations are more difficult • Index balancing issues

Approaches to Managing Multi-Tenant Data

	Separate Database	Separate Schema	Shared Schema
Pros	• Database level data isolation • Out of the box per tenant recovery	• Database level data isolation	• Guaranteed schema sync
	• Can be expensive to operate • Inherit limitations of number of databases per instance • Schema sync	• Inherit limitations of number of databases per instance • Schema sync • No simple per tenant recovery	• Per tenant operations are more difficult • Index balancing issues

The application

- ~6,000 tenants
- Over 70,000,000 transactions a week
- Average response time ~170ms
- Peak transaction volume ~20,000/min
- Average transaction volume ~4,000-7,000/min

Our approach: Release model

- Single code base for all customers (configuration over customization)
- Single release date for all customers every 8-10 weeks
- Nearly zero downtime during release
- No rollback past defined point of no-return
- Releases are done under average transaction workloads
- Typical release contains about 70-200 individual changes to the database (DML + DDL)

Management of database artifacts

- **D**ata **D**efinition **L**anguage (DDL) scripts for the schema
  - Table definitions
  - View definitions
  - Referential and other constraints
  - Indexes
  - Misc objects such as sequences etc
- O/R mapping configs
- Stored procedures/triggers etc.
- Data load/extract and migration scripts
- Test data generation scripts and test scripts

Per release script types

- DDL – Data Definition Language
  - CREATE
  - ALTER
  - DROP
  - TRUNCATE
  - RENAME
- DML – Data Manipulation Language
  - INSERT
  - DELETE
  - UPDATE

All changes are split into small incremental changes

Script naming

<file_sequence>_<release_number>_
<project_name>_<brief_description>_
[ddl|dml].sql

Script naming

<file_sequence>_<release_number>_<project_name>_<brief_description>_[ddl|dml].sql

Script naming

<file_sequence>_**<release_number>**

_

<project_name>_<brief_description>_

[ddl|dml].sql

Script naming

<file_sequence>_<release_number>_
<project_name>_<brief_description>
_ [ddl|dml].sql

Script naming

<file_sequence>_<release_number>_
<project_name>_**<brief_description>**
_ [ddl|dml].sql

Script naming

<file_sequence>_<release_number>_
<project_name>_<brief_description>_
[ddl|dml].sql

File naming example

- 110_9.5_create_we_pm_review_competency_section_table.sql
- 120_9.5_add_goal_section_weight_into_we_pm_review.sql
- 130_9.5_add_mandatory_fields_to_list_view.sql
- 140_9.5_COMP_add_comp_category.sql
- 150_9.5_COMP_add_pay_type.sql
- 160_9.5_COMP_add_behavior.sql
- 170_9.5_COMP_add_frequency.sql

Script template

- **Author** - where author name is listed
- **Project** - name of the project for which the script is
- **Description** - purpose of the script

```
-- Author: Nikola
-- Project: Taleo facelift
-- Description: Seeding new display fields

-------------------------------------------------------
```

Make scripts safely re-runnable if you can

- Provide standard pattern for DML and DDL Scripts

```
IF NOT EXISTS (select 1 from run_script where file_name = '<orig_file_name>' AND type = '<pre|post>')
BEGIN
... original sql content
END
```

```
IF EXISTS (SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME='WE_EMPLOYEE' AND
COLUMN_NAME='POTENTIAL')
    ALTER TABLE WE_EMPLOYEE DROP COLUMN POTENTIAL;
```

Common DML patterns

- Avoid bulk updates – use DB SQL and cursors with periodic commits
- Create and use library of reusable templates for common tasks
- Backup the data you alter into a table with standard naming convention:
  - BACKUP_<release_number>_<script_number>_<script_type>
- Automatically delete backup tables after defined period of time
- Make your DBA a friend

Artifact organization in source control

- Releases
  - X.X
  - Component Name
    - PRE
    - MAIN
    - POST

PRE – MAIN - POST

PRE	MAIN	POST

PRE – MAIN - POST

PRE	MAIN	POST

• Changes that keep schema backward compatible:
  • New Nullable columns
  • New Tables
  • Seeding for data compatible with the old code version		

PRE – MAIN - POST

PRE	MAIN	POST

• Changes that keep schema backward compatible:
  • New Nullable columns
  • New Tables
  • Seeding for data compatible with the old code version
	• Minimal changes that are required to make the schema compatible with the new version of the app.
	• Those changes brake backward compatibility
	• Require application downtime	

PRE – MAIN - POST

PRE	MAIN	POST

• Changes that keep schema backward compatible:
  • New Nullable columns
  • New Tables
  • Seeding for data compatible with the old code version
	• Minimal changes that are required to make the schema compatible with the new version of the app.
	• Those changes brake backward compatibility
	• Require application downtime
		• Changes requiring new application code:
		  • Not Nullable fields constraints
		  • New constraints
		  • Seeding data compatible only with the new code version

SQL Review

- Every database change gets reviewed by multiple senior engineers
- SQL reviews start mid-release dev cycle and continue until code freeze
- Tools:
  - Crucible or your favorite code review tool

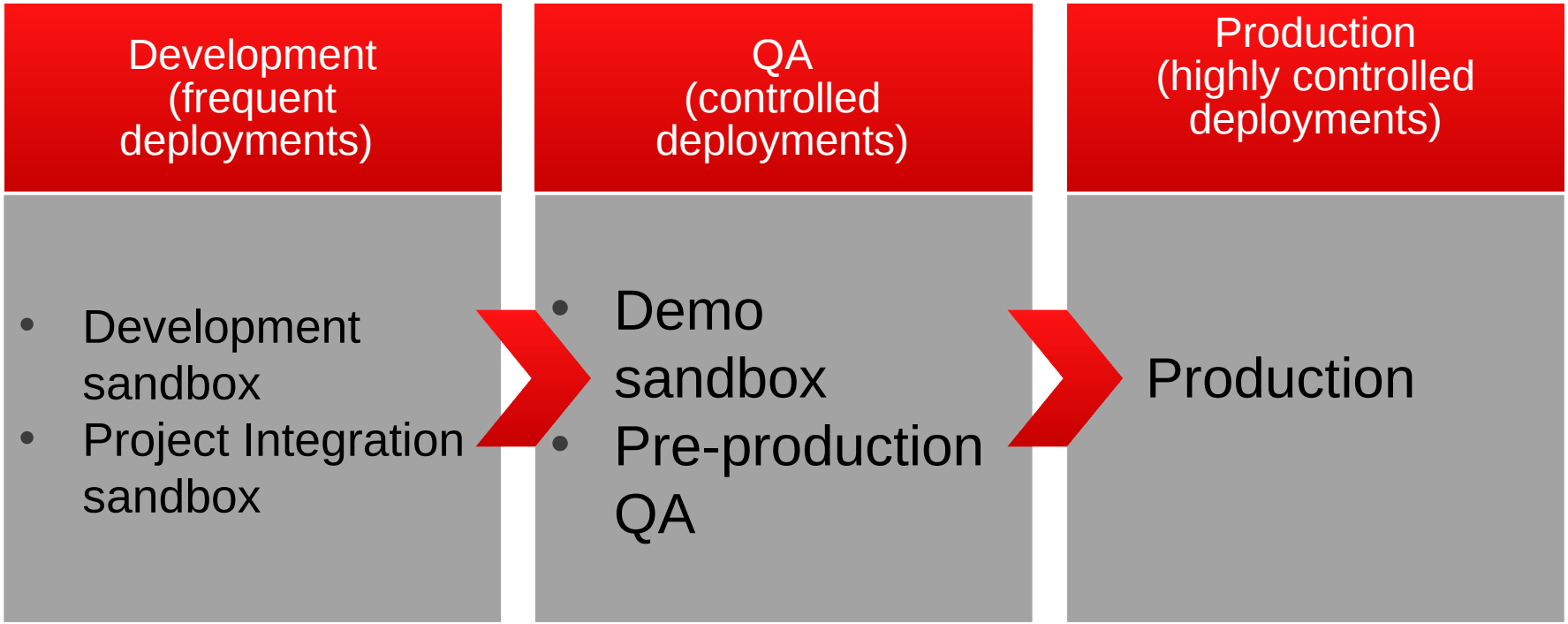
Environments

Development (frequent deployments)		
• Development sandbox • Project Integration sandbox		

Environments

Development (frequent deployments)	QA (controlled deployments)	
• Development sandbox • Project Integration sandbox	• Demo sandbox • Pre-production QA	

Environments



Environments

Development (frequent deployments)	QA (controlled deployments)	Production (highly controlled deployments)
• Apply pre main post as developed at times out of sequence		

Applying DML and DDL to Dev/QA/Production

```
CREATE TABLE run_script
(
    script_run_id bigint IDENTITY(1, 1) NOT NULL PRIMARY KEY,
    created_timestamp DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    type varchar(10) NOT NULL,
    file_name varchar(100)NOT NULL,
    username varchar(30) NOT NULL DEFAULT CURRENT_USER,
    release_number varchar(30) NULL
);
```

Environments

Development (frequent deployments)	QA (controlled deployments)	Production (highly controlled deployments)
• Apply pre main post as developed at times out of sequence	• Apply pre main post as developed at times out of sequence • Perform final integration test	

Environments

Development (frequent deployments)	QA (controlled deployments)	Production (highly controlled deployments)
• Apply pre main post as developed at times out of sequence	• Apply pre main post as developed at times out of sequence • Perform final integration test	Apply in order

Overall workflow

Overall workflow

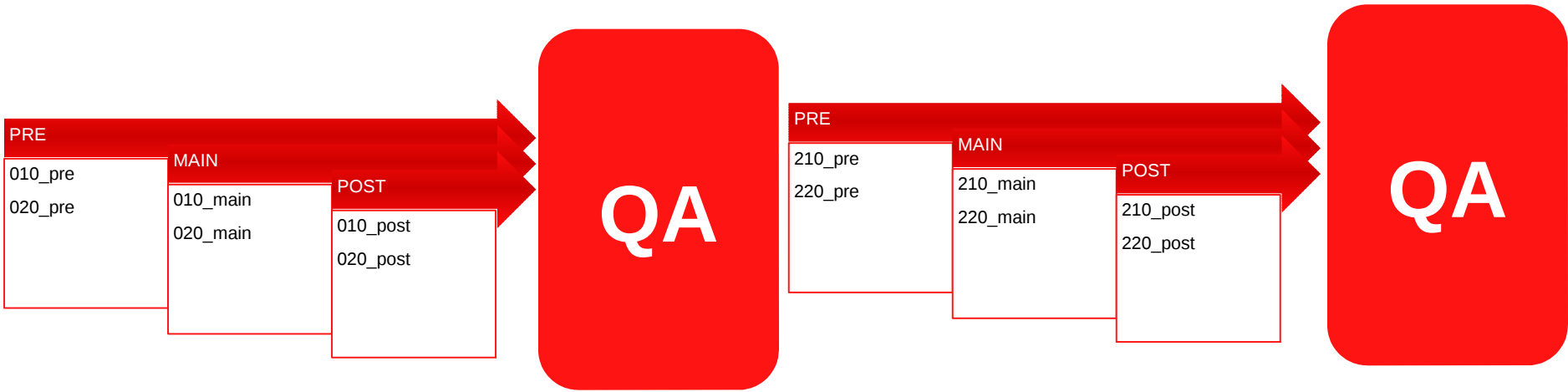
Overall workflow

Overall workflow

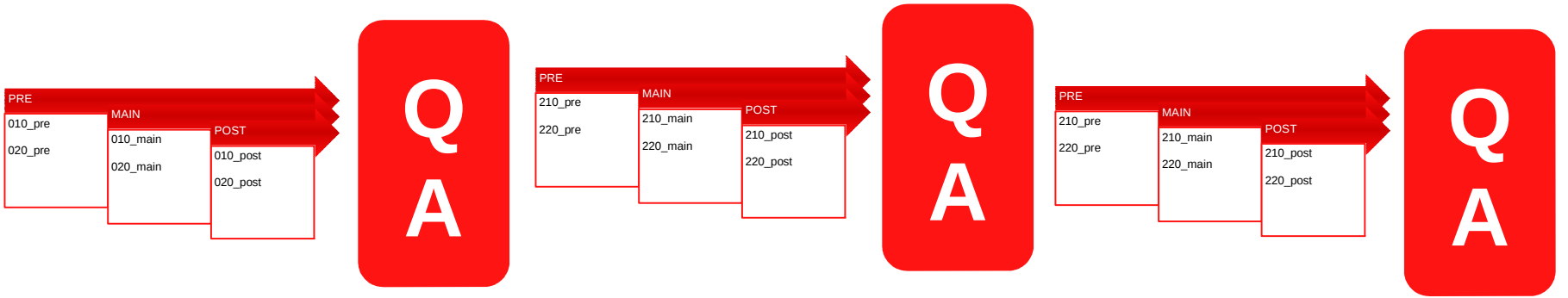
Overall workflow

Overall workflow

Overall workflow



Overall workflow



Final integration test

Final integration test

Final integration test

Final integration test

Final integration test

Final integration test

Day/Week of the release

Day/Week of the release

Common issues

- Schema inconsistencies
- Data inconsistencies
- Transaction log space
- Execution time

Database Schema Audits

- Periodic audit of schema structure
- Combination of manual and automated tools

Lessons learned

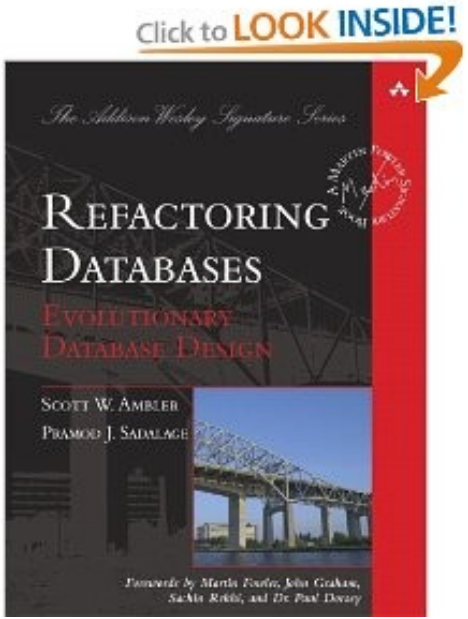
- Document your Schema change process
- Managed you database artifacts in source control
- Have your own version of an SQL Policeman
- Develop reusable templates for DML and DDL
- Have repeatable automated process to apply changes to all environments: Dev, QA, Staging and production
- Make a friend out of your DBA

Key Messages

- Database refactoring is HARD
- Automate everything to ensure repeatable process
- Review everything
- Backup crucial data
- Test as much as you reasonably can
- Don’t reinvent the wheel – follow the pattern

Further reading

- Agile data: http://www.agiledata.org/
- http://www.amazon.com/Refactoring-Databases-Evolutionary-paperback-Addison Wesley/dp/0321774515
- DBUnit
- SQLUnit



Q&A

Hardware and Software

ORACLE

Engineered to Work Together


ORACLE®