Database Delivery Best Practices

INTRODUCTION

Vladimir Khorikov
PROGRAMMER

@vkhorikov www.enterprisecraftsmanship.com

Source code versioning

Continuous delivery

Databases delivery

Common Problems with Database Delivery

Schema mismatch

Database schema mismatch in different environments

Merge conflicts

Synchronizing changes between colleagues

Multiple databases

Multiple clients have their own database instances

Prerequisites

C#

SQL

Overview

Introduction

Applying the State-based Approach to Database Delivery

Applying the Migration-based Approach to Database Delivery

Building a Database Versioning Tool

Refactoring Integration Databases Using Evolutionary Design

Database Delivery Challenges

- **Data in the database**
- **Integration databases**

Integration Databases

- ✗ Accumulates technical debt
- ✗ Lots of boilerplate in applications

Database Delivery Challenges

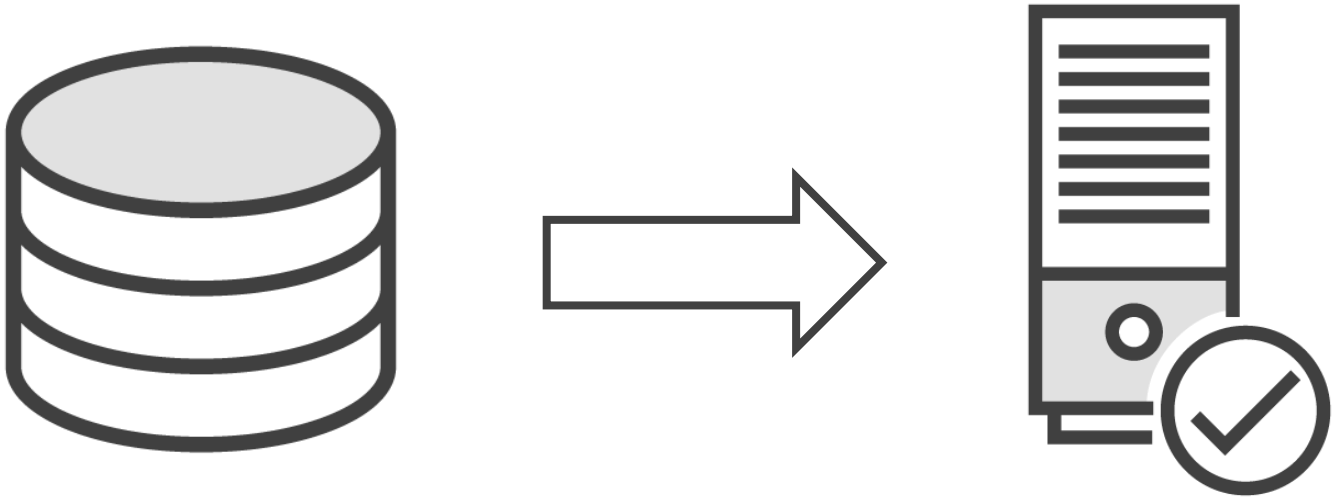
- Data in the database
- Integration databases
- Consistency across different environments
- Merge conflicts

Database Delivery Basic Principles

State-based approach

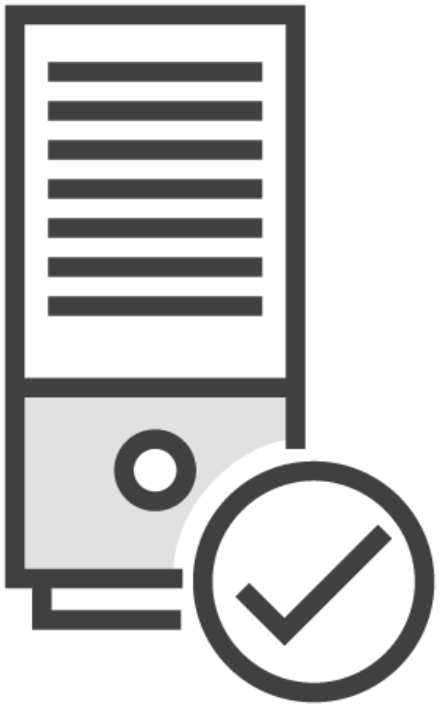
Migration-based approach

Keep Your Database in a Source Control System



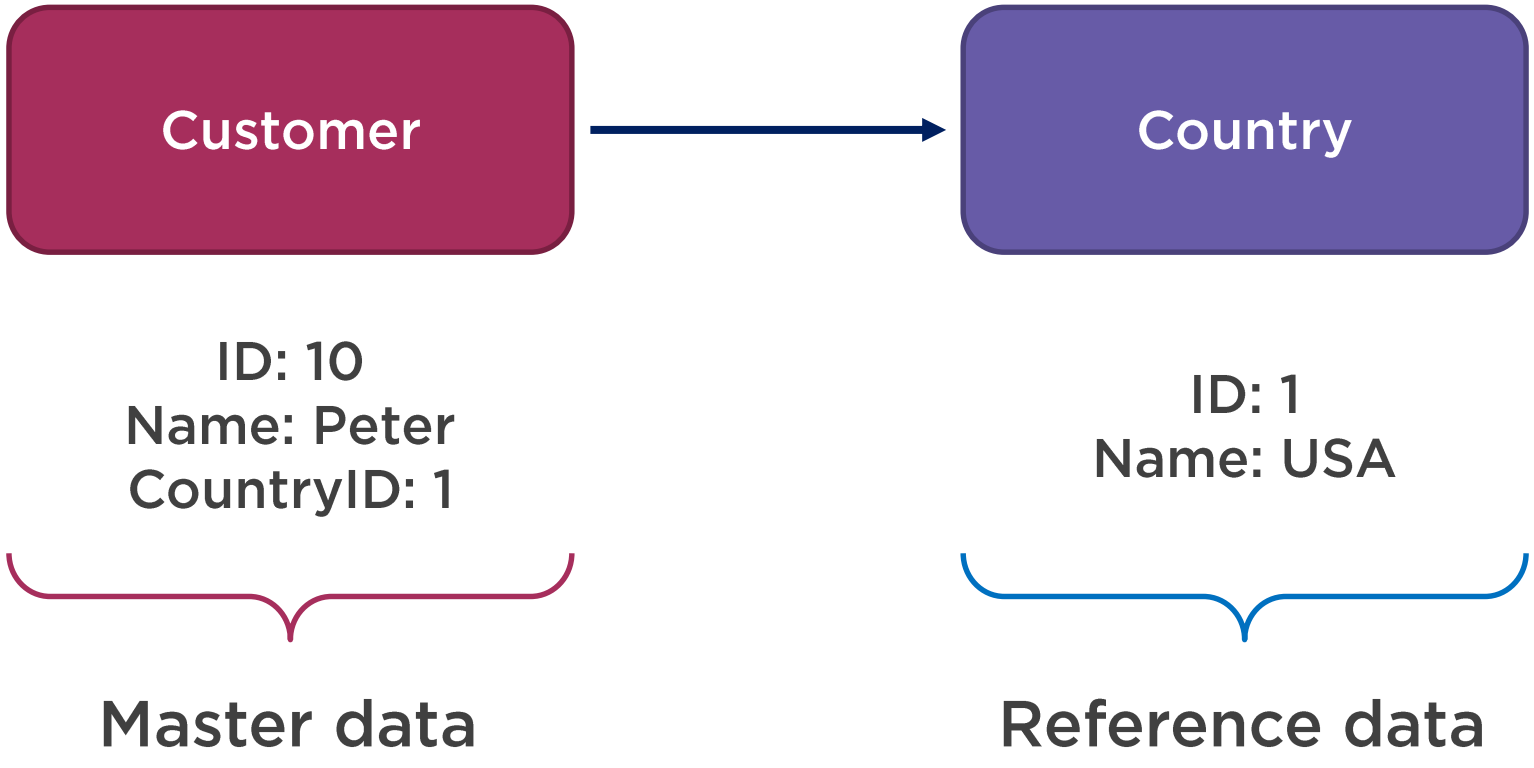
- Primary source of truth
- No changes should be allowed outside of the source control

Keep Your Database in a Source Control System

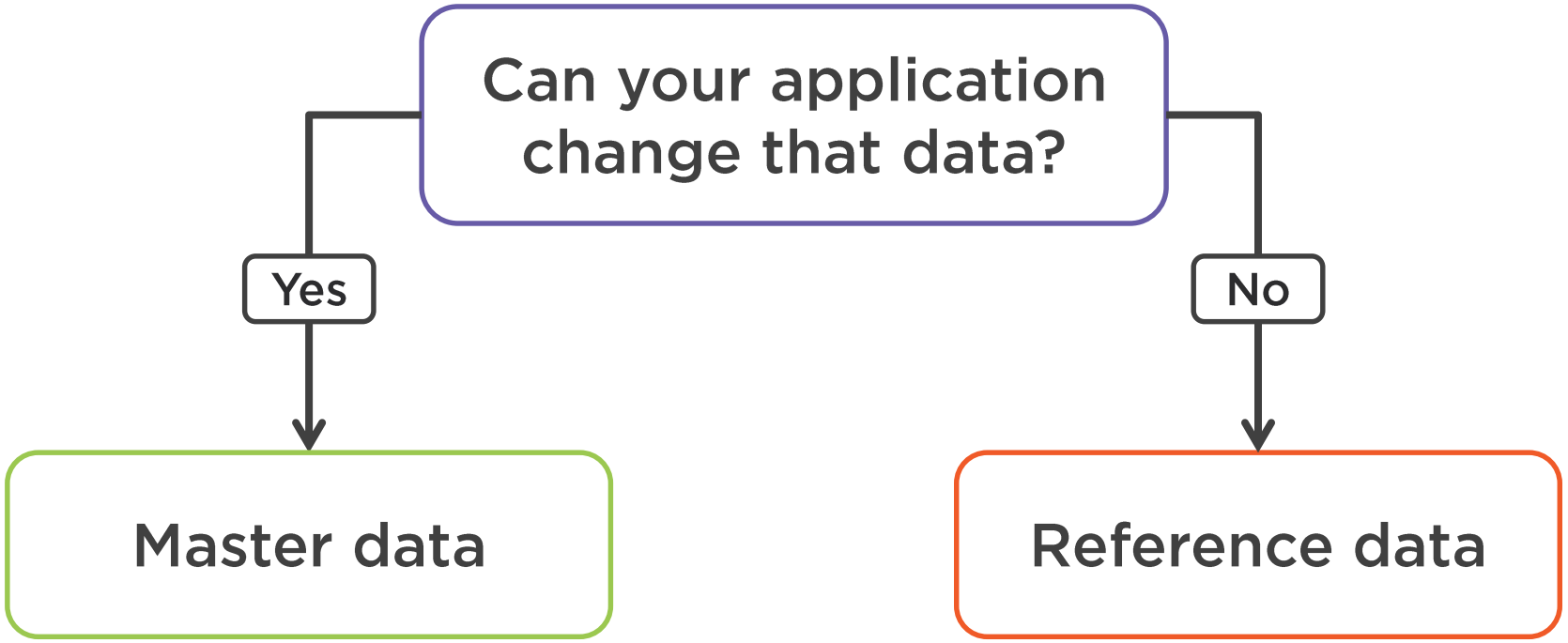


- Database schema
- Reference data

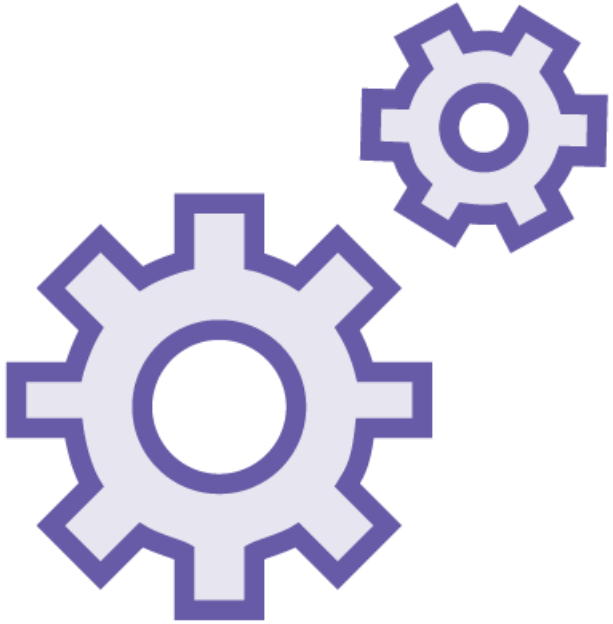
Keep Your Database in a Source Control System



Keep Your Database in a Source Control System



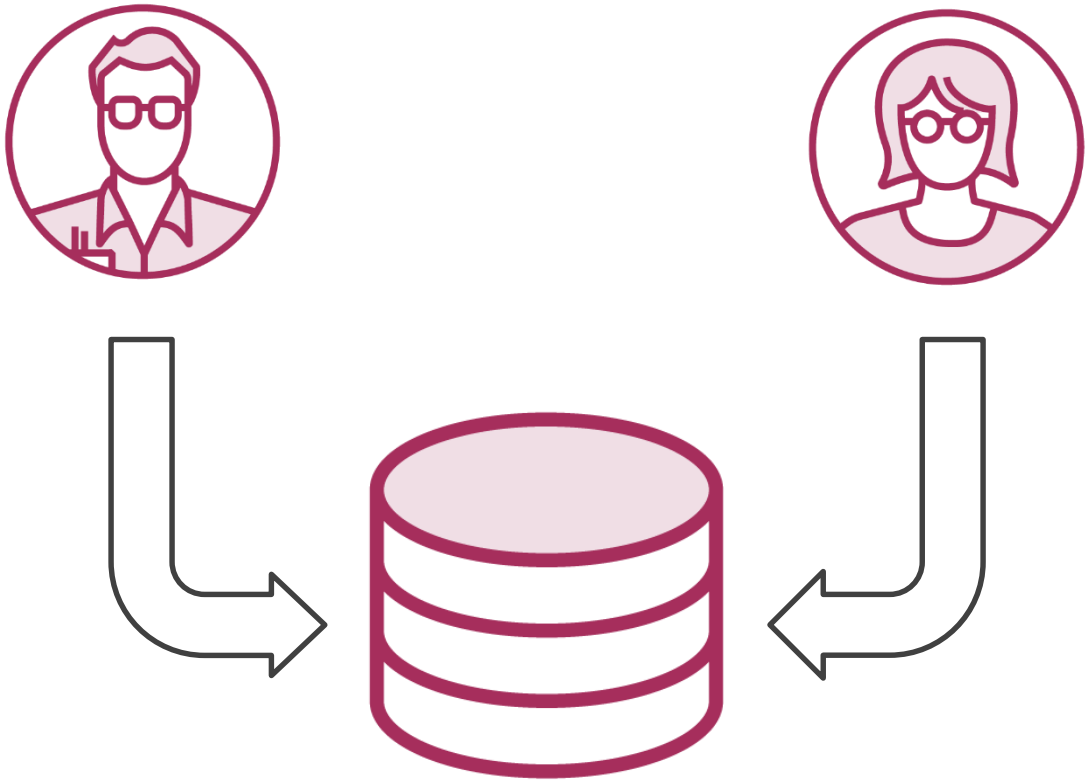
Incorporate the DB into a CI Process



Continuous integration

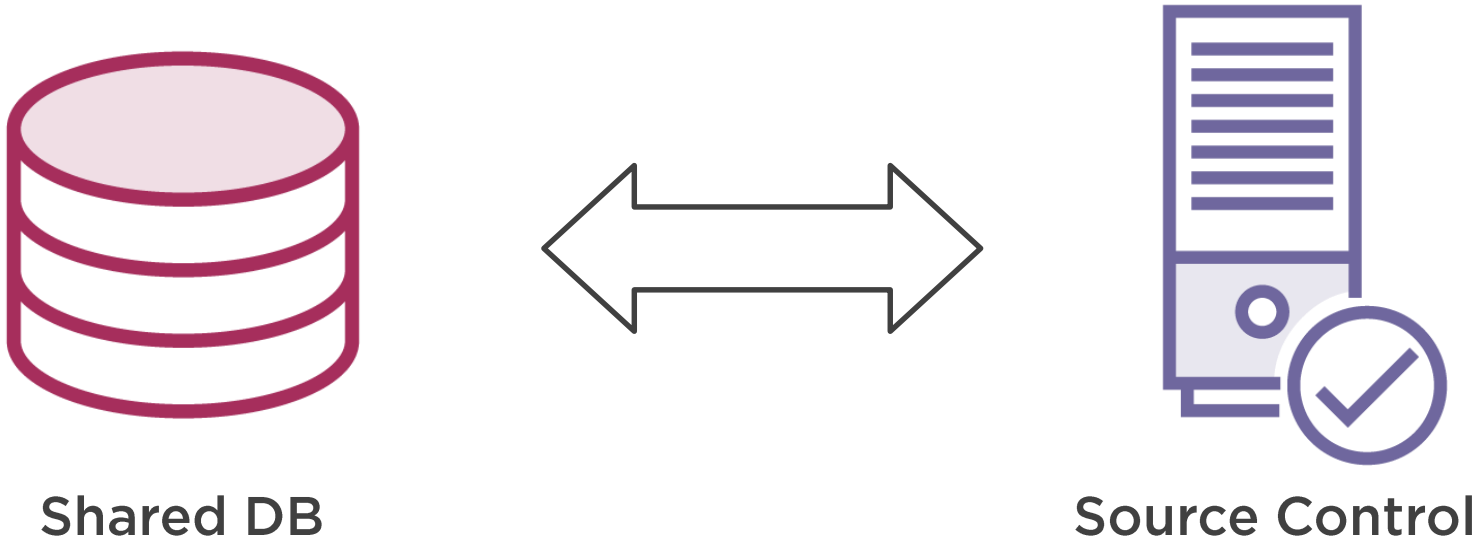
- Intrinsic part of the integration cycle
- Run auto tests on each change

Don’t Use Shared Development Database



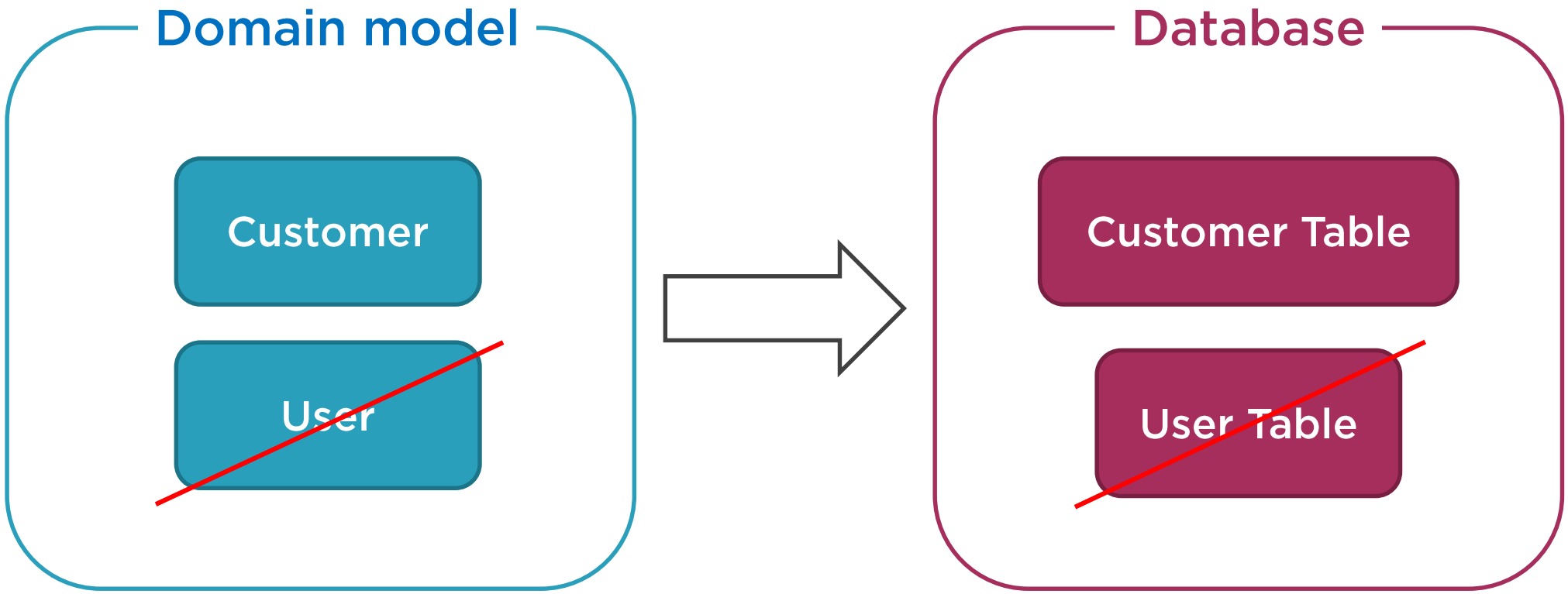
- Poor substitution for Source Control System
- Can’t track history
- Hard to deal with merge conflicts

Don’t Use Shared Development Database



- No single source of truth
- Can get out of sync
- Hard to run integration tests

Refactor Your Database Frequently



Database Delivery Basic Principles

- Store all changes in the source control system
- Integrate the DB into your CI process
- Refactor the database schema frequently
- Programmers should have a separate DB instance

State-based and Migration-based Approaches

State-based approach

Migration-based approach

State-based Approach

Maintain a snapshot of the database

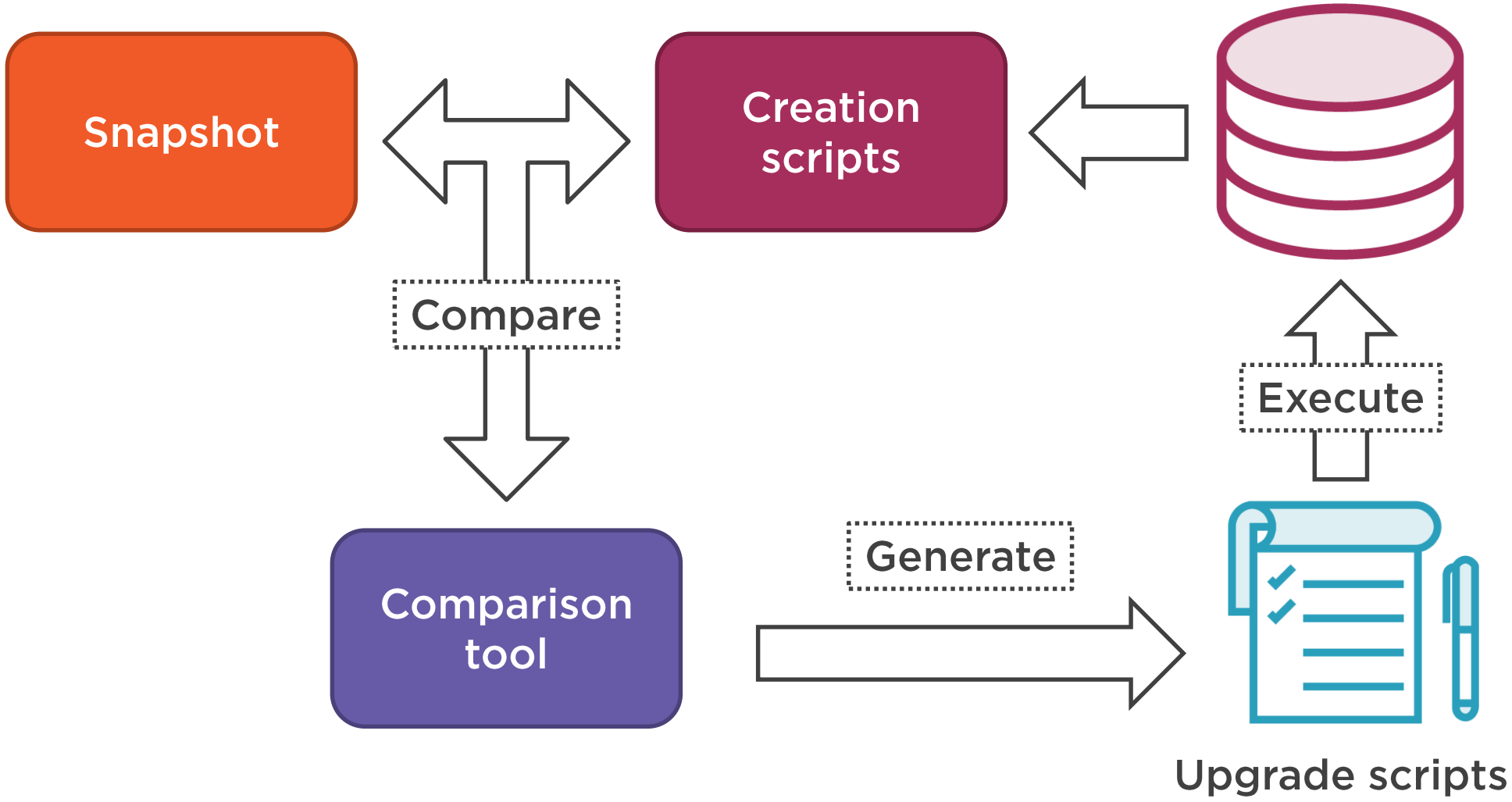
Table:

```sql
CREATE TABLE [dbo].[User] (
    [UserID]       BIGINT         NOT NULL,
    [FirstName]    NVARCHAR (100) NOT NULL,
    [LastName]     NVARCHAR (100) NOT NULL
    CONSTRAINT [PK_User] PRIMARY KEY CLUSTERED ([UserID] ASC));
```

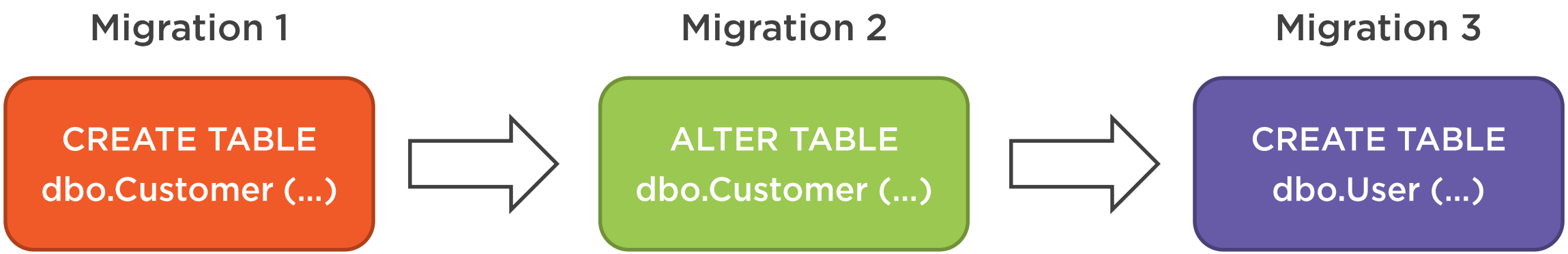
Stored procedure:

```sql
CREATE PROCEDURE [dbo].[sp_SelectUsers]
AS
BEGIN
       SELECT u.UserID, u.FirstName, u.LastName
       FROM dbo.[User] u
END
```

State-based Approach



Migration-based Approach



State-based and Migration-based Approaches

	State of the database	Migration mechanism
State-based approach	✓ Explicit	✗ Implicit
Migration-based approach	✗ Implicit	✓ Explicit

Problem Domain Introduction

Column Name	Data Type	Allow Nulls
UserID	bigint	☐
Name	nvarchar(200)	☐
PrimaryEmail	nvarchar(256)	☐
Status	nvarchar(50)	☐
		☐

UserID	Name	PrimaryEmail	Status
1	John Doe	john.doe@gmail.com	Regular
2	Jane Smith	jane@yahoo.com	Preferred
3	Simon Richardson	richardson@gmail.com	Gold

```
PROCEDURE [dbo].[sp_SelectUsers]
AS
BEGIN
    SELECT u.UserID, u.Name
FROM dbo.[User] u
END
```

Problem Domain Introduction

- Rename PrimaryEmail to Email
- Modify the stored procedure
- Split Name into FirstName and LastName
- Extract a separate UserStatus table

Summary

Database delivery challenges

- Data must be preserved at all times
- Integration database should remain backward compatible
- Inconsistencies across different environments
- Resolving merge conflicts

Basic database delivery principles

- Keep everything in the source control
- Incorporate the DB changes into your CI process
- Separate development DB per each programmer
- Refactor your database along with the application code

Summary

Overview of the state-based and migration-based approaches

Problem domain introduction

In the Next Module

Applying the state-based approach