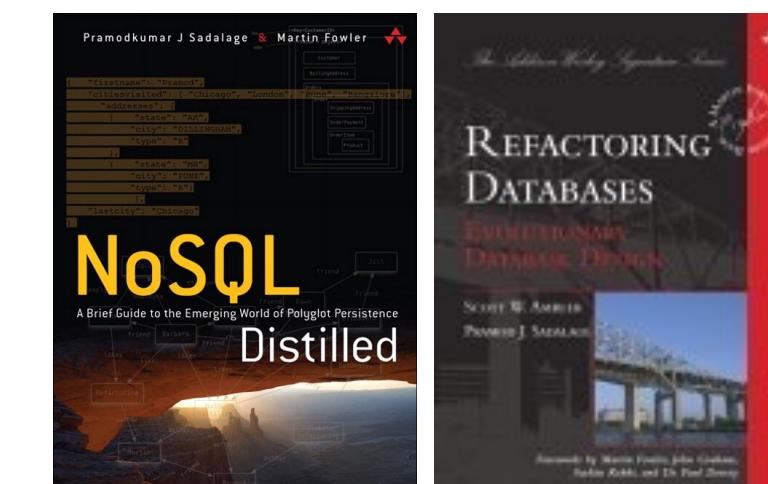


Evolutionary Database Design & Architecture

Pramod Sadalage

ThoughtWorks Inc.



Why Evolutionary?

Businesses are in
constant change

Priorities change
constantly

Learning and
responding to customer
needs is critical

Design no longer
predictive phase but
continuously evolving
activity

Making decisions
before they're
necessary is waste

Product requirements
evolve so database
needs to evolve

Deal with emerging
design

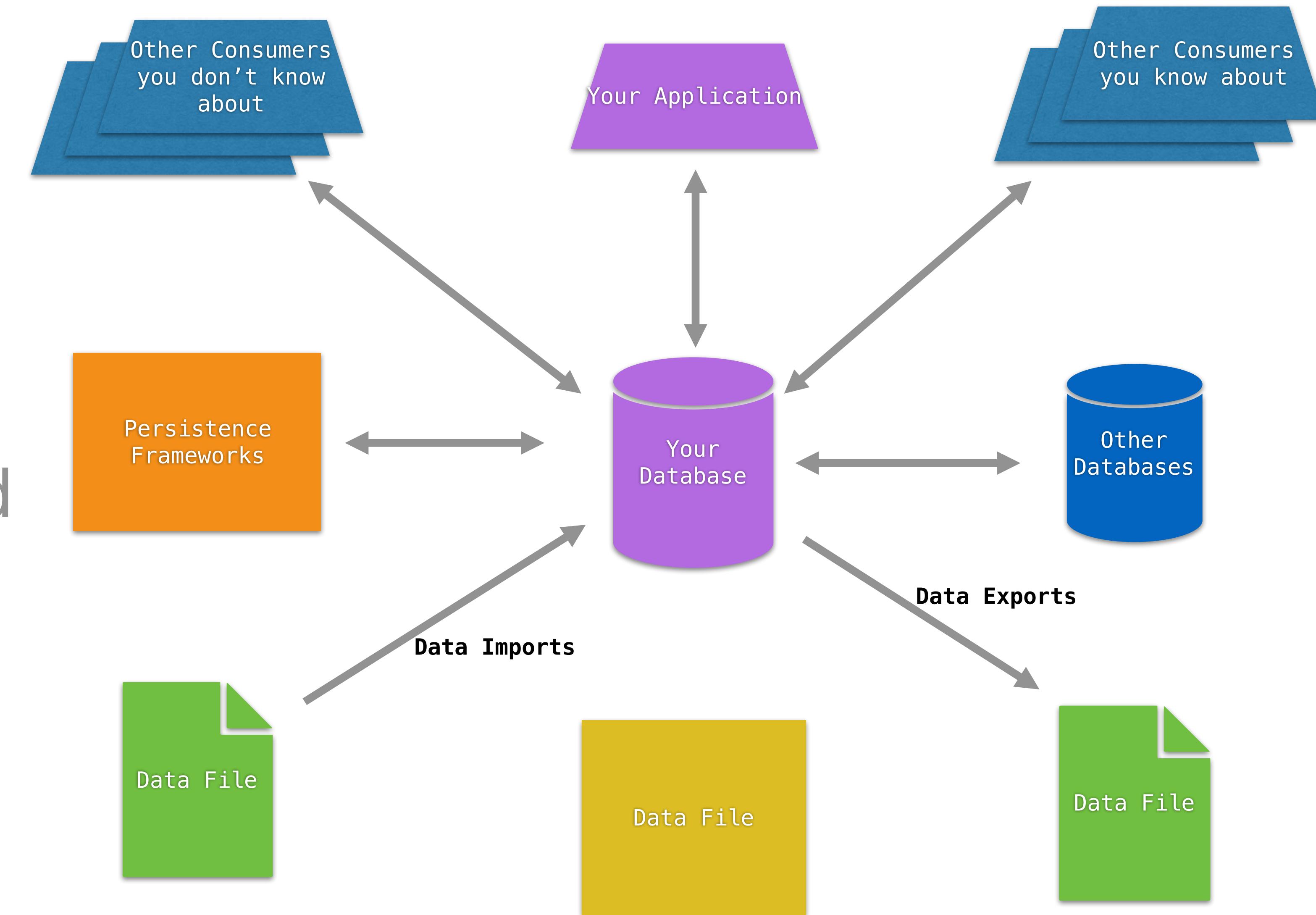
Evolving Databases

**“Refactoring is a small
change to your source
code that improves its
design without changing
its semantics”** --*Martin
Fowler*

A database refactoring is a small change to your database schema (the DDL, data, and DB code) which improves its design without changing its semantics.

A database refactoring improves its design while retaining both its **behavioral** and **informational semantics**.

Its Hard



Structural Change the table structure of your database schema

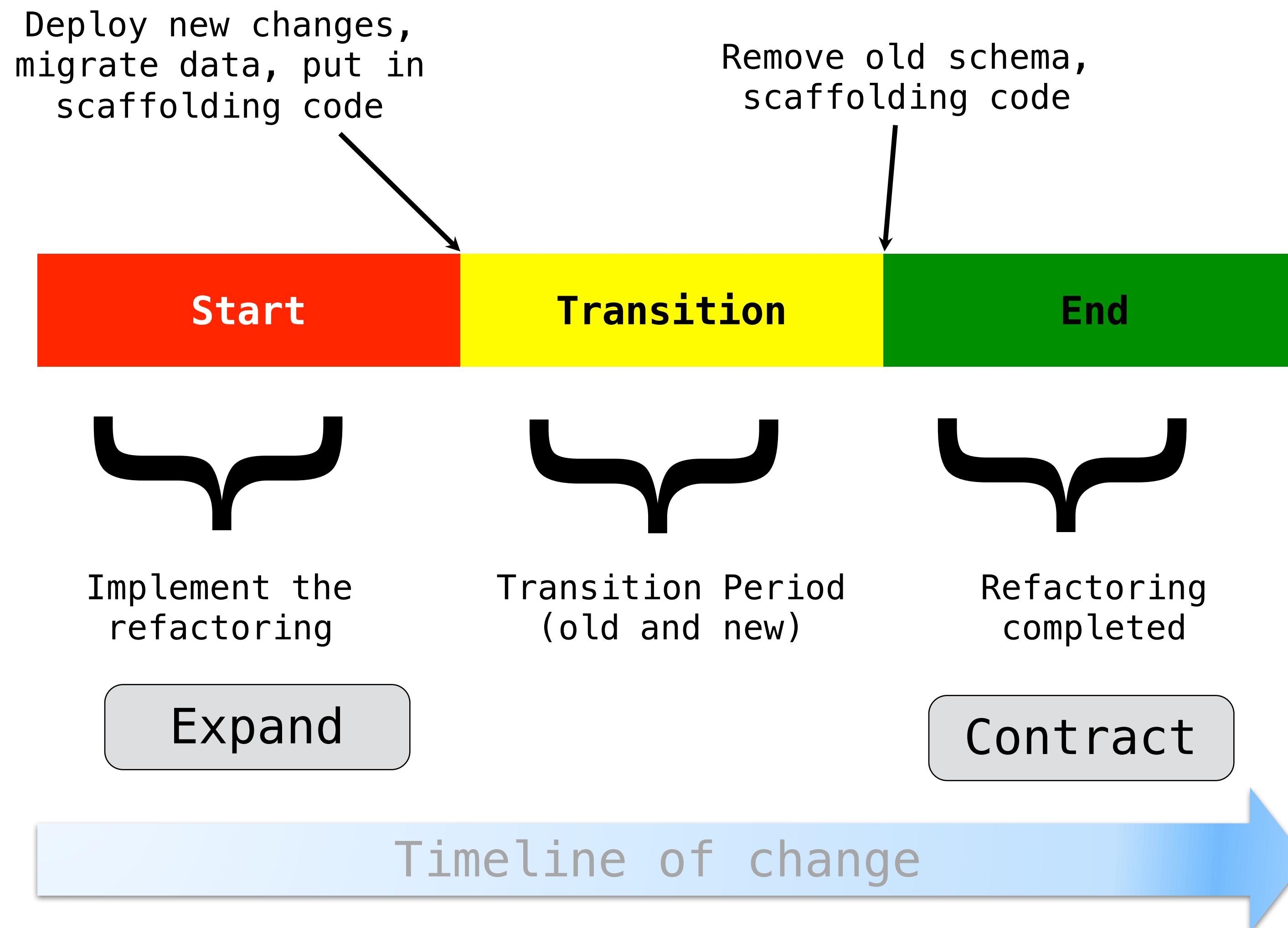
Data Quality Improve and/or ensure the consistency and usage of the values stored within the database

Architectural Improve the overall manner in which external programs interact with a database

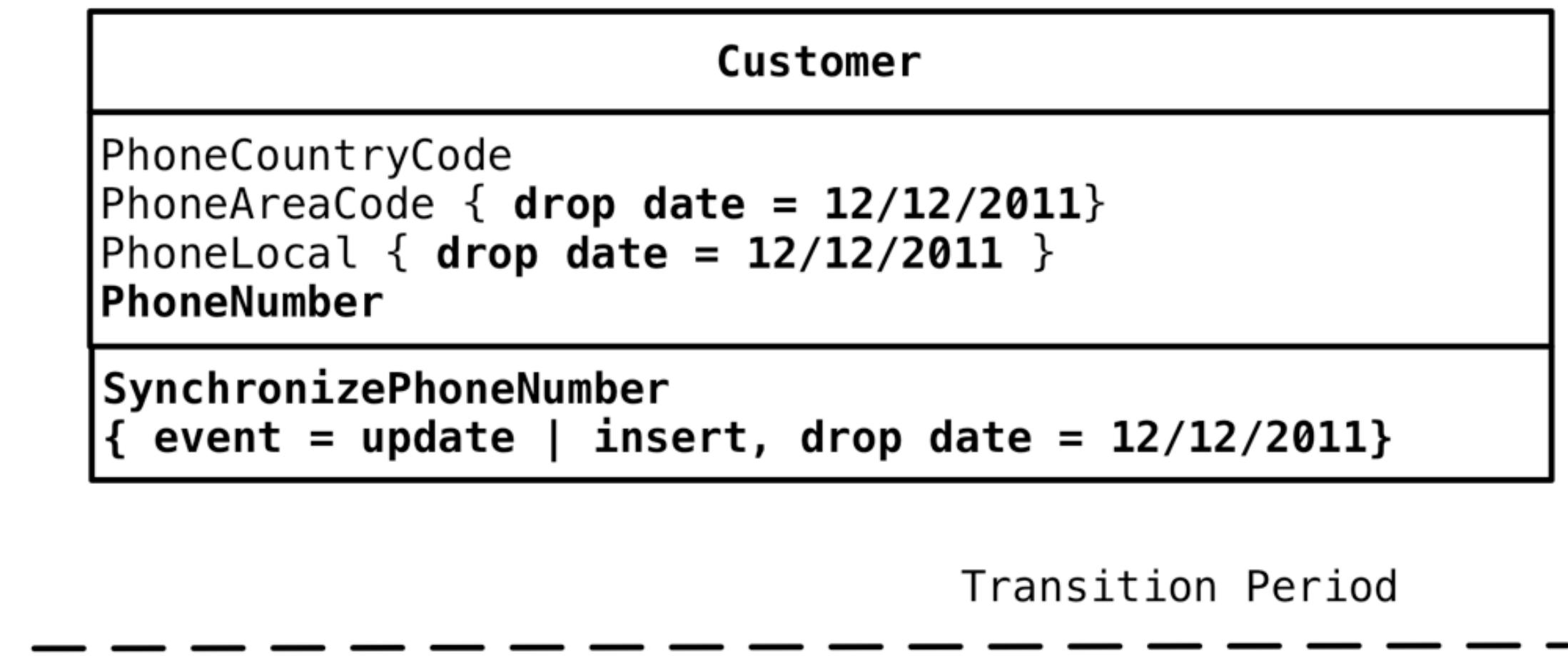
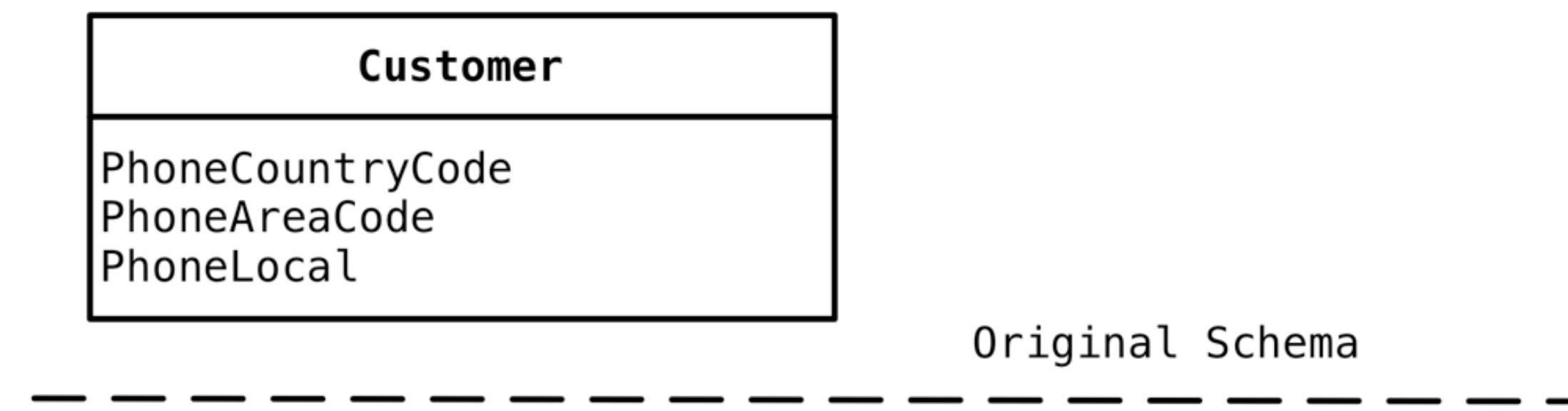
Method Apply refactoring to database code

Transformations Change the semantics of your database schema by adding new features

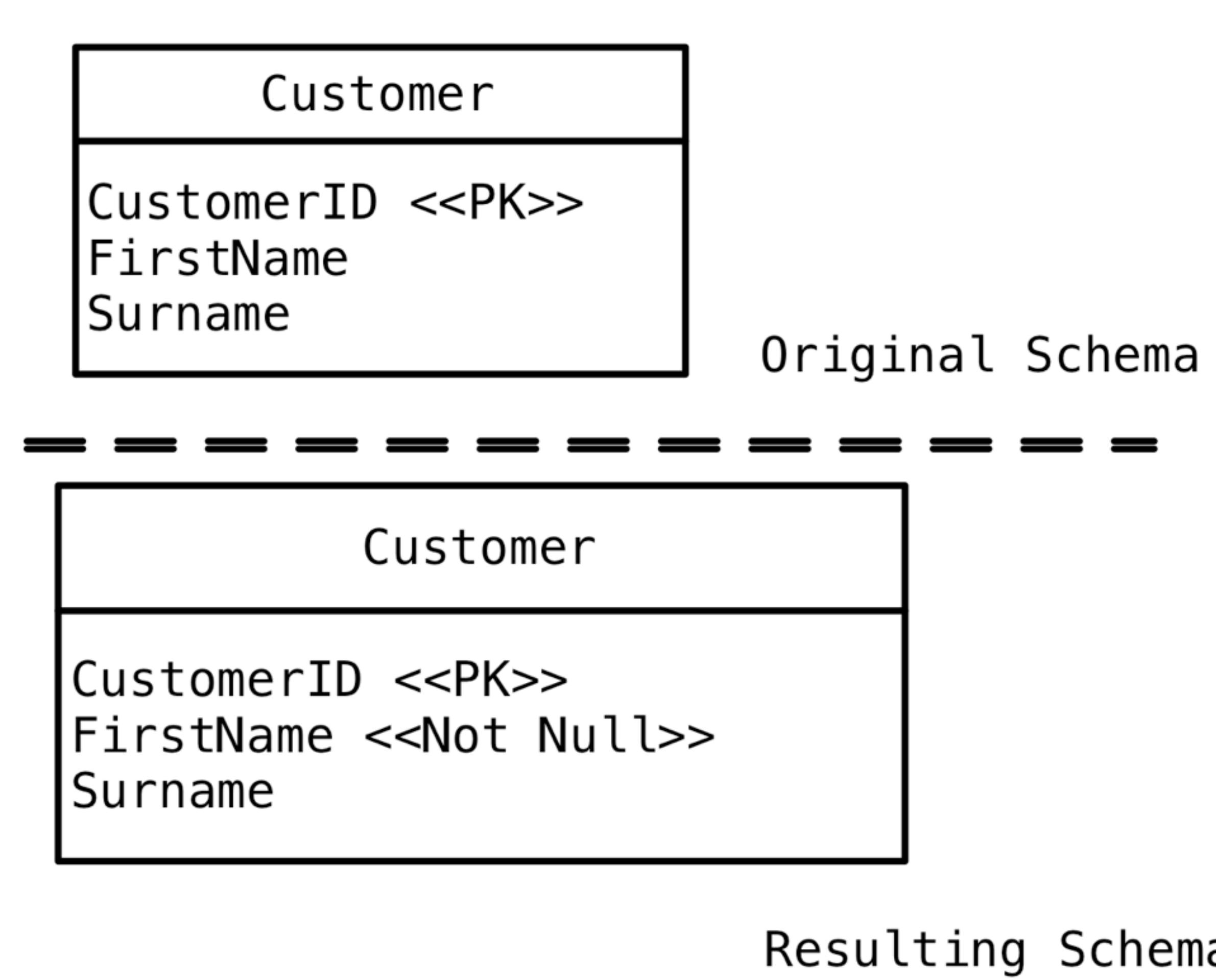
Strangler pattern applied to database



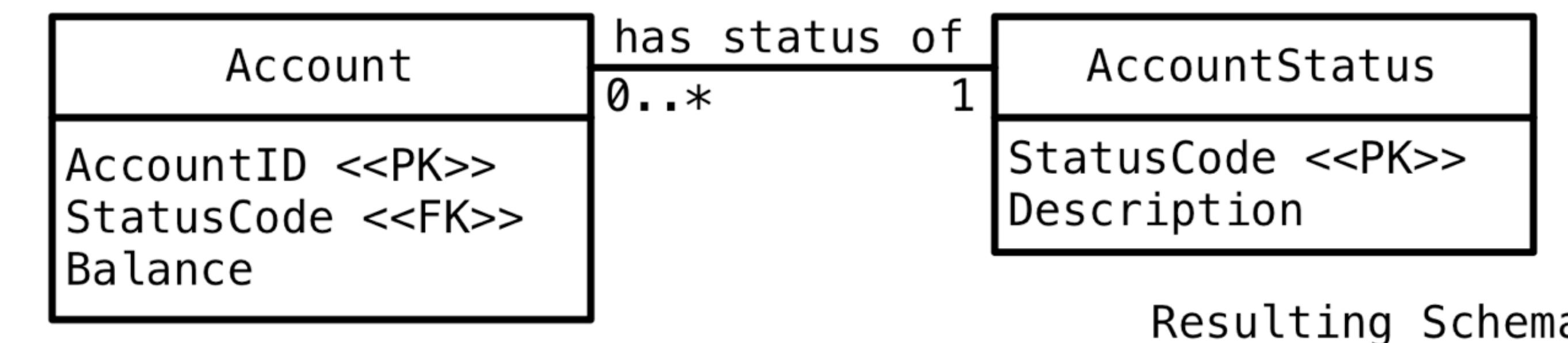
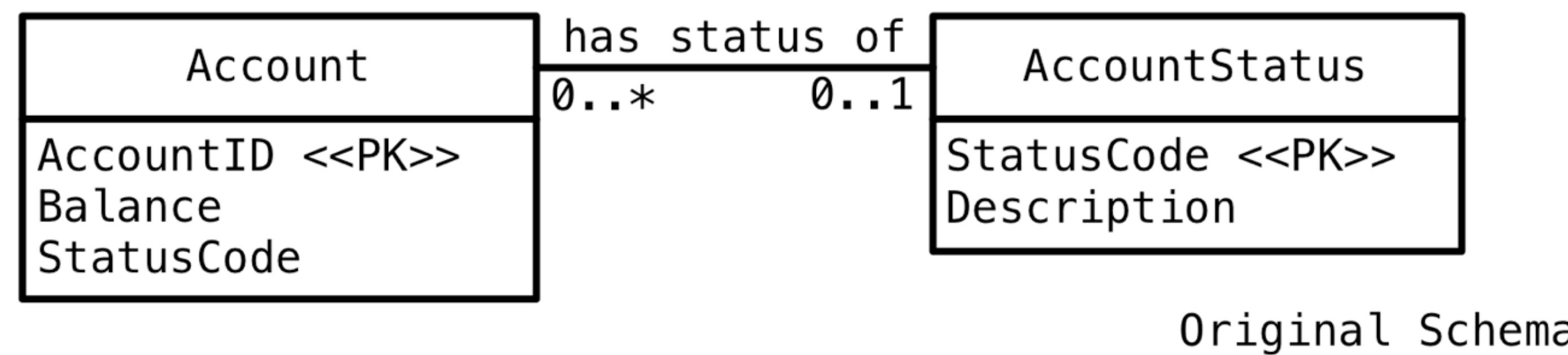
Merge Columns



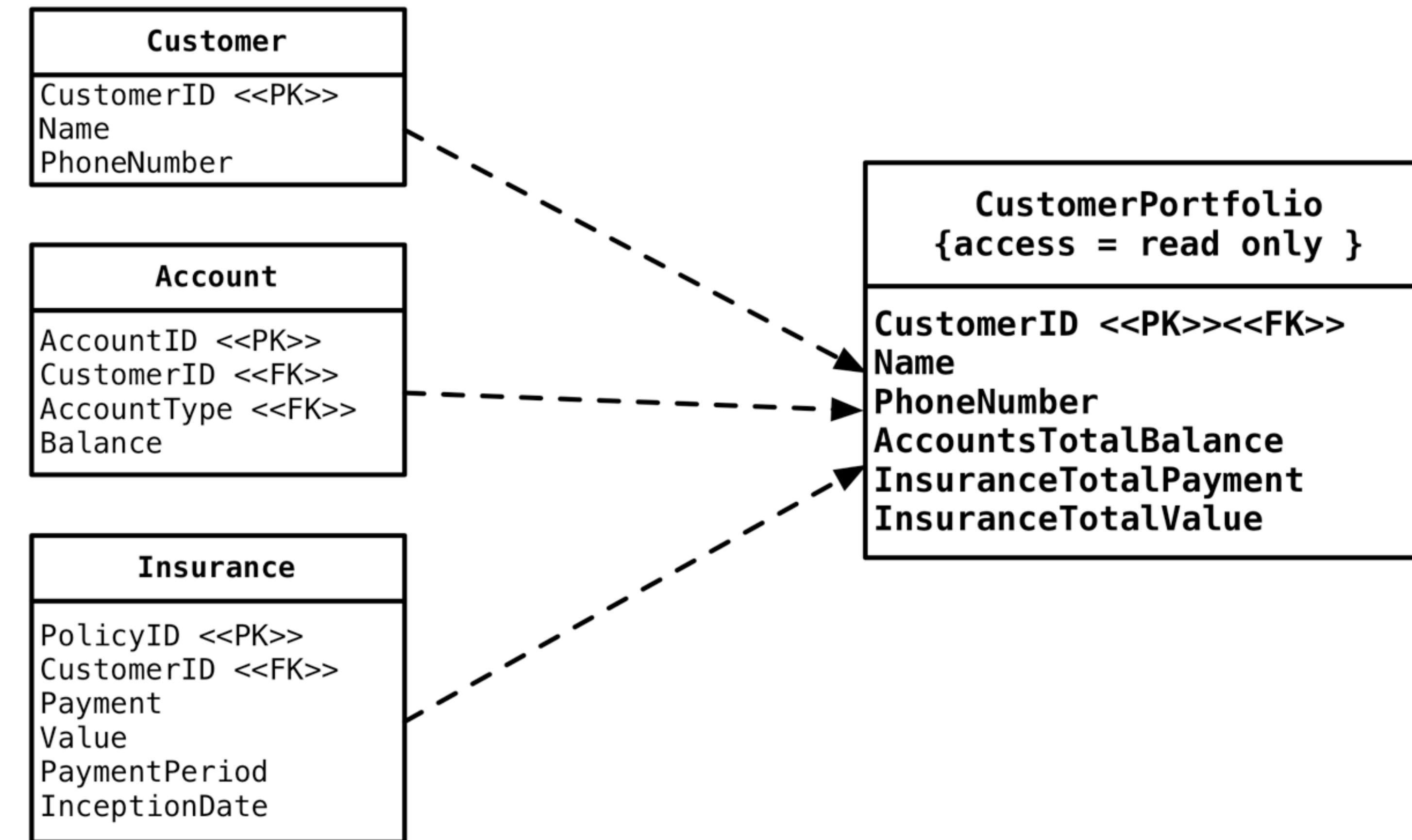
Make Column Non-Nullable



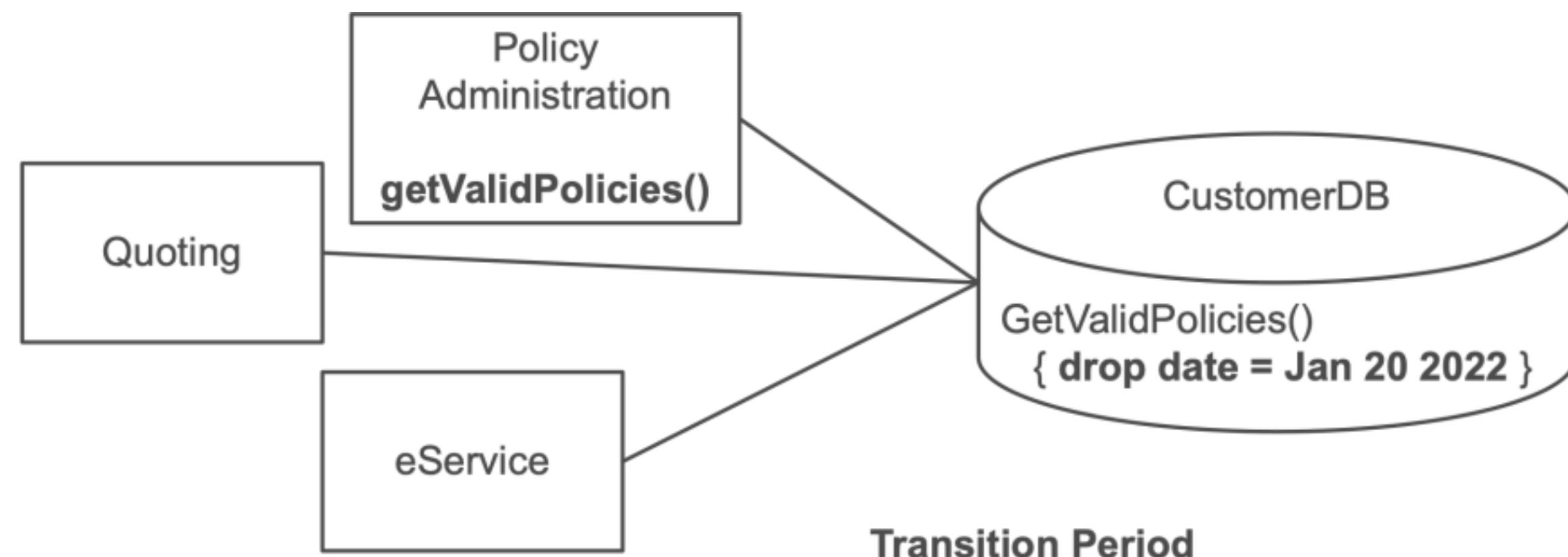
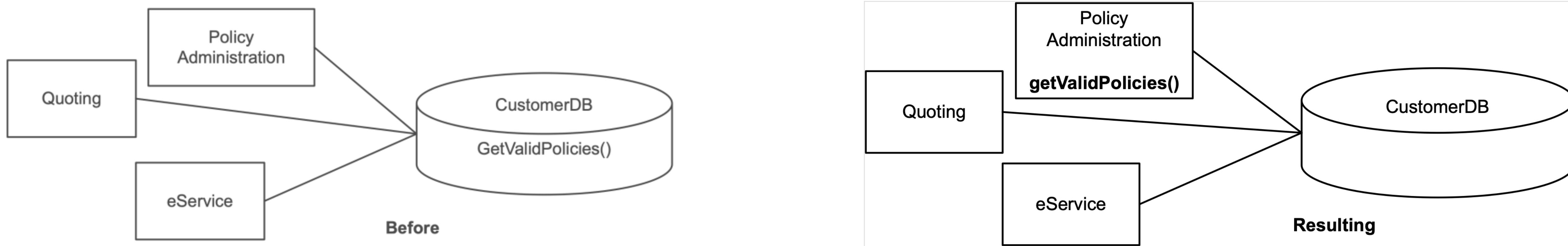
Add Foreign Key Constraint



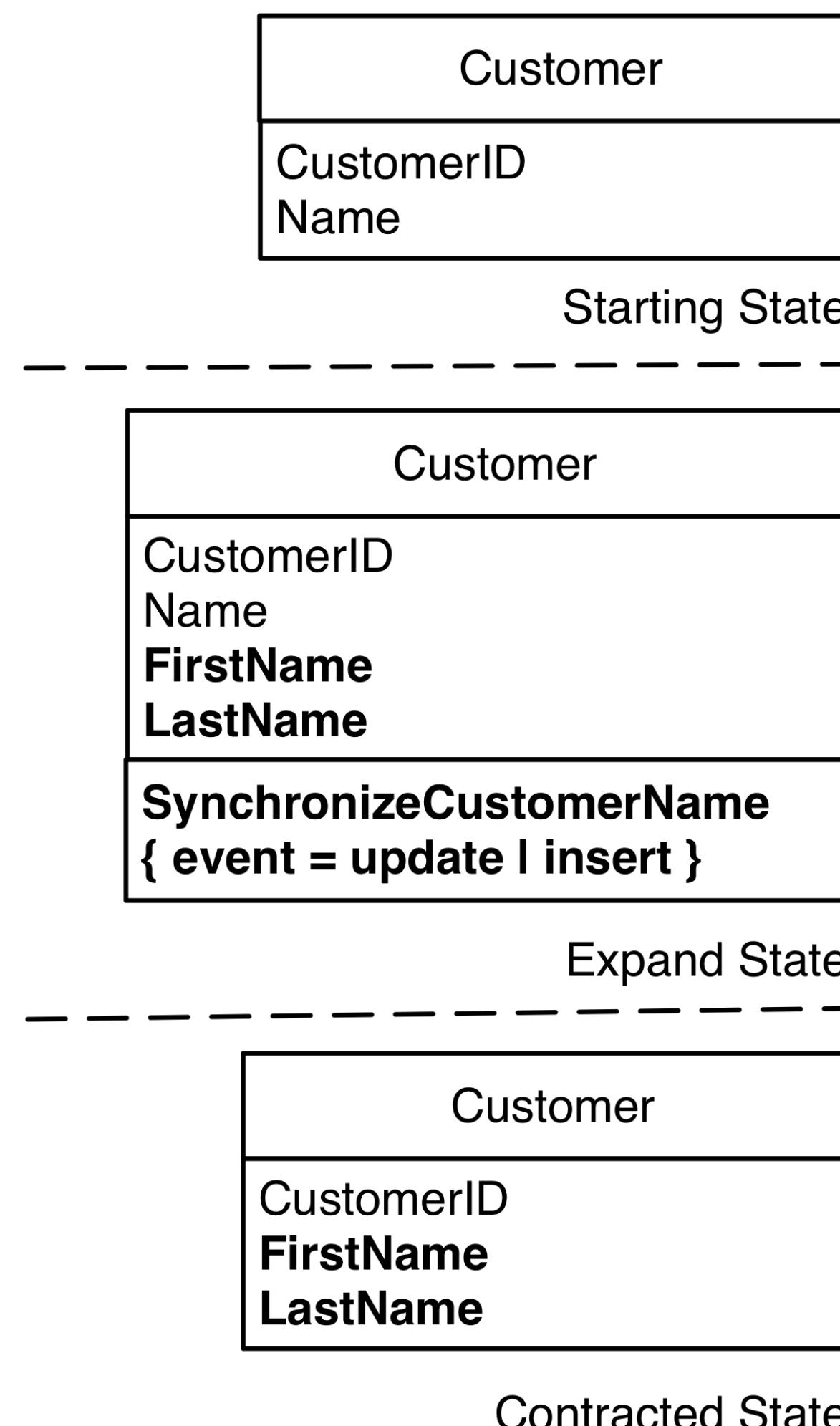
Introduce Read Only Table



Migrate method from database



Expand contract an example



Start

name = “Pramod Sadalage”

Expand

name = “Pramod Sadalage”

firstname = “Pramod”

lastname = “Sadalage”

Contract

firstname = “Pramod”

lastname = “Sadalage”

More on expand contract

Start

name = “Pramod Sadalage”

Expand

Without data migration

name = “Pramod Sadalage”
firstname = null
lastname = null

With data migration

name = “Pramod Sadalage”
firstname = “Pramod”
lastname = “Sadlage”

Simple Scenario

```
ALTER TABLE customer ADD firstname VARCHAR2(60);  
ALTER TABLE customer ADD lastname VARCHAR2(60);
```

Synchronize Data

```
ALTER TABLE customer ADD firstname VARCHAR2(60);
ALTER TABLE customer ADD lastname VARCHAR2(60);
```

```
CREATE OR REPLACE TRIGGER synchronizename
BEFORE INSERT OR UPDATE
ON customer
REFERENCING OLD AS OLD NEW AS NEW
FOR EACH ROW
BEGIN
  IF :NEW.name IS NULL THEN
    :NEW.name := :NEW.firstname||' '||:NEW.lastname;
  END IF;
  IF :NEW.name IS NOT NULL THEN
    :NEW.firstname := extractfirstname(:NEW.name);
    :NEW.lastname := extractlastname(:NEW.name);
  END IF;
END;
/
```

Migrate and Synchronize Data

```
ALTER TABLE customer ADD firstname VARCHAR2(60);  
ALTER TABLE customer ADD lastname VARCHAR2(60);
```

```
UPDATE customer  
  set firstname = extractfirstname (name);  
UPDATE customer  
  set lastname = extractlastname (name);
```

```
CREATE OR REPLACE TRIGGER synchronizename  
BEFORE INSERT OR UPDATE
```

....

Contract

```
ALTER TABLE customer DROP COLUMN  
name;
```

CONTRACT

```
ALTER TABLE customer SET UNUSED  
    name;
```

When drop
column could
take forever

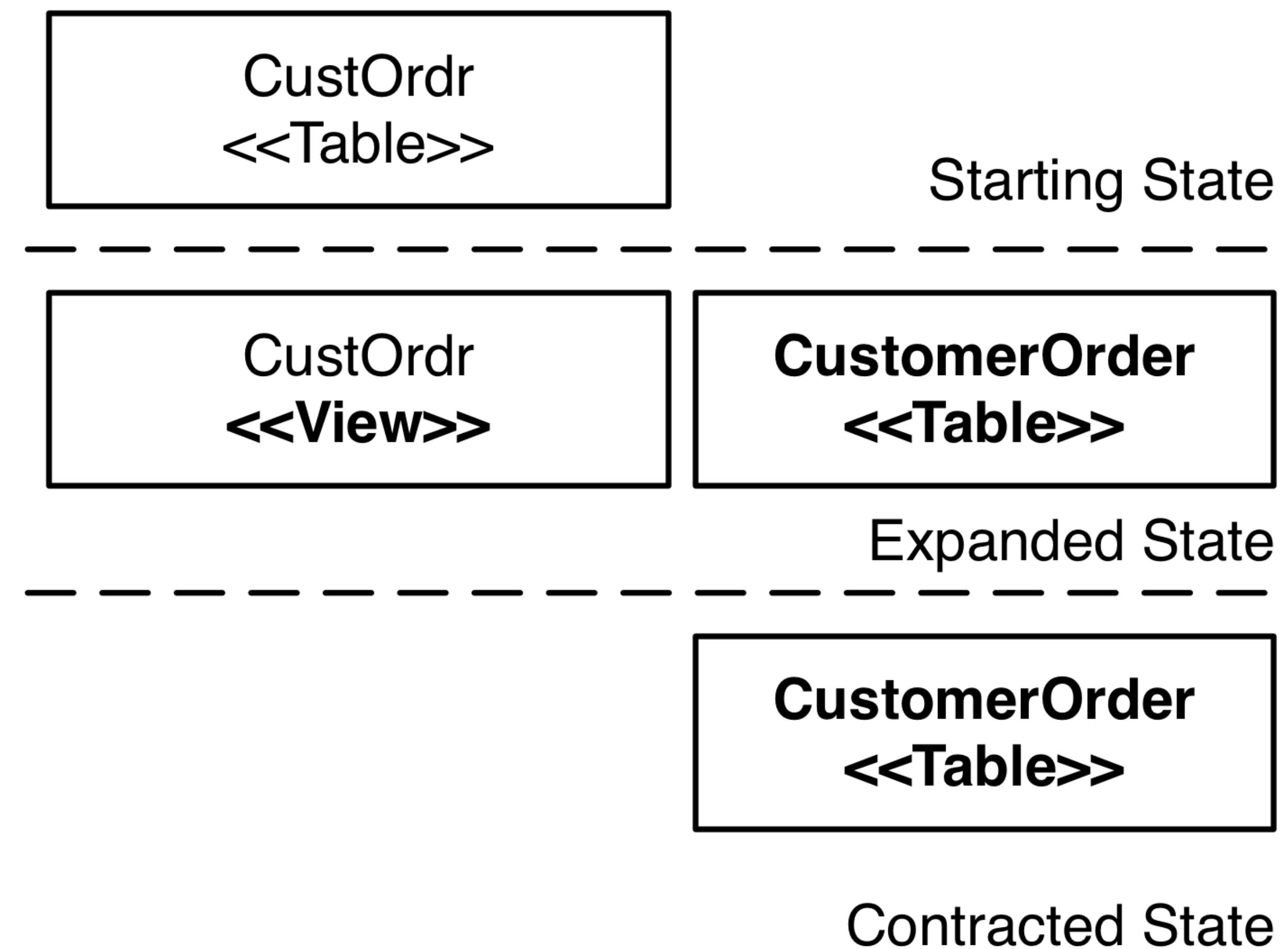
Keep legacy apps happy

```
ALTER TABLE customer DROP COLUMN name;
```

Virtual column
in Oracle,
Generated Column
in MySQL.

```
ALTER TABLE customer ADD  
(name AS (generatename (firstname, lastname)) );
```

Another example

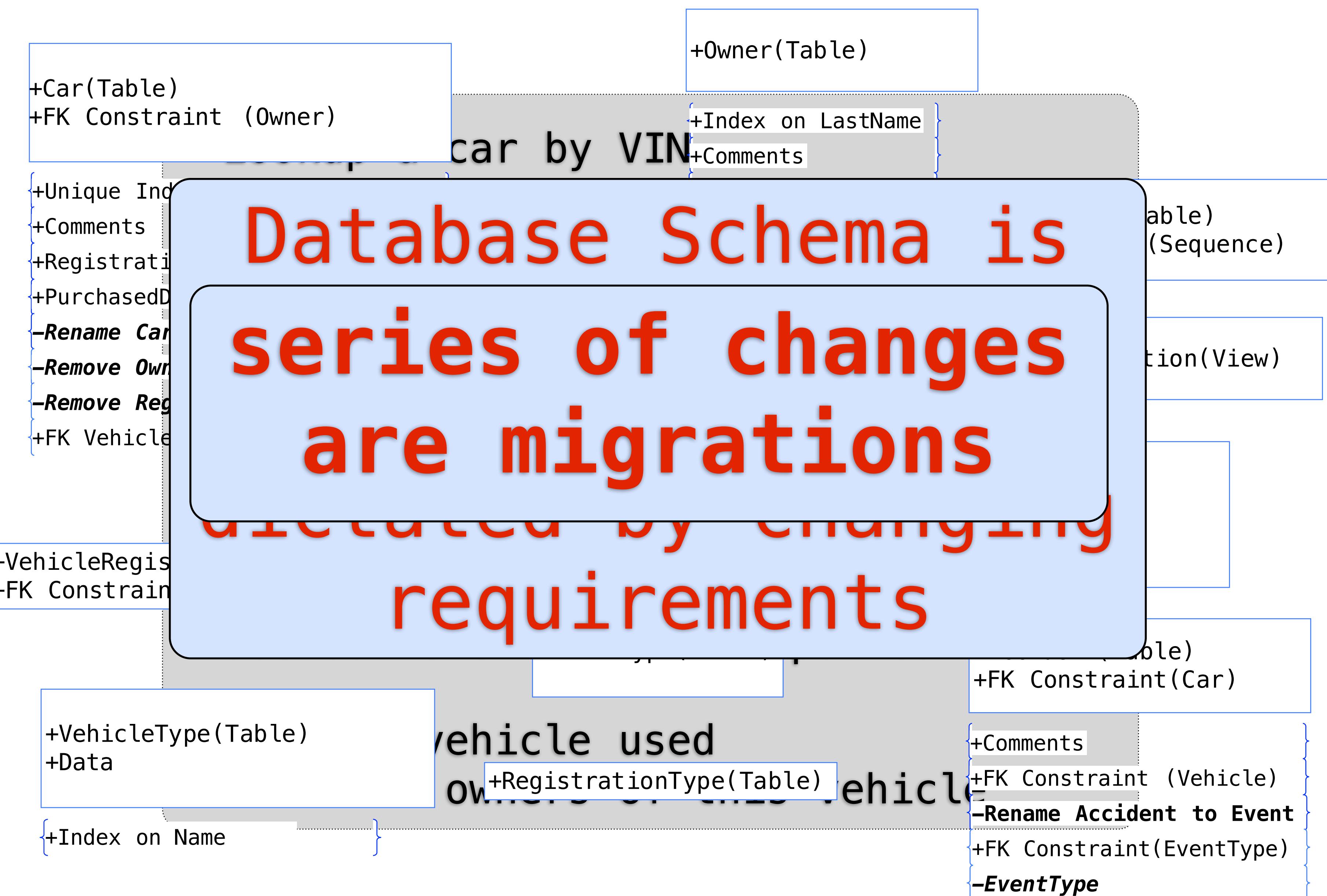


Refactoring Enablers

- Create interfaces in the database
- DB Should be able to handle multiple versions of the application
- Wrap tables with views
- Create calculated columns
- Create triggers to sync data
- Keeping Old and New working, improves uptime and ability to evolve individual consumers
- Think of them like deprecating API or methods

Guide

- Large refactoring's are risky
- Sequence of many small refactoring's can create the desired change
- Migration scripts should be checked in and run on local dev/ci/qa/uat/prod etc.
- Changes to data are also migrations
- Changes to database code can be treated as any other code, recreate all database code for every change.



An example of change

```
ALTER TABLE customer ADD firstname VARCHAR2(60);  
ALTER TABLE customer ADD lastname VARCHAR2(60);
```

Practices

Without good development
Practices and Patterns
using and implementing
Evolutionary Design is
difficult.

Test the Behavior of Database

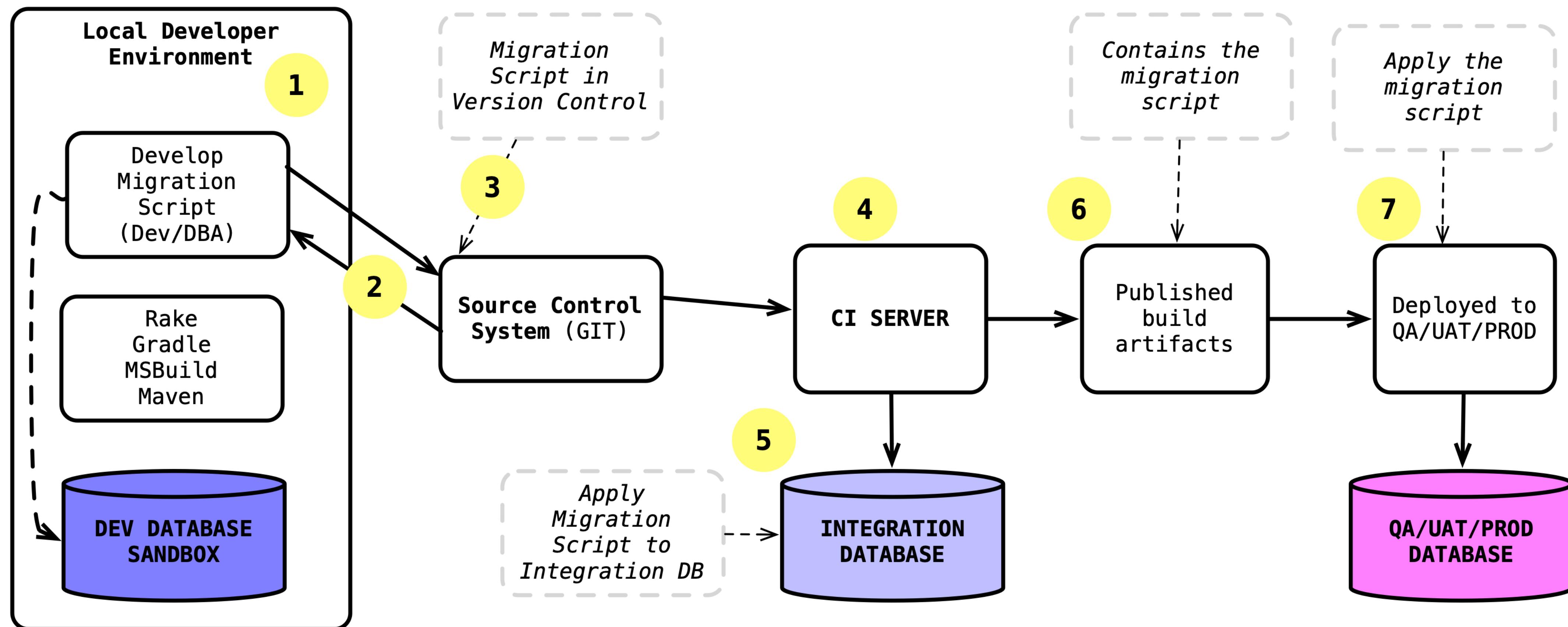
- Like objects, database has behavior
- Develop database objects using BDD style tests
- Allows easy changes to the database
- Protects against changes that affect dependent functionality
- Test using the persistence layer

- shouldNotSaveDuplicateVIN()
- shouldSaveModelYear2010()
- shouldNotSaveModelYear2004()
- shouldNotSaveNullModelName()
- shouldNotSaveNullMake()
- shouldSaveMiles5000()
- shouldNotSaveMiles12000()

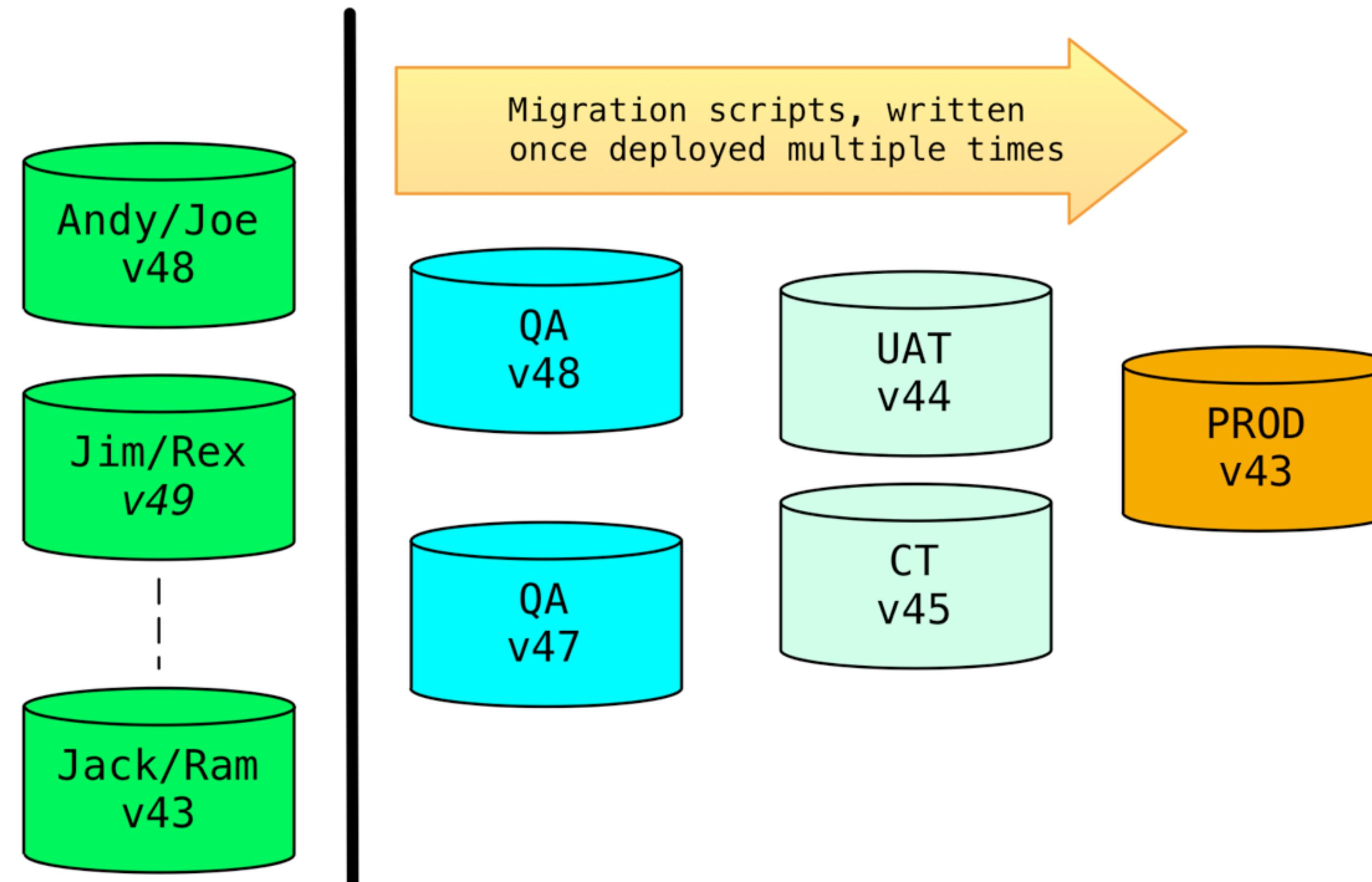
```
CREATE TABLE vehicle(
    id NUMBER(18) NOT NULL,
    vin VARCHAR2(32) NOT NULL,
    name VARCHAR2(32) NOT NULL,
    make VARCHAR2(32) NOT NULL,
    year NUMBER(4) NOT NULL,
    miles NUMBER(10) NULL,
    CONSTRAINT chk_vehicle_year_gt_2005
        CHECK (year > 2004),
    CONSTRAINT chk_vehicle_miles_lt_10001
        CHECK (miles < 10001));
CREATE UNIQUE INDEX uidx_vehicle_vin
    ON vehicle(vin);

ALTER TABLE VEHICLE ADD CONSTRAINT
    pk_vehicle PRIMARY KEY (id);
```

CI/CD database changes



Automate Deployment





Deploy Frequently

Q&A

@pramodsadalage

sadalage.com

databaserefactoring.com

devopsfordba.com