Evolutionary database design: Refactoring databases

Pramod Sadalage
@pramodsadalage
ThoughtWorks Inc.

SSID: Pramods Wifi Network
Password: Database#1

Motivations

- Continuous Delivery
- Collaboration - data team
- Changing requirements
- Devops - data team
- Deploying database changes

Why Continuous Delivery?

To get fast feedback from users, release frequently

Reduce risk of releasing

Learning and responding to customer needs is critical

Achieve Continuous Delivery

Close collaboration between everyone involved in delivery

Extensive automation of all parts involved in delivery process

Reduce the gap between development and operations

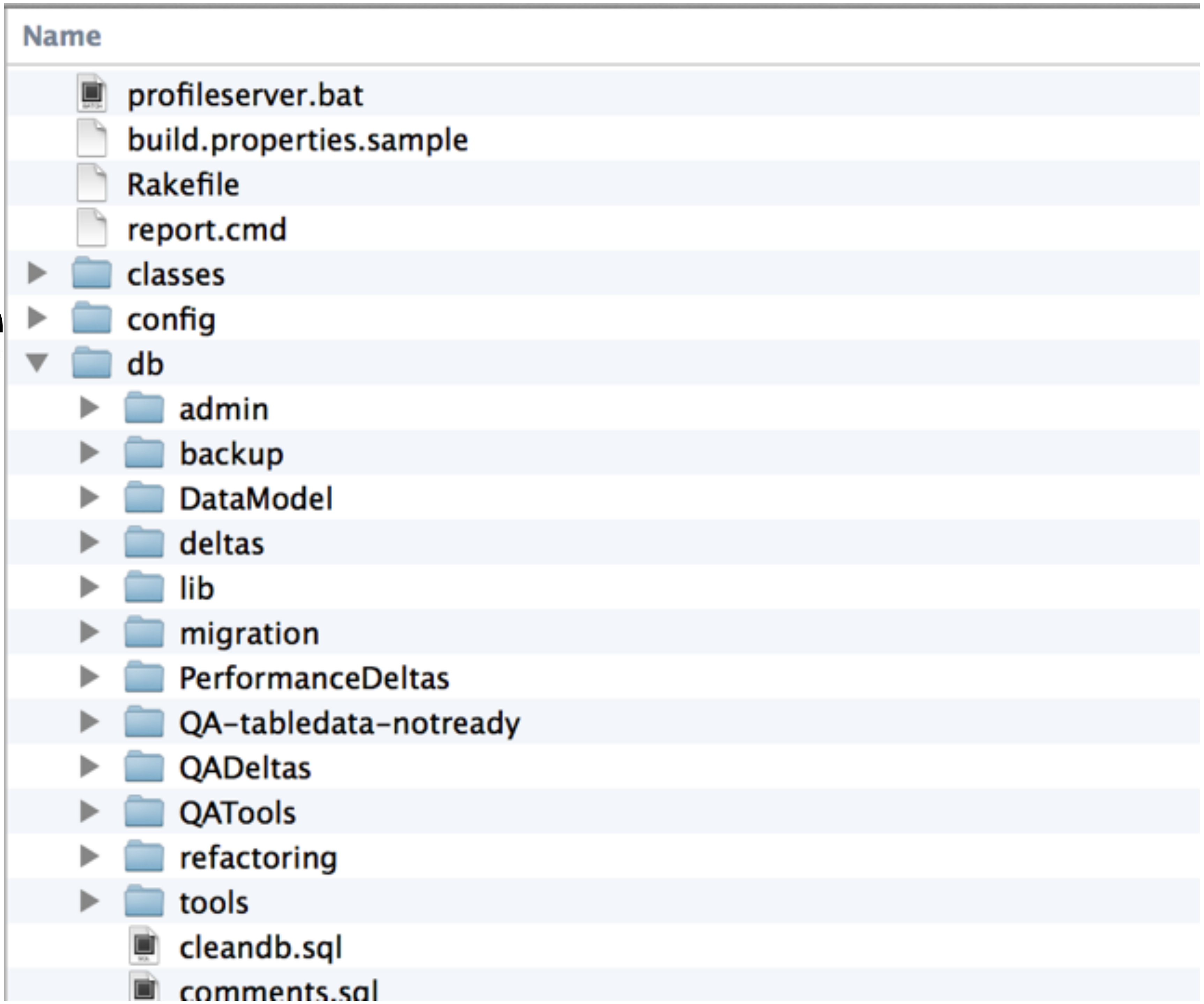
CD should involve the database team

How does continuous delivery apply to databases?

Collaboration techniques

Pair the DBA’s and Developers

Ve se


```
Name
    profileserver.bat
    build.properties.sample
    Rakefile
    report.cmd
▶   classes
▶   config
▼   db
      ▶ admin
      ▶ backup
      ▶ DataModel
      ▶ deltas
      ▶ lib
      ▶ migration
      ▶ PerformanceDeltas
      ▶ QA-tabledata-notready
      ▶ QADeltas
      ▶ QATools
      ▶ refactoring
      ▶ tools
        cleandb.sql
        comments.sql
```

Lets see our code

`github.com/pramodsadalage/database-refactoring`

Your Projects

- Is your project a single database project

- How many applications talk to your database?

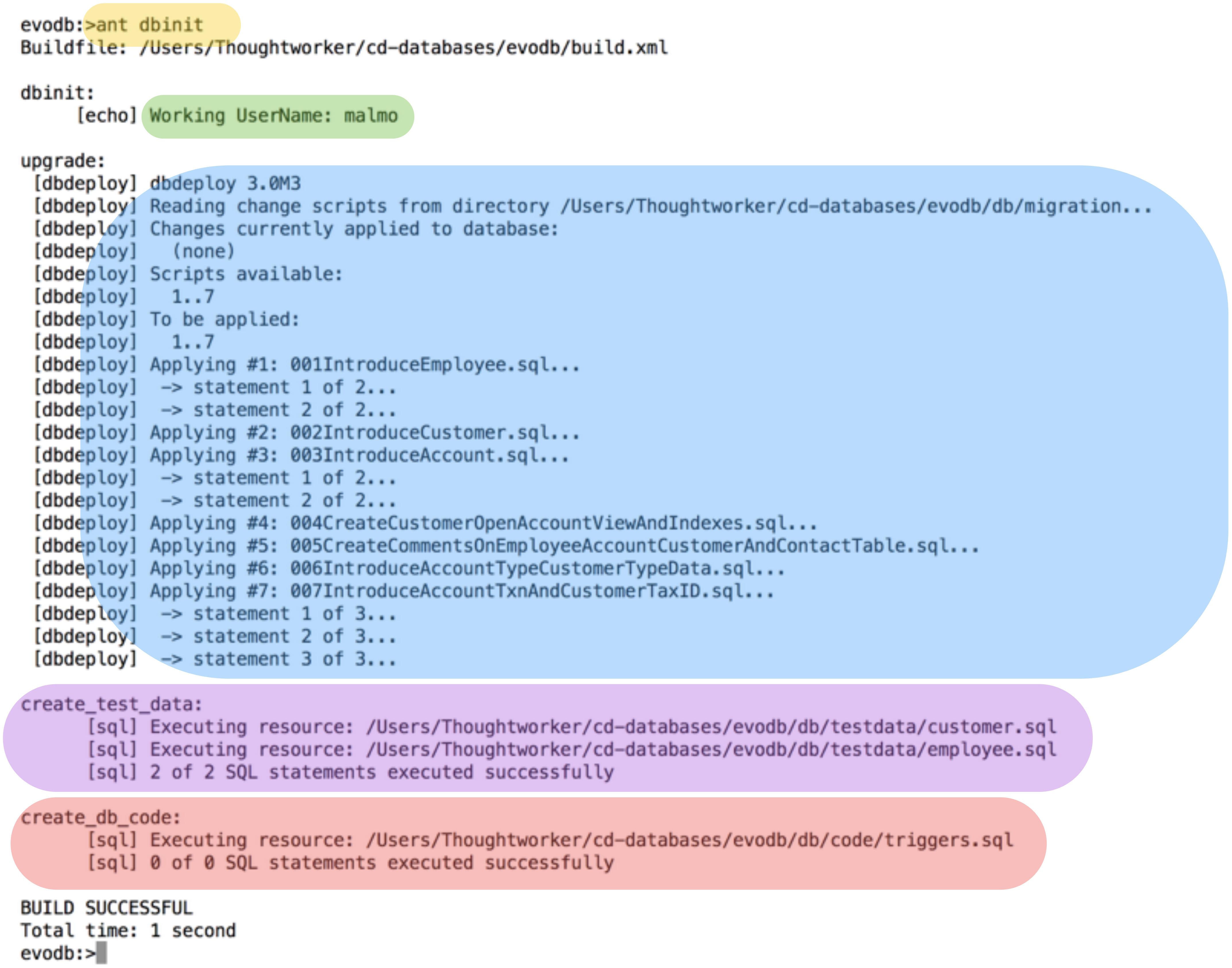
- Does your app depend on multiple databases?

- Database dependency in version control?

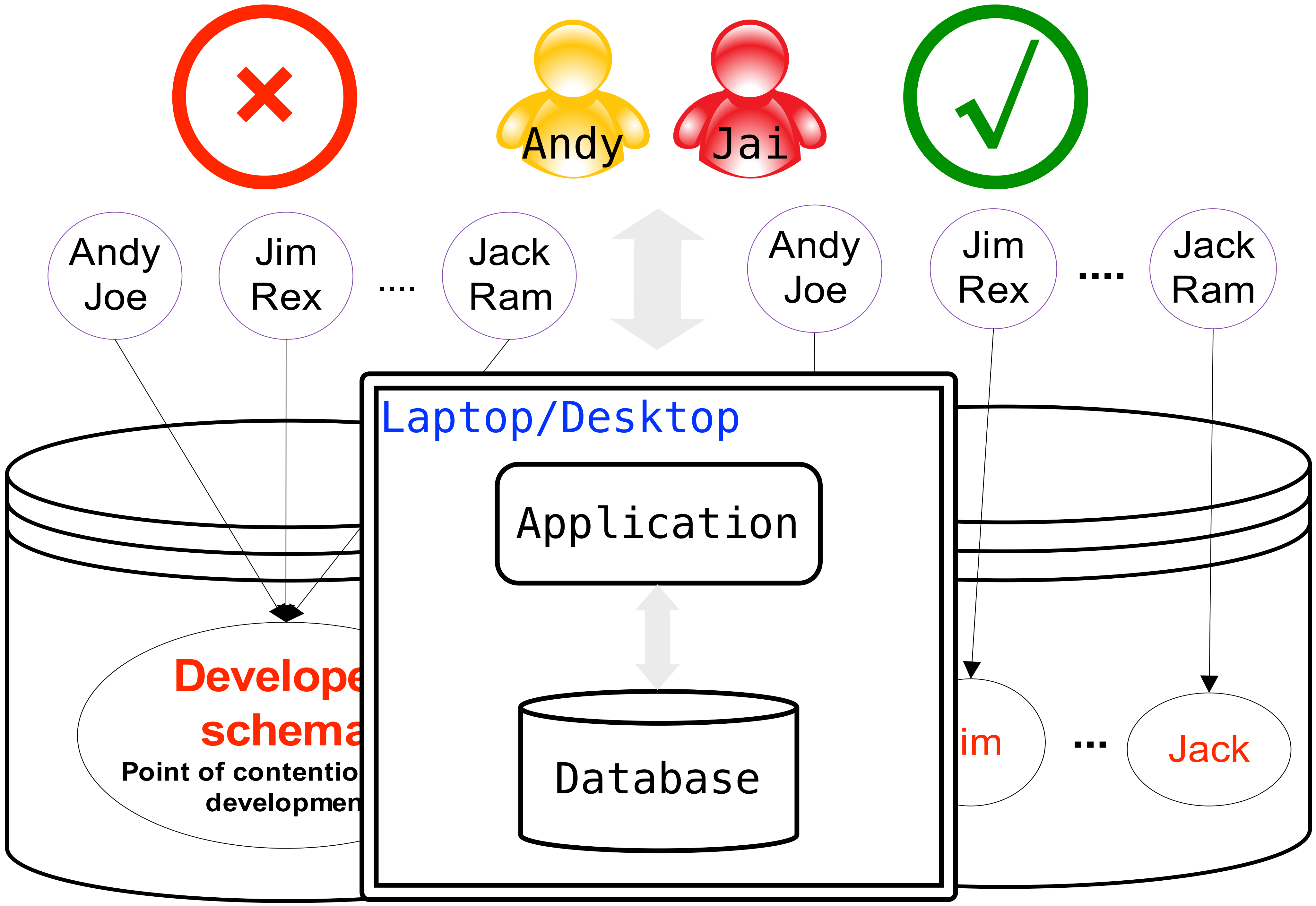
Automation techniques

Let developers provision application database

```
cd-databases:>git clone https://github.com/pramodsadalage/evodb.git
Cloning into 'evodb'...
remote: Counting objects: 115, done.
remote: Total 115 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (115/115), 7.59 MiB | 1.73 MiB/s, done.
Resolving deltas: 100% (14/14), done.
cd-databases:>cd evodb/
evodb:>cp build.properties.template build.properties
evodb:>vi build.properties
evodb:>ant create_schema
Buildfile: /Users/Thoughtworker/cd-databases/evodb/build.xml

create_schema:
     [echo] Admin UserName: system
     [echo] Creating Schema: malmo
      [sql] Executing commands
      [sql] 4 of 4 SQL statements executed successfully

BUILD SUCCESSFUL
Total time: 0 seconds
evodb:>
```



```
evodb:>ant dbinit
Buildfile: /Users/Thoughtworker/cd-databases/evodb/build.xml

dbinit:
     [echo] Working UserName: malmo

upgrade:
 [dbdeploy] dbdeploy 3.0M3
 [dbdeploy] Reading change scripts from directory /Users/Thoughtworker/cd-databases/evodb/db/migration...
 [dbdeploy] Changes currently applied to database:
 [dbdeploy]   (none)
 [dbdeploy] Scripts available:
 [dbdeploy]   1..7
 [dbdeploy] To be applied:
 [dbdeploy]   1..7
 [dbdeploy] Applying #1: 001IntroduceEmployee.sql...
 [dbdeploy]  -> statement 1 of 2...
 [dbdeploy]  -> statement 2 of 2...
 [dbdeploy] Applying #2: 002IntroduceCustomer.sql...
 [dbdeploy] Applying #3: 003IntroduceAccount.sql...
 [dbdeploy]  -> statement 1 of 2...
 [dbdeploy]  -> statement 2 of 2...
 [dbdeploy] Applying #4: 004CreateCustomerOpenAccountViewAndIndexes.sql...
 [dbdeploy] Applying #5: 005CreateCommentsOnEmployeeAccountCustomerAndContactTable.sql...
 [dbdeploy] Applying #6: 006IntroduceAccountTypeCustomerTypeData.sql...
 [dbdeploy] Applying #7: 007IntroduceAccountTxnAndCustomerTaxID.sql...
 [dbdeploy]  -> statement 1 of 3...
 [dbdeploy]  -> statement 2 of 3...
 [dbdeploy]  -> statement 3 of 3...

create_test_data:
      [sql] Executing resource: /Users/Thoughtworker/cd-databases/evodb/db/testdata/customer.sql
      [sql] Executing resource: /Users/Thoughtworker/cd-databases/evodb/db/testdata/employee.sql
      [sql] 2 of 2 SQL statements executed successfully

create_db_code:
      [sql] Executing resource: /Users/Thoughtworker/cd-databases/evodb/db/code/triggers.sql
      [sql] 0 of 0 SQL statements executed successfully

BUILD SUCCESSFUL
Total time: 1 second
evodb:>
```



Andy
Jai
Andy Joe
Jim Rex
....
Jack Ram
Andy Joe
Jim Rex
....
Jack Ram
Laptop/Desktop
Application
Database
Develope
schema
Point of contentio
developmen
im
...
Jack

Lets build our schema

```
copy template.build.properties to
build.properties
```

```
Edit build.properties
```

```
ant createschema
```

```
ant dropschema
```

Lets populate our schema

```
./flyway migrate

./flyway info

./flyway validate

ant dbset? what does it do
```

Existing Legacy Database

-Extract the schema and make that your baseline

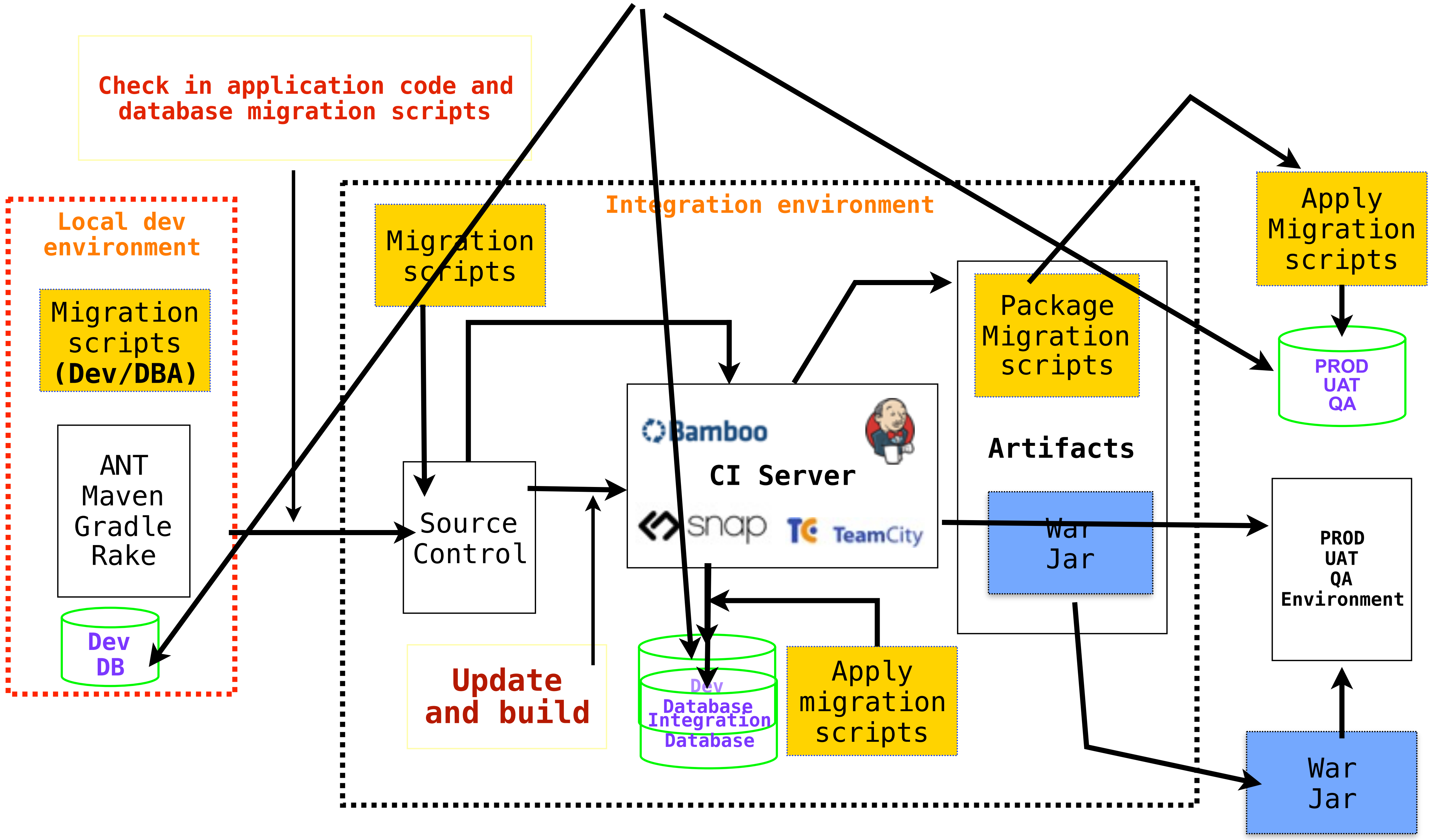
-How complex is your setup?

-Too many databases, schemas, linked schemas etc?

-Is the physical architecture too complex?

Continuously Integrate database changes

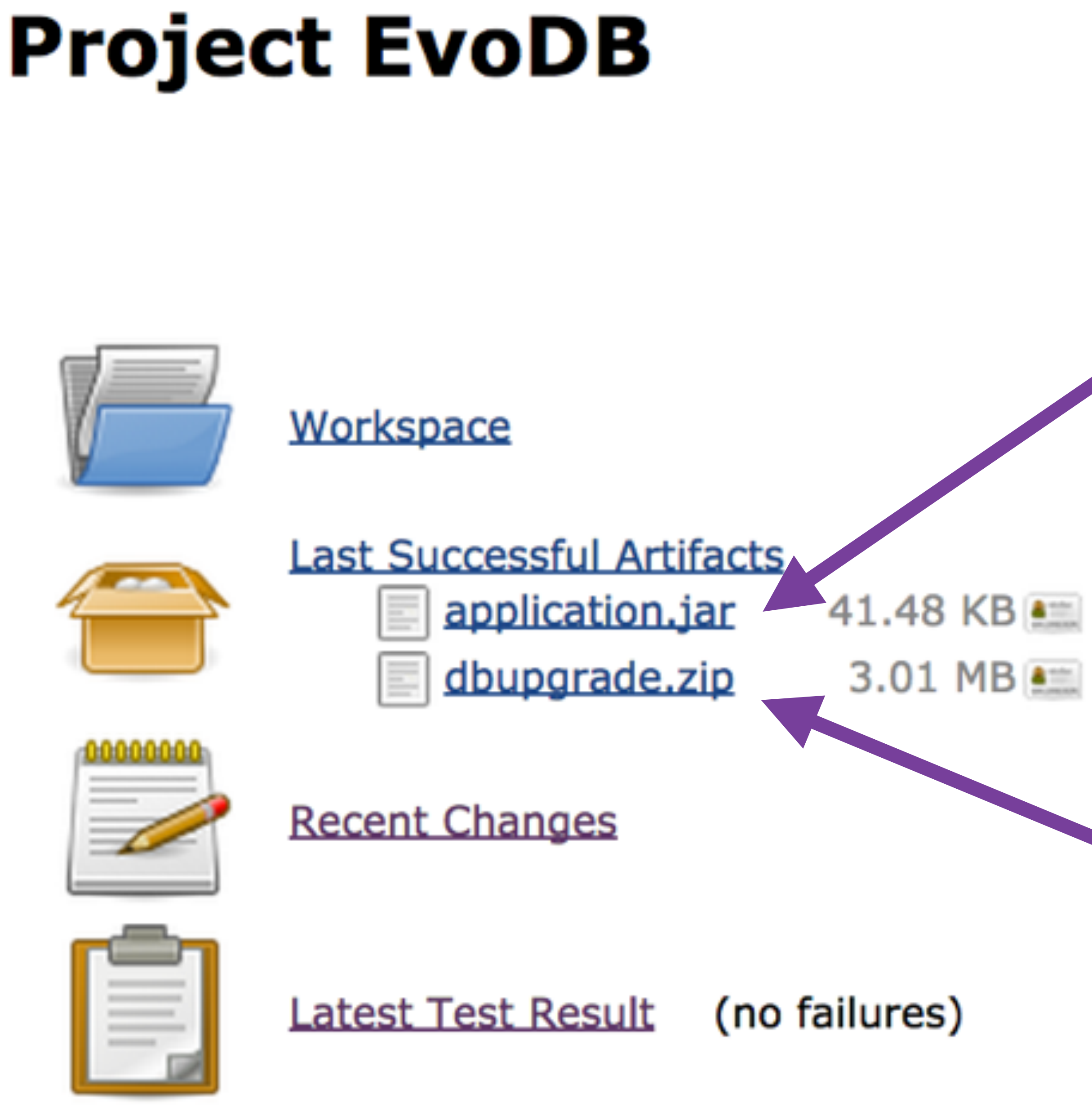
Database changes applied by DBA Team



Benefits of CI with databases

- Test application code and database at one place
- Generate code and database artifacts
- Integrate application and database changes in an independent environment
- Show current state of application and database to all

DB artifacts in CI



Project EvoDB

Workspace

Last Successful Artifacts

application.jar 41.48 KB

dbupgrade.zip 3.01 MB

Recent Changes

Latest Test Result (no failures)

application artifact for build <<n>>

database artifact for build <<n>>

Tracking Changes

- Each change is a delta/migration script
- Migration scripts are development time activity not deployment time tasks
- Package migration scripts for automated deployment
- Same scripts for: developers, QA, UAT and Production

http://10.60.0.75:8080/

Lets one of us make a change and push.

Apply change locally

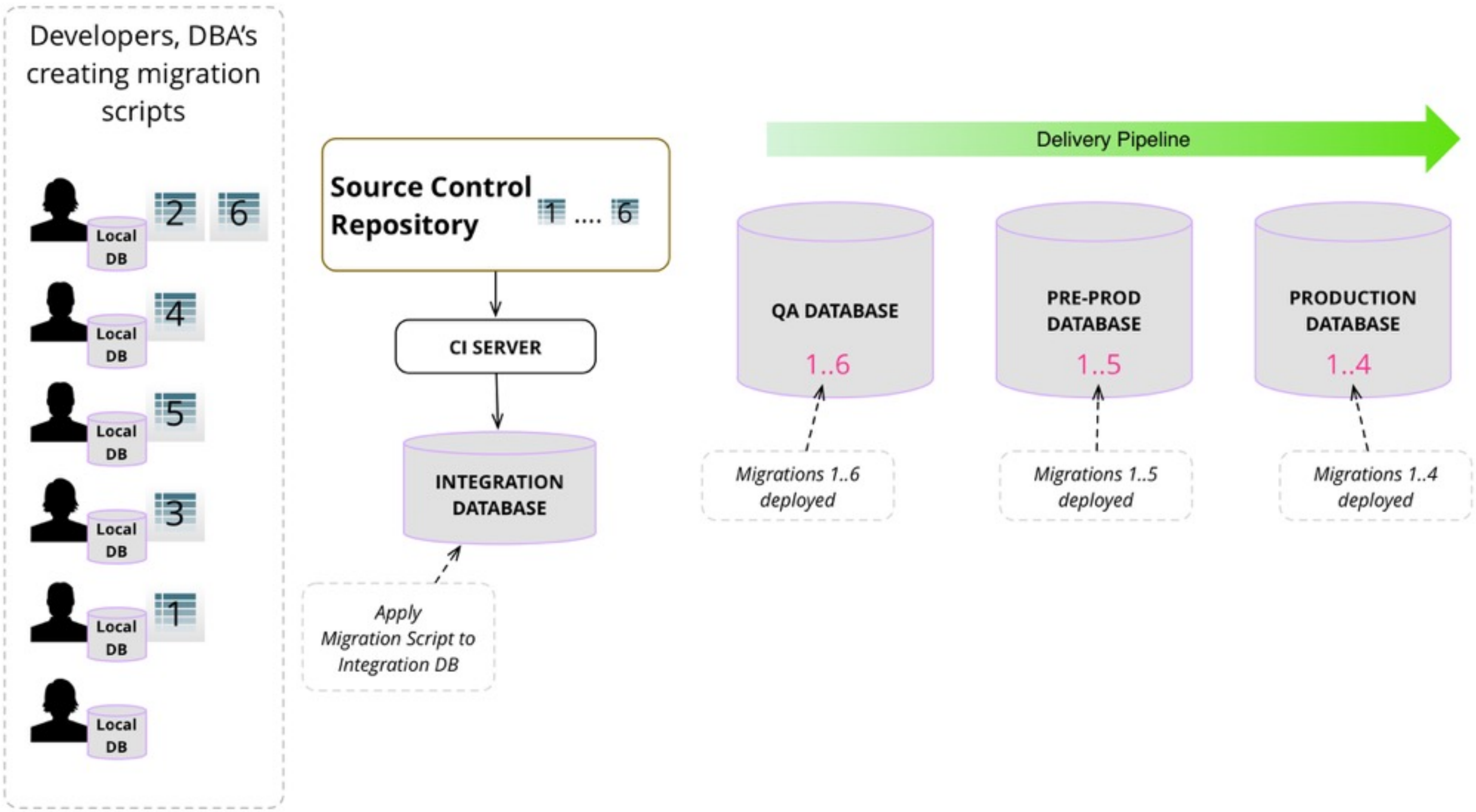
See the rest of the app works with change

Commit and Push.

See Jenkins work

How to deal with conflicts, multiple projects using same database, large teams

Tracking Changes



Deployment

- Database migration/upgrade should be a development time task not deployment time task
- Package all the migration scripts, during Continuous Integration cycle
- Apply the migration scripts
- Deploy frequently to reduce risk

Shifting to code

Download artifacts from Jenkins

What does it have?

Can you apply all the changes to a different schema?

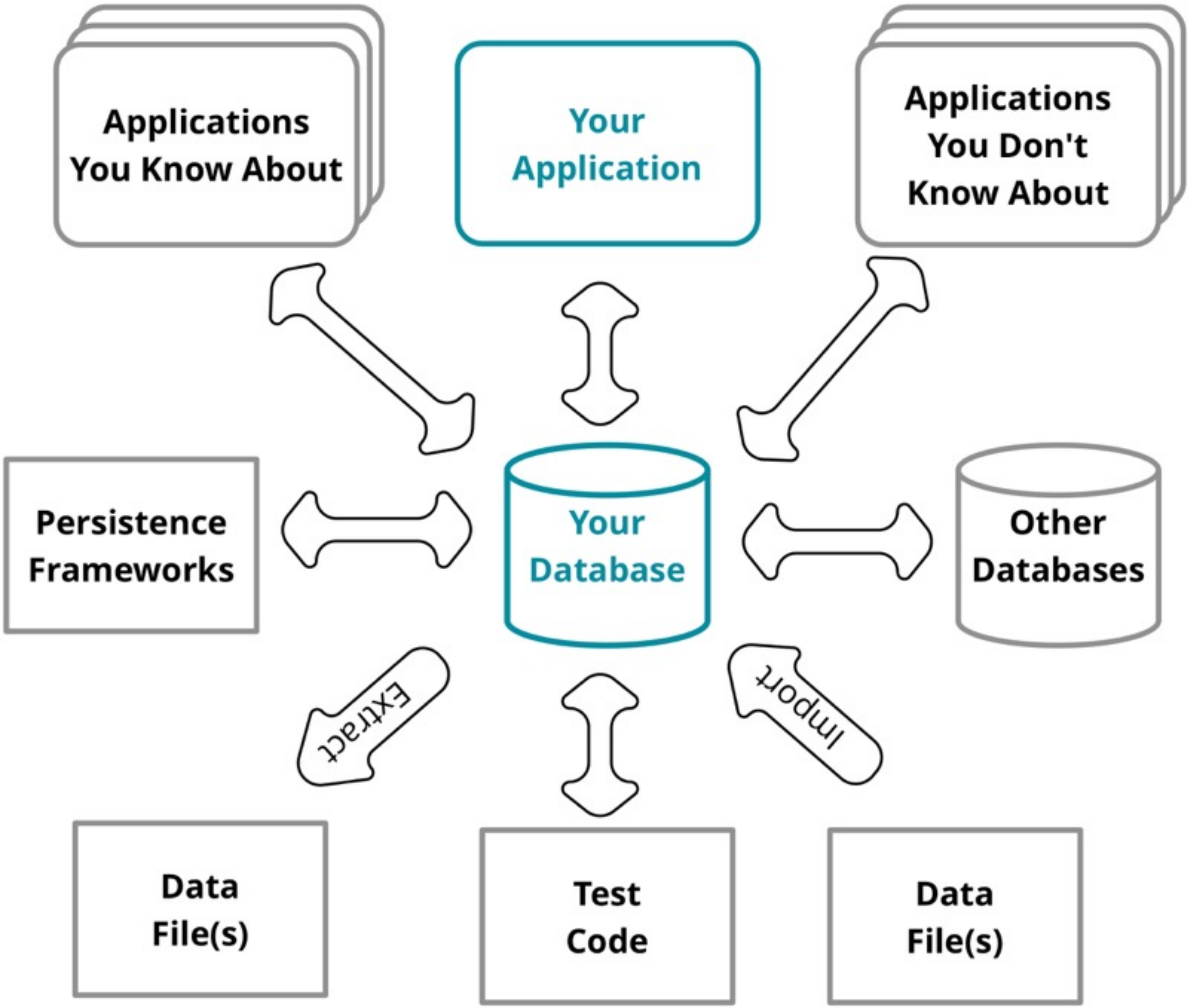
Can your devops process use this to deploy database changes along with code changes?

Database refactoring

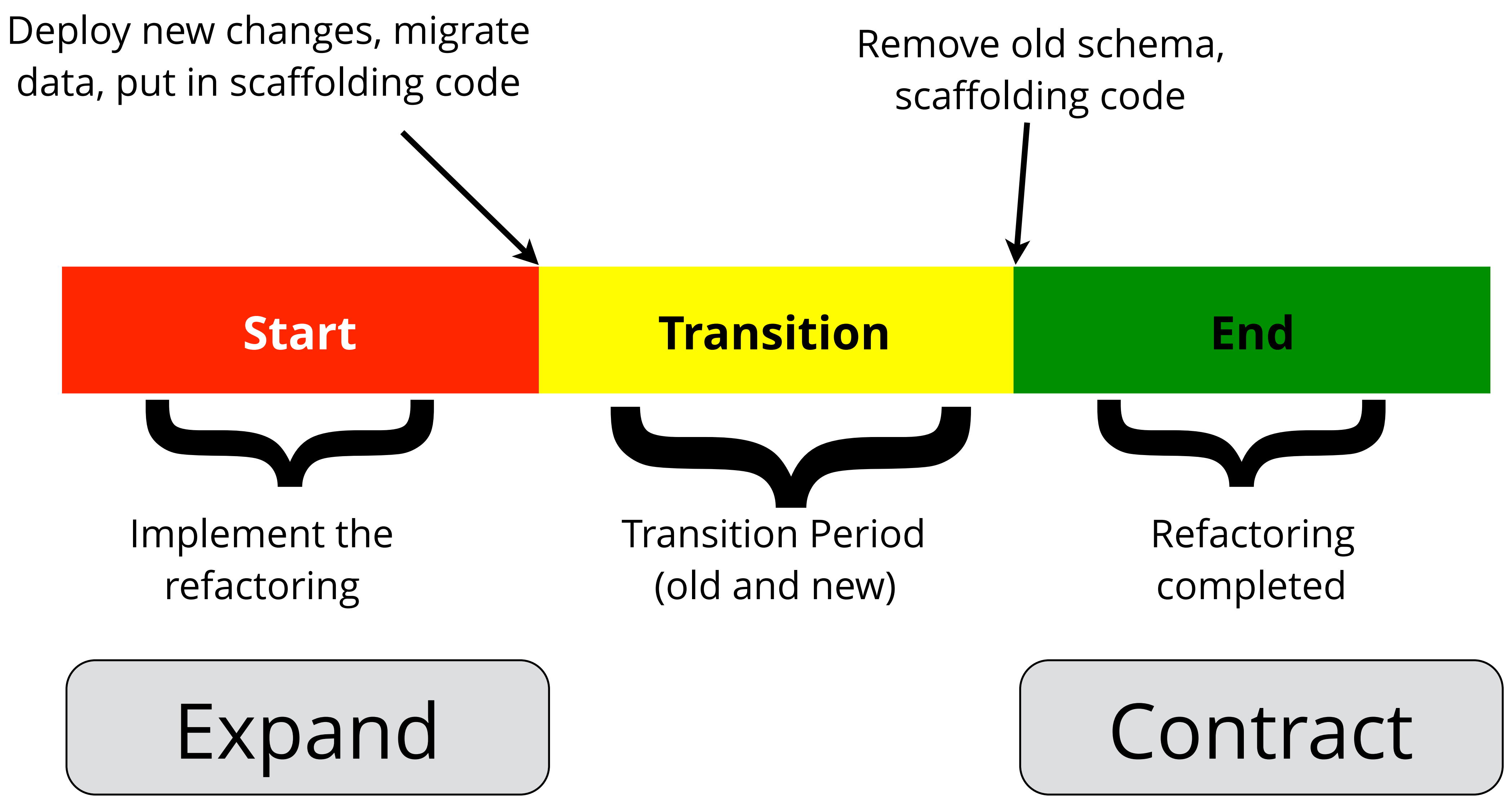
http://databaserefactoring.com/

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**.


Applications You Know About
Your Application
Applications You Don't Know About
Persistence Frameworks
Your Database
Other Databases
Extract
Import
Data File(s)
Test Code
Data File(s)

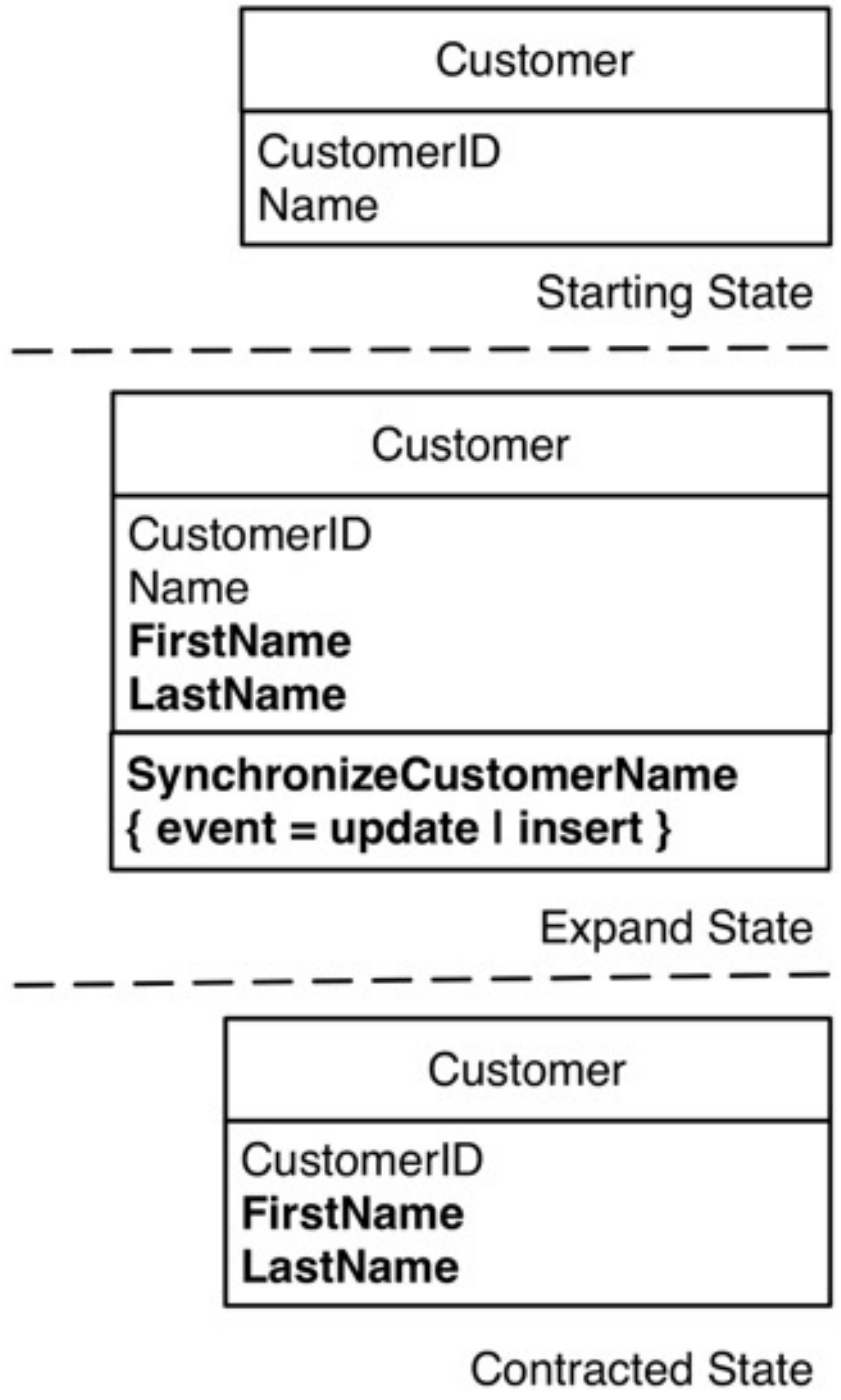
Timeline of Change


Deploy new changes, migrate data, put in scaffolding code
Remove old schema, scaffolding code
Start
Transition
End
Implement the refactoring
Transition Period (old and new)
Refactoring completed
Expand
Contract

Keeping Old and New alive

- DB Should be able to handle multiple versions of the application
- Create Interfaces in the database
- Wrap tables with views
- Create calculated columns
- Create triggers to sync data

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	With data migration
name = “Pramod Sadalage”	name = “Pramod Sadalage”
firstname = null	firstname = “Pramod”
lastname = null	lastname = “Sadalage”

Contract
firstname = “Pramod”
lastname = “Sadalage”

Simple scenario - DBDeploy

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

--//@UNDO

ALTER TABLE Customer DROP COLUMN firstname VARCHAR2(60);
ALTER TABLE Customer DROP COLUMN lastname VARCHAR2(60);
```

With synchronized 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;
/

—//@UNDO
……
```

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
….
—//@UNDO
……
UPDATE Customer set name = firstname ||’ ‘||lastname
WHERE name IS NULL;
ALTER TABLE Customer DROP COLUMN firstname;
ALTER TABLE Customer DROP COLUMN lastname;
```

Contract

```
ALTER TABLE Customer SET UNUSED name;

—//@UNDO

ALTER TABLE Customer ADD name VARCHAR2(120);
UPDATE Customer set name = firstname ||’ ‘||lastname
WHERE name IS NULL;
```

When drop takes forever

Keep legacy apps happy

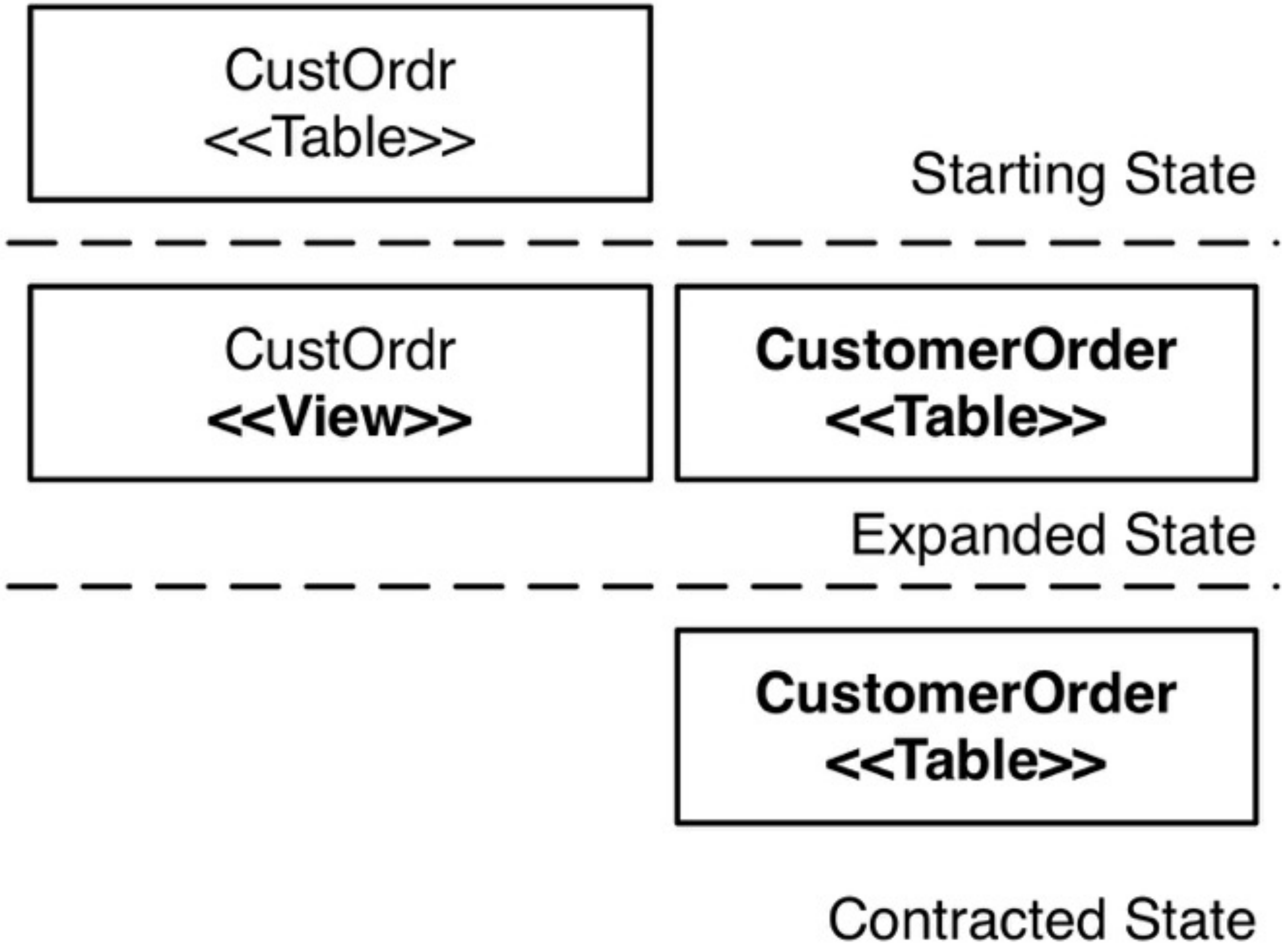
```
ALTER TABLE Customer DROP COLUMN name;
ALTER TABLE CUSTOMER ADD (name AS
          (generatename (firstname,lastname))
      );
```

Virtual column in Oracle, Generated Column in MySQL.

—//@UNDO

```
ALTER TABEL Customer DROP COLUMN name;
ALTER TABLE Customer ADD name VARCHAR2(120);
UPDATE Customer set name = firstname ||' '||lastname
WHERE name IS NULL;
```

Another example



Migration script

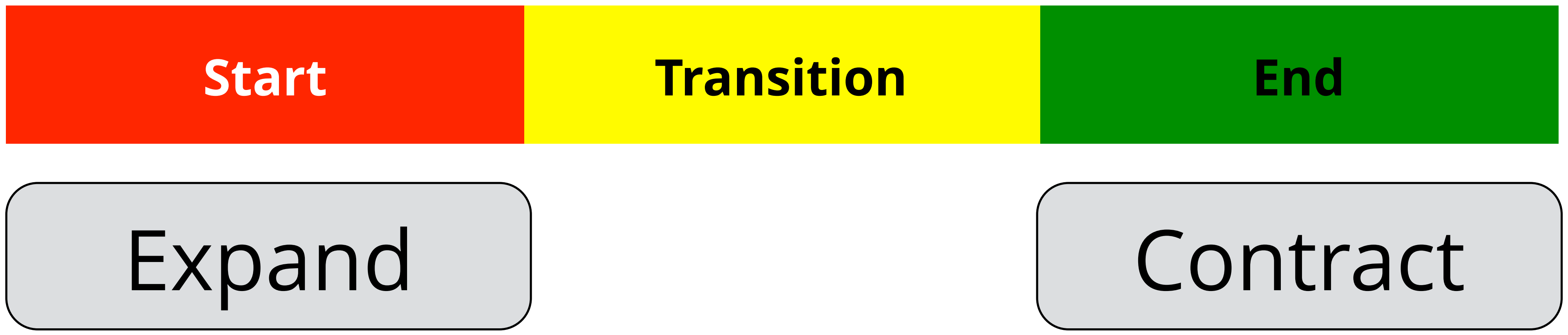
```
ALTER TABLE custordr rename to customerorder;

CREATE OR REPLACE VIEW custordr AS
SELECT custordrid, ponumber, ordrdt, shipdate, sptoadid
FROM customerorder
;
--//@UNDO

DROP VIEW custordr;
ALTER TABLE customerorder RENAME TO custordr;
```

data in migrations

```
INSERT INTO businessunit ( businessunitid, name, regionid )
VALUES ( 22, 'John Doe Services', 1 );
DELETE FROM contact ct WHERE NOT EXISTS
(SELECT 1 FROM customer p WHERE ct.contactid=p.contactid)
and EXISTS
(SELECT 1 FROM
    (SELECT customerid, COUNT(*) FROM contact
    WHERE customerid IS NOT NULL GROUP BY customerid
HAVING COUNT(*) >1) ct2
WHERE ct.customerid=ct2.customerid);
VALUES ( 3, 'Australian Dollar', 'AUD' );
INSERT INTO currency ( currencyid, name, code )
VALUES ( 4, 'EMU Euro', 'EUR' );
```

Shifting to code

```
Lets refactor our database.

See migrations/future_migrations

How would you refactor based on your
context?

What are the contexts that you need to
think about?
```



Tips

- Large refactorings are risky
- Sequence of many small refactorings 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

Devops for Database

- Devops practices help in Evolving Databases
- Production DBA's are valuable in development
- Automate
- devopsfordba.com

Devops

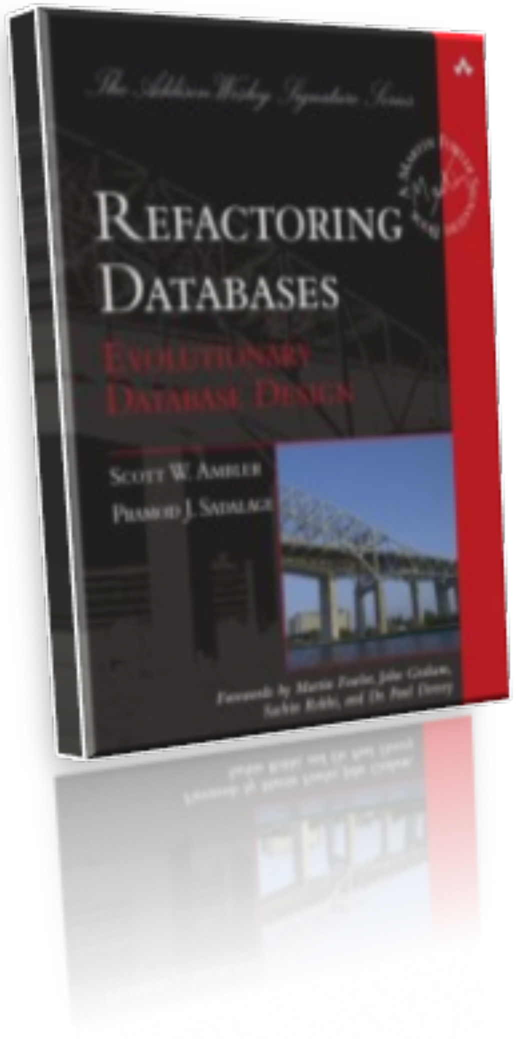
Developer databases don’t have all the production infrastructure

- Partitioning
- Linked Databases
- Synonyms vs Direct schema access

Continuous Delivery

Deployment should be easy

```
ant -propertyfile qa.properties upgrade
ant -propertyfile live.properties upgrade
```



Thanks

@pramodsadalage

sadalage.com

databaserefactoring.com

devopsfordba.com