# AURORA

An Automated Database Schema Change Logging System

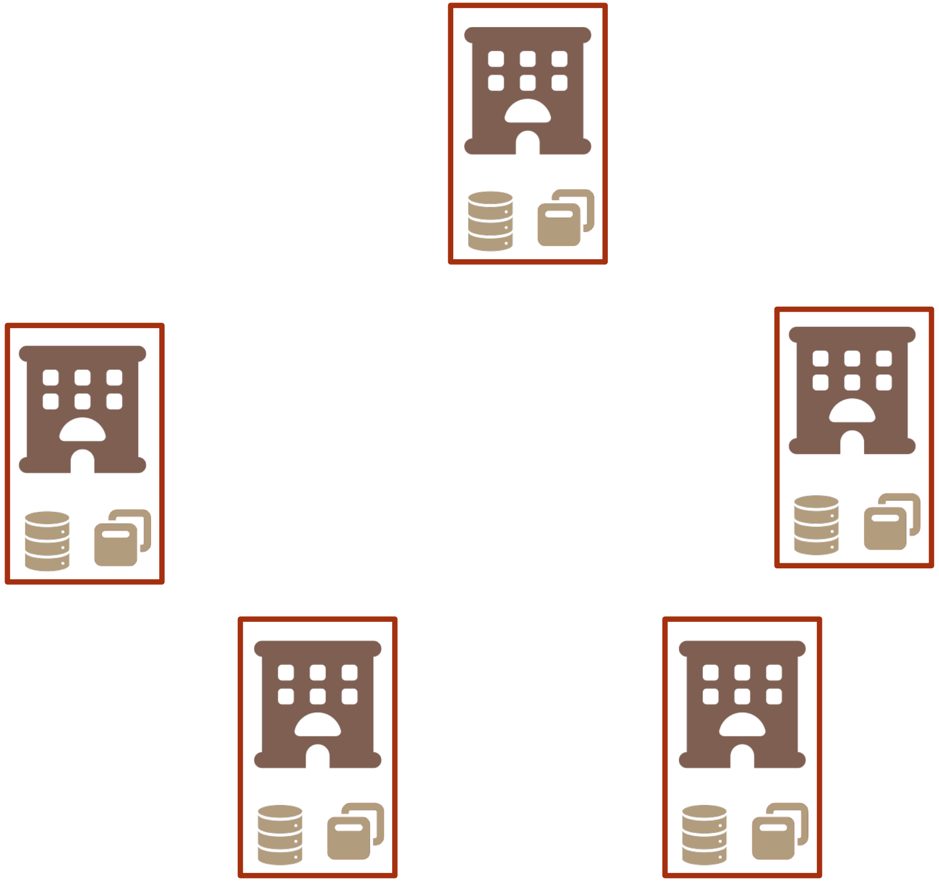
**Bradley Camilleri**
**Joseph G. Vella**

Computer Information Systems Dept.
Faculty of ICT
University of Malta

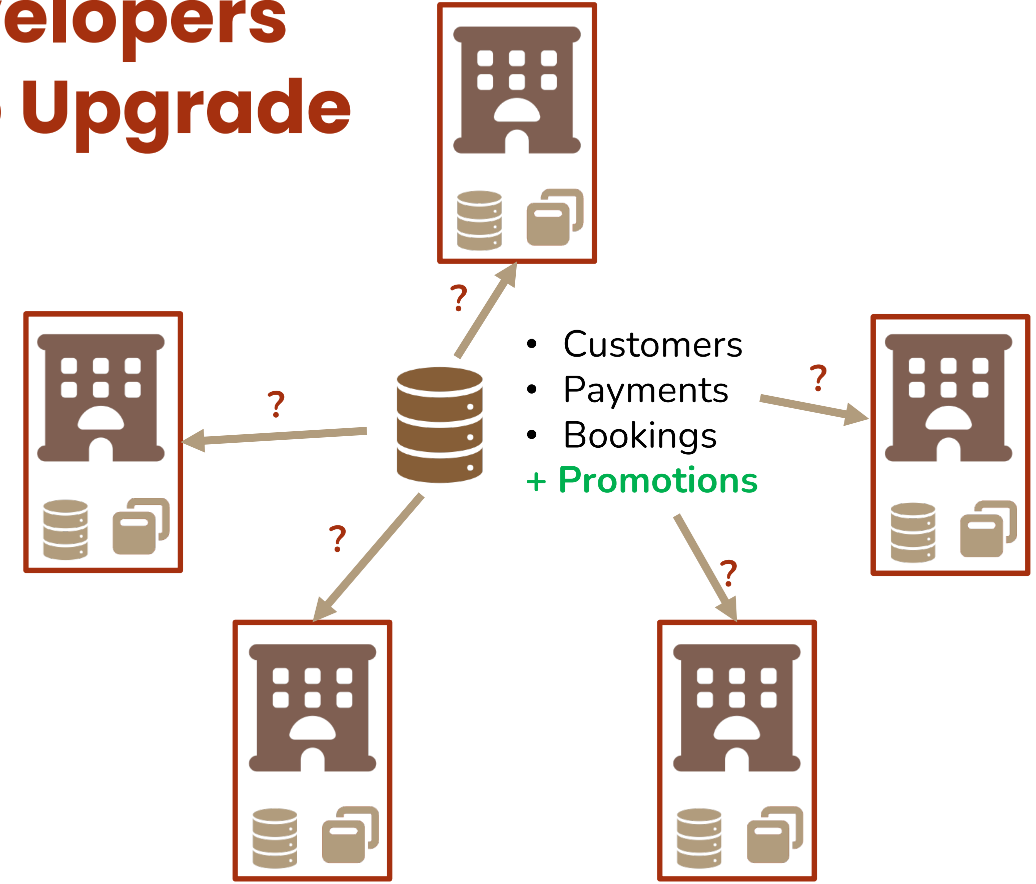
BUSTECH 2025

## Motivation

- Each copy of the system is **independent**
- Each copy has its own **unique** database state
- But each copy has the **same schema**



## The Developers Want to Upgrade



## Database Upgrades

- Database updates are **complex**
  - **No loss of user data**
    - Including user defined objects **(e.g., views)**
  - We want as little **downtime** as possible
    - We don’t want to run an **upgrade that will *definitely* fail**

# The Revolut Incident

Revolut

Get the App

Revolut app issues — 30th October. What happened, and what we did to fix it 🔧

Donato Lucia · November 01, 2019

- Database updates can go very **badly**

- In 2019, Revolut needed to **reverse a database upgrade**
- Led to approximately **2.5 hours** of downtime.

## Existing Solutions

Liquibase

Flyway

pgAdmin

atlas

SQITCH

Not Vendor-Specific

IBM DB2

Microsoft® SQL Server®

ORACLE® DATABASE

Vendors' Offerings

**They are separate from the database itself**

Developer required to generate **migration files**.

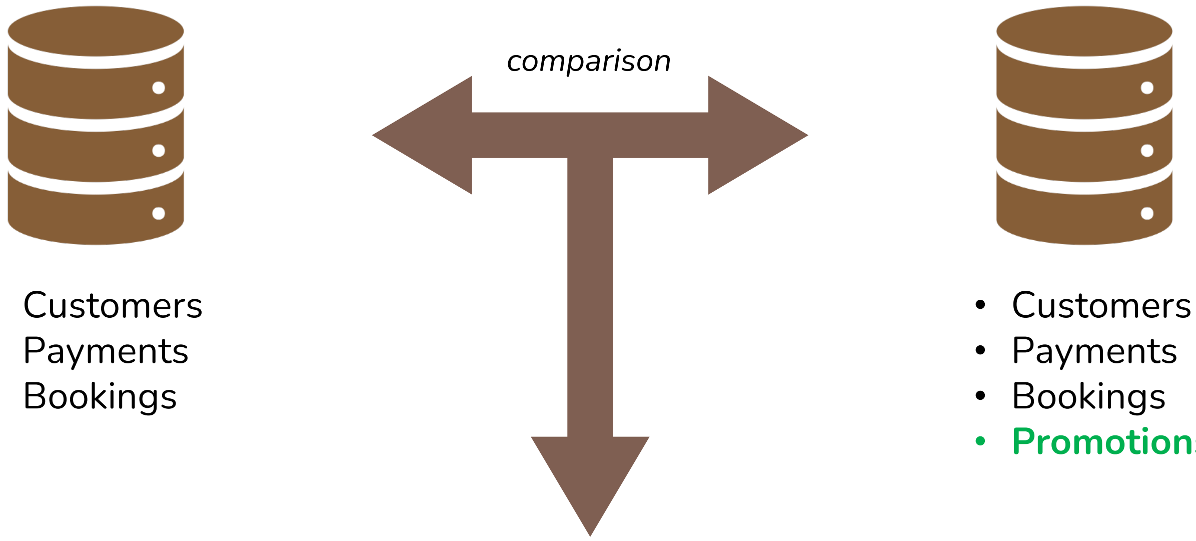
## Migration Files

- Specify the **changes** that will bring a database from one version to another.
  - E.g., **add a new table** called ‘Promotions’

- Generated in one of two ways:
  - **Schema Diff**
  - **Manually**

# Schema Diff Algorithm

Generating migration files



- Customers
- Payments
- Bookings

- Customers
- Payments
- Bookings
- Promotions

**Changes Detected:**

- New Table: Promotions

## Schema Diff Algorithm

Generating migration files

- **Problems:**
  - **Does not detect data changes**
  - Compares the database states **as-is**
    - Does not consider **what happened in between**

## The Manual Approach

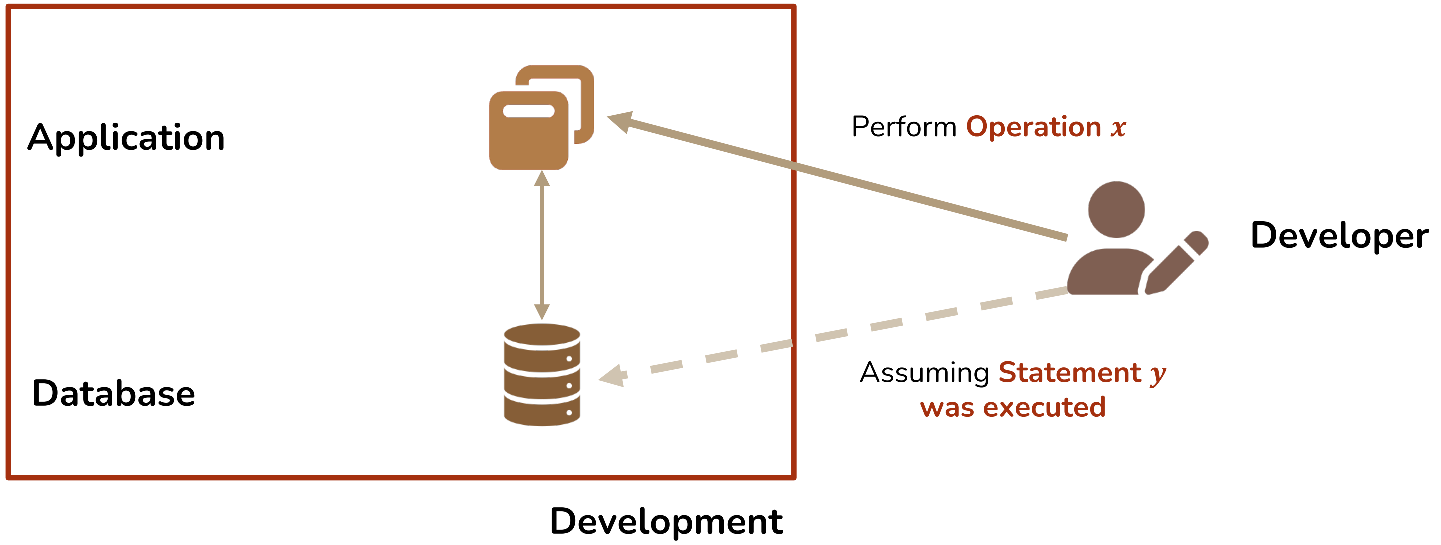
Generating migration files

- The developer **manually writes** the SQL statements that need to be executed
- This **solves all the problems** of the schema diff algorithm

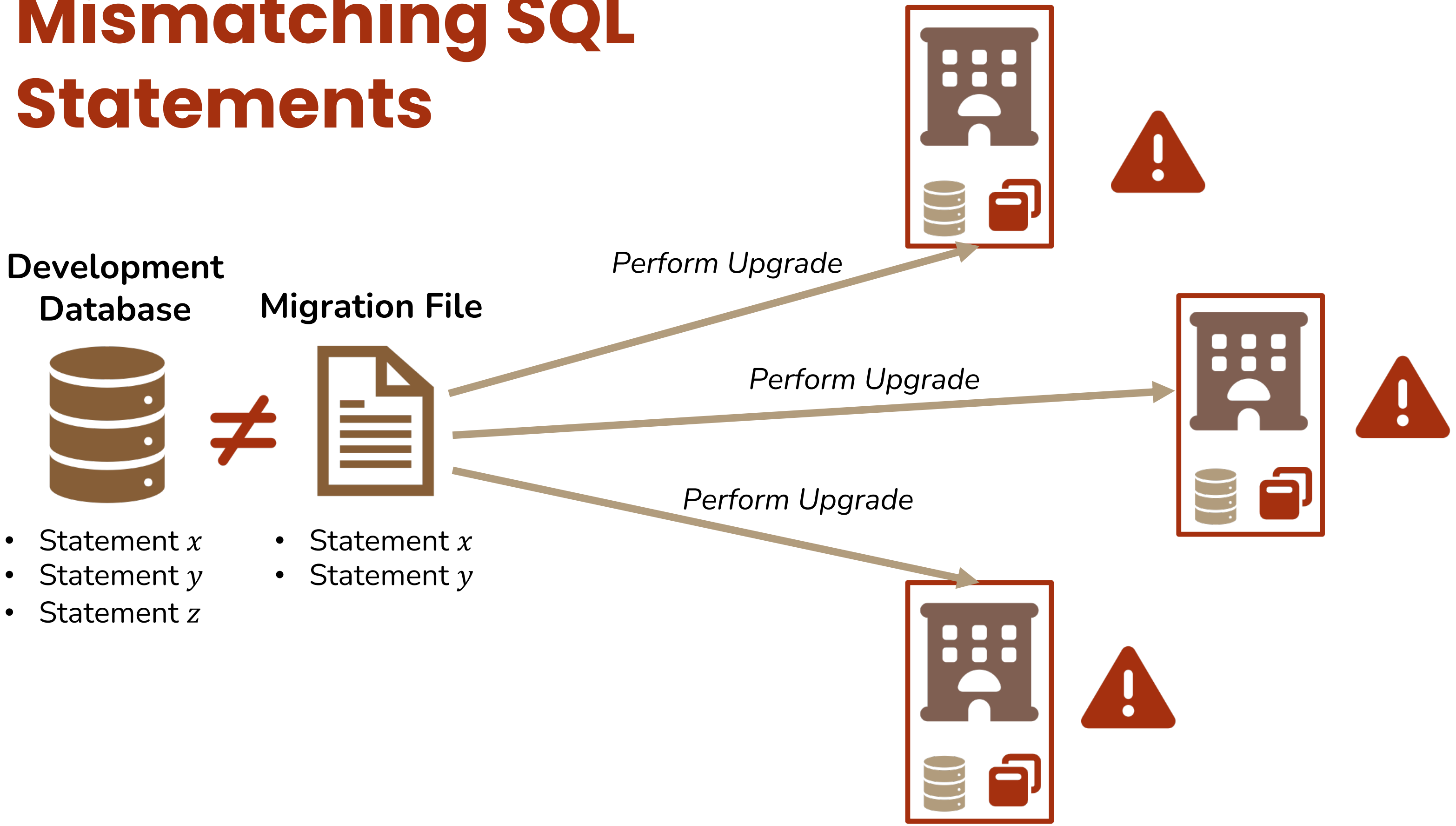
- The developer needs to **write the queries twice**
    1. **Update development database**
    2. **Generate the migration file**

- Why is this a **problem?**
    - **Mismatching SQL statements**

# Mismatching SQL Statements



# Mismatching SQL Statements



Development Database:

- Statement $x$
- Statement $y$
- Statement $z$

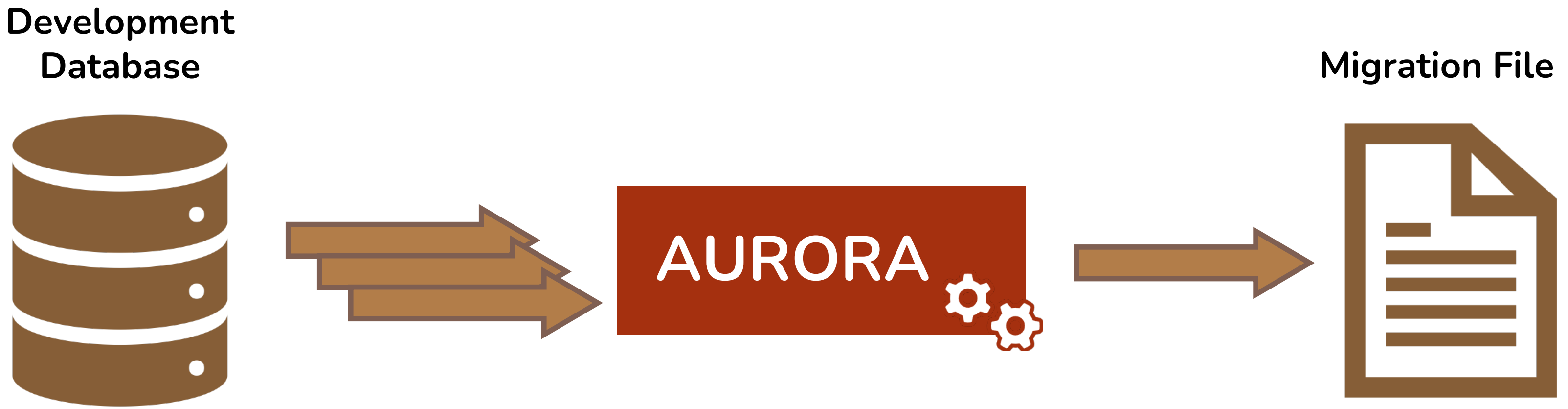
Migration File:

- Statement $x$
- Statement $y$

## How can we solve this?

- **Easy to specify changes**
- Accepts both **data and structural changes**
- No **mismatching SQL statements**.

## What if the developer does not have to generate the migration file?



- Operation $x$
- Operation $y$
- Operation $z$

- Operation $x$
- Operation $y$
- Operation $z$

## AURORA

- Automatically **tracks the changes performed on a database**

- Using this data it ***automatically***:
    - Generates an **upgrade script**
    - Generates a set of **pre-checks**
        - Ensures that the upgrade **can be executed before running it**
    - Generates a set of **post-checks**
        - Ensures that the upgrade was **executed as expected**
    - Generates an **undo script**
        - To reverse the upgrade

## Implementation

- Implemented in **PostgreSQL**
- Uses **event triggers** to automatically detect **structural** changes

- When a structural change is **detected**:
    - The event trigger checks the **data dictionary** and **stores any changes** in AURORA.

- Changes are given to a **Python script** to generate the **upgrade file**
- The **upgrade file** is then **given to the client** and run using a different Python script to **upgrade the client's database**.

## Data Dictionary

- One of the **most crucial elements** of the database management system (DBMS)
  - Without the data dictionary, the database cannot be understood by the DBMS
- **Keeps track of all the objects in the database**
  - Schemas, tables, views, functions, procedures, indexes

- PostgreSQL has **two** data dictionaries:
  - **information_schema**
  - **pg_catalogue**

## Testing

- AURORA's **test suite** includes:
  - Several **unit tests**
  - Several **SQL scripts** – both **valid** and **invalid**

- Test suite **ensures** that:
  - The developer's **queries** are:
    - **Tracked correctly** by AURORA
    - **Correctly reflected** in the **upgrade script**
  - The **correct undo queries** are generated
  - **Pre-checks** detect **compatibility issues before upgrading** a database
  - **Post-checks** detect **unexpected changes after upgrading** a database

## Evaluation

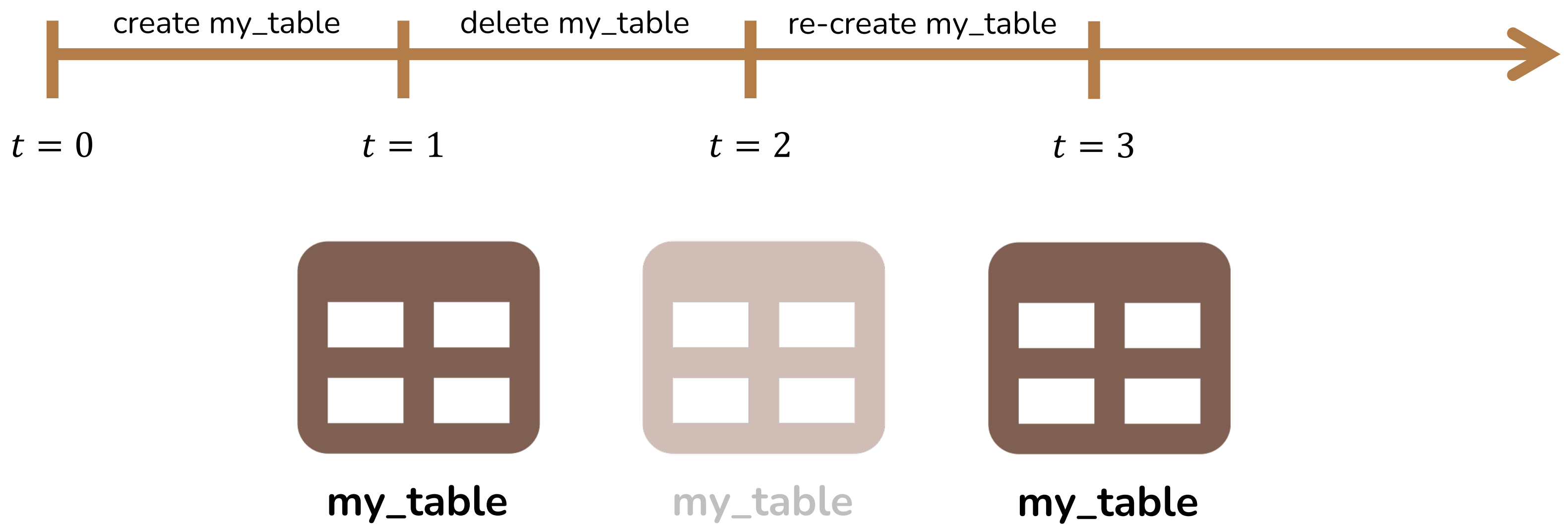
- AURORA's performance was **evaluated** in **three ways**:
  - Generating the **Scott schema**
  - Upgrading the **MediaWiki database**
  - Creating a **custom database**.

## Final Remarks

- AURORA requires a DBMS with triggers that **detect when a structural change has occurred**
  - **SQL Server and Oracle** also have these
- A **subset of DDL SQL statements** are tracked, these include
  - **Schemas, tables, constraints, views, functions, procedures, sequences, triggers.**
- AURORA does not **modify the queries** given by the developer

## Redundant Operations

Final Remarks



# Thank You

**Bradley Camilleri** | camilleribrad.com

bradley.camilleri.22@um.edu.mt
University of Malta

**Joseph G. Vella** | um.edu.mt/profile/josephgvella

joseph.g.vella@um.edu.mt
University of Malta