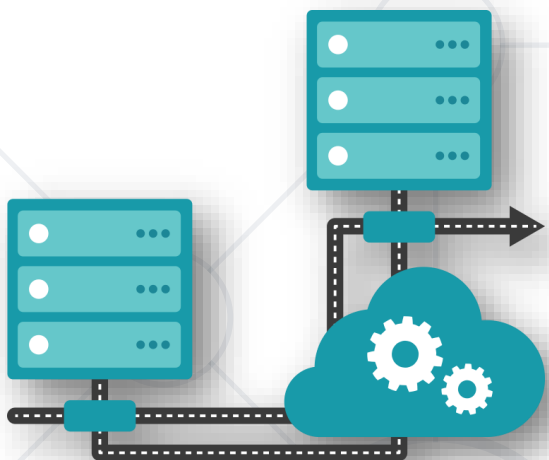


Database Programmability

User-defined Functions, Procedures, Triggers and Transactions



SoftUni Team
Technical Trainers



SoftUni



Software University

<https://softuni.bg>



sli.do
#java-db

Table of Contents

1. User-Defined Functions
2. Stored Procedures
3. Transactions
4. Triggers





User-Defined Functions

Encapsulating Custom Logic

- Extend the functionality of a MySQL Server
 - Modular programming – write **once**, call it **any number** of times
 - Faster execution – doesn't need to be reparsed and reoptimized with each use
 - Break out complex logic into **shorter code blocks**
- Functions can be:
 - Scalar – return **single value** or **NULL**
 - Table-Valued – return a **table**

Problem: Count Employees by Town

- Write a function **ufn_count_employees_by_town**(town_name) that:
 - Accepts town name as a parameter
 - Returns the count of employees in the database who live in that town

Solution: Count Employees by Town

```
CREATE FUNCTION ufn_count_employees_by_town(town_name VARCHAR(20))
```

```
RETURNS INT
```

```
DETERMINISTIC
```

```
BEGIN
```

```
    DECLARE e_count INT;
```

```
    SET e_count := (SELECT COUNT(employee_id) FROM employees AS e
```

```
        JOIN addresses AS a ON a.address_id = e.address_id
```

```
        JOIN towns AS t ON t.town_id = a.town_id
```

```
    WHERE t.name = town_name);
```

```
    RETURN e_count;
```

```
END
```

Function Name

Function Logic

Result: Count Employees by Town

- Examples of expected output:

Function Call

```
SELECT ufn_count_employees_by_town('Sofia');
```



3

```
SELECT ufn_count_employees_by_town('Berlin');
```



1

```
SELECT ufn_count_employees_by_town(NULL);
```



0

Employees
count



Stored Procedures

Sets of Queries Stored On DB Server

- Stored procedures are **logic** removed from the application and placed **on the database server**
 - Can greatly cut down traffic on the network
 - Improve the security of the database server
 - Separate data access routines from the business logic
- Stored procedures are accessed by programs using different platforms and API's

Creating Stored Procedures

- **CREATE PROCEDURE**
- Example:

DELIMITER \$\$

CREATE PROCEDURE usp_select_employees_by_seniority()

BEGIN

SELECT *

Procedure Logic

FROM employees

WHERE ROUND((DATEDIFF(NOW(), hire_date) / 365.25)) < 15;

END \$\$

Procedure Name



Executing and Dropping Stored Procedures

- Executing a stored procedure by **CALL**

```
CALL usp_select_employees_by_seniority();
```

- **DROP** Procedure

```
DROP PROCEDURE usp_select_employees_by_seniority;
```



Defining Parameterized Procedures

- To define a parameterized procedure use the syntax:

```
CREATE PROCEDURE usp_procedure_name  
(parameter_1_name parameter_type,  
parameter_2_name parameter_type,...)
```



Parameterized Stored Procedures – Example

DELIMITER \$\$

Procedure Name

CREATE PROCEDURE usp_select_employees_by_seniority(min_years_at_work INT)

BEGIN

SELECT first_name, last_name, hire_date,

Procedure Logic

ROUND(DATEDIFF(NOW(),DATE(hire_date)) / 365.25,0) AS 'years'

FROM employees

WHERE ROUND(DATEDIFF(NOW(),DATE(hire_date)) / 365.25,0) > min_years_at_work

ORDER BY hire_date;

END \$\$

Usage

CALL usp_select_employees_by_seniority(15);

Returning Values Using OUTPUT Parameters

```
CREATE PROCEDURE usp_add_numbers
```

Creating procedure

```
(first_number INT,
```

```
second_number INT,
```

```
OUT result INT)
```

```
BEGIN
```

```
SET result = first_number + second_number;
```

```
END $$
```

```
DELIMITER ;
```

```
SET @answer=0;
```

Executing procedure

```
CALL usp_add_numbers(5, 6,@answer);
```

```
SELECT @answer;
```

@answer
11

Display results

Problem: Employees Promotion

- Write a stored procedure that raises employees salaries by department name (as parameter) **by 5%**
 - Use soft_uni database

▲ employee_id	▼ first_name	last_name	middle_name	▲ job_title	📍 department_id
150	Stephanie	Conroy	A	Network Manager	11
268	Stephen	Jiang	Y	North American Sales Manager	3
288	Syed	Abbas	E	Pacific Sales Manager	3
21	Peter	Krebs	J	Production Control Manager	8

Solution: Employees Promotion

```
CREATE PROCEDURE usp_raise_salaries(department_name var  
char(50))  
BEGIN  
    UPDATE employees AS e  
        JOIN departments AS d  
        ON e.department_id = d.department_id  
        SET salary = salary * 1.05  
        WHERE d.name = department_name;  
END
```

Result: Employees Promotion

- Procedure result for 'Sales' department:

```
CALL usp_raise_salaries('Sales');
```

Data **before** procedure call:

employee_id	salary
268	48 100.00
273	72 100.00
...	...

Data **after** procedure call:

employee_id	salary
268	50 505.00
273	75 705.00
...	...

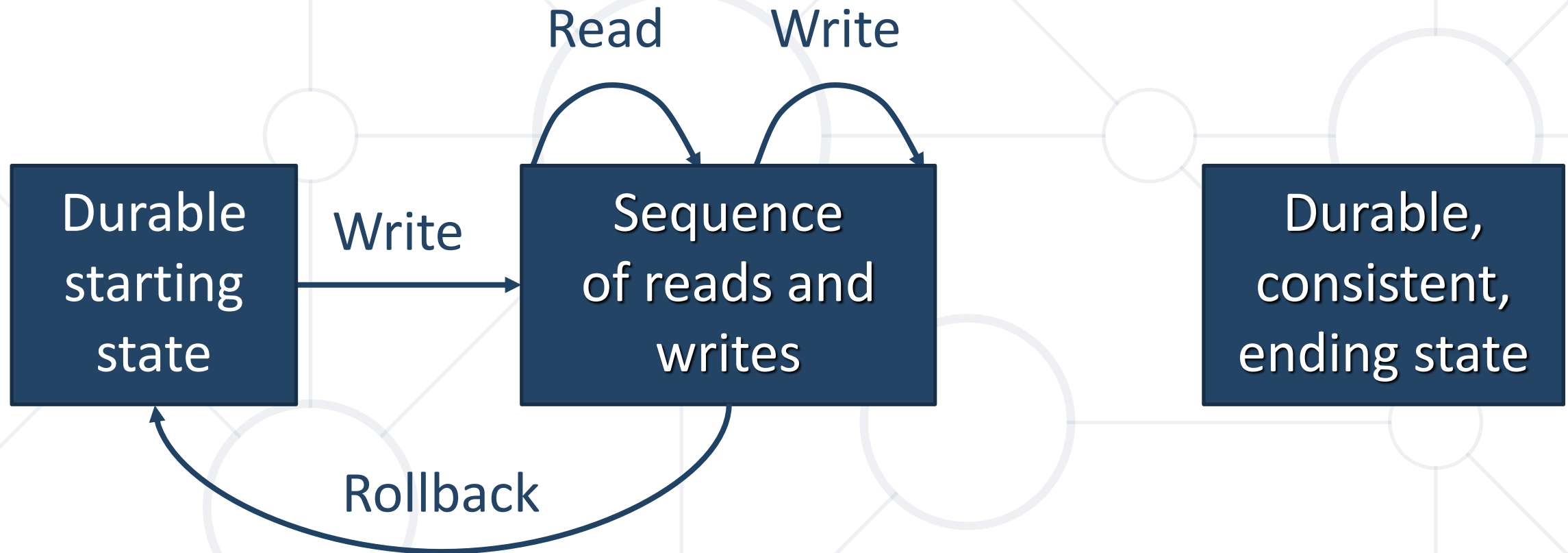


What is a Transaction?

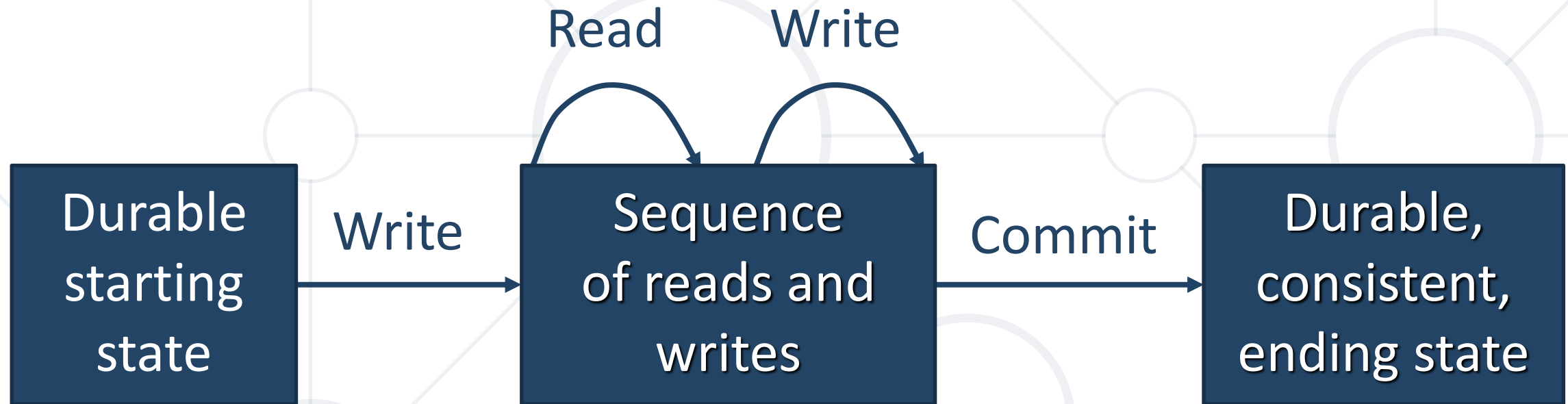
Executing Operations As a Whole

- Transaction is a **sequence of actions** (database operations) executed as a whole
 - Either **all** of them complete successfully or **none** of the them
- Example of transaction
 - A bank transfer from one account into another (withdrawal + deposit)
 - If either the withdrawal or the deposit fails **the whole operation is cancelled**

Transactions: Lifecycle (Rollback)



Transactions: Lifecycle (Commit)

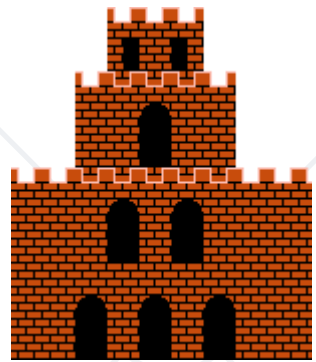


Transactions Behavior

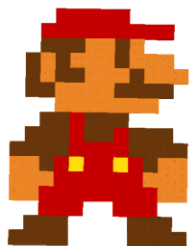
- Transactions guarantee the **consistency** and the **integrity** of the database
 - All changes in a transaction are temporary
 - Changes are persisted when **COMMIT** is executed
 - At any time all changes can be canceled by **ROLLBACK**
- All of the operations are executed as a whole



Checkpoints in Games



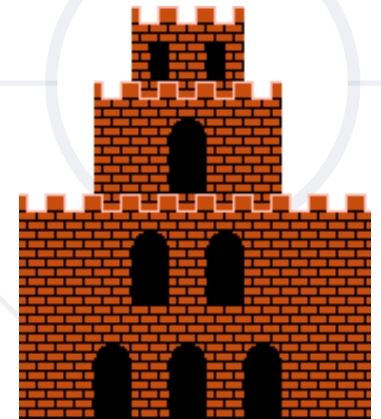
Castle 1-1



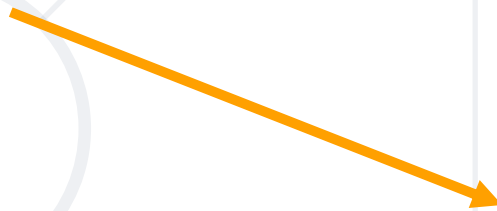
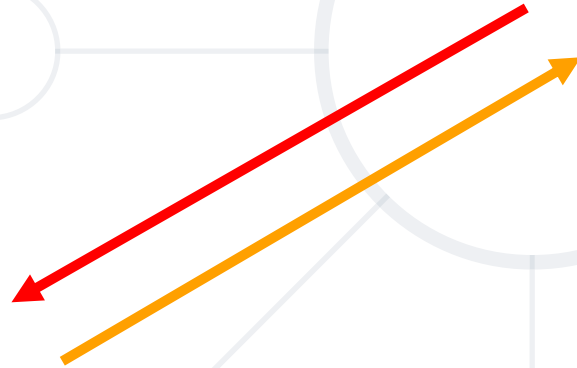
Mario

DIE

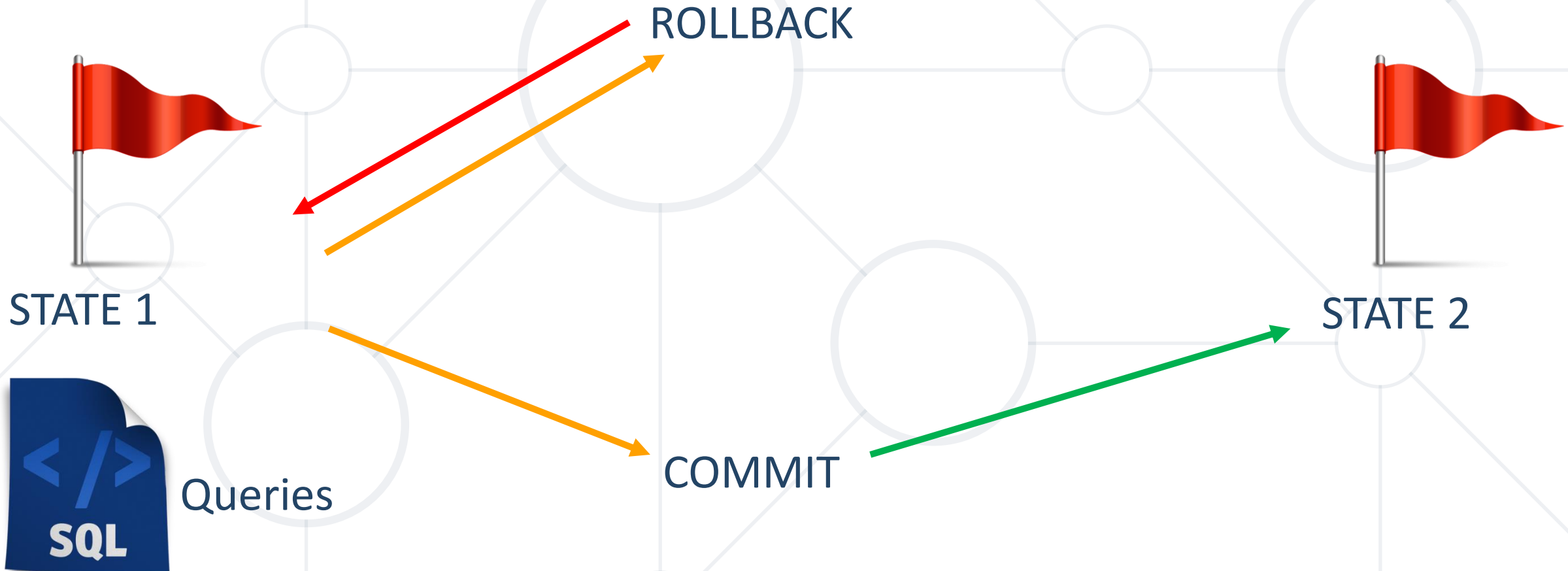
SURVIVE



Castle 1-2



What Are Transactions?



Problem: Employees Promotion by ID

- Write a transaction that raises an employee's salary by id only if the employee exists in the database
 - If not, no changes should be made
 - Use soft_uni database



Solution: Employees Promotion

```
CREATE PROCEDURE usp_raise_salary_by_id(id int)
BEGIN
    START TRANSACTION;
    IF((SELECT count(employee_id) FROM employees WHERE employee_id
like id)<>1) THEN
        ROLLBACK;
    ELSE
        UPDATE employees AS e SET salary = salary + salary*0.05
        WHERE e.employee_id = id;
    END IF;
END
```

- Modern DBMS servers have built-in transaction support
 - Implement "**ACID**" transactions
 - E.g. Oracle, MySQL, MS SQL Server, ...
- ACID means:
 - **A**tomicity
 - **C**onsistency
 - **I**solation
 - **D**urability





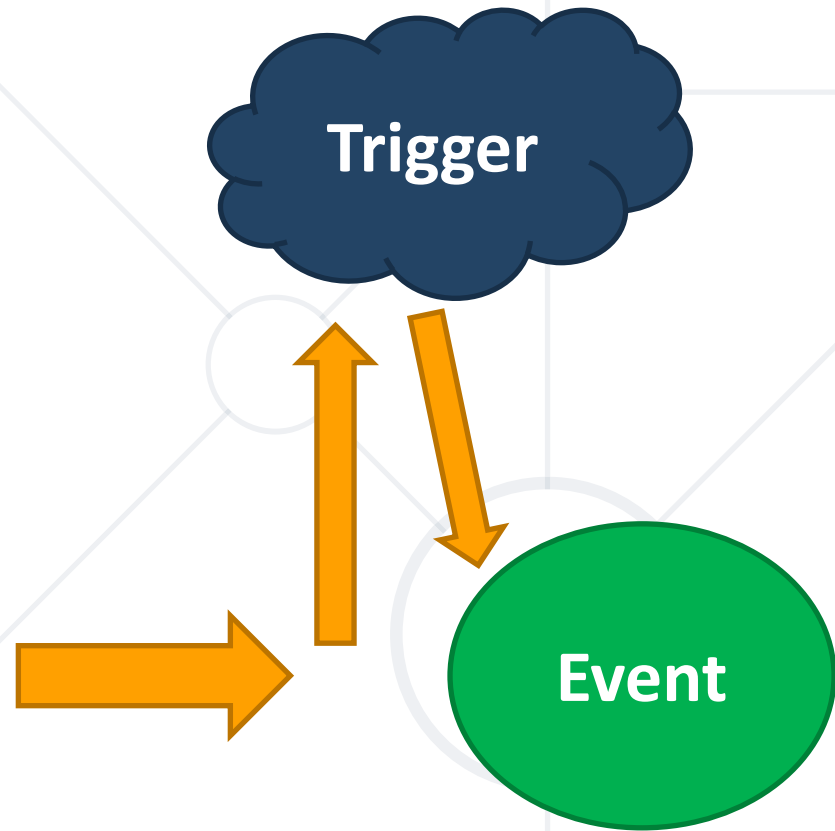
Triggers

Maintaining the Integrity of the Data

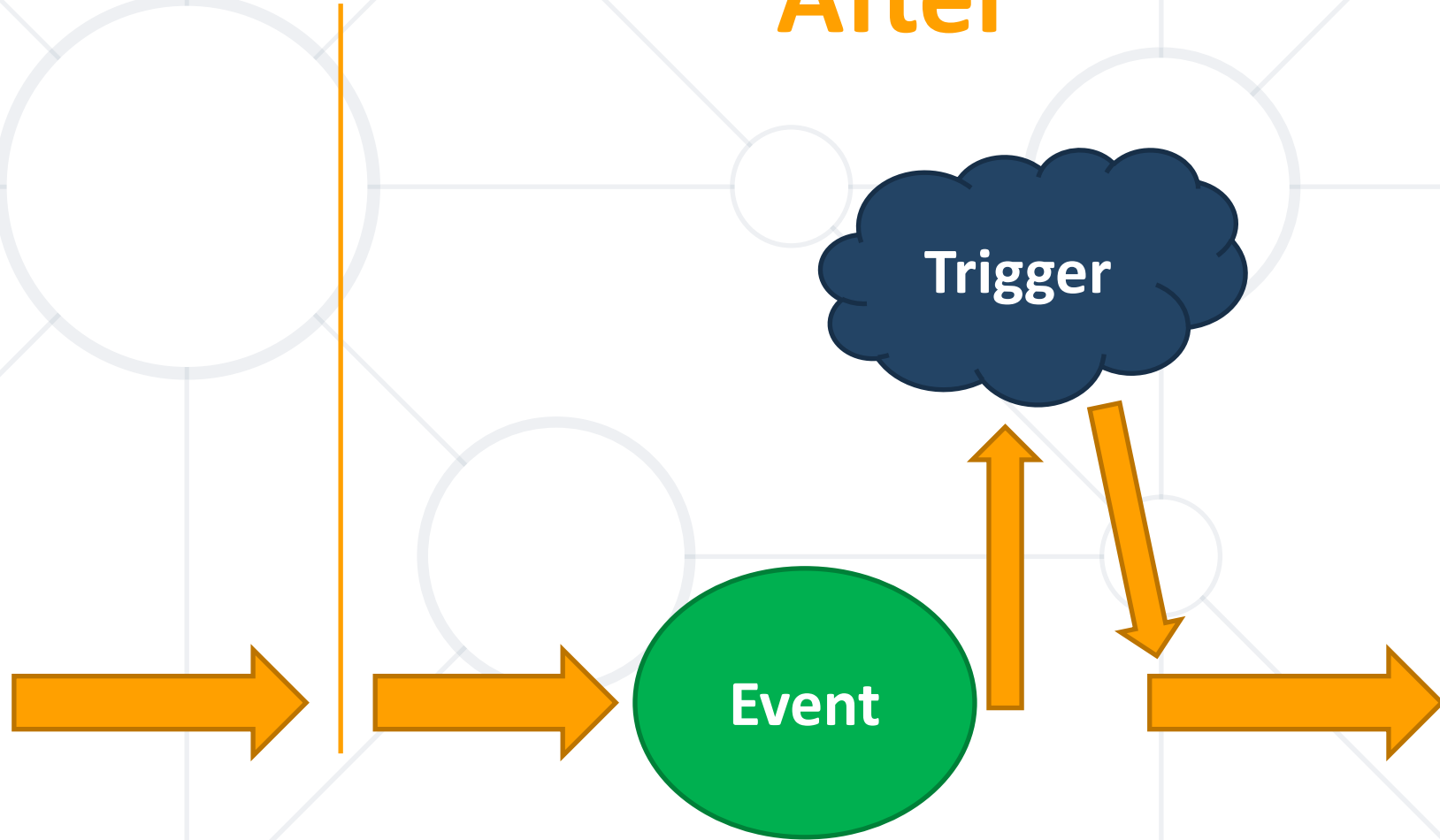
What Are Triggers?

- Triggers - small programs in the database itself, activated by the database events application layer
 - UPDATE, DELETE or INSERT queries
 - Called in case of specific **event**
- We do not call triggers **explicitly**
 - Triggers are **attached** to a table

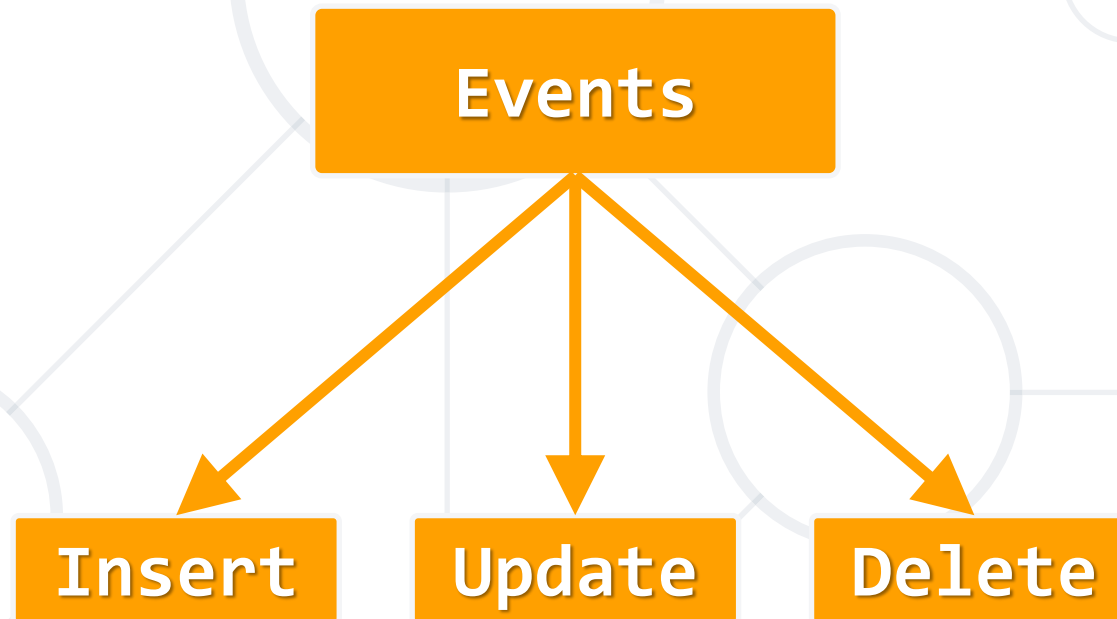
Before



After

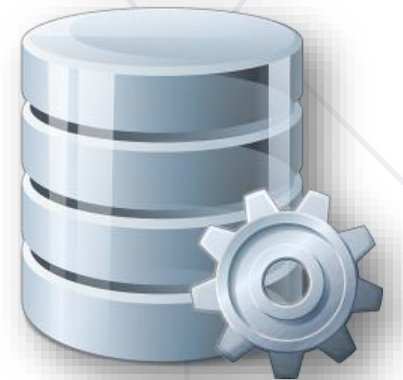


- There are three different events that can be applied within a trigger:



Problem: Triggered

- Create a table `deleted_employees` with fields:
 - `employee_id` – primary key
 - `first_name`, `last_name`, `middle_name`, `job_title`, `department_id`, `salary`
- Add a trigger to the `employees` table that logs deleted employees into the `deleted_employees` table
 - Use `soft_uni` database



```
CREATE TABLE deleted_employees(  
    employee_id INT PRIMARY KEY AUTO_INCREMENT,  
    first_name VARCHAR(20),  
    last_name VARCHAR(20),  
    middle_name VARCHAR(20),  
    job_title VARCHAR(50),  
    department_id INT,  
    salary DOUBLE  
);
```

Solution: Triggered

```
CREATE TRIGGER tr_deleted_employees  
AFTER DELETE  
ON employees  
FOR EACH ROW  
BEGIN
```

The OLD and NEW keywords allow you to access columns before/after trigger action

```
    INSERT INTO deleted_employees (first_name,last_name,  
                                   middle_name,job_title,department_id,salary)  
    VALUES(OLD.first_name,OLD.last_name,OLD.middle_name,  
            OLD.job_title,OLD.department_id,OLD.salary);
```

```
END;
```

- Trigger action result on **DELETE**:
 - NOTE: Remove foreign key checks before trying to delete employees
 - **DO NOT** submit foreign key restriction changes in the Judge System

```
DELETE FROM employees WHERE employee_id IN (1);
```

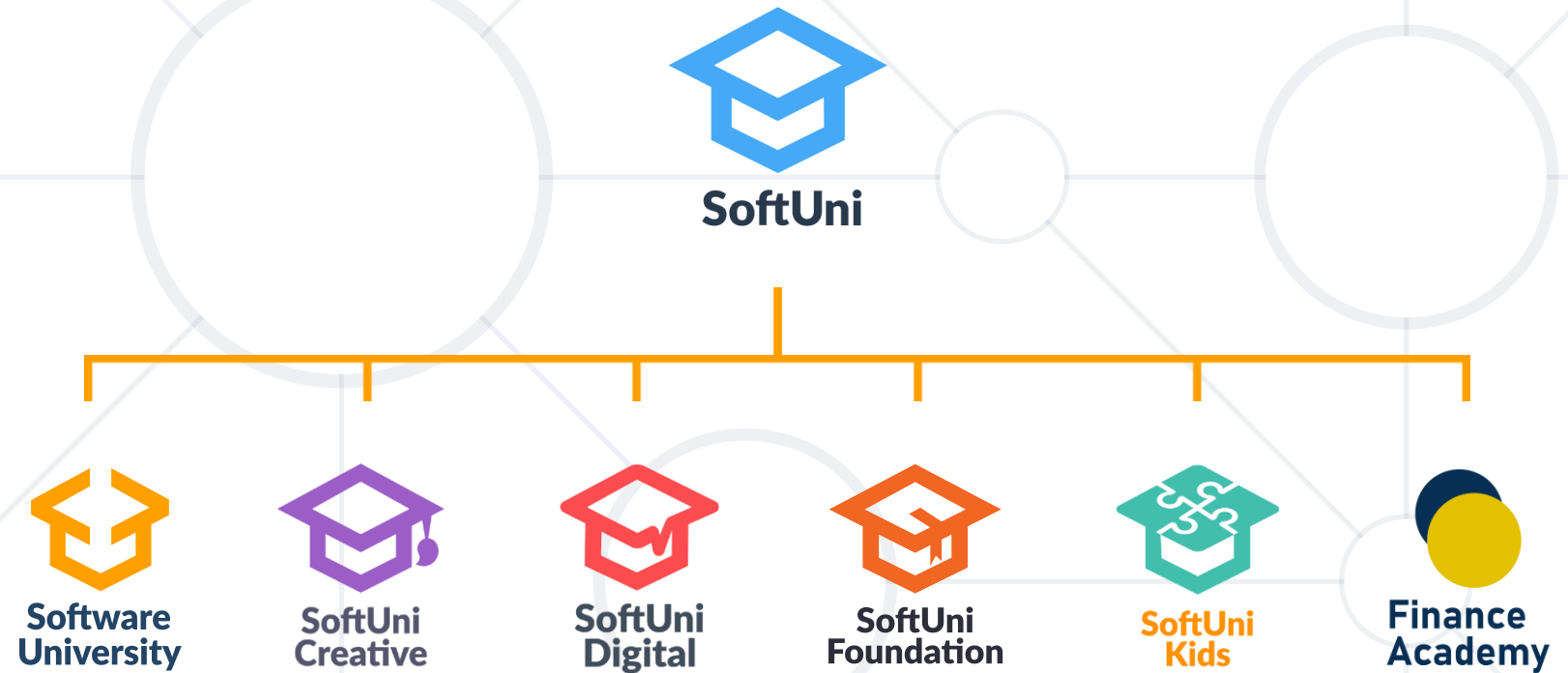
- Data in deleted_employees table:

employee_id	first_name	last_name	...
1	Guy	Gilbert	...

- We can **optimize** with User-defined **Functions**
- **Transactions** improve **security** and **consistency**
- Stored **Procedures** encapsulate repetitive logic
- **Triggers** execute **before** certain **events** on tables



Questions?



SoftUni Diamond Partners



- Software University – High-Quality Education, Profession and Job for Software Developers

- softuni.bg, about.softuni.bg

- Software University Foundation

- softuni.foundation

- Software University @ Facebook

- facebook.com/SoftwareUniversity

- Software University Forums

- forum.softuni.bg



- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://about.softuni.bg/>
- © Software University – <https://softuni.bg>

