Prof. DI Dr. Erich Gams

Stored Procedures

informationssysteme htl-wels

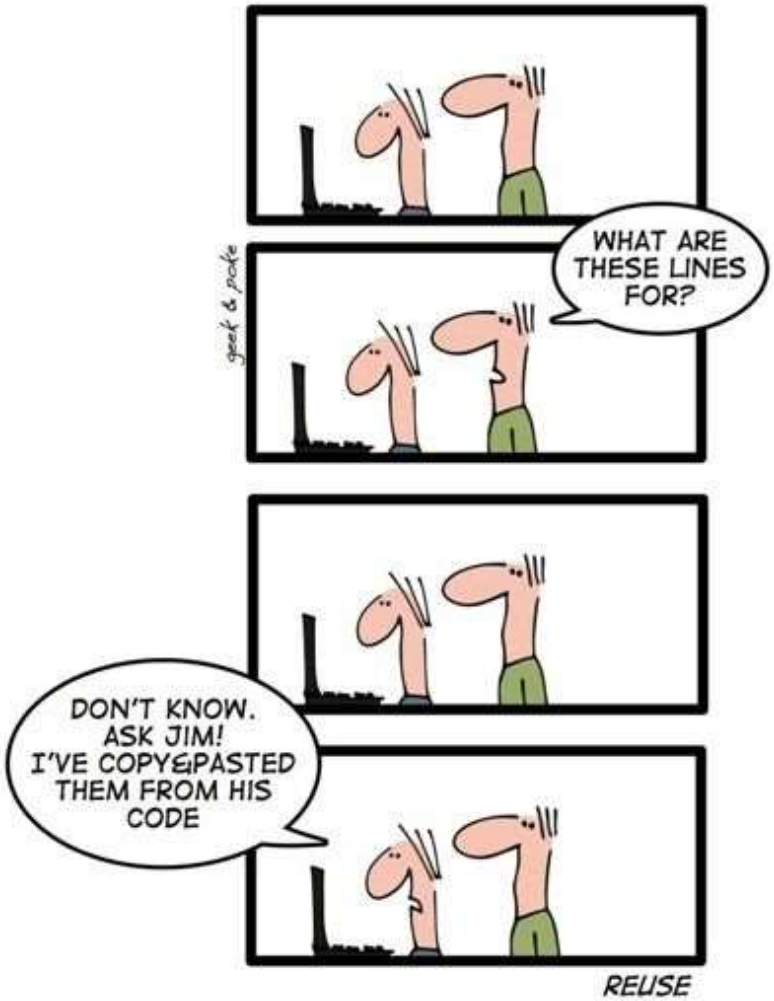
Übersicht ☛ Was lernen wir?

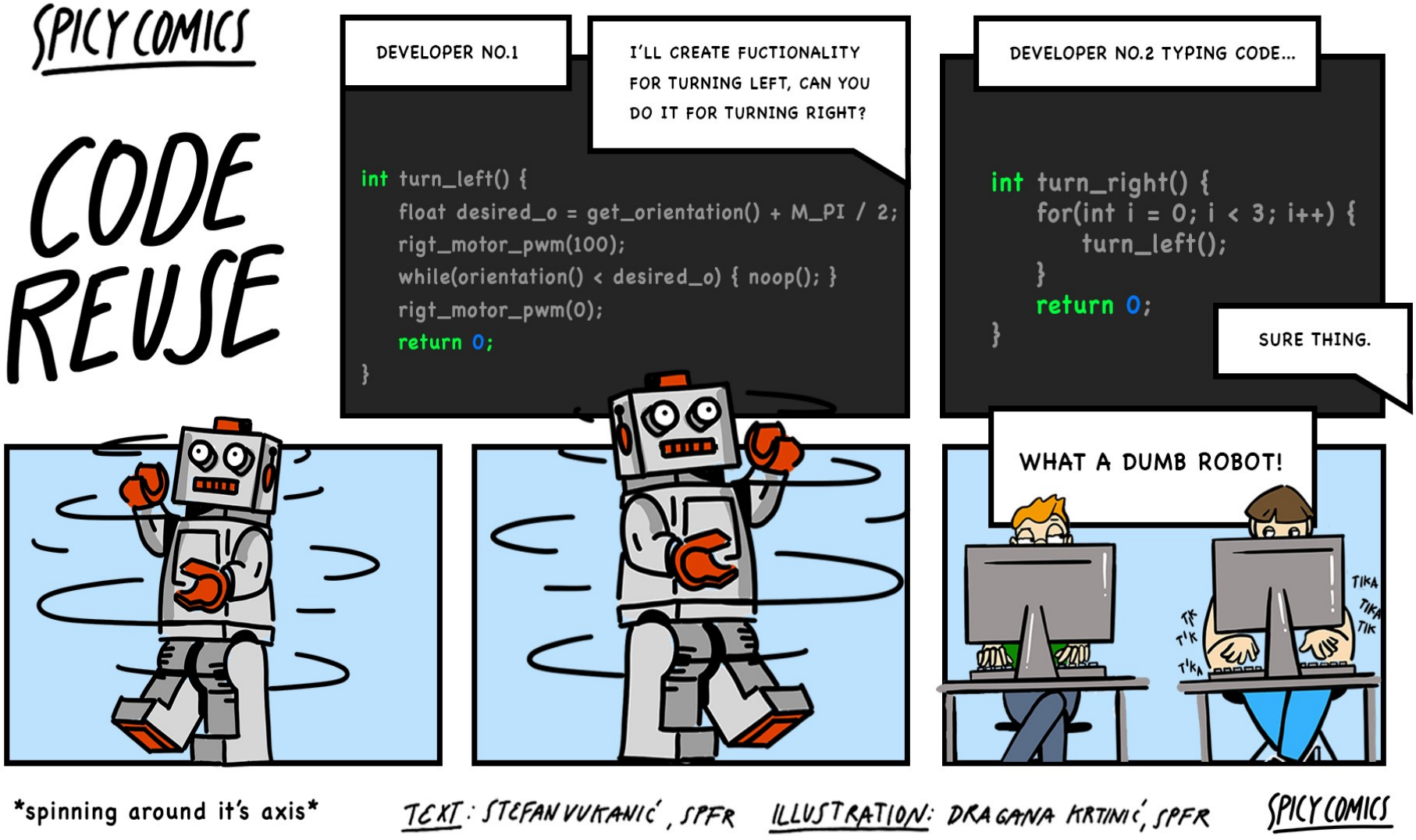
- Warum SP?
- Definition
- Elemente und Beispiele
- Übungen

What is a Stored Procedure?

- A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.
- Save it as a stored procedure, and then just call it to execute it.
- You can also pass parameters to a stored procedure

Why Stored Procedures?




SPICY COMICS
CODE REUSE
DEVELOPER NO.1
I'LL CREATE FUCTIONALITY FOR TURNING LEFT, CAN YOU DO IT FOR TURNING RIGHT?
int turn_left() {
float desired_o = get_orientation() + M_PI / 2;
rigt_motor_pwm(100);
while(orientation() < desired_o) { noop(); }
rigt_motor_pwm(0);
return 0;
}
DEVELOPER NO.2 TYPING CODE...
int turn_right() {
for(int i = 0; i < 3; i++) {
turn_left();
}
return 0;
}
SURE THING.
WHAT A DUMB ROBOT!
TIKA
TIKA
TIK
TIK
TIK
TIKA
spinning around it's axis
TEXT: STEFAN VUKANIĆ, SPFR
ILLUSTRATION: DRAGANA KRTINIĆ, SPFR
SPICY COMICS

Why?

- Reduce network traffic
- Centralize business logic in the database
- Make database more secure

History

- PL/SQL is Oracle's Procedural Language extension to SQL.
- It is loosely based on Ada (a variant of Pascal developed for the US Dept of Defense).
- PL/SQL was first released in 1992 as an optional extension to Oracle 6.
- Since version 7, Oracle did a major upgrade for PL/SQL (version 2.0) that provides more features such as procedures, functions, packages, records, collections, and some package extensions.
- Existiert für Oracle ☺, PostgreSQL, MySQL

Is PL/SQL dead?

Steven Feuerstein, Oracle Developer Advocate for PL/SQL at Oracle

Answered May 5, 2017 · Upvoted by Ramanuj Chattopahyay, 6 yrs of Development in PL/SQL and Sharath Hemmadi, Senior BI Engineer at Oracle (2018-present)

I'll offer just a few other thoughts.

- As long as Oracle Database is around, PL/SQL will be alive, thriving and critical. Does anyone think Oracle Database is going away in the next few decades? :-)
- As a database programming language, PL/SQL is generally not visible to UI developers, who mostly live in JavaScript these days. So it doesn't get lots o "love" these days, but it's still a powerful, incredibly useful language.
- I know this is going to sound funny, but PL/SQL is like toilet paper. You (say, the JavaScript developer) can live without it. But if you are building applications on top of Oracle Database, it'll sure make your life a lot easier. And more comfortable.
- It is best to think of PL/SQL as a Data API language. Its most important role is to provide package-based APIs to SQL DML operations on tables.
- If you want to build very secure, easy-to-maintain, high performing applications on top of Oracle Database, you *must* use PL/SQL.

Delimiter

- Change Delimiter to $$

```
1  DELIMITER $$
2
3  CREATE PROCEDURE sp_name()
4  BEGIN
5    -- statements
6  END $$
7
8  DELIMITER ;
```

- Change Delimiter back to ;

First Procedure

```sql
SELECT * FROM products;
```

```sql
DELIMITER //

CREATE PROCEDURE GetAllProducts()
BEGIN
    SELECT *  FROM products;
END //

DELIMITER ;
```

Executing

```
1 CALL stored_procedure_name(argument_list);
```

- Delete: DROP PROCEDURE

Variables

```
1  DELIMITER $$
2
3  CREATE PROCEDURE GetTotalOrder()
4  BEGIN
5      DECLARE totalOrder INT DEFAULT 0;
6
7      SELECT COUNT(*)
8      INTO totalOrder
9      FROM orders;
10
11     SELECT totalOrder;
12 END$$
13
14 DELIMITER ;
```

Parameters

- A parameter has one of three modes:
- IN,OUT, or INOUT.
- IN: Call-by-Value (with init value)
- OUT: Call-by-Reference (without init value)
- INOUT: Combination of IN and OUT

IN and OUT Parameter

```
CREATE PROCEDURE GetOrderCountByStatus (
    IN  orderStatus VARCHAR(25),
    OUT total INT
)
BEGIN
    SELECT COUNT(orderNumber)
    INTO total
    FROM orders
    WHERE status = orderStatus;
END$$
```

IN and OUT Parameter

- Execution

```
CALL GetOrderCountByStatus('Shipped',@total);
SELECT @total;
```

	@total
▶	303

IF Statement

```
CREATE PROCEDURE GetCustomerLevel(
    IN  pCustomerNumber INT,
    OUT pCustomerLevel  VARCHAR(20))
BEGIN
    DECLARE credit DECIMAL DEFAULT 0;

    SELECT creditLimit
    INTO credit
    FROM customers
    WHERE customerNumber = pCustomerNumber;

    IF credit > 50000 THEN
        SET pCustomerLevel = 'PLATINUM';
    ELSE
        SET pCustomerLevel = 'NOT PLATINUM';
    END IF;
END$$
```

```
1 CALL GetCustomerLevel(141, @level);
2 SELECT @level;
```

@level
PLATINUM

IF - ELSEIF

```
CREATE PROCEDURE GetCustomerLevel(
    IN  pCustomerNumber INT,
    OUT pCustomerLevel  VARCHAR(20))
BEGIN
    DECLARE credit DECIMAL DEFAULT 0;

    SELECT creditLimit
    INTO credit
    FROM customers
    WHERE customerNumber = pCustomerNumber;

    IF credit > 50000 THEN
        SET pCustomerLevel = 'PLATINUM';
    ELSEIF credit <= 50000 AND credit > 10000 THEN
        SET pCustomerLevel = 'GOLD';
    ELSE
        SET pCustomerLevel = 'SILVER';
    END IF;
END $$
```

Loops

```
[label]: LOOP
    ...
    -- terminate the loop
    IF condition THEN
        LEAVE [label];
    END IF;
    ...
END LOOP;
```

- LEAVE: Exit the Loop
- ITERATE: skip the current loop iteration and start a new iteration.

Example

- The stored procedure constructs a string from the even numbers e.g., 2, 4, and 6

```
CREATE PROCEDURE LoopDemo()
BEGIN
    DECLARE x  INT;
    DECLARE str  VARCHAR(255);

    SET x = 1;
    SET str =  '';

    loop_label:  LOOP
        IF  x > 10 THEN
            LEAVE  loop_label;
        END  IF;

        SET  x = x + 1;
        IF  (x mod 2) THEN
            ITERATE  loop_label;
        ELSE
            SET  str = CONCAT(str,x,',');
        END  IF;
    END LOOP;
    SELECT str;
END$$
```

WHILE

```
CREATE PROCEDURE LoadCalendars(
    startDate DATE,
    day INT
)
BEGIN

    DECLARE counter INT DEFAULT 1;
    DECLARE dt DATE DEFAULT startDate;

    WHILE counter <= day DO
        CALL InsertCalendar(dt);
        SET counter = counter + 1;
        SET dt = DATE_ADD(dt,INTERVAL 1 day);
    END WHILE;

END$$
```

REPEAT UNTIL

```
CREATE PROCEDURE RepeatDemo()
BEGIN
    DECLARE counter INT DEFAULT 1;
    DECLARE result VARCHAR(100) DEFAULT '';

    REPEAT
        SET result = CONCAT(result,counter,',');
        SET counter = counter + 1;
    UNTIL counter >= 10
    END REPEAT;

    -- display result
    SELECT result;
END$$
```

Functions

```
CREATE FUNCTION CustomerLevel(
    credit DECIMAL(10,2)
)
RETURNS VARCHAR(20)
DETERMINISTIC
BEGIN
    DECLARE customerLevel VARCHAR(20);

    IF credit > 50000 THEN
        SET customerLevel = 'PLATINUM';
    ELSEIF (credit >= 50000 AND
            credit <= 10000) THEN
        SET customerLevel = 'GOLD';
    ELSEIF credit < 10000 THEN
        SET customerLevel = 'SILVER';
    END IF;
    -- return the customer level
    RETURN (customerLevel);
END$$
```

```
1 SHOW FUNCTION STATUS
2 WHERE db = 'classicmodels';
```

Db	Name	Type	Definer
classicmodels	CustomerLevel	FUNCTION	root@localhost

Call a function

```
SELECT
    customerName,
    CustomerLevel(creditLimit)
FROM
    customers
ORDER BY
    customerName;
```

```
-- call the function
SET customerLevel = CustomerLevel(credit);
```

Cursor

- To handle a result set inside a stored procedure, you use a cursor.
- A cursor allows you to iterate a set of rows returned by a query and process each row individually.

Read-only, non-scrollable and asensitive

- Read-only
  - you cannot update data in the underlying table through the cursor.
- Non-scrollable
  - you can only fetch rows in the order determined by the SELECT statement. (You cannot skip rows or jump to a specific row in the result set.)
- Asensitive:
  - An asensitive cursor points to the actual data (faster),
  - whereas an insensitive cursor uses a temporary copy of the data.
  - However, any change that made to the data from other connections will affect the data that is being used by an asensitive cursor

Cursors

- First declare a Cursor:

```
DECLARE cursor_name CURSOR FOR SELECT_statement;
```

- Open the Cursor

```
OPEN cursor_name;
```

- Fetch the data

```
FETCH cursor_name INTO variables list;
```

- Close Cursor

- ```
  CLOSE cursor_name;
  ```
  ```

Cursors

- Endbedingung

```
DECLARE CONTINUE HANDLER FOR NOT FOUND SET finished = 1;
```

```
CREATE PROCEDURE createEmailList (
    INOUT emailList varchar(4000)
)
BEGIN
    DECLARE finished INTEGER DEFAULT 0;
    DECLARE emailAddress varchar(100) DEFAULT "";

    -- declare cursor for employee email
    DECLARE curEmail
        CURSOR FOR
            SELECT email FROM employees;

    -- declare NOT FOUND handler
    DECLARE CONTINUE HANDLER
        FOR NOT FOUND SET finished = 1;

    OPEN curEmail;

    getEmail: LOOP
        FETCH curEmail INTO emailAddress;
        IF finished = 1 THEN
            LEAVE getEmail;
        END IF;
        -- build email list
        SET emailList = CONCAT(emailAddress,";",emailList);
    END LOOP getEmail;
    CLOSE curEmail;

END$$
```

Exercises

- Siehe Moodle!

Auf los geht‘s los ;-)

Quellen

- http://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx
- https://www.plsqltutorial.com/what-is-plsql/
- https://www.w3schools.com/sql/sql_stored_procedures.asp