<epam>

Stored Routines

Relational Databases Basics

TRAINING
CENTER

<epam>

Read and remember!

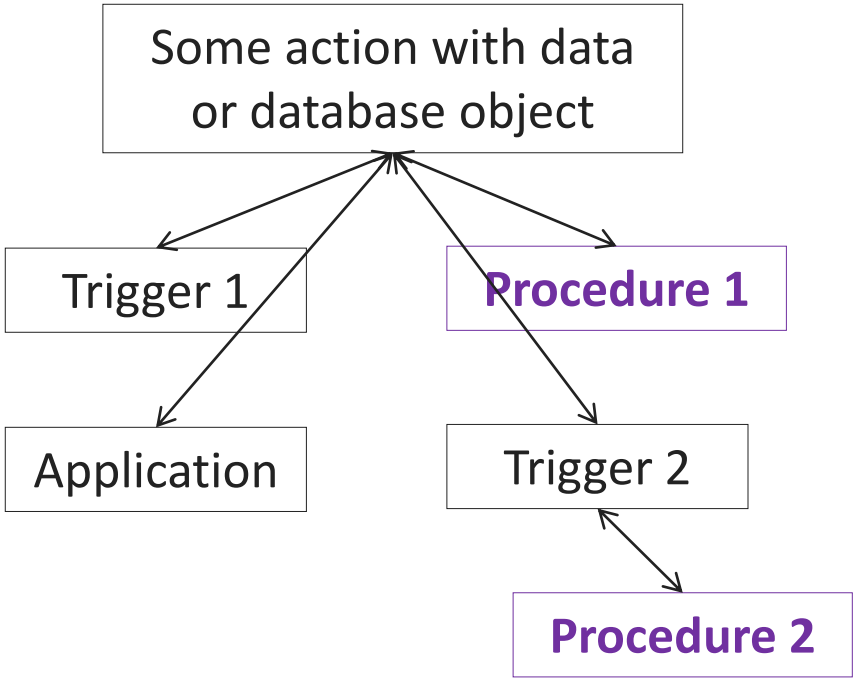
Stored Procedure – a subroutine, possibly parameterized, that is stored in database and created to perform predefined operations on database data and objects; may be called directly, or from triggers, or from other routines.

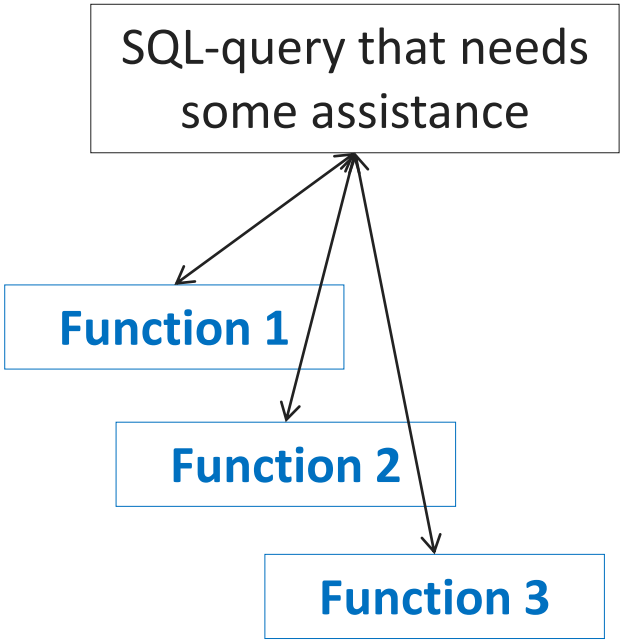
Stored Function (user-defined function, UDF) – a subroutine, possibly parameterized, that is stored in database and created to perform predefined operations enriching built-in SQL capabilities. Usually must return a value.

Stored Procedures vs Stored Functions

Stored Procedure	Stored Function
Allows several *in* and *out* parameters	Allows several *in* parameters, has to return a value (may return a table in some DBMSes)
Has its own *call* and *results processing* syntax	Allows built-in integration in any SQL-query
May call other stored procedures and stored functions	May call other stored functions only
May produce transactions	May not produce transactions
Has rich exceptions handling capabilities	Has limited exceptions handling capabilities (none in some DBMSes)
May perform any data operations	May only read data
May perform any database objects operations	May only read information about database objects
May not be used in *check* constraints	May be used in *check* constraints

Stored Procedures vs Stored Functions: typical usage





Stored Routines pros and cons

Pros	Cons
We only need to write the code once (easily to maintain)	Performance issues (in case of ignoring query plans analysis)
Routine call syntax is more “readable” than the huge code	Security issues (in case of wrong permissions settings)
No data is transmitted “outside” of DBMS (speed, security)	Testing and/or debugging may become more difficult

Stored Procedure sample 1

This stored procedure clears “today’s statistics” every midnight.

```
CREATE PROCEDURE NEW_DAY ()
BEGIN
 IF EXISTS (SELECT 1 FROM `statistics`
                    WHERE `s_actual_date` !=
                          CURRENT_DATE())
  THEN
   UPDATE `statistics` SET
    `s_users_today` = 0,
    `s_uploaded_files_today` = 0,
    `s_uploaded_volume_today` = 0,
    `s_downloaded_files_today` = 0,
    `s_downloaded_volume_today` = 0,
    `s_actual_date` = CURRENT_DATE();
   UPDATE `file` SET
    `f_download_count_today` = 0;
  END IF;
END;
```

```
CREATE EVENT `clear_statistics_at_midnight`
 ON SCHEDULE
  EVERY 1 DAY
  STARTS '2000-01-01 00:00:00'
   ON COMPLETION PRESERVE ENABLE
 DO
  CALL NEW_DAY()
```

Stored Procedure sample 2

This stored procedure clears outdated data every 30 minutes.

```
CREATE EVENT
`clear_outdated_objects_every_30_minutes`
 ON SCHEDULE
  EVERY 30 MINUTE
  STARTS '2000-01-01 00:01:00' ON
COMPLETION PRESERVE ENABLE
 DO
  CALL CLEAR_OUTDATED_OBJECTS()
```

```
CREATE PROCEDURE CLEAR_OUTDATED_OBJECTS ()
BEGIN

 UPDATE `user` SET `u_speed_bonus` = NULL
  WHERE `u_speed_bonus_exp_dt` < UNIX_TIMESTAMP();

 UPDATE `user` SET `u_volume_bonus` = NULL
  WHERE `u_volume_bonus_exp_dt` < UNIX_TIMESTAMP();

 UPDATE `user` SET `u_ban` = NULL
  WHERE `u_ban_exp_dt` < UNIX_TIMESTAMP();

 DELETE FROM `ip_blacklist`
  WHERE `ibl_exp_dt` < UNIX_TIMESTAMP();

 DELETE FROM `file`
  WHERE `f_exp_dt` < UNIX_TIMESTAMP();

 DELETE FROM `download_link`
  WHERE `dl_exp_dt` < UNIX_TIMESTAMP();

END;
```

Stored Function sample 1

This stored function determines user status based on the following set of rules.

Status	Uploaded volume	Uploaded count
Novice	< 1 GB	< 100
Experienced	1-10 GB	100-1000
Master	> 10 GB	> 1000

```
SELECT `u_id`,
       `u_login`,
       SUM(`f_size`) AS `files_volume`,
       COUNT(`f_id`) AS `files_count`,
       GET_USER_STATUS(SUM(`f_size`),
                       COUNT(`f_id`),
                       'NUMBER') AS `user_status`
  FROM `user`
  JOIN `file`
    ON `u_id` = `f_owner`
 GROUP BY (`u_id`)
```

```
CREATE FUNCTION GET_USER_STATUS(uploaded_volume BIGINT UNSIGNED,
                                uploaded_count BIGINT UNSIGNED,
                                return_mode VARCHAR(10))
RETURNS VARCHAR(150) DETERMINISTIC
BEGIN
 DECLARE uploaded_volume_status INT;
 DECLARE uploaded_count_status INT;
 DECLARE final_status INT;

 CASE
  WHEN (uploaded_volume < 1073741824) THEN SET uploaded_volume_status = 1;
  WHEN ((uploaded_volume >= 1073741824)
    AND (uploaded_volume <= 10737418240)) THEN
                                          SET uploaded_volume_status = 2;
  WHEN (uploaded_volume > 10737418240) THEN SET uploaded_volume_status = 3;
 END CASE;

 CASE
  WHEN (uploaded_count < 100) THEN SET uploaded_count_status = 1;
  WHEN ((uploaded_count >= 100)
    AND (uploaded_count <= 1000)) THEN SET uploaded_count_status = 2;
  WHEN (uploaded_count > 1000) THEN SET uploaded_count_status = 3;
 END CASE;


 SET final_status = (SELECT GREATEST(uploaded_volume_status,
                                     uploaded_count_status));

 IF (return_mode = 'NUMBER')
  THEN
   RETURN CONCAT(final_status, '');
  ELSE
   CASE
    WHEN (final_status = 1) THEN RETURN 'NOVICE';
    WHEN (final_status = 2) THEN RETURN 'EXPERIENCED';
    WHEN (final_status = 3) THEN RETURN 'MASTER';
   END CASE;
 END IF;

END;
```

Stored Function sample 2

This stored function determines file size in given measurement units.

```
SELECT NORMALIZE_SIZE(1, '2'),
       NORMALIZE_SIZE (1, '10')
UNION
SELECT NORMALIZE_SIZE (100, '2'),
       NORMALIZE_SIZE (100, '10')
UNION
SELECT NORMALIZE_SIZE (1000, '2'),
       NORMALIZE_SIZE (1000, '10')
UNION
SELECT NORMALIZE_SIZE (10000, '2'),
       NORMALIZE_SIZE (10000, '10')
UNION
SELECT NORMALIZE_SIZE (1000000, '2'),
       NORMALIZE_SIZE (1000000, '10')
UNION
SELECT NORMALIZE_SIZE (250000000, '2'),
       NORMALIZE_SIZE (250000000, '10')
UNION
SELECT NORMALIZE_SIZE (47000000000, '2'),
       NORMALIZE_SIZE (47000000000, '10')
UNION
SELECT NORMALIZE_SIZE (9800000000000, '2'),
       NORMALIZE_SIZE (9800000000000, '10')
UNION
SELECT NORMALIZE_SIZE (7100000000000000, '2'),
       NORMALIZE_SIZE (7100000000000000, '10')
UNION
SELECT NORMALIZE_SIZE (5340000000000000000, '2'),
       NORMALIZE_SIZE (5340000000000000000, '10')
```

```
CREATE FUNCTION NORMALIZE_SIZE(size BIGINT UNSIGNED,
                               measurement VARCHAR(10))
RETURNS VARCHAR(150) DETERMINISTIC
BEGIN
 DECLARE labels_2 VARCHAR(150)
  DEFAULT '["B","KiB","MiB","GiB","TiB","PiB","EiB","ZiB","YiB"]';
 DECLARE labels_10 VARCHAR(150)
  DEFAULT '["B","KB","MB","GB","TB","PB","EB","ZB","YB"]';

 DECLARE position_in_array_2 INT DEFAULT 0;
 DECLARE position_in_array_10 INT DEFAULT 0;

 DECLARE result_2 DOUBLE DEFAULT 0.0;
 DECLARE result_10 DOUBLE DEFAULT 0.0;

 SET position_in_array_2 = TRUNCATE(LOG(2, size) / LOG(2, 1024), 0);
 SET position_in_array_10 = TRUNCATE(LOG(10, size) / LOG(10, 1000), 0);

 SET result_2 = ROUND(size/POWER(1024, position_in_array_2), 2);
 SET result_10 = ROUND(size/POWER(1000, position_in_array_10), 2);

 IF (measurement = '2')
  THEN
   RETURN REPLACE(CONCAT(result_2, ' ', JSON_EXTRACT(labels_2,
                  CONCAT('$[',position_in_array_2,']'))), '"', '');
  ELSE
   RETURN REPLACE(CONCAT(result_10, ' ', JSON_EXTRACT(labels_10,
                  CONCAT('$[',position_in_array_10,']'))), '"', '');
 END IF;

END;
```

1 B	1 B
100 B	100 B
1000 B	1 KB
9.77 KiB	10 KB
976.56 KiB	1 MB
238.42 MiB	250 MB
…	

Live demo in Sparx Enterprise Architect and MySQL Workbench

<epam>

Stored Routines

Relational Databases Basics

TRAINING
CENTER

<epam>