Stored Procedures (SPs) ermöglichen es, eigene Funktionen in einer auf SQL aufbauenden Programmiersprache zu entwickeln. Erweiterungen zu den eigentlichen SQL-Kommandos ermöglichen die Verwaltung von Variablen, die Programmierung von Verzweigungen und Schleifen, …

Quelle : M.Kofler / 1.Auflage 2022 „Datenbanksysteme-Das umfassende Lehrbuch“ /© Rheinwerk Verlag GmbH

	Prozeduren	**Funktionen**
Aufruf	nur mit CALL	in allen SQL-States möglich
Rückgabe	können mehrere Werte (durch **OUT** deklariert) zurückgeben	können nur einen Wert (durch **RETURNS** deklariert) zurückgeben
Übergabe	mehrere Werte (entsprechend **IN** und **OUT** oder als **INOUT**) möglich	mehrere Werte (entsprechend der Deklaration) möglich
zulässige Kommandos	alle SQL-Kommandos (SELECT, INSERT, UPDATE ...)	alle SQL-Kommandos (SELECT, INSERT, UPDATE ...)
gegenseitiger Aufruf	dürfen andere Funktionen und Prozeduren aufrufen (auch Rekursiv)	dürfen nur Funktionen aufrufen (nicht Rekursiv)

☝	Da im Code selbst das Semikolon ; ASCII(59D) enthalten ist, muss vor und nach der Deklar-ation von SP’s das Delimiter-Zeichen (hier z.B.: mit der Anweisung **DELIMITER $$**) geändert werden.

Funktionen

```
① CREATE FUNCTION Name ([Var Typ, ...]) RETURNS Datentyp [DETERMINISTIC]
② [Label1:] BEGIN
③     [DECLARE Variable Datentyp; …]
④     Anweisung(en);
⑤     RETURN Wert;
⑥ END [:Label2] $$
```

❶ Funktionsaufrufe werden (wie vordefinierte SQL-Funktionen) in Kommandos integriert

```
SELECT Name([Wert1, Wert2,...]) FROM ... ;
```

Proceduren

```
① CREATE PROCEDURE Name ([IN Var1 Typ,… ,OUT Var2 Typ,… ,INOUT Var3 Typ])
② [Label1:] BEGIN
③     [DECLARE Variable Datentyp; …]
④     Anweisung(en);
⑤ END [:Label2] $$
```

❶ Der Prozeduraufruf erfolgt mit **CALL** (und kann Ein- und Ausgaben) beinhalten.

```
CALL Name([IN_Wert],[@Out_var,...],[@INOUT_var]);
z.B.: SET @BINOUT=5 ; CALL testProc(4,@AOUT,@BINOUT) ; SELECT @AOUT , @BINOUT ;
```

Trigger

```
① CREATE TRIGGER Triggername BEFORE|AFTER INSERT|UPDATE|DELETE ON
  Tablename FOR EACH ROW
② [Label1:] BEGIN
③     [DECLARE Variable Datentyp; …]
④     Anweisung(en);
⑥ END [:Label2] $$
```

OLD.col_name liefert den Inhalt eines existierenden Datensatzes vor UPDATE oder DELETE
NEW.col_name liefert den neuen Inhalt eines Datensatz nach INSERT oder UPDATE

❶ Trigger werden automatisch vor oder nach (entsprechend der Deklaration mit **BEFORE|AFTER**) dem SQL – State ausgelöst (**INSERT|UPDATE|DELETE**) für welche Tabelle sie deklariert wurden.

☝	Alle Elemente werden „unter“ der Datenbasis gespeichert, für die sie angelegt wurden. Mit **DROP FUNCTION|PROCEDUR|TRIGGER [IF EXISTS]** Name ; bzw. **DROP DATABASES** Name ; werden Elemente einzeln oder in der Gesamtheit gelöscht.

☝	Mit **SIGNAL** kann eine Ausnahme (EXCEPTION) ausgelöst werden. Damit wird die aktuelle Abarbeitung (z.B.: in einem BEFORE INSERT-Trigger, dass Einfügen des Datensatzes) abgebrochen. **IF** Fehlerbedingung=**TRUE THEN** **SIGNAL SQLSTATE** '45000' **SET MESSAGE_TEXT** = "Es ist ein Fehler aufgetreten" , **MYSQL_ERRNO** = 1062; **END IF;**

In allen Quellcodeteilen können benutzerdefinierte Variablen (müssen direkt nach **BEGIN** mit **DECLARE** festgelegt werden) oder Session-Variablen (also **@Variable**) gleichwertig genutzt werden.

DECLARE var_1 , var_2 , ... Datentyp [**DEFAULT** wert];

Zuweisungsmöglichkeiten:

SET var_1=$wert_1$, var_2=$wert_2$, var_3=$wert_3$ ...;
SELECT var:= wert;
SELECT wert **INTO** var;
SELECT $wert_1$[,$wert_2$,…] **INTO** var_1[,var_2,…] **FROM** …

Benutzerdefinierte Variablen sind **lokal**. Session-Variablen sind (in dieser Session) **global**.

Wie in allen bekannten Programmiersprachen stehen die gängigen Kontrollstrukturen zur Verfügung.

Verzweigung	**Auswahl**
IF Bedingung **THEN** Anweisung(en)… **[ELSEIF** Bedingung **THEN** Anweisung(en)… **]** **[ELSE** Anweisung(en)…**]** **END IF;**	**CASE** Variable **WHEN** $Wert_1$ **THEN** Anweisung(en)… **[WHEN** $Wert_2$ **THEN** Anweisung(en)…**]** **[ELSE** Anweisung(en)…**]** **END CASE;**
Kopfgesteuerte bedingte Wiederholung	**Endlos Schleife (?)** (mit LEAVE und/oder ITERATE)
[Label:] **WHILE** Bedingung **DO** Anweisung(en)… **END WHILE;**	[Label:] **LOOP** Anweisung(en)… **END LOOP;**
Zählergesteuerte Wiederholung	**Fußgesteuerte bedingte Wiederholung**
[Label:] **FOR** Variable **IN [REVERSE]** Start**..** Ende **DO** Anweisung(en)… **END FOR;**	[Label:] **REPEAT** Anweisung(en)… **UNTIL** Bedingung **END REPEAT;**
Mit den Anweisungen werden (meist in Abhängigkeit von Bedingungen) Schleifendurchläufe vorzeitig beendet **LEAVE** label springt zum Schleifenende, oder **ITERATE** label beginnt den nächsten Schleifendurchlauf.	

☝	Für alle Ausführungen gilt insbesondre zu beachten: • hierbei unterscheiden sich DBMS „MySQL“ und „MariaDB“ in einzelnen Details. Den aktuellen Entwicklungsstand findet man in der Anwenderdokumentation. • außerdem unterliegen (z.B. Trigger) der ständigen Weiterentwicklung. Dadurch ergeben sich Versions abhängige Unterschiede. • Bei der Erstellung (2/2023) wurde mit der aktuellen MariaDB Version 10.4.24 getestet.