



DB: Datenbanken

Stored Procedures

Prof. Dr. Ludger Martin

Gliederung

- ★ Einführung
- ★ Erzeugen von Stored Procedures
- ★ Verwalten von Stored Procedures
- ★ Programmsteuerung
- ★ Cursor
- ★ Fehlerbehandlung
- ★ Trigger

Einführung

- ★ Selbst definierbare SQL-Prozeduren und Funktionen
- ★ SQL-basierte Programmiersprache
- ★ Kein einheitlicher Standard vorhanden
- ★ Stored Procedures werden auf dem Server ausgeführt

Einführung

Vorteile:

☆ Höhere Geschwindigkeit:

- ☆ Daten von z.B. einem SELECT müssen nicht zum Client übertragen werden, um bearbeitet zu werden
- ☆ Code kann vom DB-Server vorverarbeitet werden (ähnlich Kompilierung)
- ☆ Stored Procedures führen zu einer höheren Belastung des DB-Servers aber Entlastung der Anwendung

Einführung

Vorteile:

- ★ Vermeidung von Coderedundanzen, bessere Wartbarkeit:
 - ★ Ähnliche Codepassagen in unterschiedlichen Anwendungen (Einfügen, Ändern von Daten)
 - ★ Bei Änderungen am Schema muss oft nur Stored Procedure geändert werden
- ★ Erhöhung von Datensicherheit
 - ★ Zugriff auf Daten z.B. nur über Stored Procedures
 - Jeder Datenzugriff kann dadurch kontrolliert oder protokolliert werden

Einführung

2 Typen von Stored Procedures

★ Prozeduren

- ★ Rückgabeparameter
- ★ Lassen mehr SQL-Kommandos zu
- ★ Werden mit `CALL` aufgerufen

★ Funktionen

- ★ Können Wert zurückgeben

Erzeugen von Stored Procedures

Die `[]` gehören nicht zur Syntax!

- ★ `CREATE FUNCTION sp_name ([arg_list])
RETURNS type [characteristic] routine_body`
- ★ `CREATE PROCEDURE sp_name ([arg_list])
[characteristic] routine_body`

★ Characteristic:

★ `[NOT] DETERMINISTIC`: Wenn eine Stored Procedure bei gegebenen Parametern immer das selbe Ergebnis liefert, ist sie deterministisch

★ Wenn Sie von Datenbanktabellen abhängt, ist sie nicht deterministisch.

★ Standard ist `NOT DETERMINISTIC`.

Erzeugen von Stored Procedures

Die [] gehören nicht zur Syntax!

★ CREATE FUNCTION `sp_name` (`[arg_list]`)
RETURNS `type` `[characteristic]` `routine_body`

★ CREATE PROCEDURE `sp_name` (`[arg_list]`)
`[characteristic]` `routine_body`

★ Characteristic

Stored Procedures werden im gewählten Datenbank-Schema gespeichert!

★ Wenn die von Datenbanktabellen abhängt, ist die nicht deterministisch

★ Standard ist NOT DETERMINISTIC

Erzeugen von Stored Procedures

- ★ Zeilenwechsel und zusätzliche Leerzeichen haben keine syntaktische Bedeutung
- ★ Groß- und Kleinschreibung wird ignoriert
- ★ Besteht eine Stored Procedure aus mehr als einer Anweisung, müssen diese zwischen `BEGIN` und `END` gestellt werden
- ★ Anweisungen in Stored Procedures werden mit `;` getrennt

Erzeugen von Stored Procedures

Wichtig:

- ★ In SQL dient ; dazu, Befehle abzuschließen. Dies verträgt sich nicht mit dem ; aus den Stored Procedures!
- ★ Im Befehlszeilen-mysql-Client kann Abhilfe durch Definition eines neuen Delimiter-Zeichens geleistet werden:


```
DELIMITER $$
```

- ★ Squirrel SQL unter Linux stört sich nicht am ; , für Windows beachte den Hinweis in moodle.

Erzeugen von Stored Procedures

★ Beispiel Funktion:

```
CREATE FUNCTION shorten (s VARCHAR(255))  
  RETURNS VARCHAR(13) DETERMINISTIC  
BEGIN  
  IF CHAR_LENGTH(s) <= 10 THEN  
    RETURN s;  
  ELSE  
    RETURN CONCAT(LEFT(s, 10), '...');  
  END IF;  
END
```



★ Aufruf:

```
SELECT shorten('12345678901234')
```

Erzeugen von Stored Procedures

- ★ Mit `SET` oder `SELECT ... INTO ...` können Werte in SQL-Variablen gespeichert werden
- ★ SQL-Variablen beginnen mit `@`
- ★ SQL-Variablen halten den Inhalt bis zum Ende der Verbindung
- ★ **Beispiel:**

```
SET @s = 'eine ganz lange Zeichenreihe';
```

Erzeugen von Stored Procedures

★ Weitere Beispiele:

★ SET @a = shorten(@s) ;

★ SELECT shorten(@s) INTO @b ;

★ SELECT @a, @b, @s ;

@a	@b	@s
eine ganz ...	eine ganz ...	eine ganz lange Zeichenreihe

Erzeugen von Stored Procedures

- ★ **Lokale Variablen** werden zu Beginn eines BEGIN ... END Blocks mit DECLARE definiert
- ★ DECLARE `var1, var2 data_type [DEFAULT value]`
- ★ Lokale Variablen haben kein @ Zeichen.
- ★ Lokale Variablen verlieren ihre Gültigkeit nach Verlassen des Blocks.



Die { } | gehören nicht zur Syntax!

Erzeugen von Stored Procedures

★ Beispiel DECLARE:

```
CREATE FUNCTION swap_name(s VARCHAR(100))
RETURNS VARCHAR(100) DETERMINISTIC
BEGIN
    DECLARE pos INT;
    SET s = TRIM(s);
    SET pos = LOCATE(' ', s);
    IF pos = 0 THEN
        RETURN s;
    END IF;
    RETURN CONCAT(SUBSTR(s, pos+1), ' ',
                  LEFT(s, pos));
END
```

Erzeugen von Stored Procedures

- ★ Bei Prozeduren muss jeder Parameter als Ein-, Aus- oder Einausgabeparameter markiert werden
 - ★ IN
 - ★ OUT
 - ★ INOUT
- ★ Prozeduren werden mit `CALL` aufgerufen

Erzeugen von Stored Procedures

★ Beispiel:

```
CREATE PROCEDURE half(IN a INT,  
                      OUT b INT)  
    DETERMINISTIC  
BEGIN  
    SET b = a/2;  
END
```

★ Aufruf:

```
CALL half(10, @result) ;  
SELECT @result ;
```

Erzeugen von Stored Procedures

★ Prozeduren

- ★ **Aufruf:** Nur mit `CALL`
- ★ **Rückgabe:** ein oder mehrere Ergebnisse
- ★ **Parameter:** Wert- und Rückgabeparameter (`IN`, `OUT`, `INOUT`)
- ★ **Zulässige Kommandos:** Alle SQL-Kommandos (`SELECT`, `INSERT`, ...)
- ★ **Gegenseitiger Aufruf:** andere Prozeduren und Funktionen

Erzeugen von Stored Procedures

★ Funktionen:

- ★ **Aufruf:** In allen SQL-Kommandos (SELECT, UPDATE)
- ★ **Rückgabe:** Geben einzelne Werte, wie bei Definition angegeben, zurück
- ★ **Parameter:** nur Wertparameter
- ★ **Zulässige Kommandos:** Keine Kommandos auf Tabellen
- ★ **Gegenseitiger Aufruf:** nur Funktionen

Verwalten von Stored Procedures

★ Löschen von Stored Procedures

★ DROP FUNCTION `sp_name`

★ DROP PROCEDURE `sp_name`

★ Anzeigen von Stored Procedures

★ SHOW FUNCTION STATUS

★ SHOW PROCEDURE STATUS

★ Quelltext ermitteln

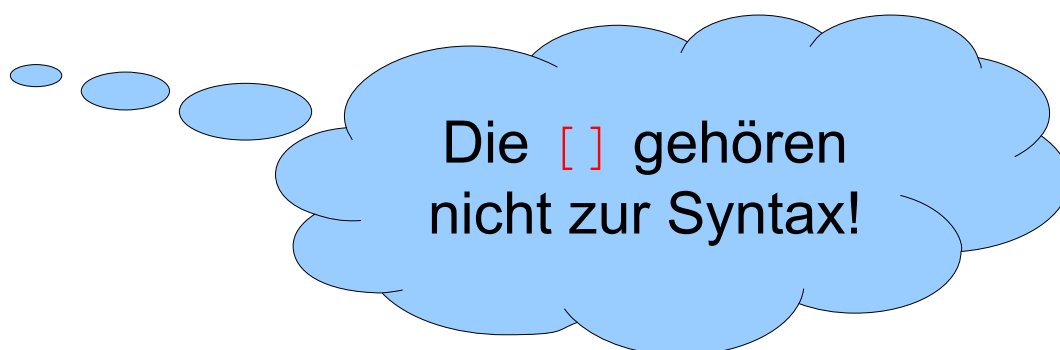
★ SHOW CREATE FUNCTION `sp_name`

★ SHOW CREATE PROCEDURE `sp_name`

Programmsteuerung

★ If-Bedingung

```
IF bedingung THEN  
    anweisungen;  
[ ELSEIF bedingung THEN  
    anweisungen; ]  
[ ELSE  
    anweisungen; ]  
END IF;
```

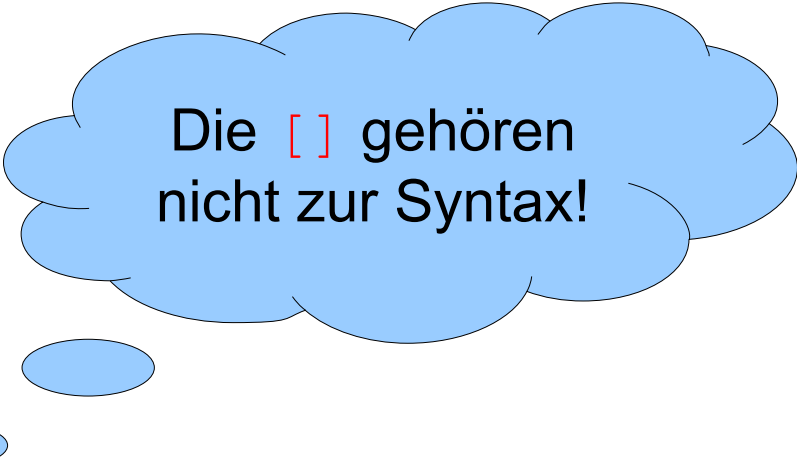


Die [] gehören
nicht zur Syntax!

Programmsteuerung

★ Case-Verzweigung:

```
CASE ausdruck
  WHEN wert THEN
    anweisungen;
  [ WHEN wert THEN
    anweisungen; ]
  [ ELSE
    anweisungen; ]
END CASE;
```



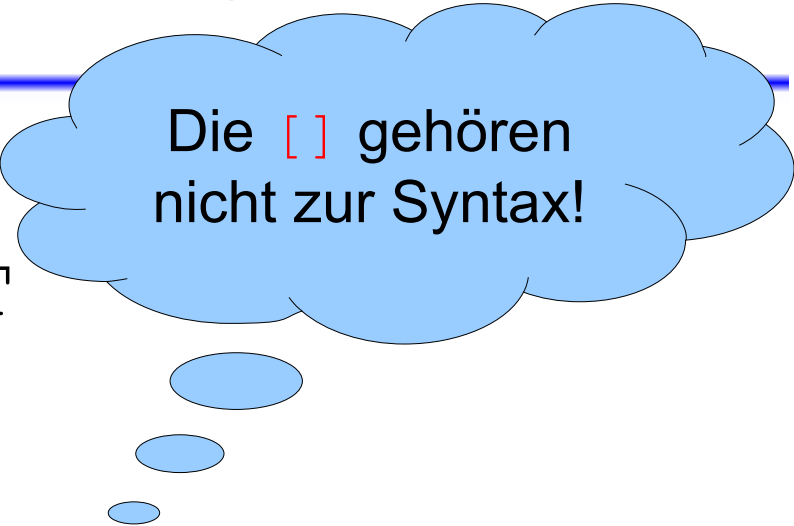
Die `[]` gehören
nicht zur Syntax!

Programmsteuerung

★ Schleifen:

★ `[schleifenname:] REPEAT
 anweisungen;
UNTIL bedingung
END REPEAT [schleifenname];`

★ `[schleifenname:] WHILE bedingung DO
 anweisungen;
END WHILE [schleifenname];`

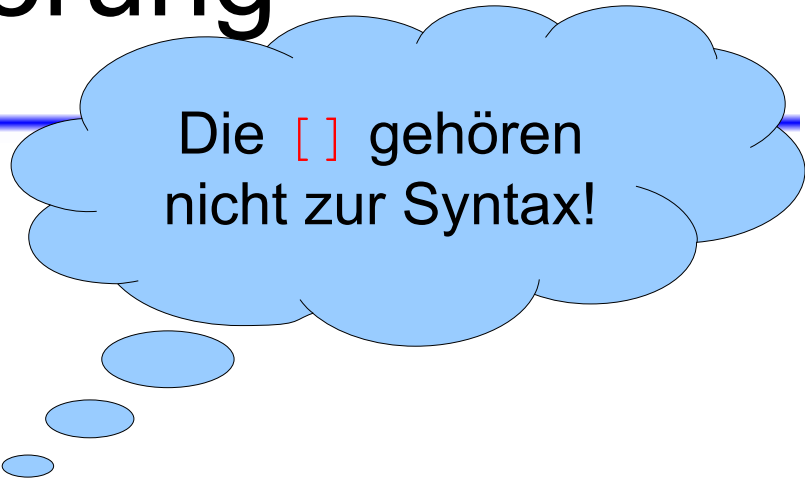


Die `[]` gehören
nicht zur Syntax!

Programmsteuerung

★ Schleifen:

```
★ [schleifenname:] LOOP  
    anweisungen;  
END LOOP [schleifenname];
```



Die `[]` gehören
nicht zur Syntax!

★ Mit `LEAVE schleifenname` kann eine Schleife beendet werden

★ Mit `ITERATE schleifenname` kann eine Schleife erneut ausgeführt werden

Cursor

- ★ Zeiger auf bestimmten Datensatz
- ★ Ermöglichen das Durchlaufen von Datensätzen aus einer Tabelle

- ★ **Vorgehensweise:**

- 1. Cursor definieren:

- ```
DECLARE cursor_name CURSOR FOR
 select_statement
```

- ★ Beliebige SELECT-Anweisung darf angegeben werden

# Cursor

## ★ Vorgehensweise: (Fortsetzung)

### 2.Fehlerhandler für NOT FOUND definieren

```
DECLARE done INT DEFAULT 0;
DECLARE CONTINUE HANDLER FOR
 NOT FOUND SET done = 1;
```

★ In Schleife Variable `done` bei Ergebnisabfrage prüfen

### 3.Cursor öffnen:

```
OPEN cursor_name
```

# Cursor

## ★ Vorgehensweise: (Fortsetzung)

### 4. Ergebnisse abfragen:

```
FETCH cursor_name INTO var_name
 [, var_name] ...
```

- ★ Holt jeweils eine Zeile aus der Ergebnismenge
- ★ Die angegebenen Variablen müssen zuvor definiert werden
- ★ Die Anzahl der Variablen muss mit den selektierten Spalten übereinstimmen

### 5. Cursor schließen:

```
CLOSE cursor_name
```

# Cursor

## ★ Beispiel Cursor (Teil 1):

```
CREATE PROCEDURE incrementStd()
BEGIN
 DECLARE _angnr, _std MEDIUMINT(5);
 DECLARE done int DEFAULT 0;
 DECLARE cur CURSOR FOR
 SELECT angnr, std FROM pilot;
 DECLARE CONTINUE HANDLER FOR
 NOT FOUND SET done = 1;
 OPEN cur;
```

# Cursor

## ★ Beispiel Cursor (Teil 2):

```
REPEAT
 FETCH cur INTO _angnr, _std;
 IF NOT done THEN
 SET _std = _std + 1;
 UPDATE pilot SET std=_std
 WHERE angnr=_angnr;
 END IF;
UNTIL done END REPEAT;
CLOSE cur;
END
```

# Fehlerbehandlung

---

- ★ In Stored Procedures können Fehler geworfen werden, die in Client abgefangen werden können.
- ★ Mit `SIGNAL` kann ein Fehler geworfen werden.
  - ★ Ein Signal hat eine Nummer und eine Nachricht.
  - ★ Nummer ist ein String aus 5 Zeichen, die ersten beiden definieren den Typ.

# Fehlerbehandlung

## ★ Mögliche Werte für SQLSTATE

★ '00xxx' → success

★ '01xxx' → warning

★ '02xxx' → not found

★ >= '03xxx' → exception

★ 'HYxxx' → general error

## ★ Syntax:

```
SIGNAL SQLSTATE '5 char'
```

```
SET MESSAGE_TEXT = 'An error message';
```

# Fehlerbehandlung

```
★ CREATE FUNCTION swap_name(s VARCHAR(100))
 RETURNS VARCHAR(100)
 BEGIN
 DECLARE pos INT;
 SET s = TRIM(s);
 SET pos = LOCATE(' ', s);
 IF pos = 0 THEN
 SIGNAL SQLSTATE '55005'
 SET MESSAGE_TEXT = 'no whitespace';
 END IF;
 RETURN CONCAT(SUBSTR(s, pos+1), ' ',
 LEFT(s, pos));
 END
```



# Trigger

---

- ★ Trigger können auch mehrere Anweisungen durch `BEGIN . . . END` enthalten
- ★ Es dürfen die gleichen Sprachelemente wie in Stored Procedures genutzt werden

# Literatur

---

- ★ Kofler, Michael: MySQL 5, 3. Auflage, Addison-Wesley, 2005
- ★ Vossen, Gottfried: Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme, 5. Auflage, Oldenburg Wissenschaftsverlag, 2008
- ★ Lubkowitz, M: Webseiten programmieren und gestalten, Galileo Press, 2004
- ★ Oracle: MySQL 5.7 Reference Manual,  
<https://dev.mysql.com/doc/refman/5.7/en/>