Funktionen, Prozeduren & Trigger

Inhalt

1. Allgemeines 2
2. Funktionen (Functions) 3
2.1. Zweck 3
2.2. Anwendungsbeispiele 3
2.3. Aufbau 4
2.4. Beispiel 5
3. Prozeduren (Procedures) 6
3.1. Zweck 7
3.1. Anwendungsbeispiele 7
3.2. Aufbau 8
3.3. Beispiel 9
4. Trigger 10
4.1. Zweck 11
4.2. Anwendungsbeispiele 11
4.3. Aufbau 12
4.4. Beispiel 13
4.5. Zusammenfassung 14
4.5.1. OLD & NEW 14
4.5.2. Trigger – Typ 14

1. Allgemeines

- SQL Funktionen, Prozeduren und Trigger sind wichtige Elemente, um bestimmte Aufgaben zu automatisieren, Daten zu verarbeiten und die Datenbankintegrität zu gewährleisten

ℹ Funktionen sind wiederverwendbare Codeblöcke, die einen einzelnen Wert oder eine Tabelle zurückgeben

ℹ Prozeduren können komplexe Anweisungen (eine Abfolge mehrerer SQL – Befehle) ausführen

ℹ Trigger sind spezielle Prozeduren, die automatisch bei Datenbankereignissen ausgelöst werden

2. Funktionen (Functions)

- Benutzerdefinierte Funktionen sind Routinen, die Parameter annehmen, eine Aktion ausführen und das Ergebnis dieser Aktion als Wert zurückgeben
- ➢ Der Rückgabewert kann ein einzelner Skalarwert oder ein Resultset sein

- z. B. können Funktionen erstellt werden, um (komplexe) Berechnung durchzuführen

2.1. Zweck

- Berechnen und zurückgeben eines Wertes.
- ➢ Werden meist in SELECTs, WHERE-Klauseln oder SET-Anweisungen verwendet
- ➢ Deterministisch (bei gleichen Eingaben immer gleiche Ausgabe)

2.2. Anwendungsbeispiele

- Formatieren von Daten (z. B. Groß-/Kleinschreibung)

- Berechnungen (z. B. Bruttobetrag aus Nettowert + Steuer)

- Prüfungen (z. B. ob eine ID gültig ist)

2.3. Aufbau

```
DELIMITER $$

CREATE FUNCTION funktionsname(parameter1 DATENTYP,
parameter2 DATENTYP, ...)
RETURNS RÜCKGABETYP
[DETERMINISTIC | NOT DETERMINISTIC]
BEGIN
    -- Lokale Variablen (optional)
    DECLARE varname DATENTYP;

    -- Logik (z. B. Berechnungen, Abfragen)
    -- ...

    -- Rückgabe des Ergebnisses
    RETURN irgendetwas;
END$$

DELIMITER ;
```

2.4. Beispiel

```
DELIMITER $$

CREATE FUNCTION berechneBrutto(netto DECIMAL(8, 2),
steuersatz DECIMAL(5, 2))
RETURNS DECIMAL(10, 2)
DETERMINISTIC
BEGIN
    RETURN netto + (netto * (steuersatz / 100));
END$$

DELIMITER ;



SELECT berechneBrutto(100, 19);
```

3. Prozeduren (Procedures)

- Prozeduren sind Anweisungen in Datenbankmanagementsystemen, mit der ganze Abläufe von Anweisungen vom Datenbank-Client aufgerufen werden können
- ➢ Prozeduren sind somit ein eigenständiger Befehl, der eine Abfolge gespeicherter Befehle ausführt

- Mittels gespeicherter Prozeduren können häufiger verwendete Abläufe auf das Datenbanksystem ausgelagert werden
- ➢ Die Prozeduren werden anschließend durch einen einzigen Aufruf (CALL oder EXECUTE) ausgeführt

- Die Abläufe würden sonst durch viele einzelne Befehle vom Client ausgeführt werden
- ➢ Dies kann zu Leistungseinbußen führen

- Gespeicherte Prozeduren tragen dazu bei, die Sicherheit einer Anwendung stark zu erhöhen
- ➢ Der Client braucht in der Regel keine DELETE-, UPDATE- oder INSERT-Zugriffsrechte mehr
- ➢ Somit ist es Angreifern nicht möglich, selbst Datenbanken zu manipulieren, z. B. durch SQL-Injections

- Der Client kann nur bereits vorgefertigte Prozeduren aufzurufen

3.1. Zweck

- Führt eine Abfolge von SQL-Anweisungen aus
- Kann mehrere Schritte, z. B. Einfügen + Protokollieren enthalten
- Kann mehrere Parameter nutzen (IN, OUT, INOUT)
- Gibt keinen Wert direkt zurück, aber kann Daten verändern oder via OUT-Parameter zurückgeben

3.1. Anwendungsbeispiele

- Insert in mehreren Tabellen
- Automatisierung von Aufgaben (z. B. Monatsabschluss)
- Batchverarbeitung von Datensätzen

3.2. Aufbau

```
DELIMITER $$

CREATE PROCEDURE prozedurname(
    IN eingabe1 DATENTYP,
    OUT ausgabe1 DATENTYP,   -- Optional
    INOUT wert DATENTYP      -- Optional
)
BEGIN
    -- Logik, z. B. Abfragen, INSERT, UPDATE
END$$

DELIMITER ;
```

- Parameterarten:
- ➢ IN → Eingabewert
- ➢ OUT → Ausgabe wird durch die Prozedur gesetzt
- ➢ INOUT → Wird gelesen und anschließend verändert

❶ Anders als Funktionen geben Prozeduren keinen RETURN-Wert zurück, sondern arbeiten mit OUT-Parametern oder verändern direkt Daten (z. B. per INSERT)

3.3. Beispiel

```
DELIMITER $$

CREATE PROCEDURE bestellungEinfuegen(
    IN p_kundenId INT,
    IN p_artikelId INT,
    IN p_anzahl INT
)
BEGIN
    DECLARE preis DECIMAL(10,2);
    DECLARE gesamt DECIMAL(10,2);

    SELECT nettopreis INTO preis
    FROM artikel
    WHERE artikelId = p_artikelId;

    SET gesamt = p_anzahl * preis;

    INSERT INTO bestellung(kundenId, artikelId, anzahl,
    gesamtpreis)
    VALUES (p_kundenId, p_artikelId, p_anzahl, gesamt);
END$$

DELIMITER ;

CALL bestellungEinfuegen(1, 2, 2);
CALL bestellungEinfuegen(1, 5, 10);
```

4. Trigger

- Ein Trigger wird automatisch ausgeführt, wenn ein Ereignis auf dem Datenbankserver auftritt

- DML-Trigger werden ausgeführt, wenn Benutzer versuchten Daten mithilfe eines DML-Ereignisses (Data Manipulation Language) zu ändern
  - ➢ Zu den DML – Befehlen zählen INSERT-, UPDATE- oder DELETE-Anweisungen für eine Tabelle oder Sicht

- Diese Trigger werden ausgelöst, sobald ein beliebiges gültiges Ereignis ausgelöst wird

- Trigger werden eingesetzt, um die Integrität der Daten zu wahren, und Aufgaben zu automatisieren

4.1. Zweck

- Automatisches Auslösen bei DML-Operationen (INSERT, UPDATE, DELETE) auf einer Tabelle

- Wird implizit (automatisch) ausgeführt

- Dient der Integritätswahrung, Protokollierung, Berechnung, etc.

4.2. Anwendungsbeispiele

- Automatisches Protokollieren von Änderungen

- Berechnung abhängiger Werte bei Datenänderungen

- Überprüfung auf ungültige Änderungen

4.3. Aufbau

```
DELIMITER $$

CREATE TRIGGER triggername
{BEFORE | AFTER} {INSERT | UPDATE | DELETE}
ON tabellenname
FOR EACH ROW
BEGIN
    -- Triggerlogik (z. B. Protokollieren, Berechnen, Prüfen)
END$$

DELIMITER ;
```

BEFORE → vor der Aktion

AFTER → nach der Aktion

NEW → Daten nach der Änderung (bei INSERT und UPDATE)

OLD → Daten vor der Änderung (bei UPDATE und DELETE)

4.4. Beispiel

```
DELIMITER $$

CREATE TRIGGER trg_check_bestand_before_insert
BEFORE INSERT ON bestellung
FOR EACH ROW
BEGIN
    DECLARE lager INT;

    # Aktuellen Lagerbestand holen
    SELECT bestand INTO lager
    FROM artikel
    WHERE artikelId = NEW.artikelId;

    # Prüfen, ob genug auf Lager ist
    IF NEW.anzahl > lager THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Nicht genug Bestand für
die Bestellung!';
    END IF;
END$$

DELIMITER ;
```

4.5. Zusammenfassung

4.5.1. OLD & NEW

DML	OLD erlaubt?	NEW erlaubt?
INSERT	☒ Nein	☑ Ja
UPDATE	☑ Ja	☑ Ja
DELETE	☑ Ja	☒ Nein

4.5.2. Trigger – Typ

Trigger – Typ	Auslöser	Typischer Anwendungsfall
BEFORE INSERT	Vor dem Einfügen	Validierungen, Standardwerte setzen
AFTER INSERT	Nach dem Einfügen	Berechnungen, Logs
BEFORE UPDATE	Vor dem Ändern	Prüfen, ob Änderung erlaubt ist
AFTER UPDATE	Nach dem Ändern	Änderungs-Log, Benachrichtigungen
BEFORE DELETE	Vor dem Löschen	Verhindern, wenn Referenzen existieren
AFTER DELETE	Nach dem Löschen	Archivieren, Protokollieren