Tiefergehende Datenbankobjekte

Index, Stored Procedure, Trigger, Sequence

Wozu diese Objekte?

- SQL kann mehr als nur Daten speichern und abfragen
- Datenbankobjekte helfen beim:
  - **Optimieren** von Abfragen
  - **Automatisieren** von Prozessen
  - **Verwalten** von IDs oder Logik

Index

- Ein **Index** ist wie ein Register/Inhaltsverzeichnis in einem Buch
- Macht **Abfragen schneller**, besonders bei großen Tabellen
- Wird auf eine oder mehrere Spalten gesetzt

```
CREATE INDEX idx_name ON Kunden(Name);
```

Vorteile:

- Schnelleres SELECT, WHERE und ORDER BY
- Kein Einfluss auf Daten

Nachteile:

- Mehr Speicherplatz
- Leicht langsameres INSERT, UPDATE und DELETE

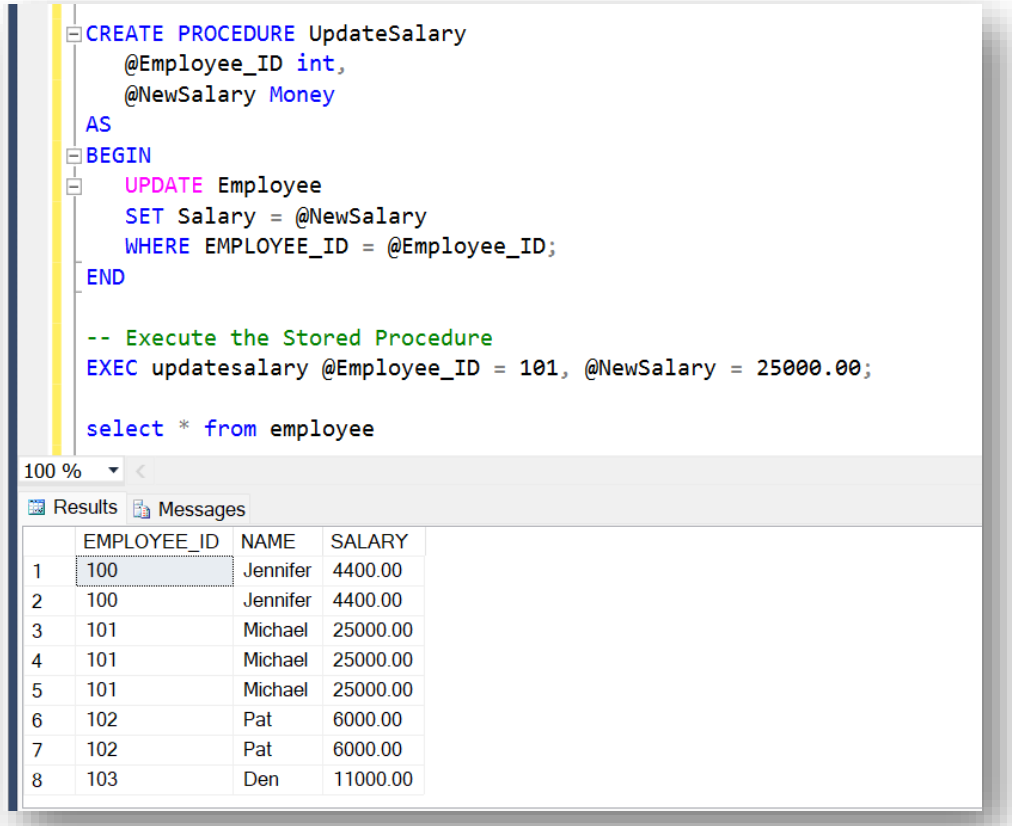
Stored Procedure – Wiederverwendbare SQL-Logik

- Eine **Stored Procedure** ist ein **gespeichertes SQL-Programm**
- Kann **mehrere Anweisungen** enthalten
- Wird direkt in der DB gespeichert und aufgerufen

```
CREATE PROCEDURE GetKunden()
BEGIN
  SELECT * FROM Kunden;
END;
```

Vorteile:

- Wiederverwendbar
- Weniger Netzwerkverkehr
- Trennung von Daten und Logik



```
CREATE PROCEDURE UpdateSalary
    @Employee_ID int,
    @NewSalary Money
AS
BEGIN
    UPDATE Employee
    SET Salary = @NewSalary
    WHERE EMPLOYEE_ID = @Employee_ID;
END

-- Execute the Stored Procedure
EXEC updatesalary @Employee_ID = 101, @NewSalary = 25000.00;

select * from employee
```

100 %

Results | Messages

	EMPLOYEE_ID	NAME	SALARY
1	100	Jennifer	4400.00
2	100	Jennifer	4400.00
3	101	Michael	25000.00
4	101	Michael	25000.00
5	101	Michael	25000.00
6	102	Pat	6000.00
7	102	Pat	6000.00
8	103	Den	11000.00

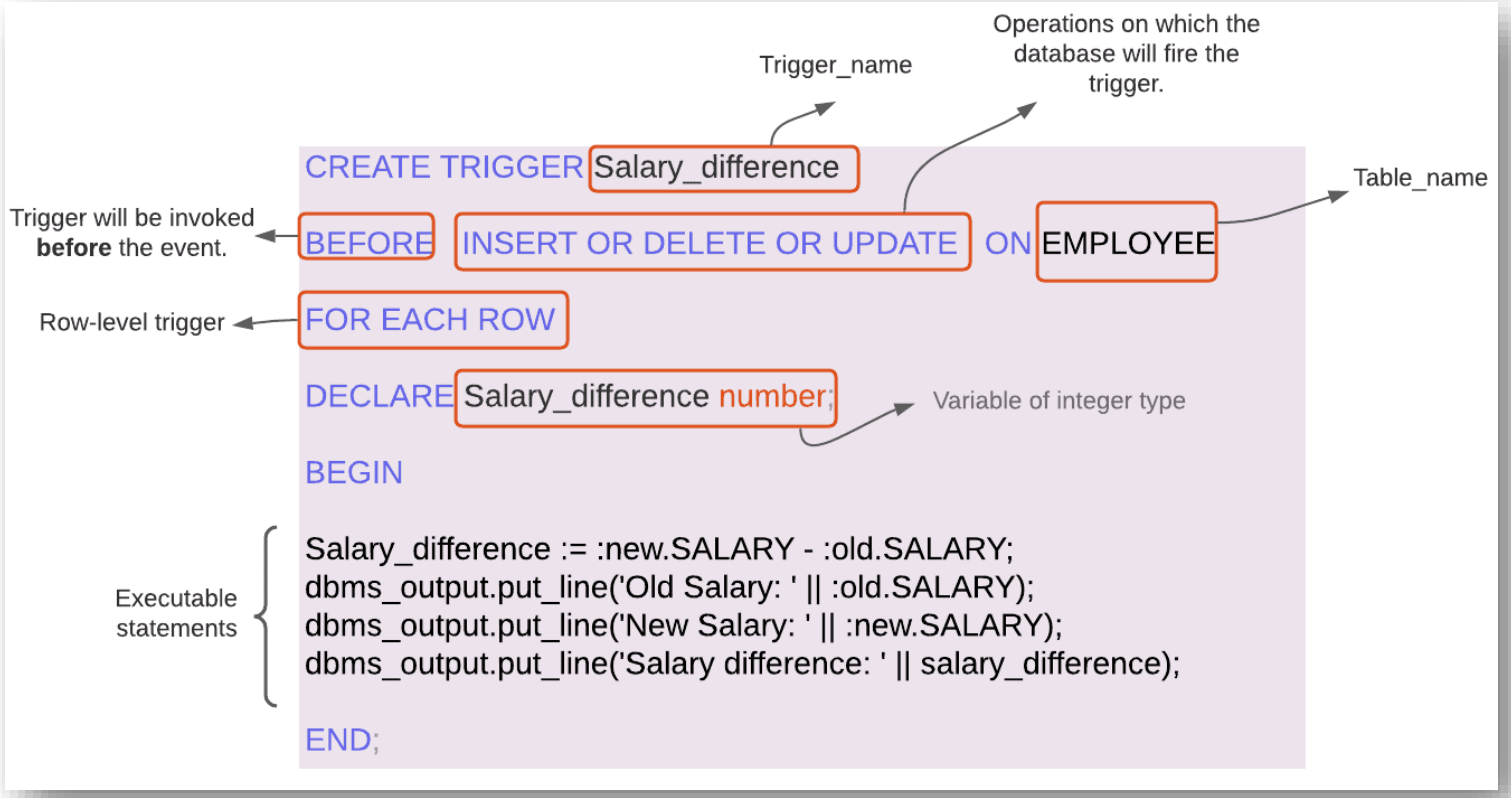
Trigger – Automatische Reaktionen

- Ein **Trigger** wird automatisch ausgelöst bei **Änderungen** an einer Tabelle
- Beispiel: Logbuch aktualisieren, wenn Datensatz geändert wird

```
CREATE TRIGGER trg_log
AFTER UPDATE ON Kunden
FOR EACH ROW
INSERT INTO Logbuch (Aktion)
VALUES ('Kunde geändert');
```

Arten von Triggern:

- BEFORE / AFTER
- INSERT, UPDATE, DELETE



Sequence – Fortlaufende Nummern erzeugen

- Eine **Sequence** erzeugt automatisch eindeutige Werte
- Typisch für **Primärschlüssel / Rechnungsnummern / Seriennummern / Bestellnummern / ..**

```
CREATE SEQUENCE kunde_seq
START WITH 1
INCREMENT BY 1;

INSERT INTO Kunden(ID, Name)
VALUES (kunde_seq.NEXTVAL, 'Max‘);
```

Vorteile:

- Kein manueller Zähler
- Paralleles Arbeiten ohne doppelte IDs

```
CREATE SEQUENCE dbo.SEQ_LieferantenID
      AS TINYINT
      START WITH 1      --optional, Default startet mit 1
      INCREMENT BY 1    --muss ganzzahlig sein
      MINVALUE 1        --optional
      MAXVALUE 100      --optional
      CYCLE             --optional, Default ist NO CYCLE
                        --(= bei Erreichen von MAXVALUE oder Ende des Da-
                        tentyps wird eine Ausnahme ausgelöst)
      CACHE             --optional, Default ist CACHE
```

Vergleich

Objekt	Zweck	Beispiel
Index	Performance verbessern	Suche, Sortierung
Stored Procedure	Logik speichern und ausführen	Kundenliste
Trigger	Automatisches Reagieren	Änderung loggen
Sequence	IDs automatisch erzeugen	Kunden-ID