**Data Definition Language (DDL)**

Die Data Definition Language (DDL) ist ein zentraler Bestandteil von SQL und wird verwendet, um die Struktur und das Schema von Datenbanken zu definieren oder zu bearbeiten. Im Folgenden werden die wichtigsten Aspekte, Codebeispiele und deren detaillierte Erklärungen behandelt.

**Datentypen**

**Kategorien von Datentypen:**

1. **Exakte Nummern:** bigint, numeric, bit, smallint, decimal, smallmoney, int, tinyint, money
   * **bigint:** Für sehr große ganze Zahlen.
   * **decimal:** Für exakte Dezimalzahlen, z. B. Preise.
   * **bit:** Boolean-ähnliche Werte (0 oder 1).
2. **Numerische Näherungswerte:** float, real
   * **float:** Für wissenschaftliche Berechnungen oder Werte mit hoher Genauigkeit.
3. **Datum und Uhrzeit:** date, datetimeoffset, datetime2, smalldatetime, datetime, time
   * **datetime:** Speichert Datum und Uhrzeit.
   * **time:** Speichert nur die Uhrzeit.
4. **Character Strings:** char, varchar, text
   * **char:** Für feste Länge von Texten.
   * **varchar:** Variable Länge, ideal für Einträge unterschiedlicher Textlänge.
5. **Unicode Character Strings:** nchar, nvarchar, ntext
   * **nvarchar:** Für mehrsprachige Zeichen.
6. **Binäre Strings:** binary, varbinary, image
   * **image:** Für große Binärdaten wie Bilder.
7. **Andere Datentypen:** cursor, rowversion, hierarchyid, uniqueidentifier, sql\_variant, xml, Spatial Geometry Types, Spatial Geography Types, table
   * **uniqueidentifier:** Für eindeutige IDs, z. B. GUIDs.

**Gruppen von Datentypen:**

* **Large Value Daten Typen:** varchar(max), nvarchar(max)
* **Large Object Daten Typen:** text, ntext, image, varbinary(max), xml

**Datenbankschemas**

Ein **Schema** dient der **organisatorischen Trennung** von Objekten innerhalb einer Datenbank und bietet eine Verwaltung von Berechtigungen.

**Beispiel:**

CREATE SCHEMA intern;

* Erstellt ein Schema namens intern.
* Schemas enthalten Tabellen, Views und Prozeduren.
* Standard-Schema in SQL Server ist dbo.

**CREATE**

**1. Erstellung von Tabellen**

* Tabellen werden mit dem Befehl CREATE TABLE erstellt.
* Sie enthalten Spalten mit spezifischen Datentypen und Constraints.

**Beispiel:**

***CREATE TABLE verkauf.bestellpositionen (***

***id INT PRIMARY KEY IDENTITY(1,1),***

***bestellid INT NOT NULL,***

***produktid INT NOT NULL,***

***menge SMALLINT NOT NULL***

***);***

**Erklärung:**

1. **id INT PRIMARY KEY IDENTITY(1,1)**:
   * id: Primärschlüssel der Tabelle.
   * IDENTITY(1,1): Automatische Nummerierung beginnend bei 1.
2. **NOT NULL**: Verhindert, dass die Spalten bestellid, produktid und menge leer bleiben.
3. **Schema-Verwendung:** verkauf organisiert die Tabelle unter einem spezifischen Bereich.

**✂️ ALTER TABLE**

**1. Spalten hinzufügen**

Mit ALTER TABLE können Spalten zu bestehenden Tabellen hinzugefügt werden.

**Beispiel:**

***ALTER TABLE verkauf.bestellpositionen***

***ADD rabatt TINYINT;***

**Erklärung:**

* Fügt die Spalte rabatt hinzu.
* TINYINT: Für kleine Ganzzahlen (0-255).

**2. Spaltenänderungen**

Spalten können hinsichtlich Datentyp oder Eigenschaften angepasst werden.

**Beispiel:**

***ALTER TABLE intern.mitarbeiter***

***ALTER COLUMN vorname NVARCHAR(50) NULL;***

**Erklärung:**

* Ändert die Spalte vorname, sodass sie Unicode-Zeichen unterstützt und optional ist.

**3. Spalten entfernen**

Nicht mehr benötigte Spalten können mit DROP COLUMN entfernt werden.

**Beispiel:**

***ALTER TABLE intern.mitarbeiter***

***DROP COLUMN kader;***

**Erklärung:**

* Entfernt die Spalte kader aus der Tabelle.

**🗑️ DROP TABLE**

Mit DROP TABLE werden Tabellen vollständig entfernt.

**Beispiel:**

***DROP TABLE verkauf.bestellungen;***

**Erklärung:**

* Löscht die Tabelle verkauf.bestellungen inklusive aller Daten und Struktur.
* Die Aktion ist irreversibel.

**❌ TRUNCATE TABLE**

Mit TRUNCATE TABLE werden alle Daten aus einer Tabelle entfernt, die Struktur bleibt erhalten.

**Beispiel:**

***TRUNCATE TABLE verkauf.bestellungen;***

**Erklärung:**

* Löscht den gesamten Inhalt der Tabelle.
* Setzt Auto-Increment-Werte zurück.
* Keine Filterung möglich (anders als bei DELETE).

**🔑 Schlüsselfelder**

**1. Primärschlüssel**

Primärschlüssel identifizieren Datensätze eindeutig und können direkt oder nachträglich definiert werden.

**Beispiel:**

***ALTER TABLE verkauf.kunden ADD PRIMARY KEY (kundenid);***

**Erklärung:**

* Fügt der Tabelle verkauf.kunden einen Primärschlüssel hinzu.

**2. Fremdschlüssel**

Fremdschlüssel stellen Beziehungen zwischen Tabellen her.

**Beispiel:**

***ALTER TABLE verkauf.bestellpositionen***

***ADD FOREIGN KEY (bestellid) REFERENCES verkauf.bestellungen(bestellid);***

**Erklärung:**

* Verknüpft die Spalte bestellid mit der Tabelle verkauf.bestellungen.

**3. Constraints**

Zusätzliche Regeln können mit Constraints definiert werden:

* **DEFAULT:** Standardwert für eine Spalte:

***ALTER TABLE intern.lohnkonto***

***ADD DEFAULT '31.12.9999' FOR gueltigbis;***

* **CHECK:** Einschränkung von Werten:

***ALTER TABLE verkauf.produkte***

***ADD CHECK (energielabel IN ('A', 'B', 'C', 'D', 'E'));***

**Zusammenfassung**

Die Data Definition Language bietet mächtige Werkzeuge zur Verwaltung der Datenbankstruktur. Ob durch die Erstellung, Änderung oder Löschung von Objekten – DDL ist essenziell für ein effizientes Datenbankdesign. Jede Operation sollte mit Vorsicht und gutem Verständnis der Auswirkungen durchgeführt werden.

**Data Manipulation Language (DML)**

Die **Data Manipulation Language (DML)** umfasst die grundlegenden SQL-Operationen zum Hinzufügen, Ändern, Löschen und Zusammenführen von Daten. Sie wird verwendet, um die Daten in relationalen Datenbanken zu manipulieren.

**Grundlagen**

* **DML-Statements:**
  + INSERT: Daten einfügen
  + UPDATE: Bestehende Daten ändern
  + DELETE: Daten löschen
  + MERGE: Daten synchronisieren (Kombination aus INSERT, UPDATE und DELETE)
* **Transaktionen:**
  + DML-Befehle können einzeln oder als Teil einer Transaktion verwendet werden, um das **ACID-Prinzip** einzuhalten.
* **WHERE-Klausel:**
  + Ermöglicht die gezielte Auswahl von Datensätzen, die durch DML-Befehle betroffen sind.

**📥 INSERT: Daten einfügen**

Mit INSERT können neue Datensätze in eine Tabelle eingefügt werden.

**Syntax:**

***INSERT [INTO] <[schema.]tabelle> (<spalte1>, <spalte2>)***

***VALUES (wert1, wert2);***

**Beispiele:**

1. **Explizite Spaltenangabe:**
2. INSERT INTO verkauf.kunden (vorname, nachname)
3. VALUES ('Hans', 'Meier');
   * Fügt einen Datensatz mit Vorname Hans und Nachname Meier in die Tabelle verkauf.kunden ein.
4. **Ohne Spaltenangabe:**
5. INSERT INTO verkauf.kunden
6. VALUES ('Peter', 'Müller');
   * Fügt Daten in alle Spalten der Tabelle ein. Die Reihenfolge der Werte muss der Tabellendefinition entsprechen.
7. **Mehrere Datensätze auf einmal einfügen:**
8. INSERT INTO verkauf.kunden (vorname, nachname)
9. VALUES ('Bruno', 'Moser'), ('Heinz', 'Manser');
   * Fügt mehrere Datensätze in einem Statement ein.
10. **Daten aus einer anderen Tabelle einfügen:**
11. INSERT INTO verkauf.kunden
12. SELECT vorname, nachname FROM firma.mitarbeiter;
    * Kopiert Daten aus der Tabelle firma.mitarbeiter in die Tabelle verkauf.kunden.

**🆕 UPDATE: Bestehende Daten ändern**

Mit UPDATE können Werte in einer Tabelle aktualisiert werden.

**Syntax:**

UPDATE <[schema.]tabelle>

SET <spalte1> = wert1, <spalte2> = wert2

[WHERE <kondition>];

**Beispiele:**

1. **Einen Datensatz aktualisieren:**
2. UPDATE verkauf.kunden
3. SET vorname = 'Pete', nachname = 'Muller'
4. WHERE kundenid = 2;
   * Ändert den Vornamen und Nachnamen des Kunden mit der kundenid 2.
5. **Basierend auf einer Bedingung:**
6. UPDATE verkauf.kunden
7. SET nachname = 'Meyer'
8. WHERE nachname = 'Meier';
   * Ändert alle Datensätze, bei denen der Nachname Meier ist, in Meyer.

**Wichtige Hinweise:**

* **Ohne WHERE:**
  + Wird die WHERE-Klausel weggelassen, werden **alle Datensätze** der Tabelle geändert.

**✂️ DELETE: Daten löschen**

Mit DELETE können Datensätze aus einer Tabelle entfernt werden.

**Syntax:**

DELETE FROM <[schema.]tabelle>

[WHERE <kondition>];

**Beispiele:**

1. **Einen spezifischen Datensatz löschen:**
2. DELETE FROM verkauf.produkte
3. WHERE produktid = 3;
   * Löscht den Datensatz mit der produktid 3.
4. **Ohne WHERE:**
5. DELETE FROM verkauf.produkte;
   * Löscht **alle Datensätze** aus der Tabelle, ohne die Struktur zu entfernen.

**Wichtige Hinweise:**

* **Unterschied zu TRUNCATE:**
  + DELETE protokolliert die Löschung und lässt die Auto-Increment-Werte unverändert.
  + TRUNCATE löscht schneller und setzt Auto-Increment zurück, kann aber keine gezielten Löschungen vornehmen.

**⛲ Cascading: Verhalten bei Fremdschlüsselverletzungen**

Cascading regelt, wie referenzierte Datensätze in Child-Tabellen automatisch angepasst oder gelöscht werden:

1. **ON UPDATE CASCADE:**
   * Änderungen am Primärschlüssel der Parent-Tabelle werden auf die Child-Tabelle übertragen.
2. ALTER TABLE verkauf.bestellungen
3. ADD CONSTRAINT fk\_bestellungen\_kunden FOREIGN KEY (kundenid)
4. REFERENCES verkauf.kunden(kundenid) ON UPDATE CASCADE;
5. **ON DELETE CASCADE:**
   * Löscht automatisch alle referenzierten Datensätze in der Child-Tabelle, wenn ein Primärschlüssel aus der Parent-Tabelle gelöscht wird.
6. ALTER TABLE verkauf.bestellungen
7. ADD CONSTRAINT fk\_bestellungen\_kunden FOREIGN KEY (kundenid)
8. REFERENCES verkauf.kunden(kundenid) ON DELETE CASCADE;
9. **Kombination:**
   * ON UPDATE CASCADE und ON DELETE CASCADE können kombiniert werden.

**🖇️ MERGE: Daten synchronisieren**

Mit MERGE können Datensätze zwischen zwei Tabellen synchronisiert werden. Es kombiniert die Funktionalitäten von INSERT, UPDATE und DELETE in einem einzigen Statement.

**Syntax:**

MERGE <ziel-tabelle> [AS TARGET]

USING <quell-tabelle> [AS SOURCE]

ON <kondition>

[WHEN MATCHED THEN <aktion>]

[WHEN NOT MATCHED BY TARGET THEN <aktion>]

[WHEN NOT MATCHED BY SOURCE THEN <aktion>];

**Beispiel:**

MERGE produkte AS TARGET

USING updatedprodukte AS SOURCE

ON (TARGET.produktid = SOURCE.produktid)

WHEN MATCHED AND (TARGET.produktname <> SOURCE.produktname OR TARGET.preis <> SOURCE.preis)

THEN UPDATE SET TARGET.produktname = SOURCE.produktname, TARGET.preis = SOURCE.preis

WHEN NOT MATCHED BY TARGET

THEN INSERT (produktid, produktname, preis) VALUES (SOURCE.produktid, SOURCE.produktname, SOURCE.preis)

WHEN NOT MATCHED BY SOURCE

THEN DELETE;

**Erklärung:**

1. **MATCHED:** Aktualisiert Werte in der Ziel-Tabelle, wenn sie sich von der Quelltabelle unterscheiden.
2. **NOT MATCHED BY TARGET:** Fügt Datensätze aus der Quelltabelle in die Ziel-Tabelle ein, die dort fehlen.
3. **NOT MATCHED BY SOURCE:** Löscht Datensätze aus der Ziel-Tabelle, die nicht mehr in der Quelltabelle vorhanden sind.

**Wide World Importers Database (WWI)**

Die WWI-Datenbank ist ein Schulungsmodell von Microsoft. Sie enthält:

* **Transaktionsdaten:** Ein- und Verkäufe.
* **Sensordaten:** Fahrzeuge und Kühlräume.
* **Schemas:**
  + **Application:** Stammdaten wie Benutzer oder Bezahlmethoden.
  + **Purchasing:** Einkaufsdaten.
  + **Sales:** Verkaufs- und Kundendaten.
  + **Warehouse:** Lagerbestände und Sensordaten.

**Datenbank wiederherstellen:**

* Lade das Backup von [GitHub](https://github.com/Microsoft/sql-server-samples/releases/tag/wide-world-importers-v1.0) herunter.
* Verwende die Datei **WideWorldImporters-Full.bak**.
* Importiere die Datenbank mit SQL Server Management Studio (SSMS).

**Zusammenfassung**

Die DML-Befehle INSERT, UPDATE, DELETE, MERGE und Cascading-Mechanismen bieten leistungsstarke Werkzeuge zur Manipulation von Daten. Die WWI-Datenbank dient dabei als Schulungsgrundlage, um diese Konzepte praktisch zu erproben.

**Zusammenfassung: SQL-Konzepte und Funktionen, DQL**

**1. Mengenoperationen**

Mengenoperationen kombinieren oder vergleichen Resultsets aus mehreren Abfragen.

**1.1 UNION**

* **Funktion:** Kombiniert die Ergebnisse von zwei Abfragen und entfernt Duplikate.
* **Verwendung:**

sql

Code kopieren

SELECT spalte1, spalte2 FROM tabelle1

UNION

SELECT spalte1, spalte2 FROM tabelle2;

* **Varianten:**
  + UNION ALL: Beibehaltung von Duplikaten.

**1.2 INTERSECT**

* **Funktion:** Liefert die Schnittmenge zweier Abfragen.
* **Verwendung:**

sql

Code kopieren

SELECT spalte1, spalte2 FROM tabelle1

INTERSECT

SELECT spalte1, spalte2 FROM tabelle2;

**1.3 EXCEPT**

* **Funktion:** Gibt Datensätze aus der ersten Abfrage zurück, die nicht in der zweiten enthalten sind.
* **Verwendung:**

sql

Code kopieren

SELECT spalte1, spalte2 FROM tabelle1

EXCEPT

SELECT spalte1, spalte2 FROM tabelle2;

**2. Unterabfragen**

Unterabfragen nutzen das Ergebnis einer Abfrage als Eingabe für eine andere Abfrage.

**2.1 WHERE-Klausel**

* **Funktion:** Filtert basierend auf einer Unterabfrage.
* **Beispiel:**

sql

Code kopieren

SELECT StockItemName, UnitPrice

FROM Warehouse.StockItems

WHERE UnitPrice > (SELECT AVG(UnitPrice) FROM Warehouse.StockItems);

**2.2 FROM-Klausel (Inline-View)**

* **Funktion:** Eine Unterabfrage wird als temporäre Tabelle verwendet.
* **Beispiel:**

sql

Code kopieren

SELECT c.CustomerID, c.CustomerName, o.OrderDate, o.AnzahlBestellungen

FROM Sales.Customers c

JOIN (

SELECT CustomerID, OrderDate, COUNT(\*) AS AnzahlBestellungen

FROM Sales.Orders

GROUP BY CustomerID, OrderDate

HAVING COUNT(\*) > 4

) AS o ON c.CustomerID = o.CustomerID;

**2.3 IN-Operator**

* **Funktion:** Prüft, ob ein Wert in einer Liste vorkommt.
* **Beispiel:**

sql

Code kopieren

SELECT \*

FROM Warehouse.StockItems

WHERE StockItemID IN (

SELECT StockItemID

FROM Sales.OrderLines

GROUP BY StockItemID

HAVING SUM(Quantity) > 6000

);

**3. Gruppierung und Aggregatfunktionen**

**3.1 Aggregatfunktionen**

* **Funktionen:**
  + SUM(): Berechnet die Summe.
  + MIN(): Kleinster Wert.
  + MAX(): Größter Wert.
  + COUNT(): Anzahl der Datensätze.
  + AVG(): Durchschnittswert.
* **Beispiel:**

sql

Code kopieren

SELECT AVG(UnitPrice) AS Durchschnitt, MIN(UnitPrice) AS Minimum, MAX(UnitPrice) AS Maximum

FROM Warehouse.StockItems;

**3.2 GROUP BY**

* **Funktion:** Gruppiert Datensätze basierend auf einer oder mehreren Spalten.
* **Beispiel:**

sql

Code kopieren

SELECT stadt, COUNT(id) AS AnzahlPersonen

FROM personen

GROUP BY stadt;

**3.3 HAVING**

* **Funktion:** Filtert Ergebnisse nach einer Gruppierung.
* **Beispiel:**

sql

Code kopieren

SELECT stadt, COUNT(\*) AS AnzahlPersonen

FROM personen

GROUP BY stadt

HAVING COUNT(\*) > 100;

**4. JOINs**

JOINs verbinden Tabellen basierend auf einer Beziehung zwischen Spalten.

**4.1 INNER JOIN**

* **Funktion:** Gibt nur die Datensätze zurück, die in beiden Tabellen übereinstimmen.
* **Beispiel:**

sql

Code kopieren

SELECT bestellung.bestellid, kunde.vorname, kunde.nachname, bestellung.preis

FROM bestellung

INNER JOIN kunde ON bestellung.kundenId = kunde.KundenId;

**4.2 LEFT JOIN**

* **Funktion:** Gibt alle Datensätze aus der linken Tabelle zurück, ergänzt mit passenden Datensätzen aus der rechten Tabelle. Fehlende Werte werden als NULL angezeigt.
* **Beispiel:**

sql

Code kopieren

SELECT bestellung.bestellid, kunde.vorname, kunde.nachname, bestellung.preis

FROM bestellung

LEFT JOIN kunde ON bestellung.kundenId = kunde.KundenId;

**4.3 RIGHT JOIN**

* **Funktion:** Gibt alle Datensätze aus der rechten Tabelle zurück, ergänzt mit passenden Datensätzen aus der linken Tabelle.
* **Beispiel:**

sql

Code kopieren

SELECT bestellung.bestellid, kunde.vorname, kunde.nachname, bestellung.preis

FROM bestellung

RIGHT JOIN kunde ON bestellung.kundenId = kunde.KundenId;

**4.4 FULL OUTER JOIN**

* **Funktion:** Gibt alle Datensätze aus beiden Tabellen zurück, ergänzt mit NULL bei fehlenden Werten.
* **Beispiel:**

sql

Code kopieren

SELECT bestellung.bestellid, kunde.vorname, kunde.nachname, bestellung.preis

FROM bestellung

FULL JOIN kunde ON bestellung.kundenId = kunde.KundenId;

**4.5 CROSS JOIN**

* **Funktion:** Bildet das kartesische Produkt zweier Tabellen.
* **Beispiel:**

sql

Code kopieren

SELECT \*

FROM kunde CROSS JOIN bestellung;

**4.6 SELF JOIN**

* **Funktion:** Verknüpft eine Tabelle mit sich selbst, häufig für Hierarchien.
* **Beispiel:**

sql

Code kopieren

SELECT MA1.Id, MA1.Vorname, MA1.Nachname, MA2.Vorname AS Vorgesetzter

FROM Mitarbeiter MA1

JOIN Mitarbeiter MA2 ON MA1.VorgesetzerId = MA2.Id;

**5. Fazit**

* **Mengenoperationen:** Kombinieren und vergleichen Datensätze aus mehreren Abfragen (UNION, INTERSECT, EXCEPT).
* **Unterabfragen:** Nutzen das Ergebnis einer Abfrage als Eingabe für eine andere (WHERE, FROM, IN).
* **Gruppierungen:** Fassen Daten zusammen und erlauben Aggregatberechnungen (SUM, COUNT, GROUP BY, HAVING).
* **JOINs:** Verbinden Tabellen basierend auf Beziehungen (INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL OUTER JOIN, CROSS JOIN, SELF JOIN).

**Zusammenfassung: Datensicherheit, Rollen und Views**

**1. Views (Sichten)**

**1.1 Was sind Views?**

* Views sind virtuelle Tabellen, die aus einer oder mehreren Tabellen erstellt werden.
* Sie speichern keine Daten selbst, sondern die zugrunde liegende Abfrage.
* Dienen der Vereinfachung komplexer Abfragen und der Einschränkung des Zugriffs auf sensible Daten.

**1.2 Erstellung von Views**

sql

Code kopieren

CREATE VIEW view\_name AS

SELECT spalte1, spalte2

FROM tabelle

WHERE bedingung;

**1.3 Beispiel: Kundendaten ohne sensible Informationen**

sql

Code kopieren

CREATE VIEW KundenSicht AS

SELECT CustomerName, City, Country

FROM Sales.Customers

WHERE Country = 'Germany';

* **Erklärung:** Diese View zeigt nur Kunden aus Deutschland und schließt sensible Daten wie Telefonnummern aus.

**1.4 Vorteile von Views**

* Vereinfachte Abfragen für Benutzer.
* Einschränkung des Zugriffs auf sensible Spalten.
* Datenkonsistenz bei komplexen Joins.

**2. Datensicherheits- und Rollenkonzepte**

**2.1 Warum Datensicherheit?**

* Schutz vor unbefugtem Zugriff auf Daten.
* Einhaltung gesetzlicher Vorschriften (z. B. DSGVO).
* Minimierung von Risiken bei Datenverlust oder Manipulation.

**2.2 Inhalt eines Datensicherheitskonzepts**

1. **Rollen und Berechtigungen:**
   * Definition von Benutzerrollen (z. B. Admin, Benutzer, Gast).
   * Zuweisung von Rechten (Lesen, Schreiben, Löschen).
2. **Sicherheitsrichtlinien:**
   * Festlegung, wer auf welche Teile der Datenbank zugreifen darf.
3. **Protokollierung und Audits:**
   * Überwachung von Änderungen und Zugriffen.
4. **Backup-Strategien:**
   * Regelmäßige Sicherung der Datenbank.

**2.3 Der Begriff "Rolle"**

* Rollen sind Gruppen von Berechtigungen, die bestimmten Benutzern zugewiesen werden.
* Rollen abstrahieren die Verwaltung von Rechten und erleichtern die Kontrolle über Zugriffe.

**2.4 Erstellung von Rollen**

sql

Code kopieren

CREATE ROLE RolleName;

GRANT SELECT, INSERT ON Tabelle TO RolleName;

* **Beispiel:**

sql

Code kopieren

CREATE ROLE Leserechte;

GRANT SELECT ON Sales.Customers TO Leserechte;

**3. Datenkontrollsprache (DCL)**

**3.1 Befehle zur Benutzer- und Rollenverwaltung**

* **CREATE USER:** Erstellen eines neuen Benutzers.

sql

Code kopieren

CREATE USER BenutzerName FOR LOGIN LoginName;

* **CREATE ROLE:** Erstellen einer neuen Rolle.

sql

Code kopieren

CREATE ROLE RolleName;

**3.2 Zuweisung von Berechtigungen**

* **GRANT:** Zuweisung von Berechtigungen.

sql

Code kopieren

GRANT SELECT ON Tabelle TO RolleName;

* **DENY:** Entzug von Berechtigungen.

sql

Code kopieren

DENY INSERT ON Tabelle TO BenutzerName;

* **REVOKE:** Aufhebung von Berechtigungen.

sql

Code kopieren

REVOKE SELECT ON Tabelle FROM RolleName;

**4. Beispiele für Sicherheitskonzepte**

**4.1 Zugriffsschutz mit Rollen**

* Admins dürfen alles:

sql

Code kopieren

GRANT ALL ON Datenbank TO AdminRolle;

* Benutzer dürfen nur lesen:

sql

Code kopieren

GRANT SELECT ON Tabelle TO BenutzerRolle;

**4.2 Dokumentation der Zugriffsrechte**

* Erstellen eines Berechtigungsplans:

| **Rolle** | **Rechte** | **Bereich** |
| --- | --- | --- |
| Admin | Lesen/Schreiben | Alle Tabellen |
| Benutzer | Lesen | Kundendaten |
| Gast | Keine | - |

**5. Zusammenfassung der DCL-Befehle**

| **Befehl** | **Beschreibung** |
| --- | --- |
| **CREATE ROLE** | Erstellt eine Rolle. |
| **CREATE USER** | Erstellt einen Benutzer. |
| **GRANT** | Erteilt Berechtigungen. |
| **DENY** | Verweigert bestimmte Rechte. |
| **REVOKE** | Hebt Berechtigungen auf. |

**6. Vorteile der Nutzung von Views und Rollen**

* **Views:**
  + Schutz sensibler Daten.
  + Vereinfachung komplexer Abfragen.
* **Rollen:**
  + Einheitliche Verwaltung von Berechtigungen.
  + Verbesserung der Sicherheit und Nachvollziehbarkeit.

**Zusammenfassung: Datensicherheit und Datenschutz**

**1. Datenschutzgesetz (DSG)**

**1.1 Bedeutung der Datensicherheit**

* Das Datenschutzgesetz (DSG) verpflichtet Organisationen, Personendaten durch **technische und organisatorische Maßnahmen** zu schützen.
* Ziel: Schutz der Daten vor unbefugtem Zugriff, Verlust, Manipulation oder Missbrauch.

**1.2 Anforderungen an die Datensicherheit**

* Datenintegrität: Verhinderung von Datenmanipulation.
* Vertraulichkeit: Sicherstellung, dass nur berechtigte Personen Zugriff auf Daten haben.
* Verfügbarkeit: Daten müssen für berechtigte Nutzer stets zugänglich sein.

**1.3 Maßnahmen im Bereich Datensicherheit**

* **Technische Maßnahmen:**
  + Verschlüsselung sensibler Daten.
  + Einsatz von Firewalls und Antivirensoftware.
  + Regelmäßige Backups.
* **Organisatorische Maßnahmen:**
  + Definierte Zugriffsrichtlinien.
  + Schulung von Mitarbeitern.
  + Erstellung eines Sicherheitskonzepts.

**2. Benutzerverwaltung und Zugriffsrechte**

**2.1 Verwaltung von Benutzern und Rollen**

* Ein sicheres Datenbanksystem erfordert klare Regeln für Benutzer und Benutzergruppen (Rollen).
* **Rollen:** Abstraktion von Benutzergruppen, um Berechtigungen effizient zu verwalten.
  + Beispiel: Rolle "Admin" mit vollständigem Zugriff, Rolle "Gast" nur mit Leserechten.

**2.2 Vergabe von Zugriffsrechten**

* Rechte können auf **Tabellen- oder Spaltenebene** zugewiesen werden.
* Mögliche Rechte:
  + **SELECT:** Lesen von Daten.
  + **INSERT:** Hinzufügen neuer Datensätze.
  + **UPDATE:** Bearbeiten bestehender Daten.
  + **DELETE:** Löschen von Datensätzen.

**2.3 Beispiele zur Rechtevergabe**

sql

Code kopieren

-- Leserechte für Benutzer vergeben

GRANT SELECT ON Sales.Customers TO BenutzerRolle;

-- Schreibrechte verweigern

DENY INSERT ON Sales.Orders TO GastRolle;

-- Rechte entfernen

REVOKE SELECT ON Warehouse.StockItems FROM BenutzerRolle;

**2.4 Vorteile der Benutzerverwaltung**

* Einheitliche Verwaltung durch Rollen vereinfacht die Rechtevergabe.
* Reduzierung des Risikos von Datenmissbrauch durch gezielte Vergabe von Berechtigungen.
* Transparenz und Nachvollziehbarkeit durch klare Richtlinien.

**3. Schlüsselprinzipien der Datensicherheit**

1. **Minimalprinzip:**
   * Benutzer erhalten nur die Rechte, die sie für ihre Aufgaben benötigen.
2. **Protokollierung:**
   * Alle Zugriffe und Änderungen werden protokolliert, um Missbrauch zu erkennen.
3. **Schulung:**
   * Regelmäßige Sensibilisierung der Mitarbeiter zum Thema Datensicherheit.

**4. Fazit**

* Die Einhaltung des Datenschutzgesetzes ist nicht nur gesetzlich vorgeschrieben, sondern auch essenziell für das Vertrauen von Kunden und Partnern.
* Eine sorgfältige Benutzerverwaltung mit klar definierten Zugriffsrechten schützt sensible Daten und minimiert Risiken.
* Rollen und Rechte vereinfachen die Verwaltung und gewährleisten die Sicherheit in einem Datenbanksystem.

**Zusammenfassung: Datensicherheit und Authentifizierung im SQL Server**

**1. Authentifizierung im SQL Server**

Der Zugriff auf Datenbanken im SQL Server erfolgt in zwei Schritten:

1. **Server-Login:** Anmeldung auf dem SQL Server.
2. **Datenbank-User:** Zugriff auf spezifische Datenbanken.

**1.1 Rollen des Logins und Users**

* **Login:** Wird global auf dem SQL Server definiert und dient der Authentifizierung.
* **User:** Wird in der spezifischen Datenbank erstellt und mit einem Login verknüpft.
* **Namenskonvention:** Es wird empfohlen, dem User denselben Namen wie dem Login zu geben, um die Verwaltung zu vereinfachen.

**2. Authentifizierungsmodi**

SQL Server unterstützt zwei Authentifizierungsarten:

**2.1 Windows-Authentifizierung**

* Verwendet Windows-Domänenkonten als Logins.
* Vorteil: Der Endanwender benötigt keine separaten SQL Server-Zugangsdaten.

**2.2 SQL Server-Authentifizierung**

* Erfordert die Erstellung eines SQL Server-Kontos mit Benutzernamen und Passwort.
* Wird für Anwender außerhalb der Windows-Domäne genutzt.

**3. Erstellung und Verwaltung von Logins und Benutzern**

**3.1 Login erstellen**

sql

Code kopieren

-- Server-Login erstellen

USE master;

CREATE LOGIN anmelde\_name

WITH PASSWORD = 'passwort', CHECK\_POLICY = OFF;

* **Parameter:**
  + CHECK\_POLICY = OFF: Deaktiviert Passwort-Richtlinien (z. B. Komplexität).
  + CHECK\_EXPIRATION = OFF: Passwort läuft nicht ab.
  + DEFAULT\_DATABASE = database: Setzt eine Standarddatenbank für das Login.

**3.2 Benutzer erstellen und Login zuordnen**

sql

Code kopieren

-- User in einer spezifischen Datenbank erstellen

USE myDatabase;

CREATE USER user\_name FOR LOGIN anmelde\_name;

* **Wichtig:**
  + Das Login muss zuerst in der Datenbank master erstellt werden.
  + Ein Benutzer wird automatisch der Datenbankrolle public zugewiesen.

**3.3 Benutzer und Login löschen**

sql

Code kopieren

-- Server-Login löschen

DROP LOGIN anmelde\_name;

-- Datenbank-User löschen

DROP USER user\_name;

**4. Vorteile des Authentifizierungskonzepts**

* **Sicherheit:** Windows-Authentifizierung integriert nahtlos in bestehende Domänensicherheitsrichtlinien.
* **Flexibilität:** SQL Server-Authentifizierung erlaubt die Anbindung von Anwendern außerhalb der Domäne.
* **Einfache Verwaltung:** Durch die Verknüpfung von Logins mit Datenbank-Usern wird die Zugriffssteuerung übersichtlicher.

**5. Fazit**

* Der zweistufige Authentifizierungsprozess des SQL Servers gewährleistet eine klare Trennung zwischen Serverzugriff und Datenbankberechtigungen.
* Die Wahl zwischen Windows- und SQL Server-Authentifizierung bietet Flexibilität für verschiedene Einsatzszenarien.
* Best Practices wie die einheitliche Namensvergabe und das Einhalten von Sicherheitsrichtlinien (z. B. Passwortkomplexität) erhöhen die Benutzerfreundlichkeit und Datensicherheit.

**Zusammenfassung: Autorisierung und Berechtigungen im SQL Server**

**1. Rollen und Berechtigungen**

**1.1 Bedeutung von Rollen**

* Rollen sind Sammlungen logischer Berechtigungen, die Benutzern oder Logins zugewiesen werden.
* Dienen der vereinfachten und übersichtlichen Berechtigungsverwaltung.
* **Zwei Hauptrollenebenen:**
  + **Serverrollen:** Berechtigungen auf Serverebene (z. B. sysadmin, serveradmin).
  + **Datenbankrollen:** Berechtigungen auf Datenbankebene (z. B. db\_datareader, db\_datawriter).

**1.2 Standardrollen**

1. **Serverrollen:**
   * sysadmin: Voller Zugriff auf den SQL Server.
   * securityadmin: Verwaltung von Logins und Berechtigungen.
2. **Datenbankrollen:**
   * db\_datareader: Lesen aller Tabellen.
   * db\_datawriter: Schreiben in alle Tabellen.
   * **public:** Automatische Standardrolle für alle Benutzer; hat minimale Berechtigungen.

**2. Rollen verwalten**

**2.1 Zuweisung und Entzug von Rollen**

sql

Code kopieren

-- Login einer Serverrolle zuweisen

USE master;

ALTER SERVER ROLE rollen\_name ADD MEMBER anmelde\_name;

-- Login aus einer Serverrolle entfernen

ALTER SERVER ROLE rollen\_name DROP MEMBER anmelde\_name;

-- User einer Datenbankrolle zuweisen

USE mydatabase;

ALTER ROLE rollen\_name ADD MEMBER user\_name;

-- User aus einer Datenbankrolle entfernen

ALTER ROLE rollen\_name DROP MEMBER user\_name;

**2.2 Vorteile von Rollen**

* **Effizienz:** Rollen ermöglichen die zentrale Verwaltung von Berechtigungen.
* **Flexibilität:** Benutzer und Logins können spezifischen Rollen zugewiesen werden, ohne individuelle Berechtigungen festzulegen.
* **Sicherheit:** Rollen verhindern, dass Benutzer unnötige Rechte erhalten.

**3. Berechtigungen (Data Control Language - DCL)**

**3.1 Arten von Berechtigungen**

1. **Anweisungsberechtigungen:**
   * Erlauben die Erstellung und Verwaltung von Datenbankobjekten.
   * Beispiele: CREATE TABLE, BACKUP DATABASE.
2. **Objektberechtigungen:**
   * Erlauben den Zugriff und die Manipulation von Datenbankobjekten.
   * Beispiele: SELECT, INSERT, UPDATE, DELETE.

**3.2 Berechtigungen erteilen**

sql

Code kopieren

-- Objektberechtigung erteilen

GRANT SELECT, INSERT, UPDATE, DELETE ON table\_name TO database\_user;

-- Anweisungsberechtigung erteilen

GRANT CREATE TABLE TO database\_user;

-- Berechtigung für alle Objekte eines Schemas

GRANT SELECT, UPDATE ON SCHEMA::schema\_name TO database\_user;

-- Berechtigung mit Weitergaberecht erteilen

GRANT INSERT ON table\_name TO database\_user WITH GRANT OPTION;

**3.3 Berechtigungen entziehen**

sql

Code kopieren

-- Objektberechtigung entziehen

REVOKE SELECT, INSERT ON table\_name FROM database\_user;

-- Anweisungsberechtigung entziehen

REVOKE CREATE TABLE FROM database\_user;

-- Berechtigung auf alle Objekte eines Schemas entziehen

REVOKE SELECT, UPDATE ON SCHEMA::schema\_name FROM database\_user;

-- Berechtigungen mit Weitergaberecht entziehen (inkl. Abhängigkeiten)

REVOKE GRANT OPTION FOR INSERT ON table\_name FROM database\_user CASCADE;

**3.4 Berechtigungen verweigern**

* Mit DENY können Berechtigungen explizit verweigert werden, auch wenn diese indirekt über Rollenmitgliedschaften gewährt wurden.

sql

Code kopieren

-- Objektberechtigung verweigern

DENY SELECT, INSERT ON table\_name TO database\_user;

-- Anweisungsberechtigung verweigern

DENY CREATE TABLE TO database\_user;

-- Berechtigung auf alle Objekte eines Schemas verweigern

DENY SELECT, UPDATE ON SCHEMA::schema\_name TO database\_user;

**4. Best Practices für Berechtigungen**

1. **Minimalprinzip:**
   * Benutzer erhalten nur die Rechte, die sie für ihre Aufgaben benötigen.
2. **Verwendung von Rollen:**
   * Zuweisung von Rechten auf Rollenebene, um die Verwaltung zu vereinfachen.
3. **Überwachung und Protokollierung:**
   * Zugriffe und Änderungen protokollieren, um Missbrauch zu erkennen.
4. **Regelmäßige Überprüfung:**
   * Überprüfen und Aktualisieren von Rollen und Berechtigungen.

**5. Fazit**

* Rollen und Berechtigungen im SQL Server bieten eine flexible und sichere Möglichkeit zur Autorisierung.
* Durch die Zuweisung von Berechtigungen an Rollen wird die Verwaltung effizienter und transparenter.
* Die DCL-Befehle GRANT, REVOKE und DENY ermöglichen eine feingranulare Kontrolle über den Datenzugriff und die Sicherheit.

**Zusammenfassung: Row Level Security (RLS) im SQL Server**

**1. Was ist Row Level Security (RLS)?**

* RLS ermöglicht die Steuerung des Tabellenzugriffs auf **Zeilenebene** basierend auf Benutzerkonten.
* Jeder Benutzer sieht nur die Datensätze, die für ihn bestimmt sind, basierend auf einer vordefinierten Logik.
* **Anwendungsfall:** Benutzer „Fred“ sieht nur zwei Datensätze, die für ihn relevant sind, während Benutzer „Chris“ andere Datensätze sieht.

**2. Komponenten der RLS-Implementierung**

**2.1 Prädikats-Funktion**

* Eine Tabellenwertfunktion, die entscheidet, ob ein Benutzer Zugriff auf eine Zeile hat.
* Funktioniert wie eine dynamische **WHERE-Klausel**, prüft aber spezifische Bedingungen.

**2.2 Sicherheits-Prädikat**

* Bindet die Prädikats-Funktion an eine Tabelle.
* Zwei Typen:
  + **Filter-Prädikat:** Beschränkt Lesezugriffe (z. B. SELECT).
  + **Block-Prädikat:** Beschränkt Schreiboperationen (z. B. INSERT, UPDATE, DELETE).

**2.3 Sicherheits-Policy**

* Eine Sammlung von Sicherheits-Prädikaten, die Regeln für den Datenzugriff definiert.

**3. Schritt-für-Schritt-Anleitung zur Implementierung von RLS**

**Schritt 1: Benutzer erstellen**

* Benutzer "Sophia" und "Eric" werden erstellt.

sql

Code kopieren

USE WideWorldImporters;

CREATE USER Sophia WITHOUT LOGIN;

CREATE USER Eric WITHOUT LOGIN;

**Schritt 2: Prädikats-Funktion erstellen**

* Eine Funktion wird erstellt, die prüft, ob der Benutzer Zugriff hat:
  + Benutzername wird mit der Spalte PreferredName abgeglichen.
  + Überprüfung, ob der Benutzer ein Verkäufer ist (IsSalesperson = 1).

sql

Code kopieren

CREATE SCHEMA Security;

CREATE FUNCTION Security.fn\_SecurityPredicateOrders(@personID INT)

RETURNS TABLE

WITH SCHEMABINDING

AS

RETURN (

SELECT 1 AS fn\_SecurityPredicateOrders\_Result

FROM Application.People

WHERE @personID = PersonID

AND PreferredName = USER\_NAME()

AND IsSalesperson = 1

);

**Schritt 3: Sicherheits-Policy erstellen**

* Bindet die Prädikats-Funktion an die Tabelle Sales.Orders, um die Datenfilterung basierend auf SalespersonPersonID zu erzwingen.

sql

Code kopieren

CREATE SECURITY POLICY Security.fn\_SecurityPredicates

ADD FILTER PREDICATE Security.fn\_SecurityPredicateOrders(SalespersonPersonID)

ON Sales.Orders

WITH (STATE = ON);

**Schritt 4: Rechte erteilen und testen**

* Den Benutzern werden Lese-Rechte erteilt.
* Mithilfe von EXECUTE AS wird die Policy getestet.

sql

Code kopieren

GRANT SELECT ON Sales.Orders TO Sophia;

GRANT SELECT ON Sales.Orders TO Eric;

EXECUTE AS USER = 'Sophia';

SELECT \* FROM Sales.Orders;

REVERT;

EXECUTE AS USER = 'Eric';

SELECT \* FROM Sales.Orders;

REVERT;

**4. Vorteile von RLS**

* **Datenintegrität:** Benutzer sehen nur relevante Daten.
* **Zentrale Steuerung:** Regeln werden an zentraler Stelle definiert.
* **Flexibilität:** Anpassbar durch Sicherheits-Policies und Prädikats-Funktionen.

**5. Fazit**

RLS bietet eine mächtige Möglichkeit, Zugriffsrechte auf Zeilenebene dynamisch zu steuern, ohne dass die Anwendungsschicht explizite Filter implementieren muss. Es sorgt für mehr Sicherheit und Benutzerfreundlichkeit, indem es Benutzern nur die für sie bestimmten Daten anzeigt.

**Zusammenfassung: Dynamic Data Masking (DDM) im SQL Server**

**1. Was ist Dynamic Data Masking (DDM)?**

Dynamic Data Masking ist ein Sicherheitsfeature des SQL Servers, das sensitive Daten für unbefugte Benutzer maskiert. Dabei bleiben die Daten in der Datenbank unverändert, und die Maskierung erfolgt nur auf der Anzeigeebene.

**2. Vorteile von DDM**

* **Sicherheit:** Verhindert unbefugten Zugriff auf sensible Daten.
* **Flexibilität:** Maskierung erfolgt individuell pro Spalte und Benutzer.
* **Effizienz:** Daten werden maskiert, ohne dass zusätzliche Tabellen oder Änderungen erforderlich sind.

**3. Maskierungsfunktionen**

SQL Server bietet vier Funktionen zur Maskierung:

| **Funktion** | **Beschreibung** | **Beispiel** |
| --- | --- | --- |
| **Default** | Zeigt keine Informationen der Spalte, maskiert den gesamten Inhalt. | XXXX |
| **Email** | Zeigt das erste Zeichen und die Domain-Endung, maskiert den Rest. | aXXX@XXXX.com |
| **Random** | Generiert eine Zufallszahl innerhalb eines bestimmten Bereichs. | 531264 |
| **Custom String** | Zeigt Teile des realen Werts und fügt einen benutzerdefinierten String ein. | aMASKEDg |

**4. Syntax und Beispiele**

**4.1 Default-Maskierung**

Die gesamte Spalte wird maskiert.

sql

Code kopieren

ALTER TABLE Application.People

ALTER COLUMN LogonName ADD MASKED WITH (FUNCTION = 'default()');

**4.2 Email-Maskierung**

Zeigt das erste Zeichen der E-Mail-Adresse und die Domain-Endung, maskiert den Rest.

sql

Code kopieren

ALTER TABLE Application.People

ALTER COLUMN EmailAddress ADD MASKED WITH (FUNCTION = 'email()');

**4.3 Zufallswert-Maskierung**

Generiert eine Zufallszahl zwischen einem definierten Minimum und Maximum.

sql

Code kopieren

ALTER TABLE Sales.Orders

ALTER COLUMN PickedByPersonID ADD MASKED WITH (FUNCTION = 'random(1, 99)');

**4.4 Benutzerdefinierte String-Maskierung**

Zeigt Teile des ursprünglichen Inhalts und fügt einen benutzerdefinierten String ein.

sql

Code kopieren

ALTER TABLE Application.People

ALTER COLUMN PhoneNumber ADD MASKED WITH (FUNCTION = 'partial(1, "XXXXXXX", 0)');

**5. Berechtigungen**

* **Standardbenutzer:** Sehen nur die maskierten Werte.
* **UNMASK-Berechtigung:** Benutzer mit dieser Berechtigung können die unmaskierten Originalwerte anzeigen.

**Vergabe der UNMASK-Berechtigung**

sql

Code kopieren

GRANT UNMASK TO Benutzername;

**Entzug der UNMASK-Berechtigung**

sql

Code kopieren

REVOKE UNMASK FROM Benutzername;

**6. Beispiele zur Maskierung und Anzeige**

**6.1 Testen der Maskierung**

sql

Code kopieren

SELECT LogonName, EmailAddress, PickedByPersonID, PhoneNumber

FROM Application.People;

* Benutzer ohne UNMASK-Rechte sehen die maskierten Werte (z. B. XXXX oder aXXX@XXXX.com).
* Benutzer mit UNMASK-Rechten sehen die Originalwerte.

**7. Fazit**

Dynamic Data Masking ist eine leistungsstarke Methode, um Daten auf der Anzeigeschicht vor unbefugten Benutzern zu schützen. Es bietet Flexibilität und Sicherheit, indem es sensitive Informationen maskiert, ohne die zugrunde liegenden Daten zu verändern. Dieses Feature ist besonders nützlich in Anwendungen, bei denen Benutzer eingeschränkten Zugriff auf sensible Daten benötigen.

**Zusammenfassung: Verschlüsselung in SQL Server**

**1. Was ist Verschlüsselung in SQL Server?**

SQL Server bietet verschiedene Methoden, um sensitive Daten zu schützen, darunter:

* **Transparent Data Encryption (TDE):** Verschlüsselt Datenbanken und Logfiles auf Dateiebene.
* **Spaltenverschlüsselung:** Verschlüsselt gezielt bestimmte Spalten in einer Tabelle.
* **Always Encrypted:** Verschlüsselt Daten, bevor sie in der Datenbank gespeichert werden, und hält die Schlüssel außerhalb der Datenbank.

**2. Transparent Data Encryption (TDE)**

* **Zweck:** Schützt ruhende Daten durch Dateiverschlüsselung.
* **Funktionsweise:**
  + TDE verwendet einen symmetrischen **Database Encryption Key**, der durch ein Zertifikat in der master-Datenbank geschützt wird.
  + Die Datenbank wird real-time verschlüsselt.

**Vorgehen: TDE aktivieren**

sql

Code kopieren

USE master;

-- Master Key erstellen

CREATE MASTER KEY ENCRYPTION BY PASSWORD = '123\_My\_VeryStrong\_Password!';

-- Zertifikat erstellen

CREATE CERTIFICATE myservercert WITH SUBJECT = 'my key certificate';

-- Encryption Key erstellen

USE WideWorldImporters;

CREATE DATABASE ENCRYPTION KEY WITH ALGORITHM = AES\_256 ENCRYPTION BY SERVER CERTIFICATE myservercert;

-- Verschlüsselung aktivieren

ALTER DATABASE WideWorldImporters SET ENCRYPTION ON;

* **Wichtig:** Master Key und Zertifikat müssen sicher aufbewahrt werden.

**3. Verschlüsselung von einzelnen Spalten**

* **Zweck:** Nur spezifische sensitive Informationen wie Bankdaten verschlüsseln.
* **Vorgehen:**
  1. Erstelle einen **Master Key** und ein **Zertifikat**.
  2. Definiere einen **symmetrischen Schlüssel**.
  3. Verschlüssele die Daten in einer neuen Spalte.

**Beispiel: Bankdaten verschlüsseln**

sql

Code kopieren

-- Master Key erstellen

CREATE MASTER KEY ENCRYPTION BY PASSWORD = '123\_MyComplexPassword';

-- Zertifikat erstellen

CREATE CERTIFICATE SuppliersBAN WITH SUBJECT = 'Suppliers Bank Account Number';

-- Symmetrischen Schlüssel erstellen

CREATE SYMMETRIC KEY SSN\_Key\_01 WITH ALGORITHM = AES\_256 ENCRYPTION BY CERTIFICATE SuppliersBAN;

-- Neue Spalte für verschlüsselte Daten hinzufügen

ALTER TABLE Purchasing.Suppliers ADD EncryptedBAN varbinary(128);

-- Symmetrischen Schlüssel öffnen

OPEN SYMMETRIC KEY SSN\_Key\_01 DECRYPTION BY CERTIFICATE SuppliersBAN;

-- Daten verschlüsseln und speichern

UPDATE Purchasing.Suppliers

SET EncryptedBAN = EncryptByKey(Key\_GUID('SSN\_Key\_01'), BankAccountNumber);

-- Daten überprüfen

SELECT BankAccountNumber, EncryptedBAN,

CONVERT(nvarchar, DecryptByKey(EncryptedBAN)) AS DecryptedBankAccountNumber

FROM Purchasing.Suppliers;

-- Schlüssel schließen

CLOSE SYMMETRIC KEY SSN\_Key\_01;

* **Hinweis:** Verschlüsselte Daten werden in Binärform gespeichert und können nur mit dem richtigen Schlüssel entschlüsselt werden.

**4. Always Encrypted**

* **Zweck:** Verschlüsselt Daten durch eine Client-Anwendung, bevor sie in die Datenbank gelangen.
* **Besonderheit:**
  + Die Datenbank selbst hat keinen Zugriff auf die Schlüssel.
  + Nur der Client mit dem **Column Master Key** kann die Daten entschlüsseln.
* **Vorteil:** Schutz vor internen und externen Bedrohungen, einschließlich DB-Administratoren.

**Funktionsweise von Always Encrypted**

* **Column Master Key (CMK):** Verschlüsselt einen oder mehrere **Column Encryption Keys (CEK)**.
* **Column Encryption Key (CEK):** Verschlüsselt die tatsächlichen Daten in der Spalte.
* **Speicherung:** CMK und CEK werden außerhalb der Datenbank (z. B. in einem Key Store) verwaltet.

**5. Zusammenfassung der Verschlüsselungsmethoden**

| **Methode** | **Einsatzbereich** | **Vorteil** | **Beispiel** |
| --- | --- | --- | --- |
| **TDE** | Vollständige Datenbank | Einfach, schützt gesamte Datenbank | Real-time Verschlüsselung |
| **Spaltenverschlüsselung** | Einzelne Spalten | Präzise Kontrolle über zu schützende Daten | Bankdaten verschlüsseln |
| **Always Encrypted** | Externe Verschlüsselung | Schutz vor internen Angriffen, Schlüssel außerhalb DB | Ideal für Cloud-Speicherung |

**Fazit**

SQL Server bietet flexible Verschlüsselungsmechanismen, die auf verschiedene Szenarien zugeschnitten sind:

* **TDE** eignet sich für umfassenden Schutz ruhender Daten.
* **Spaltenverschlüsselung** ist ideal für gezielte Sicherheit sensibler Informationen.
* **Always Encrypted** bietet maximale Sicherheit, indem sensible Daten nie in unverschlüsselter Form in der Datenbank gespeichert werden.

**🧾Sicherheits- und Rollenkonzept**

Handlungsziel 1

**Überlegungen und Fragestellungen**

Beim Erstellen eines Sicherheits- und Rollenkonzepts sollte man möglichst den Gesamtkontext im Auge behalten und sich u.a. an den folgenden Fragen orientieren:

* Wird von ausserhalb des unternehmensinternen Netzwerks auf die Daten zugegriffen?
* Worauf benötigt der User Zugriff: eine oder mehrere Datenbanken?
* Auf welche Objekte benötigt der User Zugriff?
* Existiert schon ein Server Login für den User?
* Gibt es passende vordefinierte Rollen für den User?
* Welche Gruppen von Usern dürfen auf welche Daten zugreifen?
* Welche Daten sind besonders schützenswert?

**Aufgabe (in Gruppen)**

Erstelle bezogen auf **Wide World Importers** ein Use Case-Diagramm mit möglichen Akteuren und Anwendungsfällen von Datenbankzugriffen.

Ein Bild, das Diagramm, Reihe, weiß, Text enthält.

Automatisch generierte Beschreibung

Aufgabe 15 login und user erstellen berechtigungen. db\_datareader = leserecht