Datenbanken

Robin Rausch, Florian Maslowski, Ozan Akzebe

14. Mai 2023

Inhaltsverzeichnis

1 Grundlagen

1.1 Ebenen

Interne: wirkliche Speicherung der Daten im internen Speicher (SQL)

Konzeptionelle: beschreibt welche Daten gespeichert werden sollen und wie die Beziehung der Daten untereinander aufgebaut ist. (ERM, UML)

Externe: Sicht des Kunden auf die Daten. Hier kann für jede Benutzergruppe eine eigene Sicht auf die Daten zur Verfügung gestellt werden.

1.2 Begrifflichkeiten

Entität: Entitäten einer Datenbank sind Objekte, die sich eindeutig von anderen Objekten des gleichen Entitätstyps abgrenzen lassen. Die Entitäten stehen in der Regel mit sich selbst oder mit anderen Entitäten in Beziehung.

Relation: Tabelle, in welcher jedes Tupel eine Entität darstellt.

Attribut: In einem relationalen Datenbankmodell ist ein Attribut eine Spalte einer Tabelle. Jede Entität besitzt eine definierte Anzahl an Attributen (Eigenschaften), die sich eindeutig von anderen Entitäten des gleichen Entitättyps abgrenzen.

Beziehung: Verbindung zwischen Entitäten. Gibt Kardinalitäten an

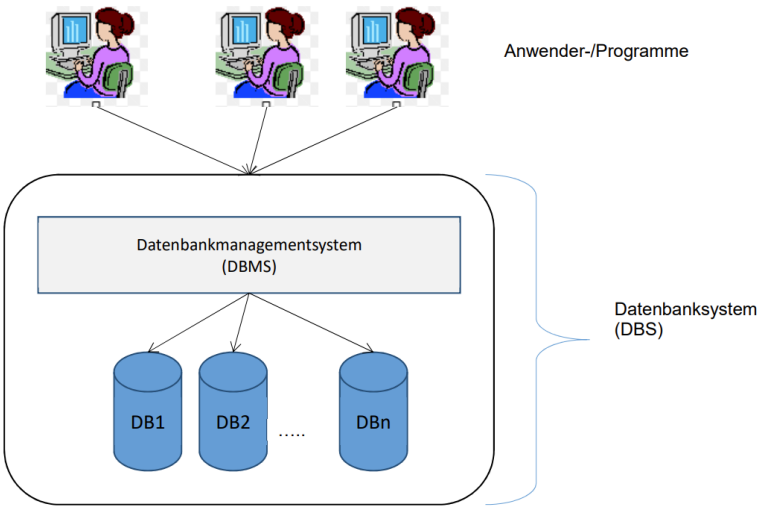
Tupel: Zeile in Tabelle

Indizes: Datenbankindex, welcher von der Datenstruktur getrennt ist. Beschleunigt Suche und Sortierung nach bestimmten Feldern

Optimizer: Macht DB effizient. Sucht besten Weg eine Anfrage auszuführen. Indizes spielen eine wichtige Rolle.

Schema: Struktur einer Tabelle. Gesammtheit der Attribute und deren Eigenschaften.

Instanz/Ausprägung: Spezielle Ausprägung eines Schemas mit zugehörigen Werten



1.3 Historie

1.3.1 Hierarchisches Modell

Das Hierarchische Modell kann nur 1:N Beziehungen modellieren, da dieses wie eine Baumstruktur aufgebaut ist.

1.3.2 Relationale Datenbanken

Bei Relationalen Datenbanken liegt der Fokus auf den Relationen. Hier können auch N:M Beziehungen modelliert werden.

1.4 Coddschen Regeln

Integration: Keine Dopplungen der Daten

Operation: CRUD - create, read, update und delete

Katalog: Im Katalog werden Informationen abgelegt, die die Daten in einer Datenbank beschreiben.

Benutzersichten: Daten können verschieden gelesen/interpretiert werden

Konsistenzüberwachung/Integritätssicherung: Die Korrektheit der Daten muss zu jedem Zeitpunkt gewährleistet sein. Es dürfen keine Anomalien auftreten. Die Datenspeicherung muss nach einem vorgegebenen Schema erfolgen.

Zugriffskontrolle: Nur notwendige Personen sollen auf die Daten zugreifen können.

Transaktionen: Bündelung mehrerer Anweisungen zu einer einzigen Transaktion, welche als eine funktionale Einheit ausgeführt werden – entweder ganz oder gar nicht.

Synchronisation: Parallel ausgeführte Transaktionen müssen den gleichen Datenbankzustand hervorrufen wie irgendeine serielle Ausführung der Transaktionen.

Datensicherung: Das Datenbanksystem muss nach einem Systemfehler in der Lage sein, den letzten konsistenten Datenbankzustand mittels automatischer Datensicherungs- und Wiederherstellungsmechanismen herzustellen.

1.5 Von der Idee zur Datenbank

Vorgehen:

1. Anforderungsanalyse
2. Kozeptioneller Entwurf (ERM)
3. Logischer Entwurf (Überführung des ERM in ein Datenbankmodell)
4. Datenbank-Definition (DDL)
   Kann zusammen mit Logischem Entwurf zu sog. ***Implementierungsentwurf*** *zusammengefasst werden*
5. Physikalischer Entwurf

Ebenso muss das Sichtenmodell beachtet werden. Dieses beschreibt die Problematik, dass unterschiedliche Personen/Rollen unterschiedliche Sichten auf ein Modell haben. Bei der Planung müssen deshalb diese Sichten kombiniert werden, um eine universelle Lösung zu finden.

1.6 Referentielle Integrität

RI meint Bedingungen, die zur Sicherung der Datenintegrität bei Nutzung relationaler Datenbanken beitragen können. Nach der RI-Regel dürfen Datensätze (über ihre Fremdschlüssel) nur auf existierende Datensätze verweisen.

2 Normalisierung

Dient dazu redundante Speicherung von Informationen und damit Inkonsistenz und Anomalien zu vermeiden.

2.1 Begrifflichkeiten

Funktionale Abhängigkeit: Attribut B ist voll funktional abhängig von A, wenn B durch A eindeutig identifiziert werden kann. ($f(A) = B$)

Triviale funktionale Abhängikeiten: sind funktionale Abhängigkeiten auf sich selbst: $f(A) = B$ mit $B \subseteq A$ bzw. $X \rightarrow X$

Verlustfreie Zerlegung: Alle Tupel der ursprünglichen Tabelle können durch einen Join aus den abgeleiteten Relationen wiederherstellen lassen. Sie sichert damit die Wiederherstellbarkeit der ursprünglichen Relation.

Abhängigkeitsbewahrend: Wenn alle Funktionalen Abhängigkeiten, nach Zerlegung der (Ursprungs)-Relation, auf einer der Teilrelationen dargestellt werden können.

2.2 Schlüssel

2.2.1 Superschlüssel

Jede Teilmenge $S_i \subseteq$ [R] für die gilt $S_i \rightarrow$ [R] heißt Superschlüssel von R.
Ein Superschlüssel ist die Menge von Attributen einer Relation, welche ein Tupel in dieser Relation eindeutig identifizieren. Es sagt noch nichts darüber aus, ob es sich um eine minimale Menge von Attributen handelt. Ein Superschlüssel wäre zum Beispiel die Menge aller Attribute einer Relation gemeinsam ([R] $\rightarrow$ [R]).

2.2.2 Schlüsselkandidat

Eine minimale Teilmenge der Attribute eines Superschlüssels, welche die Identifizierung der Tupel ermöglicht (Schlüsselkandidaten $\subseteq$ Superschlüssel). In der Regel wird einer dieser Kandidaten als Primärschlüssel ausgewählt. Eine Relation kann mehrere Schlüsselkandidaten haben.

2.2.3 Primärschlüssel

Der Primärschlüssel ist dann ein (willkürlich) ausgewählter Schlüsselkandidat, der zur eindeutigen Identifikation der Tupel in der Relation verwendet wird.

2.2.4 Primattribute

Ein Attribut wird Primattribut oder prim genannt, wenn es in irgendeinem Schlüsselkandidaten von [R] vorkommt.
Formal: Wenn K die Menge aller Schlüsselkandidaten von [R] ist, dann heißen alle Attribute in der Vereinigung über alle j von K_j, prim oder Primattribute. Alle Attribute in $\bigcup_j K_j$ heißen prim.

2.3 Normalformen

2.3.1 1. Normalform

Eine Relation erfüllt die erste Normalform, wenn jede Entität (jedes Tupel) für jedes Attribut der Relation nur einen Datenwert besitzt.
Atomar => Attribute können nicht weiter aufgeteilt werden.

2.3.2 2. Normalform

Eine Relation erfüllt die zweite Normalform (2NF), wenn diese sich in der ersten Normalform befindet und alle Nichtschlüsselattribute nur durch den gesamten Primärschlüssel festgelegt werden.
Also: 1. Normalform + Nichtschlüsselattribute sind von jedem Schlüsselkandidaten voll funktional abhängig.
Nichtschlüsselattribute müssen vom gesammten Schlüssel abhängig sein. Dazu müssen eventuell die Tabellen gesplittet werden.

2.3.3 3. Normalform

Eine Relation ist in der 3.Normalform (3NF), wenn diese die 2. Normalform erfüllte und keine transitiven Abhängigkeiten zwischen einem Nichtschlüsselattribut und einem Schlüsselkandidaten bestehen.
Also: 2. Normalform + keine transitiven Abhängigkeiten von Nichtschlüsselattribut zu Schlüsselkandidat.
Das heißt: Wenn B von A abhängig ist (A -> B), darf B nicht auf z.B. C abbilden(B->C). Hier müssen, die von B abhängigen Attribute in eine extra Tabelle ausgelagert werden. Dazu müssen eventuell die Tabellen gesplittet werden.

Formale Definition:
Ein Relationsschema [R] ist in 3NF, wenn für jede für [R] geltende funktionale Abhängigkeit der Form $X \rightarrow \alpha,\ X \sqsubseteq [R]\ und\ \alpha \in [R]$
mindestens eine der drei Bedingungen gilt:

- $\alpha \in X$, (dann ist die funktionale Abhängigkeit trivial)
- Das Attribut α ist in einem Schlüsselkandidaten von [R] enthalten (dies bedeutet α ist „prim“)
- X ist ein Superschlüssel von [R]

2.3.4 Boyce-Codd Normalform

Die Boyce-Codd Normalform ist eine Verschärfung der 3NF. Die Boyce-Codd-Normalform (BCNF) ist eine Weiterentwicklung der Dritten Normalform (3NF). In der Dritten Normalform kann es vorkommen, dass ein Teil eines Schlüsselkandidaten funktional abhängig ist von einem Teil eines anderen Schlüsselkandidaten. Die Boyce-Codd-Normalform verhindert diese funktionale Abhängigkeit.

Definition:
Ein Relationsschema [R] ist in Boyce-Codd Normalform (BCNF), wenn für jede [R] geltende funktionale Abhängigkeit der Form $X \rightarrow \alpha,\ X \sqsubseteq [R]\ und\ \alpha \in [R]$, wenn mindestens eine der beiden Bedingungen gilt:

- $\alpha\ \in X$, (dann ist die funktionale Abhängigkeit trivial)
- X ist ein Superschlüssel von [R]

Eine Relation ist in Boyce-Codd Normalform, wenn jeder Determinante ein Superschlüssel ist.

2.4 Zerlegung/Dekomposition von Relationen

Relationsschema [R]					
a	**b**	**c**	**d**	**e**	**f**
…	…	…	…	…	…
…	…	…	…	…	…

$$a \rightarrow bc$$
$$d \rightarrow e$$
$$ad \rightarrow f$$

1. Normalform wir als gegeben vorausgesetzt.

Schlüsselkandidat ist nur ad.
Zerlegung in mehrere Tabellen:
R1: a, d, f
R2: d, e
R3: a, b, c
Da $[R1] \wedge [R2 = d]$

3 Datenbankmanagementsystem (DBMS)

3.1 PostgreSQL

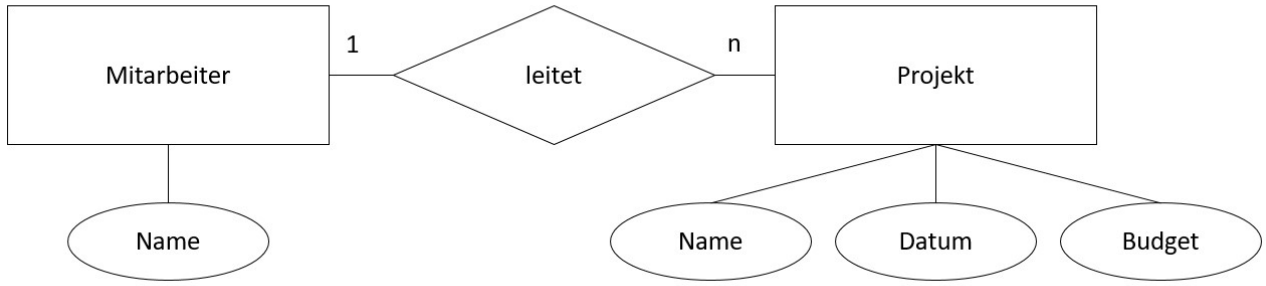
flo?

3.2 MySql

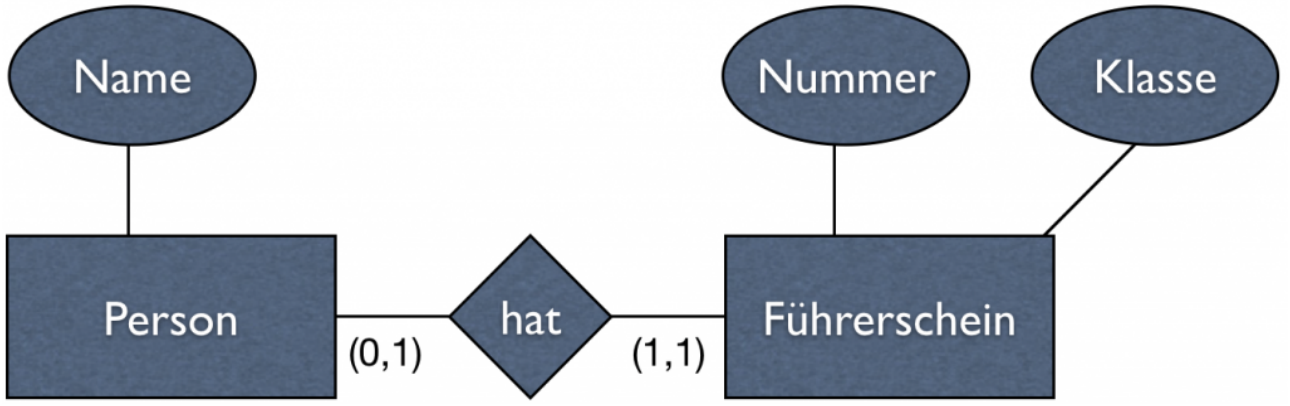
flo?

4 Entity Relationship Model

4.1 Chen-Notation

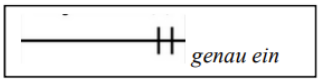


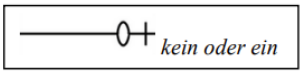
4.2 Min-Max-Notation

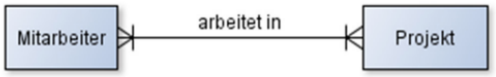


4.3 Krähenfß-Notation

Chen-Notation	Krähenfuß-Notation
1:1	
1:N	
M:N	







5 SQL

SQL lässt sich in folgende Formen von Kommandos einteilen:
Data Manipulation Language (DML), Data Definition Language (DDL), Data Control Language(DCL), Data Query Language (DQL), Transaction Control Language (TCL).
SQL setzt sich aus folgenden Sprachen zusammen:

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Data Control Language (DCL)

- Data Query Language (DQL)
- Transaction Control Language (TCL)

5.1 Data Manipulation Language (DML)

Merkspruch: Sag Für Wen Geht Hilde Ohne Lippenstift.
Befehle: SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY, LIMIT

SELECT [columns] (Funktionen wie: AVG, SUM, ... möglich)

FROM [table]

WHERE condition(s) (Wird auf individuelle Zeilen angewandt)

GROUP BY [column]

HAVING [condition] (Wird auf Gruppen angewandt, Funktionen können verwendet werden.)

ORDER BY [column] ASC/DESC

LIMIT [number of rows];

UPDATE [columns]

SET [$column_n = value_n$]

WHERE [condition];

INSERT INTO [table] [(column$_1$, column$_2$, ...)]

VALUES [$column_n = value_n$];

DELETE FROM [table]

WHERE [condition];

Zusätzlich gibt es Löschregeln. Diese sind CASCADE, RESTRICT und SET NULL. CASCADE löscht alle Einträge in jeder Tabelle die mit dem gelöschten Element in Verbindung stehen. RESTRICT verbietet das Löschen und SET NULL löscht das genannte Element und setzt alle Referenzen des gelöschten Elementes auf NULL.

DISTINCT schließt doppelte Werte aus (SELECT DISTINCT Vorname FROM Mitarbeiter). Man kann auch Funktionen anwenden, welche z.B. den Durchschnitt berechnen (AVG, SUM, COUNT, MIN, MAX).

5.1.1 Unterabfragen

Select Vorname, Nachname, Gehalt from Mitarbeiter Where Gehalt > (Select Avg(Gehalt) from Mitarbeiter) Order By Gehalt;
Diese müssen in Klammern stehen. Bei den Unterabfragen können auch Tabellen herauskommen. In diesem Fall muss ein Ausdruck davor gesetllt werden(ANY, ALL, EXISTS, IN).

5.2 Data Definition Language (DDL)

CREATE TABLE [table]

[column$_n$] [datatype] [constraints];

ALTER TABLE [table]

[column$_n$] [datatype] [constraints];

DROP TABLE [table];

5.3 Begriffe, etc.

CONSTRAINTS: Beschränkungen/Bedingungen für Spalten (Bsp.: UNIQUE, NOT NULL, CHECK, FOREIGN KEY, PRIMARY KEY, etc.)

5.4 ACID-Regeln

Atomicity: Transaktionen müssen ganz oder garnicht ausgeführt werden

Consistency: Datenbank muss nach Transaktionen einen gültigen Zustand annehmen. Bei Fehlern wird vorheriger Zustand beibehalten.

Isolation: Abfragen von verschiedenen Usern/Prozessen müssen voneinander abgegrenzt sein.

Durability: Werte müssen nach Transaktionen dauerhaft gespeichert sein.

5.5 Relationale Algebra

5.5.1 Selektion σ

$\sigma_{Abteilung=ANr}(Mitarbeiter \times Abteilung)$ oder $\sigma_{Geschlecht='w'}(Mitarbeiter)$
Damit bekommt man alle Zeilen für welche das zutrifft. Vergleichbar mit WHERE.

5.5.2 Projektion π

$\pi_{Name,Abteilung}(Mitarbeiter)$
Damit bekommt man nur die selektierten Attribute der jeweiligen Zeilen. Vergleichbar mit SELECT.

5.5.3 Verkettung

Diese Grundfunktionen lassen sich auch verketten:
$\pi_{Vorname}(\sigma_{Geschlecht='w'}(Mitarbeiter))$

5.5.4 Vereinigung und Differenz

Vereinigung:
$\pi_{Nachname}(Mitarbeiter) \cup \pi_{Nachname}(Kunde)$
Differenz:
$\pi_{Nachname}(Mitarbeiter) - \pi_{Nachname}(Kunde)$

5.5.5 Durchschnitt

Der Durchschnitt wird gleich verwendet wie die Vereinigung und Differenz. Dieser gibt alle Zeilen der Einträge zurück die nicht in beiden Tabellen liegen. Das Zeichen ist $\cap$.

5.5.6 Kreuzprodukt

$\sigma_{Mitarbeiter.Gehaltsstufe=Vergtungsgruppe.Gehaltsstufe}(Mitarbeiter \times Vergtungsgruppe)$

5.5.7 JOIN

Wird durch $\bowtie$ dargestellt.

5.5.8 Theta-Join $\bowtie_p$

Das Theta-Join ist eine Erweiterung des Kreuzprodukts, das zusätzliche Bedingungen verwendet, um die Verknüpfung von Zeilen in zwei Tabellen zu steuern. Es wird mit dem Operator $\bowtie_p$ dargestellt.
Syntax:
$\sigma_{Bedingung}(Tabelle_1 \bowtie_p Tabelle_2)$
Beschreibung:
Das Theta-Join kombiniert Zeilen aus zwei Tabellen, indem es die angegebene Bedingung überprüft. Nur die Zeilen, die die Bedingung erfüllen, werden im Ergebnis zurückgegeben.
Beispiel:
$\sigma_{Tabelle_1.Spaltename=Tabelle_2.Spaltename}(Tabelle_1 \bowtie_p Tabelle_2)$
Dieser Syntax führt das Theta-Join durch, indem es die Spalten Spaltename in $Tabelle_1$ und $Tabelle_2$ vergleicht und nur die Zeilen zurückgibt, in denen die Werte übereinstimmen.

5.5.9 Equi-Join $\bowtie_{[L],[R]}$

Der Equi-Join ist eine spezielle Form des Theta-Joins, bei dem die Bedingung auf einer Gleichheit zwischen zwei Spalten basiert. Es wird häufig verwendet, um Zeilen aus zwei Tabellen basierend auf übereinstimmenden Werten in den angegebenen Spalten zu verknüpfen.
Syntax:
$\sigma_{Tabelle_1.Spaltename=Tabelle_2.Spaltename}(Tabelle_1 \bowtie_{[L],[R]} Tabelle_2)$
Beschreibung:
Der Equi-Join vergleicht die Werte der angegebenen Spalten in $Tabelle_1$ und $Tabelle_2$ und gibt nur die Zeilen zurück, in denen die Werte übereinstimmen.
Beispiel:
$\sigma_{Tabelle_1.ID=Tabelle_2.ID}(Tabelle_1 \bowtie_{[ID],[ID]} Tabelle_2)$

5.5.10 Natürlicher Join

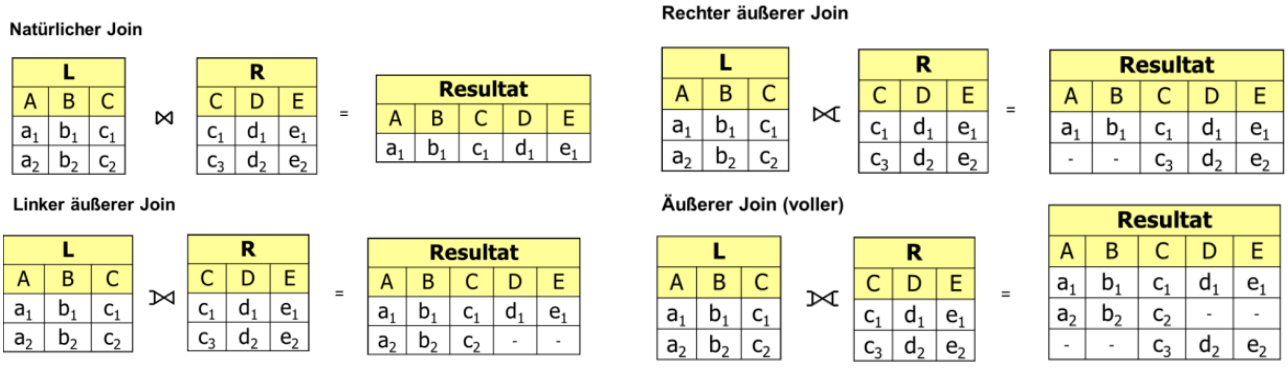
Der Natürliche Join ist eine spezielle Form des Equi-Joins, bei dem die Verknüpfung basierend auf allen gemeinsamen Spaltennamen in den beiden Tabellen erfolgt. Es werden automatisch die Spalten verglichen, die den gleichen Namen haben.
Syntax:
$Tabelle_1 \bowtie Tabelle_2$
Beschreibung:
Der Natürliche Join vergleicht automatisch alle gemeinsamen Spalten in $Tabelle_1$ und $Tabelle_2$ und gibt nur die Zeilen zurück, in denen die Werte in allen gemeinsamen Spalten übereinstimmen.
Beispiel:
$Tabelle_1 \bowtie Tabelle_2$

5.5.11 Left Outer-Join

Der Left Outer-Join ist eine Verknüpfungsoperation, die alle Zeilen der linken Tabelle und die übereinstimmenden Zeilen der rechten Tabelle zurückgibt. Wenn keine Übereinstimmungen vorhanden sind, enthält das Ergebnis NULL-Werte für die Spalten der rechten Tabelle.
Syntax:
$Tabelle_1 \bowtie Tabelle_2$ ON Bedingung
Beispiel:
$Mitarbeiter \bowtie Abteilung$ ON Mitarbeiter.Abteilung = Abteilung.ANr
Dieser Syntax führt einen Left Outer-Join durch, indem er die Spalte Abteilung in der Tabelle Mitarbeiter mit der Spalte ANr in der Tabelle Abteilung vergleicht und alle Zeilen der Tabelle Mitarbeiter und übereinstimmende Zeilen der Tabelle Abteilung zurückgibt.

5.5.12 Right Outer-Join

Der Right Outer-Join ist eine Verknüpfungsoperation, die alle Zeilen der rechten Tabelle und die übereinstimmenden Zeilen der linken Tabelle zurückgibt. Wenn keine Übereinstimmungen vorhanden sind, enthält das Ergebnis NULL-Werte für die Spalten der linken Tabelle.
Syntax:
$Tabelle_1 \bowtie Tabelle_2$ ON Bedingung
Beispiel:
$Mitarbeiter \bowtie Abteilung$ ON Mitarbeiter.Abteilung = Abteilung.ANr
Dieser Syntax führt einen Right Outer-Join durch, indem er die Spalte Abteilung in der Tabelle Mitarbeiter mit der Spalte ANr in der Tabelle Abteilung vergleicht und alle Zeilen der Tabelle Abteilung und übereinstimmende Zeilen der Tabelle Mitarbeiter zurückgibt.



5.5.13 Umbenennung

Wird durch ρ dargestellt. Die Umbenennung einer Relation kann dann erforderlich werden, wenn wir in einer Abfrage eine Relation mehrfach verwenden möchten (siehe Joins).

5.6 Views

Views in Datenbanken sind virtuelle Tabellen, die aus den Daten einer oder mehrerer Basis- oder anderen Views erstellt werden. Sie bieten eine Möglichkeit, Daten aus verschiedenen Tabellen logisch zu kombinieren oder bestimmte Daten aus einer Tabelle auszuwählen und sie als separate Tabelle darzustellen. Views stellen eine abstrahierte Sicht auf die zugrunde liegenden Daten dar, wobei die tatsächlichen Daten unverändert bleiben.
Views bieten mehrere Vorteile. Erstens ermöglichen sie eine verbesserte Datenintegrität, da sie es ermöglichen, komplexe Regeln und Einschränkungen auf die darunterliegenden Tabellen anzuwenden. Zweitens ermöglichen Views eine verbesserte Sicherheit, da sie den Zugriff auf sensible Daten einschränken können, indem sie nur bestimmte Spalten oder Zeilen zugänglich machen. Drittens erleichtern sie die Datenbankverwaltung, da sie komplexe Abfragen vordefinieren und die Wiederverwendung von Abfragen ermöglichen.
Views können in Abfragen genauso verwendet werden wie reguläre Tabellen. Benutzer können auf sie zugreifen, Daten abfragen, Aktualisierungen durchführen oder sie in anderen Views verwenden. Änderungen, die an den Daten einer View vorgenommen werden, können sich auf die zugrunde liegenden Tabellen auswirken, je nach den festgelegten Regeln und Einschränkungen.
Zusammenfassend bieten Views in Datenbanken eine flexible und effiziente Möglichkeit, Daten logisch zu organisieren, komplexe Abfragen zu vereinfachen, die Sicherheit zu verbessern und die Datenintegrität zu gewährleisten.

5.7 Transaktionen

Eine Transaktion ist eine logische Arbeitseinheit (Bündelung mehrerer Anweisungen), welche dafür sorgt, dass logisch zusammengehörige Folgen von Operationen ohne negative Begleiterscheinungen auf der Datenbank ausgeführt werden können. Transaktionen folgen ebenfalls den ACID Anforderungen. Verhindern dass zwei gleichzeitig auftretende Anfragen an eine Datenbank die Integrität zerstören oder falsche Ergebnisse liefern. Sobald ein Fehler bei einer Transaktion auftritt, wird ein Rollback ausgeführt der die Änderungen rückgängig macht.