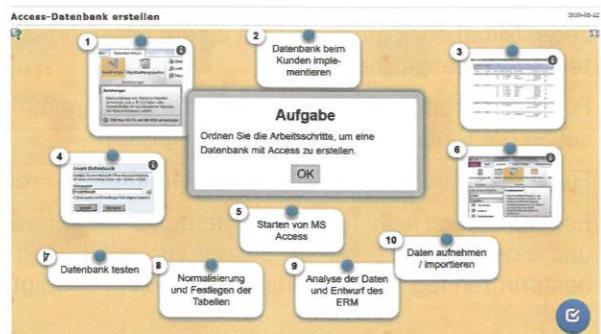


5.8 Digitale Inhalte zu Kapitel 5

Hinweis: Um die Aufgaben online zu bearbeiten, bitte den QR-Code scannen oder den Link eingeben.

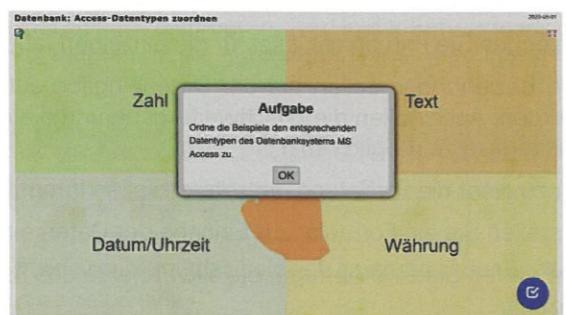
Aufgabe 1

Erstellen einer Datenbank mit Access
<https://vel.plus/UFga>



Aufgabe 2

Datentypen zuordnen
<https://vel.plus/j0jG>



Aufgabe 3

Fachbegriffe einsetzen
<https://vel.plus/YLTI>



Aufgabe 4

Kahoot-Quiz
 Suchbegriffe in der Kahoot-App: "36087" und "ACCESS"



6 Die Datenbanksprache SQL

SQL (Structured Query Language = strukturierte Abfragesprache) ist eine Datenbanksprache zur Definition von Datenstrukturen in relationalen Datenbanken sowie zum Bearbeiten (Einfügen, Verändern, Löschen) und Abfragen von Datenbeständen.

6.1 SQL-Standards

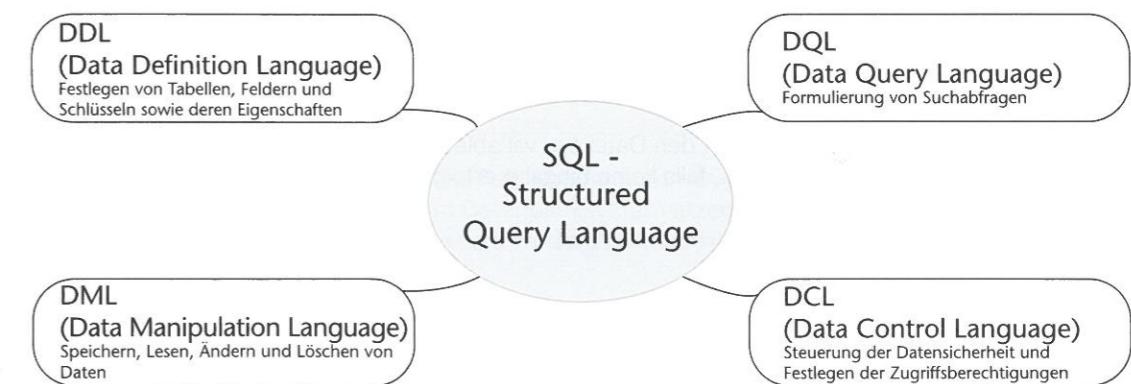
SQL war zunächst eine Sprache für den Endbenutzer von relationalen Datenbanksystemen. Sie wurde hier durch die weit verbreiteten grafischen Oberflächen ersetzt. Da SQL aber mächtiger ist als viele grafische Systeme für Datenzugriffe, ist sie für Datenbankentwickler ein wichtiges Softwarewerkzeug. Auch als Schnittstellensprache und als Zugriffssprache auf andere Datenbanksysteme hat sie große Bedeutung, da sie nahezu unabhängig von Betriebssystem, Betriebsart und Oberfläche ist.

Man unterscheidet verschiedene Standards:

Jahr	Norm	Standard
1986	ANSI-SQL1	SQL1-Standard
1989	ISO SQL-1	SQL1 mit Integritätserweiterungen
1992	ISO SQL-92	SQL2-Standard
1999	ISO/IEC 9075:1999	SQL3-Standard, Trigger, rekursive Abfragen
2003	ISO/IEC 9075:2003	Verknüpfung mit XML (eXtensible Markup Language)
2011	ISO/IEC 9075:2011	Erweiterungen für Windows functions
2016	ISO/IEC 9075:2016	Unterstützung von JavaScript Object Notations (JSON)
2019	SQL/MDA:2019	Erweiterung für Datentyp „mehrdimensionales Feld“

Gegenwärtig ist in den Datenbanksystemen vor allem der SQL2-Standard verwirklicht, während die neueren Standards nur in ausgewählten Systemen umgesetzt werden.

Die Sprache SQL besteht aus verhältnismäßig wenig Befehlen, die jedoch vielfältig eingesetzt werden können. Sie dient unter anderem sowohl zur Tabellen- und Felddefinition als auch zur Abfrage und Veränderung von Daten.



6.2 Erzeugen, Ändern und Löschen von Tabellen

Hinweis:

Da in Access-SQL einige Definitionsbefehle nicht zur Verfügung stehen, wird die Erzeugung von Datenbanken und Tabellen anhand des Datenbanksystems MariaDB erläutert. Die Schreibweise unterscheidet sich zum Teil auch durch die Begrenzung von Tabellennamen mit oder ohne Hochkommata.

Erstellen einer Datenbank

Mit dem Befehl `CREATE DATABASE` werden Datenbanken erzeugt.

Beispiel:

Erstellen Sie eine Datenbank mit dem Namen `mydb`.

Lösung:

```
CREATE DATABASE IF NOT EXISTS 'mydb';
USE 'mydb';
```

Der Befehl erzeugt die Datenbank `mydb`. Durch die Ergänzung wird der Abbruch und eine Fehlermeldung verhindert, falls die Datenbank bereits existiert.

Der Befehl `USE 'mydb'`; stellt sicher, dass die nachfolgenden Anweisungen innerhalb dieser Datenbank durchgeführt werden.

Erstellen von Tabellen

Mit dem Befehl `CREATE TABLE 'tabellenname'` werden Tabellen einer Datenbank erzeugt.

Die allgemeine Syntax lautet:

```
CREATE TABLE Tabellenname (
    Spalte_1 Datentyp_für_Spalte_1,
    Spalte_2 Datentyp_für_Spalte_2,
    ...);
```

Beispiel:

Es wird eine neue Tabelle `orte` erzeugt.

Lösung:

```
CREATE TABLE IF NOT EXISTS 'orte' (
    'Ortplz' INT(5) NOT NULL,
    'Ortname' VARCHAR(20) DEFAULT NULL,
) ENGINE=InnoDB;
```

Das erste Datenfeld heißt `Ortplz` und ist vom Typ `INTEGER`. Es kann ganze Zahlen mit einer Länge von vier Bytes speichern. Da anhand der Einträge im Feld `Ortplz` die einzelnen Datensätze identifiziert werden sollen, wird zusätzlich festgelegt, dass das Feld immer einen Wert enthalten muss (`NOT NULL`).

Das zweite Feld `Ortname` hat den Datentyp `variables Zeichenfeld` mit maximal 20 Zeichen und den Standardwert `NULL`, falls keine Eingabe erfolgt.

Mögliche Parameter der Felddeklaration sind:

NULL	Dieser Parameter legt fest, dass das Datenfeld standardmäßig keinen Wert (auch nicht 0 oder eine leere Zeichenkette) enthält und ein Datensatz in diesem Feld auch keinen Wert enthalten muss.
NOT NULL	Dieser Parameter erzwingt die Eingabe eines Wertes für das entsprechende Datenfeld. Die Angabe <code>NOT NULL</code> ist für Schlüsselfelder verpflichtend.
DEFAULT standardwert	Der Parameter <code>DEFAULT</code> definiert einen Standardwert für das Datenfeld. Erhält dieses Datenfeld bei der Eingabe der Daten keinen Wert, wird der Standardwert verwendet.
AUTO_INCREMENT (MariaDB)	Der Wert dieses Datenfeldes wird automatisch beim Anlegen eines neuen Datensatzes aus dem Wert des Datenfeldes des vorherigen Datensatzes plus eins errechnet. Dieser Wert kann vom Benutzer nicht geändert werden. Diese Einstellung ist besonders für den Primärschlüssel empfehlenswert, da dadurch automatisch ein eindeutiger Schlüsselwert erzeugt wird
SERIAL (PostgreSQL)	

Festlegen eines Primärschlüssels

Mit der Deklaration `PRIMARY KEY` wird das Datenfeld `Ortplz` als Primärschlüssel festgelegt. Der Primärschlüssel ermöglicht es, einen Datensatz eindeutig zu identifizieren. Dieses Datenfeld darf nicht leer bleiben und muss für jeden Datensatz unterschiedliche Werte enthalten.

Mit dem Befehl `ENGINE=InnoDB` wird der Typ der Datenbank-Speicherengine festgelegt. InnoDB erlaubt die Verwaltung großer Datenmengen und die problemlose Indizierung und Zuweisung von Fremdschlüsseln.

Hinweis:

Primärschlüsselfelder dürfen nicht leer sein (`NOT NULL`) und müssen stets eindeutig sein (z. B. Datentyp `INTEGER` mit `AUTO_INCREMENT`).

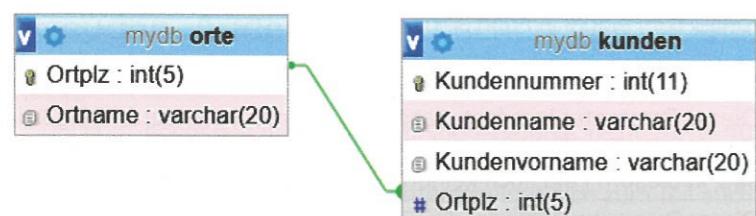
Beispiel:

Erzeugen Sie eine weitere Tabelle `kunden`.

Lösung:

```
CREATE TABLE IF NOT EXISTS 'mydb'.'kunden' (
    'Kundennummer' INT NOT NULL AUTO_INCREMENT,
    'Kundenname' VARCHAR(20) NULL,
    'Kundenvorname' VARCHAR(20) NULL,
    'Ortplz' INT NOT NULL,
    PRIMARY KEY ('Kundennummer'),
    CONSTRAINT 'kunden_fk_1' FOREIGN KEY ('Ortplz')
        REFERENCES 'orte'('Ortplz') ON UPDATE CASCADE)
    ENGINE=InnoDB;
```

Die Tabelle `Kunden` enthält das Feld `Kundennummer`, das nicht leer ("`NOT NULL`", gesprochen „nall“) sein darf und vom Datenbanksystem erzeugt und fortlaufend hochgezählt wird (`AUTO_INCREMENT`). Dieses Feld wird als Primärschlüsselfeld zur Kennung der einzelnen Datensätze festgelegt.



Festlegen eines Fremdschlüssels

Der Ort in der Tabelle kunden wird durch das Feld Ortplz eindeutig bestimmt. Die Postleitzahl Ortplz kann in der Tabelle orte nachgeschlagen werden. Um dies dem Datenbanksystem anzuzeigen, wird in der Tabelle kunden ein Fremdschlüssel deklariert, der auf den Primärschlüssel der Tabelle orte verweist (referenziert).

Diese Beschränkung (CONSTRAINT) definiert diesen Fremdschlüssel (FOREIGN KEY) mit Verweis (REFERENCES) auf das Feld 'orte'('Ortplz'). Dadurch wird die Tabelle kunden zur Child-Tabelle, die Tabelle orte zur Parents-Tabelle.

Das Feld Ortplz in der Tabelle kunden kann nur noch Werte annehmen, die in der Parents-Tabelle orte im Feld Ortplz bereits angelegt sind.

Beispiel:

Geben sie in die Tabelle kunden die folgenden Werte ein:

Kundename	Mayer
Kundenvorname	Hans
Ortplz	89077

Ergebnis:

Das Datenbanksystem bricht die Aktion mit einer Fehlermeldung ab:



Die Ursache liegt in einer Verletzung der referentiellen Integrität.

Zunächst muss in der Parents-Tabelle orte der Wert 89077 eingetragen und einem Ortnamen zugeordnet werden. Erst dann kann in der Child-Tabelle kunden der Wert 89077 als Ortplz verwendet werden.

Diese Regel der Reihenfolge ist beim Befüllen der Tabellen mit Daten unbedingt zu beachten.

Die Option ON UPDATE CASCADE bewirkt, dass eine Änderung (ON UPDATE) des Primärschlüsselfeldes in der Parents-Tabelle orte weitergegeben wird in die untergeordnete Tabelle kunden. Wird also eine Postleitzahl in der Tabelle orte geändert, so wird diese Änderung auch in der Tabelle kunden durchgeführt. Ob dies sinnvoll ist muss der Designer der Datenbank bereits beim Entwurf berücksichtigen.

Fremdschlüssel dienen der referentiellen Integrität: die Datensätze in verschiedenen Tabellen müssen dauerhaft zueinander passen.

Einem Kunden muss genau eine Postleitzahl zugewiesen werden. Diese Postleitzahl muss in der Datenbank existieren.

Mögliche CONSTRAINT-Bedingungen für Fremdschlüsselfelder sind:

NO ACTION	Alle Änderungen werden verweigert.
CASCADE	Eine Änderung des Wertes in der Primärtabelle wird an die Detailtabelle weitergegeben.
RESTRICT	Eine Änderung wird sowohl in der Primärtabelle als auch in der Detailtabelle verweigert.
SET NULL	Eine Änderung oder Löschung des Wertes in der Primärtabelle bewirkt, dass der Wert in der Detailtabelle auf NULL gesetzt wird.
SET DEFAULT	Eine Änderung oder Löschung des Wertes in der Primärtabelle bewirkt, dass der Wert in der Detailtabelle auf einen Standardwert gesetzt wird.

Diese Bedingungen können sowohl für Änderungsversuche (ON UPDATE) der Parents-Werte als auch für Löschversuche (ON DELETE) festgelegt werden.

Beispiel:

```
CONSTRAINT 'kunden_fk_1' FOREIGN KEY ('Ortplz')
  REFERENCES 'orte'('Ortplz')
  ON UPDATE CASCADE
  ON DELETE RESTRICT)
```

Ergebnis:

Das Datenbanksystem gibt eine Änderung (ON UPDATE) eines Wertes Ortplz in der Tabelle orte an alle betroffenen Felder der Tabelle kunden weiter.

Soll ein Wert Ortplz in der Tabelle orte gelöscht werden (ON DELETE) und es existiert in der Tabelle kunden ein Datensatz mit diesem Wert, so wird der Löschevorgang sowohl für die Parents-Tabelle als auch für die Child-Tabelle verweigert.

Aufgabe:

Bewerten Sie die Gefahr der folgenden Fremdschlüssel-Deklaration. Was geschieht, wenn versehentlich die Postleitzahl 89077 in der Parents-Tabelle gelöscht wird.

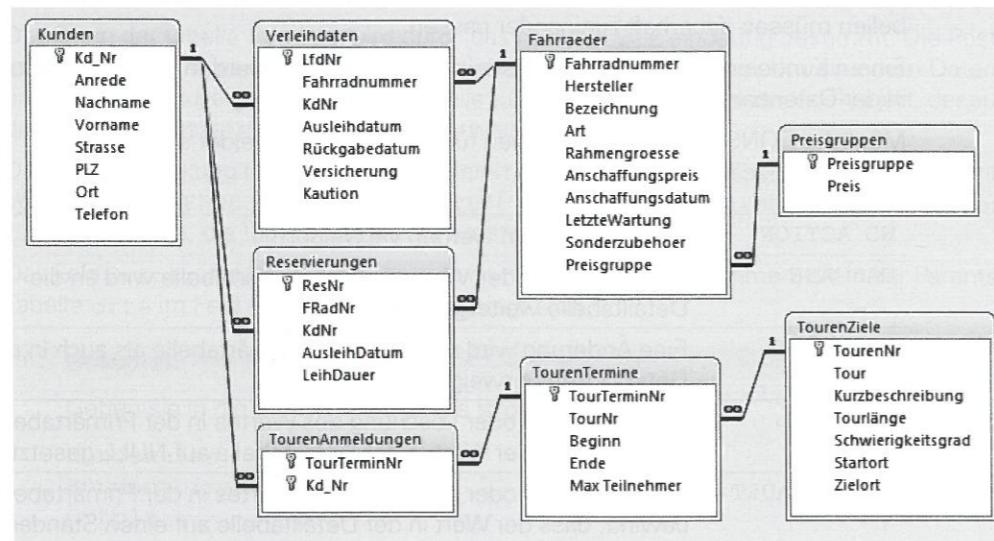
```
CONSTRAINT 'kunden_fk_1' FOREIGN KEY ('Ortplz')
  REFERENCES 'orte'('Ortplz')
  ON UPDATE CASCADE
  ON DELETE CASCADE)
```

6.3 Auswahlabfragen mit SELECT

Für **Auswahlabfragen** steht der Befehl SELECT zur Verfügung. Dieser Befehl besitzt eine umfangreiche Syntax und ist dadurch vielseitig einsetzbar.

Hinweise:

Die folgenden Beispiele beziehen sich auf die Datenbank „Faradiso“, die einen Fahrradverleih abbildet.



Dabei werden neben den Stammdaten der Kunden und der Fahrräder sowohl Verleihdaten als auch Reservierungsdaten aufgenommen.

Zusätzlich werden Radtouren für verschiedene Ziele angeboten, zu denen sich die Kunden anmelden können.

6.3.1 Eingrenzen von Auswahlabfragen mit Bedingungen

Beispiel:

Eine SQL-Abfrage soll alle Fahrräder mit einer Preisgruppe kleiner als 5 auswählen.

Lösung:

```
SELECT Fahrradnummer, Hersteller, Bezeichnung, Preisgruppe
FROM Fahrraeder
WHERE Preisgruppe < 5;
```

Nach dem Schlüsselwort SELECT werden die anzuzeigenden Spalten, durch Kommas getrennt, festgelegt, z. B. Fahrradnummer, Hersteller, Bezeichnung, Preisgruppe. Die Ausgabe erfolgt dabei in der Reihenfolge hinter der SELECT-Anweisung. Zu beachten ist die gleiche Schreibweise der Feldnamen wie in den Tabellen, da sie sonst nicht erkannt werden. Auch werden bestimmte Sonderzeichen wie der Bindestrich „-“ z. B. als Rechenanweisung minus interpretiert.

Hinweis:

Sonderzeichen und Leerzeichen verursachen in Tabellennamen und Feldnamen Ausführungsfehler.

Das Zeichen * kann als Platzhalter für sämtliche Felder einer Tabelle eingesetzt werden, z. B. zeigt die Anweisung

```
SELECT * FROM Fahrraeder;
alle Datensätze der Tabelle Fahrraeder an.
```

Die Tabelle, mit der gearbeitet werden soll, z. B. Fahrraeder, wird nach dem Schlüsselwort FROM (= von, aus) angegeben. Werden mehrere Tabellen verwendet, so werden sie mit Kommas getrennt. Nach dem reservierten Wort WHERE (= wo) steht eine Bedingung, mit der die Auswahl der angezeigten Datensätze eingegrenzt wird. Sie besteht in der Regel aus einem Spaltennamen, gefolgt von einem Vergleichsoperator und einem Vergleichswert.

Hinweis:

WHERE-Bedingungen müssen immer aus einem logischen Vergleich bestehen, der mit „true“ oder „false“ beantwortet werden kann.

Im Beispiel wird die Bedingung Preisgruppe < 5 verwendet. Das Datenbanksystem prüft bei jedem Datensatz der Tabelle Fahrraeder, ob für ihn diese Bedingung „true“ oder „false“ erfüllt ist. Als Ergebnis der Abfrage werden die Datensätze, für die Preisgruppe < 5 wahr (true) ist, am Bildschirm ausgegeben.

Die allgemeine Syntax der WHERE-Bedingung lautet:

```
... WHERE [Spaltenname] [Vergleichsoperator] [Vergleichswert]
z. B.
... WHERE Preisgruppe
```

Operator	Beispiel	Bedeutung, Wirkung
=	Kd_Nr = 24	gleich
<	Preisgruppe < 5	kleiner als
>		größer als
<>		ungleich
<=		kleiner gleich
>=		größer gleich
LIKE	Ort LIKE 'U%'	Vergleicht das Feld Ort mit dem Textmuster; es werden alle Orte berücksichtigt, die mit U beginnen.
AND	PLZ = 89077 AND Ort = 'Ulm'	Datensatz wird nur berücksichtigt, wenn Bedingung 1 UND Bedingung 2 erfüllt ist
OR	Nr > 200 OR Nr < 100	Datensatz wird berücksichtigt, wenn Bedingung 1 ODER Bedingung 2 erfüllt ist
NOT	NOT (PLZ = 89077)	Datensatz wird berücksichtigt, wenn die Bedingung nicht erfüllt ist

6.3.2 Darstellung von Feldinhalten in WHERE-Bedingungen

Inhalte von Zahlenfeldern, auch Währungsfelder, werden ohne weitere Kennzeichnung verwendet, z. B. Preis = 150.

Inhalte von Textfeldern werden innerhalb der WHERE-Bedingung in Hochkommas eingeschlossen, z. B. Wohnort = 'Stuttgart'.

Hinweis:

Bei Vergleichen von Textfeldern ist der Operator LIKE statt = zu verwenden.

Zur Definition von Textmustern können Platzhalter (Wildcards) verwendet werden:

Platzhalter	Erläuterung
%	Platzhalter für beliebige Zeichen in beliebiger Anzahl (einschließlich der Anzahl 0).
_ (under-score)	Platzhalter für genau ein beliebiges Zeichen.

Beispiele für den Vergleich eines Spaltennamens mit Textmustern für Maria DB und PostgreSQL:

Ort LIKE 'U%'	Vergleicht das Feld Ort mit einem Textmuster. Im Beispiel werden alle Orte berücksichtigt, die mit U beginnen und beliebig lang sind.
Ort LIKE '%au'	Berücksichtigt alle Orte, die mit beliebiger Zeichenfolge beginnen und mit den Buchstaben au enden, z. B. Blumenau.
Ort LIKE 'L%au'	Ergebnis: Orte, die mit L beginnen und au enden, z. B. Lindau, Langenau.
Ort LIKE 'L__au'	Ergebnis: Orte, die mit L beginnen und au enden und dazwischen genau drei beliebige Buchstaben enthalten, z. B. Lindau (nicht aber Langenau).
Ort LIKE '%au%'	Ergebnis: Orte, in deren Namen die Buchstabenfolge au an beliebiger Stelle vorkommt, z. B. Blaubeuren, Laupheim.

Beispiel:

Mit einer SELECT-Anweisung werden alle Kunden ausgegeben, deren Wohnort mit dem Buchstaben U beginnt.

Lösung:

```
SELECT Nachname, Vorname, Straße, PLZ, Ort, Telefon
FROM Kunden
WHERE Ort LIKE 'U*';
```

Beim Vergleich eines Textfeldes mit einem Textmuster stellt der Operator LIKE klar, dass das Textfeld mit einem Textmuster verglichen werden soll und somit im Beispiel alle Orte mit dem Anfangsbuchstaben U ausgegeben werden sollen.

Um nach Textmustern zu suchen, in denen die Zeichen % oder _ vorkommen (z. B. Passwort LIKE 'password%'), müssen diese als tatsächliche Zeichen kenntlich gemacht werden. Dies geschieht durch einen vorgestellten Escape-Strich \ (Backslash).

Beispiel:

```
SELECT *
FROM User
WHERE password LIKE '%\%';
```

Das erste Zeichen % wird als Wildcard für beliebig viele Zeichen interpretiert. Durch den Backslash \ wird das nächste Zeichen als Prozentzeichen interpretiert. Als gefunden werden somit alle Passwörter angezeigt, die als letztes Zeichen das Zeichen % enthalten, z. B. 'password%' und 'HjuZtJ_k556L%.'

Beispiel:

Eine Abfrage soll alle Passwörter ausgeben, die mit dem Zeichen 1 beginnen und mit dem Zeichen % enden.

Lösung:

```
SELECT *
FROM User
WHERE password LIKE '1%\%';
```

Platzhalter in Jet-SQL (MS Access):

In Access stehen folgende Platzhalter zur Verfügung:

Zeichen	Beschreibung	Beispiel
*	Dieses Zeichen entspricht einer beliebigen Anzahl von Zeichen. Sie können den Stern (*) an einer beliebigen Stelle in einer Zeichenfolge verwenden.	wo* findet wohin und woher, aber nicht warum.
?	Dieses Zeichen entspricht einem einzelnen Buchstaben an einer bestimmten Stelle.	B?llen findet Bullen, Ballen und Bellen.
#	Dieses Zeichen entspricht einer einzelnen Ziffer.	1#3 findet 103, 113 und 123, nicht aber 1223
[]	Für die Zeichen in der Klammer wird nach einer Entsprechung gesucht.	B[ae]llen findet Ballen und Bellen, aber nicht Bullen.
!	Mit diesem Zeichen werden Zeichen in den Klammern ausgeschlossen.	B[!ae]llen findet Bollen und Bullen, aber nicht Ballen und Bellen.
-	Dieses Zeichen bestimmt einen Zeichenbereich. Er muss in aufsteigender Reihenfolge angegeben werden (A-Z, nicht Z-A).	b[a-c]d findet bad, bbd und bcd.

Beispiel:

Es sollen Daten zurückgegeben werden, die mit dem Buchstaben „P“ beginnen, gefolgt von einem beliebigen Buchstaben von A bis F und drei Ziffern.

Lösung:

```
Like 'P[A-F]###'
```

Um nach Sonderzeichen zu suchen, werden sie im Textmuster in eckige Klammern eingeschlossen.

Beispiel:

Das Textmuster 'a[*]a' findet die Zeichenfolge 'a*a', nicht aber 'aaa'

Die Argumente können kombiniert werden:

Beispiel:

Analysieren Sie den folgenden LIKE-Operator:

```
... LIKE 'a[ !b-m] #'
```

Ergebnis:

Das erste Zeichen muss 'a' oder 'A' sein, das zweite Zeichen ist beliebig außer der Buchstaben b bis m, das dritte Zeichen muss eine Ziffer sein. Das Textmuster findet z. B. 'An9', 'az0', 'a99', nicht aber 'abc' oder 'aj0'.

Datumsfelder werden in Datenbanksystemen unterschiedlich eingegeben und angezeigt.

Datenbanksystem	Format eines Datumsfeldes
Oracle, Informix, DB2	'2020/10/03'
Access	#03.10.2020# oder #2020/10/03#
MySQL, Maria DB	'2020-10-03'

Obwohl es meist Möglichkeiten gibt, zwischen deutscher und amerikanischer Schreibweise umzuwandeln, wird die amerikanische Schreibweise (Jahr/Monat/Tag), z. B. #2020/10/04#, bevorzugt.

6.3.3 DISTINCT

Das Schlüsselwort DISTINCT (= verschieden) unterdrückt in der Abfrage doppelte Datensätze (Duplikate). Sollen z. B. nur die Namen der Hersteller ausgegeben werden, von denen Fahrräder bezogen wurden, so wird mit

```
SELECT DISTINCT Hersteller FROM Fahrraeder
jeder betroffene Hersteller nur einmal angezeigt.
```

6.3.4 Der Operator BETWEEN

Der Operator BETWEEN (= zwischen) wird zur Formulierung von Bereichsabfragen eingesetzt. Er kann bei Textfeldern, Datumsfeldern und bei numerischen Feldern eingesetzt werden. BETWEEN wählt die Daten zwischen einem unteren und einem oberen Grenzwert aus. Die Grenzwerte sind in der Auswahl mit enthalten.

Beispiel:

Das Ergebnis einer SQL-Anweisung listet alle Fahrräder auf, welche im Jahr 2018 angeschafft wurden.

Lösung (in Access):

```
SELECT Fahrradnummer, Hersteller, Bezeichnung, Anschaffungsdatum,
Anschaffungspreis
FROM Fahrraeder
WHERE Anschaffungsdatum BETWEEN #2018/01/01# AND
#2018/12/31#;
```

Ergebnis:

Fahrradnummer	Hersteller	Bezeichnung	Anschaffungsdatum	Anschaffungspreis
3	Panasonic	FirstClass	17.01.2018	1.960,00 €
4	Miyata	Devant	17.03.2018	2.800,00 €
6	Scott	Executive	18.03.2018	2.350,00 €
19	Hirsch	Klettergemse	16.06.2018	2.400,00 €
24	Ivecu	Stralys	31.12.2018	1.270,00 €

6.3.5 Der Operator IN

Der Operator IN dient zum Vergleich eines Feldinhaltes mit einer Liste von möglichen Inhalten. Um alle Kunden aus dem Großraum Stuttgart zu erfassen, kann z. B. die folgende Bedingung verwendet werden:

```
... WHERE Ort IN ('Stuttgart', 'Esslingen', 'Fellbach', 'Waiblingen');
```

Beispiel:

Eine Abfrage sucht alle Kunden aus den Hauptstädten der baden-württembergischen Regierungsbezirke aus.

Lösung:

```
SELECT Nachname, Vorname, Strasse, PLZ, Ort
FROM Kunden
WHERE Ort IN ('Stuttgart', 'Karlsruhe', 'Tübingen', 'Freiburg');
```

Hinweis:

Der Operator IN kann auf Text-, Datums- und numerische Felder angewendet werden.

6.3.6 Umgang mit NULL-Werten

Spalten, in denen keine Werte eingegeben worden sind, haben den Wert NULL (sprich: null). Dem Wert NULL entspricht weder die Zahl 0 noch eine Zeichenfolge mit Leerzeichen. Rechenoperationen mit einem Feld, das einen NULL-Wert enthält, ergeben als Ergebnis den Wert NULL. Mit der Eigenschaft NOT NULL werden Felder mit einem Eintrag gesucht.

Beispiel:

Die Abfrage

```
SELECT Nachname, Vorname, Strasse, PLZ, Ort, Email
FROM Kunden
WHERE Email IS NOT NULL;
wählt alle Kunden mit E-Mail-Adresse aus.
```

Wenn ein Feld einer Tabelle keinen NULL-Wert enthalten darf, also nicht leer sein darf, so ist dies bereits beim Entwurf der Tabelle vorzusehen. Primärschlüsselfelder dürfen z. B. nicht leer sein. Auch für andere Felder, z. B. Nachname oder Postleitzahl, kann es zweckmäßig sein, NULL-Werte nicht zuzulassen.

6.3.7 Daten sortieren

Mit der Anweisung ORDER BY können Ausgaben nach Feldinhalten sortiert werden. Dabei wird die Sortierreihenfolge mit dem Zusatz ASC (ascending = aufsteigend) oder DESC (descending = absteigend) festgelegt. Standardmäßig (ohne Zusatz) wird aufsteigend sortiert.

Beispiel:

Die Kunden werden alphabetisch angezeigt:

```
SELECT *
FROM Kunden
ORDER BY Nachname, Vorname ASC
```

KdNr	Anrede	Nachname	Vorname	Strasse	PLZ	Ort	Telefon
16	Frau	Adler	Astrid	Günthersburgstr. 56	40213	Düsseldorf	0211 543210
25	Herr	Amini	Hans	Buchenlandweg 120	89075	Ulm	0731 4538254
23	Herr	Bader	Michael	Blautalallee 235	89075	Ulm	4567890
6	Frau	Müller	Andrea	Adelheidstr. 13	88446	Biberach	07364/894623
3	Herr	Müller	Hans	Friedberger Landstr. 204	89077	Augsburg	0821/434343
2	Frau	Müller	Herta	Ringelstr. 2	86416	Krumbach	08282/465432
5	Frau	Winter	Susanne	Am Waldrand 1	89077	Ulm	0731/4892
8	Herr	Zwiebel	Karl	Hügelstr. 123	89123	Elchingen	07321/745645

Werden zwei Feldinhalte als Sortierkriterium angegeben, dann wird nach dem ersten (hier Nachname) sortiert, bei gleichen Feldinhalten (z. B. 'Müller') zusätzlich nach dem zweiten Feld (hier Vorname).

Folgt nach dem Wort `SELECT` das Symbol `*`, so werden alle Felder zur Anzeige ausgewählt.

Die Sortierreihenfolge kann für zwei Kriterien unterschiedlich sein.

Beispiel:

Die Fahrräder sollen nach ihrem Anschaffungsdatum absteigend vom neuesten zum ältesten sortiert werden. Bei gleichem Anschaffungsdatum soll aufsteigend nach Hersteller sortiert werden:

```
SELECT *
FROM Fahrraeder
ORDER BY Anschaffungsdatum DESC, Hersteller;
```

Fahrrad-number	Hersteller	Bezeichnung	Art	Anschaffungspreis	Anschaffungsdatum	Letzte Wartung
23	Hercules	Davos	Tourenrad	890,00 €	18.08.2020	08.09.2021
24	Peugeot	Stralis	Tourenrad	1.270,00 €	18.08.2020	01.01.2021
26	Yamaha	Sutra	Rennrad	4.560,00 €	18.08.2020	01.01.2021
7	Hercules	GoClimb	Moutain-Bike	850,00 €	26.04.2019	08.01.2020
8	Panasonic	EasyRide	Tourenrad	960,00 €	26.04.2019	08.05.2019
18	Hirsch	Schneller Hirsch	Rennrad	2.100,00 €	13.01.2019	19.06.2019
20	Panasonic	FirstClass	Tourenrad	1.325,00 €	05.10.2008	10.03.2019

6.3.8 Abfrage-Ergebnisse einschränken

Mit dem Schlüsselwort `TOP` werden Abfrage-Ergebnisse auf eine bestimmte Anzahl von Datensätzen beschränkt. `TOP 3` z. B. gibt nur die ersten 3 Datensätze einer Abfrage aus.

Beispiel:

Es sollen die 5 Fahrräder ausgegeben werden mit den teuersten Anschaffungspreisen.

Lösung:

```
SELECT TOP 5 *
FROM Fahrraeder
ORDER BY Anschaffungspreis DESC;
```

Ergebnis:

Fahrrad-number	Hersteller	Bezeichnung	Art	Anschaffungspreis	Anschaffungsdatum	Letzte Wartung
5	Mercier	Excalibur Ultra Light	Rennrad	3.750,00 €	27.04.2016	08.01.2020
15	Techno-bike	Supertandem	Tourenrad	3.200,00 €	28.12.2016	03.06.2020
17	Staiger	Supertandem		2.950,00 €	14.01.2016	19.06.2020
13	Systemo	Hurrican	Moutain-Bike	2.665,00 €	30.07.2016	12.04.2019
4	Miyata	Devant	Tourenrad	2.585,00 €	17.03.2018	29.11.2020

Da die Datensätze innerhalb einer Tabelle in der Regel unsortiert gespeichert sind, wird die Abfrage mit der Sortierung `ORDER BY Anschaffungspreis DESC` ergänzt.

Mit der Funktion `TOP` kann auch ein bestimmter Prozentsatz der Datensätze ausgegeben werden. Die Syntax lautet z. B. `SELECT TOP 10 percent ...`

Aufgabe

Schreiben Sie eine Abfrage, um 10 Prozent der neuesten Fahrräder auszugeben.

Lösung:

```
SELECT TOP 10 percent *
FROM Fahrraeder
ORDER BY Anschaffungsdatum DESC;
```

Die Sortieranweisung ist auch hier `DESC`, da neuere Datumswerte als größer betrachtet werden als frühere Werte.

6.3.9 Funktionen in SELECT-Abfragen

Aggregatfunktionen

Aggregatfunktionen (auch: Gruppenfunktionen) werten eine oder mehrere Spalten einer Tabelle nach bestimmten Kriterien aus. Das Ergebnis, z. B. die Summe einer Spalte, wird in einem Feld ausgegeben.

Funktion	Ergebnis
MIN(Spaltenname)	Minimalwert der Feldinhalte einer Spalte
MAX(Spaltenname)	Maximalwert
COUNT(Spaltenname)	Anzahl der vorhandenen Einträge in einer Spalte
COUNT(*)	Anzahl aller Datensätze
SUM(Spaltenname)	Summe der Feldinhalte einer Spalte
AVG(Spaltenname)	Arithmetischer Mittelwert der Feldinhalte, AVG von engl.: average = Durchschnitt, Mittelwert

Beispiel

Eine SQL-Anweisung gibt die Summe der Anschaffungspreise aller Fahrräder aus.

Lösung:

```
SELECT SUM(Anschaffungspreis) AS Summe
FROM Fahrraeder;
```

Ergebnis:

Summe
41.337,00 €

Um eine Spalte bei der Ausgabe mit einer gewünschten Überschrift zu versehen, wird die Anweisung `AS (= wie)`, gefolgt von der neuen Überschrift, z. B. `Summe`, verwendet.

Das Ergebnis einer Aggregatfunktion besteht in der Regel aus einem einzigen Feld. Der Versuch, mit dem gleichen Befehl die Inhalte einer weiteren Spalte, z. B. die Fahrradnummern, ausgeben zu wollen, führt zu einer Fehlermeldung.

Hinweis:

In einer einfachen SELECT-Anweisung mit Aggregatfunktion können keine weiteren Spaltenwerte angezeigt werden.

Die Funktion `COUNT (Spaltenname)` zählt die Anzahl der Felder in der Spalte, die nicht leer sind. Die Anweisung `SELECT COUNT (Email) FROM Kunden` ermittelt die Anzahl der Kunden, für die eine E-Mail-Adresse eingetragen ist.

Durch die Anweisung `SELECT COUNT (*) FROM Kunden` werden alle Datensätze der Tabelle Kunden gezählt.

Beispiel:

In der Tabelle Fahrraeder soll die Anzahl der verschiedenen Hersteller gezählt werden.

Lösung in Standard-SQL:

```
SELECT COUNT(DISTINCT 'Hersteller') AS 'Anzahl Hersteller'
FROM 'fahrraeder'
```

Ergebnis:**Anzahl Hersteller**

13

Das Schlüsselwort DISTINCT unterdrückt die doppelte Zählung der einzelnen Hersteller.

Hinweis:

Jet-SQL (Access) lässt die direkte Kombination von COUNT() und DISTINCT nicht zu. Deshalb muss man sich mit einer geschachtelten Lösung behelfen:

```
SELECT COUNT(*) AS 'Anzahl Hersteller'
```

```
FROM
```

```
(
```

```
SELECT DISTINCT Hersteller FROM Fahrraeder
```

```
);
```

Mit SELECT DISTINCT ... werden zunächst alle Namen der Hersteller ohne Duplikate gesucht. Die übergeordnete COUNT-Funktion zählt anschließend die Anzahl der Ergebnisse dieser Unterabfrage.

Hinweise:

Um die gesamte Anzahl aller Datensätze einer Tabelle zu erfassen, ist die Suche mit COUNT (*) zu verwenden.

Die Funktionen SUM () und AVG () können nur bei numerischen Spalteninhalten angewendet werden.

Die Funktionen MIN () und MAX () können auch bei Text- und Datumsfeldern verwendet werden, wobei Textfelder nach dem ASCII-Code behandelt werden: A ist „kleiner“ als Z und Kleinbuchstaben sind „größer“ als Großbuchstaben, allerdings „kleiner“ als Sonderzeichen. Ein späteres Datum ist „größer“ als ein früheres.

Beispiel:

Die folgende SQL-Anweisung ermittelt das Anschaffungsdatum des neuesten Fahrrades.

```
SELECT MAX(Anschaffungsdatum)
FROM Fahrraeder;
```

Rechenoperationen

Mit den Werten aus numerischen Spalten können Rechenoperationen wie z. B. Addition, Subtraktion, Multiplikation und Division durchgeführt werden.

Dies gilt auch für Datumswerte, da diese als Integerzahlen (ganze Zahlen) abgelegt sind.

Datumsfunktionen

Mit Datumsfunktionen lassen sich Datumswerte verarbeiten. Z. B. kann die Jahreszahl aus einem Datum herausgefiltert werden oder das aktuelle Datum dynamisch verarbeitet werden.

Beispiel:

Um alle Fahrräder aufzulisten, deren Wartungsdatum länger als 100 Tage zurückliegt, ist die folgende Anweisung notwendig:

```
SELECT Fahrradnummer, Bezeichnung, LetzteWartung
FROM Fahrraeder
WHERE LetzteWartung < DATE() - 100;
```

Die Funktion DATE () liefert das Systemdatum. Von diesem Wert wird die Zahl 100 subtrahiert und das Ergebnis jeweils mit dem Inhalt des Feldes LetzteWartung verglichen. Alle Datensätze mit einem kleineren Wert werden ausgegeben.

Hinweis:

Datumsfunktionen gehören nicht zum SQL-Standard, sind jedoch in nahezu allen Datenbanksystemen vorhanden. Sie unterscheiden sich jedoch zum Teil erheblich. Sie sind sehr vom verwendeten Datenbanksystem abhängig. Gegebenenfalls ist im jeweiligen Handbuch bzw. der Hilfefunktion nachzuschlagen.

In Access gibt es neben der Funktion DATE(), die das aktuelle Datum liefert, weitere Datumsfunktionen:

Funktion	Ergebnis
DATE()	Aktuelles Datum
DAY(datum)	Tag vom Feldinhalt 'datum' als Zahl zwischen 1 und 31
MONTH(datum)	Monat als Zahl zwischen 1 und 12
YEAR(datum)	Jahr als vierstellige Zahl
ISDATE(datum)	Prüft, ob Datumsangabe korrekt ist
FORMAT(ausdruck, 'format')	Formatiert das Datum in ausdruck nach Vorgabe: D,DD: Tageszahl, DDD, DDDD: Wochentag, M, MM: Monatszahl, MMM, MMMM: Monatsname YY, YYYY: Jahreszahl

Mit der Anweisung FORMAT (ausdruck, 'format') können z. B. Datumswerte formatiert ausgegeben werden.

Beispiel:

Das Rückgabedatum der Felder der Tabelle Verleihdaten soll in der Form „Freitag, 26. April 2019“ ausgegeben und umbenannt werden.

```
SELECT FORMAT(Rückgabedatum, 'DDDD, DD. MMMM YYYY') AS Rechnungsdatum
FROM Verleihdaten;
```

Ergebnis:

Rechnungsdatum
Dienstag, 06. Februar 2018
Mittwoch, 04. Oktober 2019
Dienstag, 12. März 2021
Samstag, 27. April 2021

Formatierung von Währungsfeldern in Jet-SQL (MS Access):

Mit der Funktion FormatCurrency () können Währungsfelder in Access formatiert werden.

Die allgemeine Syntax lautet:

```
FormatCurrency (Feldname [, AnzStellenNachDezimal] [, FührendeNullAnzeigen] [, KlammernFürNegativeZahl] [, StellenGruppieren] )
```

Die eckigen Klammern sind nicht zu schreiben, sie zeigen optionale Argumente:

Argument	Beschreibung
Ausdruck	Erforderlich. Der zu formatierende Ausdruck.
AnzStellenNachDezimal	Optional. Numerischer Wert, der angibt, wie viele Stellen rechts vom Dezimaltrennzeichen angezeigt werden. Der Standardwert (-1) bedeutet, dass die Landes-/Regionaleinstellungen des Computers verwendet werden.
FührendeNullAnzeigen	Optional. Eine von drei Konstanten (siehe Tabelle unten), die angibt, ob für Dezimalzahlen <1 (z. B. 0,45) eine führende Null angezeigt werden soll oder nicht.
KlammernFürNegativeZahlen	Optional. Eine von drei Konstanten (siehe Tabelle unten), die angibt, ob negative Werte in Klammern gesetzt werden oder nicht.
StellenGruppieren	Optional. Eine von drei Konstanten (siehe Tabelle unten), die angibt, ob Zahlen mit dem Gruppentrennzeichen gruppiert werden, das in den Landes-/Regionaleinstellungen des Computers angegeben ist.

Die Argumente FührendeNullAnzeigen, KlammernFürNegativeZahlen und StellenGruppieren können die folgenden Werte annehmen:

Konstante	Wert	Beschreibung
vbTrue	-1	True
vbFalse	0	False
vbUseDefault	-2	Die Landes-/Regionaleinstellungen des Computers werden verwendet.

Beispiel:

Die Buchungen eines Kontos sollen mit 4 Nachkommastellen und bei Werten zwischen 0 und 1 mit führender Null angezeigt werden. Negative Zahlen sollen in Klammern gesetzt und die Ziffern landestypisch gruppiert werden.

Lösung:

```
SELECT BuchungsNr, FormatCurrency(Buchung, 4, -1, -1, -2)
FROM Konto;
```

Buchungs Nr	Buchung
1	1'525,0000 €
2	0,5901 €
3	(16,4500) €

Zeichenkettenfunktionen

Die verschiedenen Datenbanksysteme stellen unterschiedliche Funktionen für die Bearbeitung und Umwandlung von Zeichenketten zur Verfügung. Die Auswirkungen der entsprechenden Funktionen sind jedoch sehr ähnlich. So liefert im Datenbanksystem Oracle die Funktion ASCII(n) die Codenummer eines ASCII-Zeichens, während in Access die Funktion ASC(n) lautet und das Gleiche bewirkt.

Funktion	Wirkung, Beispiele
ASC(n)	Liefert die ASCII-Codenummer von n, ASC('M') → 77.
LCASE(string)	Wandelt um in Kleinbuchstaben, LCASE('Müller') → 'müller'
LEFT(string, längte)	Gibt linksbündige Teilzeichenkette aus, LEFT('Müller',2) → 'Mü'.
LEN(string)	Gibt die Zeichenanzahl aus, LEN('Müller') → 6.
RIGHT(string, längte)	Gibt rechtsbündige Teilzeichenkette aus, siehe LEFT().
UCASE(string)	Wandelt in Großbuchstaben um, UCASE('text') → 'TEXT'
&	Operator zur Verkettung (in Access) 'Hans' & ' ' & 'Müller' → 'Hans Müller'

Beispiel:

Unabhängig von der gespeicherten Schreibweise werden von der folgenden Anweisung die Nachnamen der Kunden so ausgegeben, dass der erste Buchstabe als Großbuchstabe erscheint, die folgenden Buchstaben jedoch in Kleinschrift.

```
SELECT Left(UCASE(Nachname), 1) &
LCASE(Right(Nachname, Len(Nachname) - 1)) AS ['Nachname']
FROM Kunden;
```

Die versehentliche Eingabe des Nachnamens „müller“ wird durch diese Anweisung umgewandelt in „Müller“.

Umwandlungsfunktionen

Mit Umwandlungsfunktionen werden z. B. Zeichenketten in Zahlen gewandelt und umgekehrt.

Beispiele sind:

Funktion	Wirkung, Beispiele
CDATE('datumsstring')	Zeichenkette, Zahl in DatumsTyp DATE CDATE('12.03.2022') → 12.03.2022
STR(zahl)	Zahl in Zeichenkette, STR(89077) → '89077'
VAL(string)	Zeichenkette in Zahl, VAL('34') → 34
CINT(zahl)	Wandelt um in INTEGER, runden auf CINT(45,8) → 46 CINT(-67,4) → -67

Die Funktion CDATE('datum') wandelt z. B. eine aus Zahlen bestehende Zeichenkette oder eine Ganzzahl in ein Datum um, soweit es vom ursprünglichen Typ her möglich ist.

Die Funktion CINT() runden auf den nächsthöheren Ganzahlwert auf und gibt dies als Ergebnis zurück.

Beispiel:

Die folgende SQL-Anweisung zeigt die Verleihpreise der einzelnen Fahrräder in der Tabelle Preisgruppen um 5% erhöht an, wobei sie auf volle €-Beträge aufgerundet werden.

```
SELECT CINT(Preis*1.05)
FROM Preisgruppen;
```

Mathematische und logische Funktionen

Um mathematische Aufgaben zu lösen, z. B. die Quadratwurzel eines Feldinhaltes zu bilden, werden folgende Funktionen verwendet:

Funktion	Ergebnis
ABS (zahl)	Absoluter Wert (ohne negatives Vorzeichen)
SQR (zahl)	Wurzel einer Zahl: SQR (16) = 4
RND ()	Zufallszahl zwischen 0 und 1
ISNULL (ausdruck)	Wert = 1, wenn z. B. Feldinhalt leer ist, Wert = 0, wenn der Feldinhalt nicht leer ist.
EXP (Wert)	Ergibt e^{Wert} mit der Eulerschen Zahl $e = 2.71828\dots$ als Basis.
LOG (Wert)	Der natürliche Logarithmus, d. h. $\ln(\text{EXP}(\text{Wert})) = \text{Wert}$.
LOG10 (Wert)	Logarithmus von der Zahl Wert zur Basis 10
SIGN (Wert)	Liefert -1 für negative Werte und 1 für positive Werte
MOD (Wert, Divisor)	Restwert der (ganzzahligen) Division von Wert/Divisor

Beispiel:

Aus einer Tabelle mit Leitungsquerschnitten werden die Durchmesser errechnet.

Lösung:

```
SELECT Flaeche, 2*SQR(Flaeche/(3.1415*2)) AS Durchmesser
FROM Leitung;
```

Die Funktion RND () erzeugt eine Zufallszahl als Gleitkommazahl >0 und <1, z. B. 0,3452419.

Die Funktion ISNULL (ausdruck) prüft, ob der Ausdruck, z. B. ein Feldinhalt, leer ist. Trifft dies zu, so liefert die Funktion den Wert 1 ('wahr'), sonst den Wert 0 ('falsch').

6.3.10 Gruppieren von Daten

Gruppieren bedeutet, die Daten nach Feldern mit gleichen Feldinhalten zusammenzufassen. Dies geschieht mit dem SQL-Ausdruck GROUP BY (= zusammenfassen aufgrund) gefolgt von einem Feldnamen.

Beispiel:

Die folgende Anweisung stellt das Einzugsgebiet des Unternehmens „Faradiso“ fest, indem der Wohnort und die Anzahl der Kunden aus den einzelnen Orten ausgegeben werden:

```
SELECT Ort, COUNT(KdNr) AS Anzahl
FROM Kunden
GROUP BY Ort;
```

Ort	Anzahl
Berlin	2
Biberach	2
Jettingen-Scheppach	2
Düsseldorf	35
Neckarsulm	1
Haan-Gruiten	12
Ulm	11
Zwickau	1
...	...

Zunächst wird durch den Ausdruck GROUP BY Ort aus allen Datensätzen, die den gleichen Eintrag im Feld Ort besitzen, jeweils eine Gruppe gebildet. Anschließend werden innerhalb jeder dieser Gruppen aufgrund der Funktion COUNT (Kd_Nr) die Anzahl der Datensätze anhand der Kd_Nr festgestellt.

Hinweis:

Alle Felder, die nach dem Schlüsselwort SELECT zusätzlich zur Aggregatfunktion aufgelistet werden, müssen auch hinter der Gruppierungsanweisung GROUP BY angegeben werden, da es sonst zu einer Fehlermeldung kommt.

Bedingungen für Gruppierungen – HAVING:

Die Anzeige der einzelnen Gruppen kann durch die Anweisung HAVING Bedingung zusätzlich von Bedingungen abhängig gemacht werden.

Beispiel:

```
SELECT Hersteller, SUM(Anschaffungspreis) AS Summe
FROM Fahrraeder
GROUP BY Hersteller
HAVING SUM(Anschaffungspreis) > 4000;
```

Hersteller	Summe
Hirsch	5.932,00 €
Panasonic	6.095,00 €

Die Anweisung HAVING SUM (Anschaffungspreis) >4000 bewirkt z. B., dass nur die Gruppen ausgegeben werden, deren aufsummierter Anschaffungspreis („Summe“) mehr als 4000 € beträgt.

Hinweis:

Die Bedingung nach dem Schlüsselwort HAVING muss immer auf die gesamte Gruppe anwendbar sein.

6.3.11 Abfragen über mehrere Tabellen (JOINS)

Joins (to join = verknüpfen) stellen logische Verbindungen zwischen mehreren Tabellen her. Um z.B. alle Verleihdaten mit den dazugehörigen Kundennamen auszugeben, müssen Felder der beiden Tabellen Kunden und Reservierungen angezeigt werden. Die Anweisung lautet dann:

```
SELECT KdNr, Kunden.Nachname, Kunden.Vorname,
       Reservierungen.AusleihDatum
  FROM Kunden, Reservierungen;
```

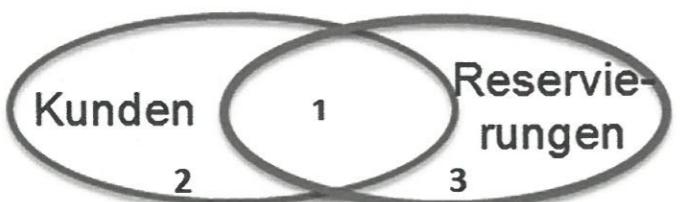
Das Ergebnis ist eine Ausgabe, die alle Kunden mit allen Verleihdaten kombiniert. Aus einer Tabelle mit 50 Einträgen und einer Tabelle mit 400 Einträgen wird somit eine Ergebnismenge mit 20'000 Einträgen erzeugt.

Die Anzahl der Einträge multipliziert sich mit der Anzahl der Datensätze jeder Tabelle, die man hinzufügt.

Arten von Joins

Am Beispiel der beiden Tabellen Kunden und Reservierungen werden die unterschiedlichen Joins erläutert. Es gibt

- Kunden, die Reservierungen in Auftrag gegeben haben (1),
- Kunden, die keine Reservierung beauftragt haben (2) und
- Reservierungen, die noch keinem Kunden zugeordnet sind (3).



Name	Darstellung	Erläuterung	Syntaxbeispiel
INNER JOIN, EQUI JOIN		Ausgabe von Kunden, die reserviert haben, mit den zugehörigen Reservierungen.	SELECT * FROM Kunden INNER JOIN Reservierungen ON Kunden.KdNr = Reservierungen.KdNr;
NATURAL JOIN		Ergebnis wie INNER JOIN	SELECT * FROM Kunden ,Reservierungen WHERE Kunden.KdNr = Reservierungen.KdNr;
LEFT JOIN,		Ausgabe aller Kunden und (falls vorhanden) die dazu gehörenden Reservierungen.	SELECT * FROM Kunden LEFT JOIN Reservierungen ON Kunden.KdNr = Reservierungen.KdNr;
RIGHT JOIN,		Ausgabe aller Reservierungen und – falls gespeichert – der entsprechenden Kunden.	SELECT * FROM Kunden RIGHT JOIN Reservierungen ON Kunden.KdNr = Reservierungen.KdNr;
LEFT OUTER JOIN		Ausgabe nur der Kunden, die nicht reserviert haben.	SELECT * FROM Kunden LEFT JOIN Reservierungen ON Kunden.KdNr = Reservierungen.KdNr WHERE Reservierungen.KdNr IS NULL;

Name	Darstellung	Erläuterung	Syntaxbeispiel
RIGHT OUTER JOIN		Ausgabe der Reservierungen, die keinem Kunden zugeordnet sind.	SELECT * FROM Kunden RIGHT JOIN Reservierungen ON Kunden.KdNr = Reservierungen.KdNr WHERE Reservierungen.KdNr IS NULL;
FULL OUTER JOIN		Ausgabe aller Kunden ohne Reservierung und aller nicht zugeordneten Reservierungen	SELECT * FROM Kunden FULL JOIN Reservierungen ON Kunden.KdNr = Reservierungen.KdNr WHERE Reservierungen.KdNr IS NULL; (Wird von ACCESS-SQL nicht unterstützt)

Inner-Joins, Equi-Joins, Natural Joins

Für Abfragen, die mehrere Tabellen betreffen, werden Verknüpfungen auf Gleichheit, Inner-Joins (auch: Equi-Joins equi = gleich, to join = verknüpfen;) verwendet. Im Beispiel dürfen nur die Kunden berücksichtigt werden, deren Kundennummer auch in der Tabelle Reservierungen aufgeführt ist. Dazu muss eine eindeutige Verknüpfungsvorschrift im SQL-Befehl definiert werden. Dies geschieht durch die Bedingung

WHERE Kunden.KdNr = Reservierungen.KdNr.

Die vollständige Anweisung lautet somit:

SELECT Kunden.KdNr, Kunden.Nachname, Kunden.Vorname, Reservierungen.AusleihDatum
FROM Kunden, Reservierungen

Um gleichnamige Felder in den verschiedenen Tabellen zu unterscheiden, wird der Tabellename, gefolgt von einem Punkt, dem Spaltennamen vorangestellt, z. B. Kunden.KdNr. Die Verknüpfung (Joins) mehrerer Tabellen werden in einer SQL-Abfrage z. B. in Form einer WHERE-Bedingung beschrieben.

Beispiel:

Alle Reservierungen mit den Kundendaten (Kundennummer, Nachname, Vorname) und den Fahrraddaten (Fahrradnummer, Hersteller) werden ausgegeben. Die Anweisung lautet:

```
SELECT K.KdNr, Nachname, Vorname, Ort,  
LeihDauer AS LDauer, F.Fahrradnummer AS FNr, Hersteller  
FROM Kunden AS K, Fahrraeder AS F, Reservierungen AS R  
WHERE K.KdNr=R.KdNr And F.Fahrradnummer=R.FRadNr;
```

In einem zusätzlichen Feld GesPreis wird der Gesamtleihpreis mit dem Ausdruck P.Preis*LDauer AS GesPreis berechnet und ausgegeben. Das Ergebnis lautet nun:

```
SELECT K.KdNr, Nachname, Vorname,  
LeihDauer AS LDauer, F.Fahrradnummer AS FNr, Hersteller,  
P.Preis*LDauer AS GesPreis  
FROM Kunden AS K, Preisgruppen AS P, Fahrraeder AS F,  
Reservierungen AS R
```

```
WHERE K.KdNr=R.KdNr AND F.Fahrradnummer=R.FRadNr AND
P.Preisgruppe=F.Preisgruppe;
```

Die Ausgabe sieht dann folgendermaßen aus:

KdNr	Nachname	Vorname	LDauer	FNr	Hersteller	GesPreis
2	Müller	Herta	9	23	Hercules	108,00 €
3	Müller	Hans	18	13	Systemo	486,00 €
5	Winter	Susanne	12	8	Panasonic	132,00 €
5	Winter	Susanne	9	21	Hercules	135,00 €
8	Zwiebel	Karl	9	8	Panasonic	99,00 €
10	Hase	Hanna	12	21	Hercules	180,00 €

Da die Tabelle Reservierungen über einen Join mit der Tabelle Kunden verbunden ist, wird dies ebenso als WHERE-Bedingung angegeben, entsprechend den Beziehungen der Tabellen Reservierungen – Fahrräder – Preisgruppen. Deshalb muss beim Entwurf einer komplexen SQL-Abfrage über mehrere Tabellen hinweg die Art der Beziehungen und der betroffenen Felder bekannt sein.

Zur Vereinfachung der SQL-Syntax können Tabellennamen innerhalb der Abfrage umbenannt werden. Dazu wird in der FROM-Anweisung dem Tabellennamen ein kennzeichnender Buchstabe, z. B. K, P, R, F (also Fahrräder F), hinzugefügt. Dadurch wird der Tabelle ein neuer (temporärer) Name zugewiesen. Die Tabelle ist dann innerhalb der Abfrage allein durch diesen Buchstaben ansprechbar.

Mit dem Schlüsselwort AS werden Felder in der Ausgabe umbenannt. Dies ist z.B. für berechnete Felder wichtig, für die sonst kein Name definiert ist.

Im Beispiel oben ist dies: P.Preis*LDauer AS GesPreis

INNER-Joins können auch mit den Schlüsselworten INNER JOIN beschrieben werden.

Beispiel:

Alle reservierten Fahrräder (Fahrradnummer, Bezeichnung, Hersteller) sollen mit wichtigen Reservierungsdaten (ResNr, AusleihDatum, LeihDauer) angezeigt werden. Die Anweisung lautet:

```
SELECT F.Fahrradnummer, F.Bezeichnung, F.Hersteller,
      F.LetzteWartung, R.ResNr, R.AusleihDatum, R.LeihDauer
   FROM Fahrräder F INNER JOIN Reservierungen R
     ON F.Fahrradnummer = R.FRadNr;
```

Die Ausgabe ergibt z. B.:

Fahrradnummer	Bezeichnung	Hersteller	Letzte Wartung	ResNr	Ausleih-Datum	Leih-Dauer
4	Devant	Miyata	29.08.2020	20	22.01.2021	2
4	Devant	Miyata	29.11.2020	22	01.04.2022	5
8	Galata	Panasonic	08.05.2020	7	18.01.2022	9
13	Hurrican	Systemo	12.04.2019	5	16.06.2020	3
14	DownHill-Racer	Hitachi	21.05.2019	8	15.04.2021	9
19	Kletter-gemse	Hirsch	27.02.2020	12	15.12.2022	6

Hinweis:

INNER-Joins, EQUI-Joins und Natural Joins erzeugen jeweils dasselbe Ergebnis. Sie werden lediglich mit unterschiedlicher Syntax beschrieben. Bei Natural Joins wird die Verbindung der Tabellen in der WHERE-Klausel beschrieben, bei INNER-Joins (=EQUI-Joins) geschieht dies in der FROM-Klausel.

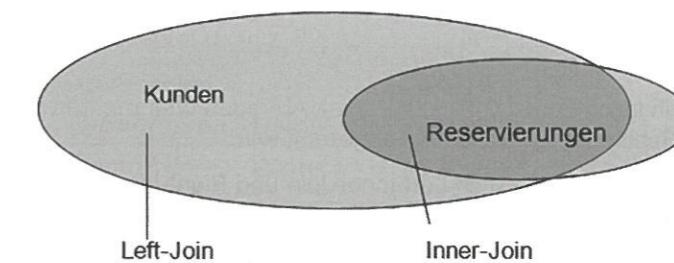
Left-Joins (auch Left-Inner-Joins)

Der LEFT-JOIN nimmt alle Einträge der einen Tabelle (z. B. Kunden) und kombiniert sie entweder mit allen gültigen Einträgen in der anderen Tabelle (Reservierungen) oder mit NULL, falls in der zweiten Tabelle kein zugehöriger Eintrag zu finden ist. Es ist wichtig, das entsprechende ER-Modell für die zwei Tabellen zu kennen:



Die Beziehung besagt, dass jedem Kunden mehrere oder auch gar keine Reservierungen zugeordnet sein können. Im Fall, dass keine Reservierungen zugeordnet werden, enthält das Ergebnis des LEFT JOIN in der Spalte AusleihDatum den Wert NULL.

Sollen in die Abfrage der Reservierungen auch die Kunden mit aufgenommen werden, die bisher noch kein Fahrrad reserviert haben, so muss die Verknüpfungsbedingung geändert werden. Während der Equi-Join der Schnittmenge der beiden Bereiche Kunden und Reservierungen entspricht, muss nun der linke Bereich der beiden Mengen mit eingeschlossen werden.



Ein solcher Left Join kann in Access mit der allgemeinen Syntax

```
... FROM tab1 LEFT JOIN tab2 ON tab1.feld = tab2.feld ...  
definiert werden.
```

Beispiel:

```
SELECT K.KdNr, Nachname, Vorname, Ort, AusleihDatum AS AusDat,  
LeihDauer AS LDauer
```

```
FROM Kunden K LEFT JOIN Reservierungen R ON K.KdNr=R.KdNr;
```

Die Ausgabe lautet z. B.:

KdNr	Nachname	Vorname	Ort	AusDat	LDauer
1	Palmert	Carlo	Jettingen		
2	Müller	Herta	Krumbach	16.10.2022	9
2	Müller	Herta	Krumbach	22.01.2022	3
3	Müller	Hans	Ulm	27.02.2022	18
4	Schulze	Anna	Ulm		
5	Winter	Susanne	Ulm	16.10.2021	9
5	Winter	Susanne	Ulm	01.05.2021	12
6	Müller	Andrea	Biberach		
7	Zwiebel	Karl	Elchingen	18.01.2021	9

Outer-Joins

Outer-Joins (z.B. ein Left-Outer-Join) zeigen als Ergebnis nur die Datensätze, die keine Referenzdaten in der zweiten Tabelle besitzen.

Beispiel:

Es sollen Daten der Kunden angezeigt werden, die noch keine Reservierung gebucht haben.

```
SELECT K.KdNr, Nachname, Vorname, Ort, AusleihDatum AS AusDat, Leih
      Dauer AS LDauer
FROM Kunden K LEFT JOIN Reservierungen R ON K.KdNr=R.KdNr
WHERE R.KdNr IS NULL;
```

Die Ausgabe lautet nun:

KdNr	Nachname	Vorname	Ort	AusDat	LDauer
1	Palmert	Carlo	Jettingen		
4	Schulze	Anna	Ulm		
6	Müller	Andrea	Biberach		
9	Winkler	Claus	Thalflingen		
10	Hase	Hanna	Günzburg		
11	Hahn	Isabelle	Senden		

Merke:

Inner-Join (auch Equi-Join, Natural Join): Abfragen über mehrere Tabellen, bei denen nur die Schnittmenge der Daten angezeigt wird.

Left-Join / Right Join (auch Left-Inner-Join und Right-Inner-Join): Außer der Schnittmenge wird auch der linke / rechte Bereich der Teilmengen eingeschlossen.

Outer-Join: schließt die Schnittmenge aus und zeigt nur die Datensätze an, die keine Einträge in der verbundenen Tabelle besitzen. Er wird in Access gebildet durch den Zusatz WHERE [Feldname] IS NULL.

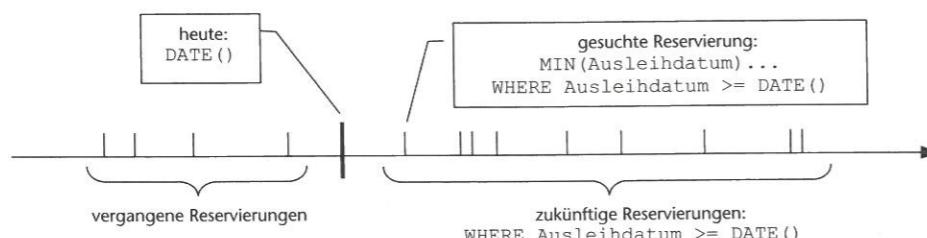
6.3.12 Unterabfragen

Unterabfragen (Subqueries) benötigt man, wenn die Suchbedingung vom Ergebnis einer anderen Abfrage abhängig ist. Sie liefern nur solche Daten, die innerhalb des WHERE-Ab schnittes zu einer logisch bewertbaren Bedingung führen, sonst erfolgt eine Fehlermeldung.

Beispiel:

Wir erstellen eine Abfrage, die den Namen des Kunden anzeigt, dessen Reservierung als nächstes fällig ist. Auch die Fahrradnummer und Bezeichnung des Fahrrades werden ausgegeben.

Zweckmäßig ist es, zunächst eine Abfrage zu bilden, die das nächste Reservierungs datum liefert. Die Funktion MIN(Ausleihdatum) gibt das früheste gespeicherte Reservierungsdatum zurück, die WHERE-Bedingung Ausleihdatum >= DATE() schränkt dann auf das kleinste Datum ab heute ein.



Die Abfrage zur Suche des gewünschten Datumswertes, die anschließend als Unterabfrage dient, lautet somit:

```
SELECT MIN(Ausleihdatum)
FROM Reservierungen
WHERE Ausleihdatum >= Date();
```

Das Ergebnis dieser Abfrage stellt den Wert des kleinsten zukünftigen Ausleihdatums dar, z. B.

Unterabfrage

16.10.202X

Nun muss der entsprechende Datensatz in der Tabelle Reservierungen durch die Bedingung WHERE Ausleihdatum = [Subquery] gesucht werden. Da Daten aus mehreren Tabellen ausgelesen werden, müssen die Tabellen Kunden, Reservierungen und Fahrräder über die notwendigen Equi-Joins miteinander in Beziehung gesetzt werden.

Die gesamte Abfrage lautet nun:

```
SELECT F.Fahrradnummer, F.Bezeichnung, Nachname, Ausleihdatum
FROM Fahrräder AS F, Reservierungen AS R, Kunden AS K
WHERE K.KdNr = R.KdNr
      AND F.Fahrradnummer = R.FRadNr
      AND Ausleihdatum = (
          SELECT MIN(Ausleihdatum)
          FROM Reservierungen
          WHERE Ausleihdatum >= Date()
      );
```

Natural Joins

Subquery

Die Ausgabe lautet dann z. B.:

NächsteAusleihe			
FNr	Bezeichnung	Name	Ausleihdatum
21	StormRide	Winter	16.10.202X

Hinweis:

Das Ergebnis der Unterabfrage muss vom Datentyp mit dem Vergleichsfeld übereinstimmen. Auch darf eine Unterabfrage nur dann mehrere Ergebnisse liefern, wenn sie z. B. Bestandteil einer Bedingung mit dem IN-Operator ist und somit einer Feldliste entspricht.

Komplexe Abfragen sollten stets modular entwickelt werden. Dadurch werden Fehler rechtzeitig entdeckt und können leichter beseitigt werden.

Beispiel:

Alle Daten der Kunden, die in einem Ort der Fahrradhersteller wohnen, werden aufgelistet. Voraussetzung ist, dass zunächst eine Tabelle „Hersteller“ mit den Daten der Fahrradhersteller erzeugt wird.)

```
SELECT *
FROM Kunden
WHERE Kunden.Ort IN
      (
          SELECT Hersteller.Ort
          FROM Hersteller
      );
```

6.4 Daten bearbeiten mit SQL

SQL bietet als Data Manipulation Language (DML) die Möglichkeit, Daten zu bearbeiten, indem neue Datensätze eingefügt, bestehende gelöscht oder geändert werden können.

6.4.1 Einfügen von Datensätzen

Um die Datenbestände entsprechend den laufenden Geschäftsprozessen zu aktualisieren, werden neue Daten hinzugefügt und überflüssige Daten gelöscht. Einfügen von Datensätzen geschieht in SQL mit der Schlüsselanweisung `INSERT`.

Die allgemeine Syntax der `INSERT`-Anweisung lautet:

```
INSERT INTO Tabelle (Feld1, Feld2, ...)
VALUES (Inhalt1, Inhalt2, ...);
INSERT INTO Tabelle (Feld1, Feld2, ...)
SELECT (Feld1, Feld2, ...) ...;
```

Beispiel:

Wir fügen mit einer SQL-Anweisung einen neuen Datensatz für das Tourenrad „Easygo“ des Herstellers Stiegl und der Fahrradnummer 12 in die Fahrradtabelle ein.

Lösung:

```
INSERT INTO Fahrraeder (Fahrradnummer, Hersteller, Bezeichnung,
Art)
VALUES (12, 'Stiegl', 'Easygo', 'Tourenrad');
```

Hinweis:

Die Reihenfolge der nach `INSERT` aufgelisteten Felder entspricht den nach `VALUES` aufgelisteten Werten. Der Datentyp der Felder muss gleich dem Datentyp des entsprechenden Wertes sein.

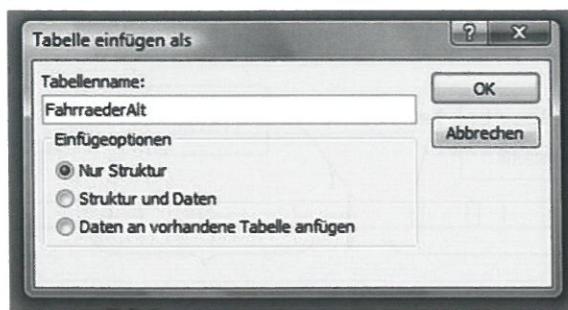
Werden die einzutragenden Daten aus einer bestehenden Tabelle ausgewählt, so wird eine `SELECT`-Anweisung eingebunden. Die Struktur der `INSERT`-Anweisung lautet dann:

Beispiel:

Fahrräder, die vor 2014 angeschafft wurden, sollen ausgemustert werden. Um die Daten nicht endgültig zu verlieren, benötigt man eine Anweisung, die zur Sicherung die Fahrraddaten in eine Tabelle `FahrraederAlt` einfügt.

```
INSERT INTO FahrraederAlt
SELECT *
FROM Fahrraeder
WHERE Anschaffungsdatum < #1/1/2014#;
```

Es wird eine Tabelle `FahrraederAlt` benötigt mit der gleichen Tabellenstruktur. Diese kann man mit einer `CREATE TABLE`-Anweisung erstellen oder die bestehende Tabelle `Fahrraeder` z. B. in Access kopieren, indem man im Kontextmenü erst Kopieren und dann Einfügen wählt. Dort wird die Option Nur Struktur markiert.



Im Beispiel werden über die `SELECT`-Anweisung alle Datensätze der Fahrräder gesucht, die vor 2014 angeschafft wurden, und anschließend in die neue Tabelle gespeichert.

Hinweis:

Datentypen und Namen der korrespondierenden Felder der beiden Tabellen müssen bei dieser Methode übereinstimmen.

Beispiel:

Mit einer `INSERT`-Anweisung sollen die Daten des neuen Kunden Fritz Kleidermann aus 70173 Stuttgart, Mantelstr. 128 und der Kundennummer Kd_Nr = 2658 eingefügt werden.

Lösung:

```
INSERT INTO Kunden (Kd_Nr, Nachname, Vorname, PLZ, Ort, Strasse)
VALUES (2658, 'Kleidermann', 'Fritz', 70173, 'Stuttgart', 'Mantelstr. 128');
```

Hinweis:

Bei dieser Methode des Einfügens definierter Werte mit Hilfe des Schlüsselworts `VALUES` muss in jedes Feld, das wegen des Attributs NOT NULL nicht leer sein darf, ein Wert eingefügt werden.

6.4.2 Löschen von Datensätzen

Das Löschen der ausgelagerten Datensätze geschieht mit dem SQL-Befehl `DELETE`. Die allgemeine Syntax lautet:

```
DELETE Attributname/* FROM Tabelle
WHERE Bedingung;
```

Beispiel:

Eine SQL-Anweisung soll die Fahrräder, die vor 2014 angeschafft wurden, aus der Tabelle `Fahrraeder` löschen.

Lösung:

```
DELETE *
FROM Fahrraeder
WHERE Anschaffungsdatum < #1/1/2014#;
```

Hinweis:

Der Löschbefehl `DELETE` kann nicht mehr rückgängig gemacht werden. Deshalb ist es empfehlenswert, zunächst mit dem Befehl `SELECT` nach den zu löschen Daten zu suchen. Nach Überprüfung werden die Daten gelöscht.

Empfehlenswert ist stets, eine Sicherungskopie der Daten aufzubewahren.

6.4.3 Aktualisieren von Daten

Zum Verändern von Feldinhalten dient der Aktualisierungsbefehl `UPDATE`, z. B. wenn viele Datensätze gleichsinnig geändert werden sollen.

Die allgemeine Syntax lautet:

```
UPDATE Tabelle
SET Attributname = Wert
WHERE Bedingung;
```

Beispiel:

Preisgruppen dienen im Beispiel der Datenbank Faradiso zur Zusammenfassung von Fahrrädern mit gleichem Verleihpreis.

Der Verleihpreis der Fahrräder, die vor 2016 angeschafft wurden, soll um eine Preisgruppe vermindert werden. Die Änderungen sollen in den betroffenen Datensätzen gespeichert werden.

Lösung:

```
UPDATE Fahrraeder
SET Preisgruppe = Preisgruppe - 1
WHERE YEAR(Anschaffungsdatum) < 2016;
```

Hier wird in allen Datensätzen der Tabelle Fahrraeder, die der WHERE-Bedingung entsprechen und vor dem Jahr 2016 angeschafft wurden, der Inhalt des Feldes Preisgruppe um 1 vermindert.

6.5 Konsistenz der Datenbank

Die **Konsistenz** einer Datenbank beschreibt die Korrektheit der internen Speicherungsstrukturen und der Zugriffspfade, um die Datenbestände stets widerspruchsfrei nutzen zu können.

Da Datenbanken in der Regel von mehreren Anwendern gleichzeitig verwaltet werden, können z. B. Löschtätigkeiten von Daten nicht mehr rückgängig gemacht werden. Denn nach der Löschtätigkeit könnten sich weitere Anwender auf die inzwischen geänderten Datenbestände bereits verlassen haben. Durch diese Verhinderung des Rückgängigmachens werden somit Anomalien (logische Widersprüche) im Datenbestand verhindert, die Daten sind weiterhin konsistent (widerspruchsfrei).

Beispiel:

Die Datenbankstruktur muss sicherstellen, dass eine einmal gespeicherte Kundenadresse nicht einer weiteren Adressangabe dieses Kunden widerspricht. Sonst wäre das Zusenden einer Rechnung nicht mehr möglich.

Bereits bei der Planung einer Datenbank zu verhindern Anomalien sind Löschanomalien, Änderungsanomalien und Einfügeanomalien.

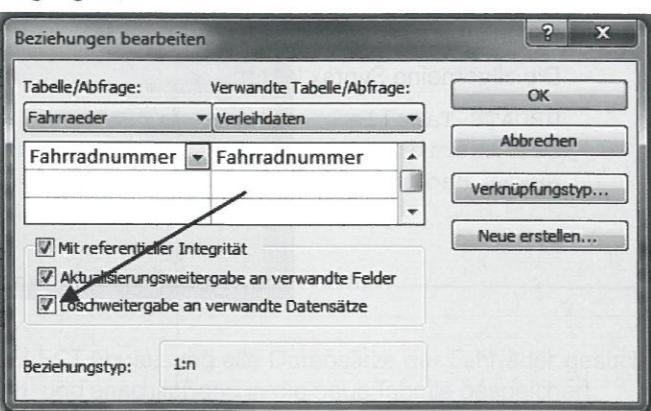
Löschanomalien entstehen, wenn ein Datensatz gelöscht wird, auf dessen Primärschlüsselwert sich ein abhängiger Datensatz einer anderen Tabelle bezieht. Würde z. B. der Datensatz des Kunden mit der Kundennummer 15 gelöscht werden, so könnten Verleihdaten, die sich auf die Kundennummer 15 beziehen, nicht mehr zugeordnet werden.

Änderungsanomalien entstehen z. B. durch Mehrfachspeicherung von Daten. Werden die Daten (z. B. die Kundenadresse) in einer Tabelle geändert, so widersprechen sie den Daten in einer anderen Tabelle.

Einfügeanomalien entstehen dann, wenn nach dem Einfügen von Datensätzen Werte mehrfach in der Datenbank vorkommen und sich dadurch widersprechen.

Um **Anomalien** zu verhindern, werden bereits beim Entwurf der Datenbank die Beziehungen so festgelegt, dass Löschtätigkeiten überwacht werden.

Wird z. B. in der Tabelle Fahrraeder ein Fahrrad gelöscht, auf dessen Fahrradnummer ein Datensatz in der Verleihtabelle verweist, so sollen ebenfalls die Verleihdatensätze dieser Fahrradnummer gelöscht werden. Dies geschieht z. B. in Access durch Auswahl der Checkbox Löschtätigkeit an verwandte Datensätze im Fenster Beziehungen bearbeiten:



6.6 Transaktionen

Datenbanken müssen zu jedem Zeitpunkt, auch nach einem Hardware- oder Softwarefehler, konsistent sein. Bei einer Überweisung zwischen Konten zweier Banken darf die Überweisung nur dann gültig sein, wenn das Konto A um den Betrag vermindert worden ist und dem Konto B dieser Betrag gutgeschrieben worden ist.



Erfolgt nach der Buchung 1 ein Stromausfall, so wären die Kontostände nicht mehr konsistent, da der Kontostand A bereits um 250 € vermindert, der Kontostand B aber noch nicht erhöht wurde.

Um sicher zu gehen, dass die Überweisung vollständig durchgeführt wird, werden die beiden Buchungen in Form einer Transaktion ausgeführt.

Definition:

Unter einer **Transaktion** versteht man eine Folge von SQL-Anweisungen, die logisch zusammengehören und den Datenbestand konsistent erhalten.

Hinweis:

Beim Ausführen einer Transaktion muss sichergestellt werden, dass die Transaktion entweder komplett ausgeführt wird oder im Fehlerfall völlig rückgängig gemacht wird.

Nachfolgende Anweisungen werden nur dann ausgeführt, wenn die vorhergehenden erfolgreich durchgeführt wurden.

Um Datenbestände immer gültig zu erhalten, wird in einzelnen Schritten vorgegangen:

Schritte	Aufgabe
1. Lesen der Daten	Die Daten werden vom Speichermedium eingelesen.
2. Merken der bisherigen Daten	Die zu ändernden Daten werden in die Logdatei geschrieben (Before-Image).
3. Ändern der Daten	Ändern der Daten im Arbeitsspeicher, Sperren dieser Einträge für andere Benutzer (Datensatz loggen).
4. Merken der geänderten Daten	Die geänderten Daten werden in die Logdatei geschrieben (After-Image).
5. Transaktionsende mit COMMIT (= Bestätigung)	Schreiben aller Images und Metadaten in die Logdatei. Transaktionsende in der Logdatei vermerken. Sperren freigeben.
6. Transaktionsende mit ROLLBACK (= zurückdrehen)	Rücksetzen der Metadaten der Transaktion. Geänderte Daten, die bereits in die Datenbank geschrieben wurden, werden für ungültig erklärt. Sperren freigeben.
7. Änderungen speichern	Die geänderten Daten werden in die Datenbank geschrieben.

Zunächst werden die Daten geladen, also der Kontostand A mit 1300 €. Die noch nicht geänderten Daten werden nun als Before-Image (= Abbild vor der Änderung) zur Sicherung in die Logdatei geschrieben. Dies ist eine nichtflüchtige Datei innerhalb des Datenbanksystems.

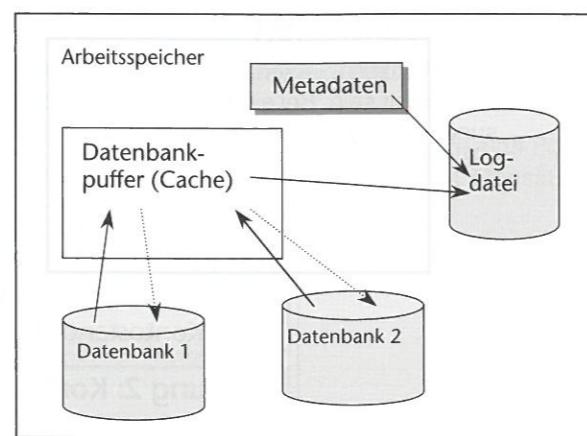
Jetzt werden die Änderungen der Daten im Arbeitsspeicher mit den notwendigen Befehlen (UPDATE, DELETE, INSERT) durchgeführt. Dabei entstehen Metadaten (= vorläufige geänderte Daten), die für andere Benutzer gesperrt sind. Der Kontostand A wird also mit Update um 250 € auf 1050 € vermindert. Dieser geänderte Kontostand wird innerhalb der Logdatei als After-Image (= Abbild nach der Änderung) gesichert.

Die entsprechenden Vorgänge müssen nun mit dem Kontostand B durchgeführt werden. Die neuen Daten stehen somit als After-Image in der Logdatei zur Verfügung.

Ist bisher kein Fehler aufgetreten, so wird mit dem Befehl COMMIT das Ende der Transaktion in der Logdatei vermerkt. Sperren werden freigegeben. Anschließend werden die Metadaten in die Datenbank geschrieben.

Ist vor dem COMMIT-Befehl ein Fehler aufgetreten, so wird mit der Anweisung ROLLBACK die gesamte Transaktion mit allen Metadaten zurückgesetzt. Die bisherigen Daten sind noch in den Before-Images innerhalb der Logdatei gesichert. Da bei einem Ende einer Transaktion ein Vermerk in die Logdatei geschrieben wird, kann bei einem Systemabsturz jederzeit erkannt werden, welche Änderungen zu einer bereits beendeten Transaktion gehören.

Wird die Logdatei nicht auf demselben Speichermedium angelegt, auf dem sich auch die Datenbank befindet, so ist es sehr unwahrscheinlich, dass beide Datenbestände, Logdatei und Datenbank, gleichzeitig zerstört werden. Deshalb kann mithilfe der letzten Sicherung und der inzwischen weitergeführten Logdatei der Datenbestand jederzeit wiederhergestellt werden, um wieder einen konsistenten Datenbestand zu erhalten.



6.7 Aufgaben zu Kapitel 6

Aufgabe 1

Für eine Arztpraxis soll eine Datenbank entworfen werden mit folgenden Forderungen:

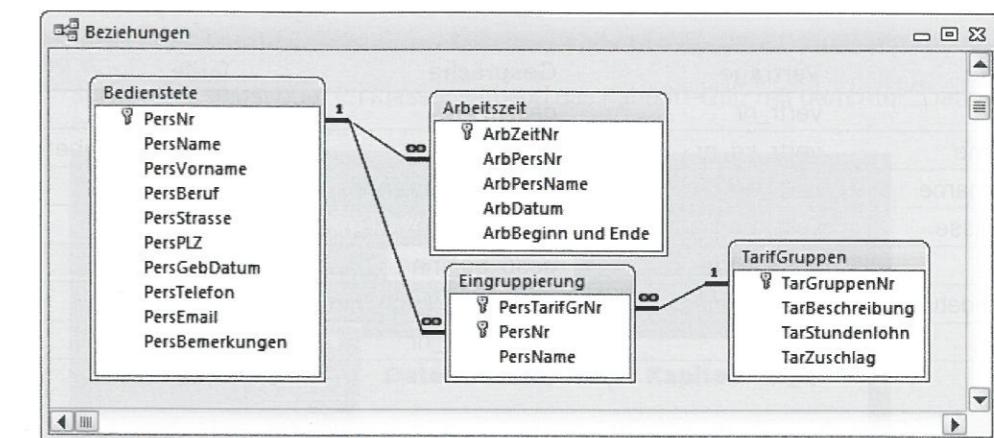
- ▶ Es sollen die einzelnen Behandlungen der Patienten verwaltet werden. Dafür sind das Datum sowie die Diagnose und evtl. Bemerkungen zu speichern.
- ▶ Jeder Patient ist in genau einer Krankenversicherung Mitglied. Privat Versicherte brauchen nicht berücksichtigt zu werden.
- ▶ Leistungen (wie Blutabnahme, Blutdruckkontrolle, Laboruntersuchungen, eingehende Untersuchung, Überweisung, Krankheitsbescheinigung, Rezeptausstellung) werden direkt in der Praxis durchgeführt. Hier werden zuweilen pro Behandlungstermin mehrere erbracht. Die Leistungen sind laut einem Katalog mit Nummer, Bezeichnung der Leistung und dem Preis zu erfassen.
- ▶ Bei Verschreibungen sollen die Daten des verschriebenen Medikaments, der Hersteller, sowie der Preis und der Hauptwirkstoff nachvollziehbar sein.
- ▶ Für jeden Patienten sollen die Kosten jeder Behandlung einschließlich der erbrachten Leistungen abrufbar sein. (Mögliche Kostensteigerungen brauchen aber nicht vorgesehen werden.)

Entwerfen Sie eine geeignete Datenbank.

Erstellen Sie auch die notwendigen Beziehungen.

Aufgabe 2

Für die Lohnabrechnung der Bediensteten in der Arztpraxis dient die folgende Erweiterung der Datenbank „Arztpraxis“.



- In welchen Punkten verstößt dieser Entwurf gegen die Normalisierungsregeln der ersten, zweiten und dritten Normalform? Machen Sie jeweils Vorschläge zur Vermeidung der Verstöße.
- Erstellen Sie SQL-Abfragen, die folgende Aufgaben erfüllen:
- Es sollen alle Bediensteten aus dem Postleitzahlbereich 46*** ausgegeben werden.
- Wie viele Bedienstete sind in der Arztpraxis beschäftigt?
- Welche Personen haben am 23.12.2022 Dienst getan?

Aufgabe 3

In einem Unternehmen soll eine Projektverwaltung mithilfe einer Datenbank erstellt werden. Jeder Mitarbeiter muss sich mit einem Passwort anmelden.

Dieses ist in einer Tabelle gespeichert. Für jedes Projekt soll der Name, sowie das Start-, das Enddatum und der Projektleiter gespeichert werden.

Für jedes Projekt wird ein Mitarbeiter als Projektleiter ausgewählt.

Es soll möglich sein für jeden Mitarbeiter die Arbeitszeit (Dauer in Stunden) und eine Beschreibung der ausgeführten Arbeiten zu erfassen, die dieser an einem bestimmten Tag für ein bestimmtes Projekt erledigt hat.



- Überprüfen Sie die Felder der einzelnen Tabellen entsprechend den Anforderungen.
- Stellen Sie die Beziehungen der Tabellen grafisch mit einem erweiterten ER-Diagramm dar und geben Sie den jeweiligen Beziehungstyp zwischen den einzelnen Tabellen an.
- Wie viele Mitarbeiter sind in den einzelnen Projekten beschäftigt? Erstellen Sie eine SQL-Abfrage, die die Projektnummern und den Namen sowie die Anzahl der Mitarbeiter ausgibt.

Aufgabe 4

Gegeben sind die folgenden Relationen (Tabellen) einer Datenbank eines Mobilfunkanbieters.

Kunden	Verträge	Gespräche	Tarife
kd_nr	vertr_nr	gesp_nr	tarif_nr
kd_name	vertr_kd_nr	gesp_vertr_nr	tarif_minutenbetrag
kd_vorname	vertr_art	gesp_zielnummer	
kd_strasse	vertr_beginn	gesp_datum	
kd_plz	vertr_ende	gesp_beginn	
kd_gebdatum	vertr_grundpreis	gesp_dauer (in min)	
		gesp_tarif_nr	

- a) Bestimmen und kennzeichnen Sie die Primär- und Fremdschlüssel der Tabellen.

Erstellen Sie SQL-Abfragen, die folgende Aufgaben erfüllen:

- b) Welcher Kunde hat das Gespräch Nr.34890 geführt?
 c) Es sollen alle Gespräche mit Beginn und Dauer sowie der Tarif und der jeweilige Gesprächspreis angezeigt werden, die der Kunde Hermann Maier am 25.03.2015 geführt hat.
 d) Welcher Vertrag (Ausgabe mit Name des Kunden) endet als nächstes?
 e) Die Anzahl der Gespräche der einzelnen Kunden sind gesucht.
 f) Was bewirkt die SQL-Anweisung „HAVING ...“ (mit Beispiel)?

Aufgabe 5:

Eine Gemeinschaftspraxis von Ärztinnen und Ärzten bietet ihren Patienten an, Arzttermine online zu vereinbaren. Dazu wird eine Datenbank entworfen, die u. a. folgende Relationen enthält:

Arzt (ArztID, Name, Spezialgebiet, TelNr)

Termin (TerminID, ArztID, Datum, Anfangszeit, Endzeit, PatientID)

Patient (PatientID, Name, GebDatum)

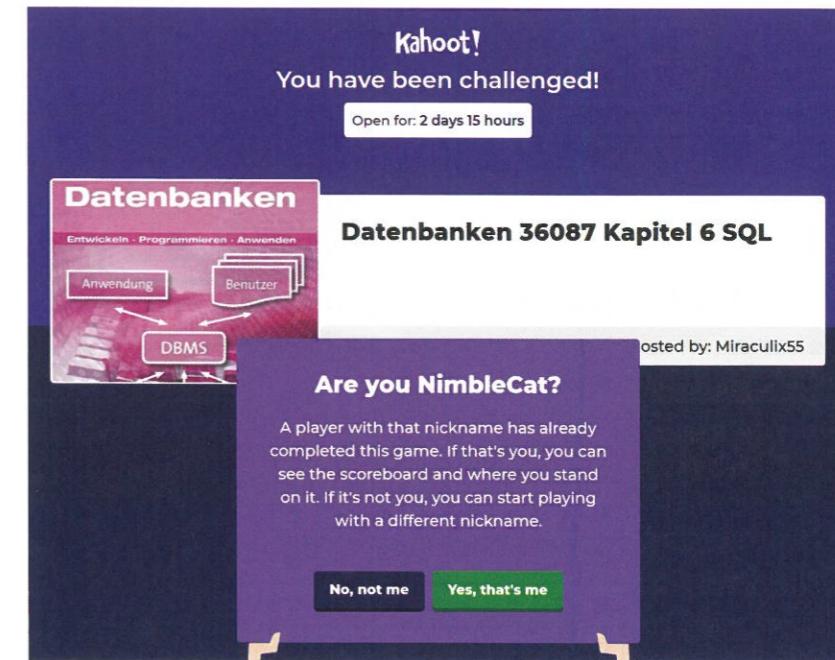
- a) Kennzeichnen Sie die Primärschlüsselfelder und die Fremdschlüsselfelder.
 b) Der Arzthelfer möchte eine Liste aller Patiententermine von Dr. Gallenbitter für den heutigen Tag in zeitlicher Reihenfolge anzeigen lassen.
 Formulieren Sie eine SQL-Abfrage für die erwünschte Liste.
 c) Für Dr. Müller (ArztID=2) soll am 17. Juni um 16:30 ein 20-Minuten-Termin für den Patienten Peter Piercing (PatientID=27) eingetragen werden.
 Erstellen Sie eine SQL-Anweisung, die diesen Datensatz anfügt.
 (Die TerminID wird vom Datenbank-Management-System automatisch vergeben.)

6.8 Digitale Inhalte zu Kapitel 6

Hinweis: Um die Aufgaben online zu bearbeiten, bitte den QR-Code scannen oder den Link eingeben.

Aufgabe 1

Spielen Sie selbst oder im Klassenverband das Kahoot!-Quiz mit dem Titel „Datenbanken 36087 Kapitel 6 SQL“



Quelle: Kahoot-App oder www.kahoot.com

Aufgabe 2

Suchen Sie im Wortgitter Fachbegriffe zu SQL
<https://vel.plus/rJYI>



B	L	Q	K	D	H	Ö	O	Z	A	A	I	U	K	Z	N
O	Z	J	B	P	D	R	S	X	Y	G	H	M	Y	Q	Z
N	U	G	Z	Ü	W	H	G	F	B	G	G	B	J	C	M
K	L	T	Ö	E	J	Y	L	U	O	R	R	H	C	T	V
I	P	Ä	C	G	H	E	Q	Ä	O	E	Z	I	H	E	Ä
C	J	S	O	I	I	S	W	F	D	G	S	T	M	W	P
T	C	O	A	M	A	F	Z	R	S	A	Q	Ä	P	Ä	K
H	O	G	V	L	Y	H	W	O	L	T	R	Y	P	G	Ö
Ö	Ü	X	Ü	H	I	Ü	J	U	S	F	K	Ä	L	V	P
S	X	Z	F	Y	D	W	J	Ä	Ä	U	R	W	L	Ö	G
K	A	Ü	W	Z	D	H	T	S	X	N	F	S	G	Q	I
U	R	I	D	B	F	K	F	Q	G	K	E	Y	Y	S	G
A	R	A	Z	I	J	H	G	A	R	T	U	Ä	Ü	D	R
U	C	N	D	J	P	E	P	N	U	I	S	T	C	N	X
H	R	R	S	F	U	Ü	P	Ä	P	O	Z	X	Ä	N	U
O	E	D	Q	A	Q	J	R	Z	P	N	G	O	P	F	S
P	A	U	B	C	Ä	O	H	P	I	E	G	S	A	J	T

- _____ Auswahlabfragen
- _____ zwischen
- _____ ... verhindert Duplikate
- _____ korrekte Datenstrukturen
- _____ Bedingungen
- _____ logische Abfolge von Datenänderungen
- _____ Zusammenfassen von Daten
- _____ Widersprüche durch Löschen von Daten
- _____ Logische Verknüpfungen von Tabellen

Aufgabe 3

Bilden Sie Paare aus den Fachbegriffen
<https://vel.plus/id4t>



Aufgabe 4

Suchen Sie Begriffe zu SQL aus einem Wortgitter.
<https://vel.plus/wROh>



E	L	V	Ä	E	Y	X	C	A	M	P	W	S	R	J
M	T	D	D	I	M	T	N	L	H	B	F	Q	Ä	Ö
V	V	H	A	V	I	N	G	T	E	B	T	S	Ö	B
A	U	E	Ü	W	H	E	R	E	L	J	U	J	Ü	C
S	K	I	X	R	C	Ü	O	R	D	E	R	-	B	Y
J	Z	Q	Ö	L	Z	S	U	H	K	L	S	E	T	O
O	E	X	W	K	T	U	P	D	A	T	E	S	D	Y
I	R	Ä	S	O	A	Y	-	R	Ü	X	L	V	D	Q
N	D	K	L	Q	I	Y	B	E	T	W	E	E	N	L
S	U	B	Q	U	E	R	Y	Ö	E	I	C	R	Z	Ä
Ä	Y	D	Ö	Ü	Ä	Ä	K	J	O	X	T	O	D	Ü

1. _____
Gruppierung (2 Worte)
 2. _____
Bedingung für einen Datensa
 3. _____
Aktualisierungsabfrage
 4. _____
Unterabfrage
 5. _____
Bedingung für Gruppierunge
 6. _____
Änderungsabfrage
 7. _____
Abfrage eines Bereiches
 8. _____
Auswahlabfrage
 - 9r. _____
Sortierung (2 Worte)
 10. _____
Verknüpfungen von Tabellen

Aufgabe 5

gabe 5:
Tabellen und Schlüsseln mit SQL definieren
Wählen Sie jeweils die richtigen Antworten.
<https://vel.plus/9NSh>



Tabellen und Schlüsselelementen mit SQL definieren

2021-12-19

Datenbanken

Wozu dient ein Primärschlüssel
(Primary Key)?

1 / 4

Entwickeln, Programmieren, Anwenden

- Um eine Tabelle als erstes zu erstellen.
- Damit wird eine Tabelle eindeutig gekennzeichnet.
- Um einen Datensatz eindeutig zu kennzeichnen.
- Um Beziehungen zu anderen Tabellen herzustellen zu können.

DBMS

Aufgabe 6

CONRAINT-Optionen in SQL-Anweisungen
Die CONSTRAINT-Option einer Felddefinition beschreibt verschiedene Beschränkungen der Feldinhalte.
Wählen Sie zu jedem Begriff die richtige Beschreibung.
<https://vel.plus/spNx>



The diagram illustrates the Oracle CONSTRAINT Options:

- RESTRICT**: A box containing the text "Eine Änderung des Wertes in der Primärtafel wird an die Detailtabellen weitergegeben." with a red arrow pointing to it.
- CASCADE**: A box containing the text "Eine Änderung des Wertes in der Primärtafel wird sowohl in der Primärtafel als auch in der Detailtafel verweigert." with a red arrow pointing to it.
- Aufgabe**: A central box containing:
 - "Die CONSTRAINT-Option einer Felddefinition beschreibt verschiedene Beschränkungen der Feldinhalte."
 - "Wählen Sie zu jedem Begriff die richtige Beschreibung."
- OK**: A button below the central box.
- SET NULL**: A box containing the text "Die Änderung der Löschung Wertes in der Detailtafel bedeutet, dass der Wert in der Detailtafel auf NULL gesetzt wird." with a red arrow pointing to it.
- SET DEFAULT**: A box containing the text "Alle Daten werden auf NULL gesetzt." with a red arrow pointing to it.
- ACTION**: A box containing the text "Alle Änderungen werden verweigert." with a red arrow pointing to it.