MS ACCESS: Fortgeschrittene Jointechniken

Inhalt

1 Einführung

In diesem Beitrag werden *„Fortgeschrittene Jointechniken“* im Datenbanksystem *ACCESS* nachgebildet, die ursprünglich für das Datenbanksystem *MySQL* entwickelt wurden. Alle 6 Tabellen einer MySQL-Beispieldatenbank (Quelle: [1]) werden übernommen. Zielsetzung dieses Beitrags ist, die Kompatibilität der beiden Datenbanksysteme bezüglich bestimmter Auswahlabfragen mit SQL[1] zu überprüfen.

Mit einer *JOIN*-Operation lassen sich Verknüpfungen zwischen mehr als einer Datenbanktabelle oder auch nur innerhalb einer einzigen Datenbanktabelle herstellen und zu einer einzigen Ergebnismenge kombinieren. Voraussetzung ist jeweils ein Feld, das in der/den Datenbanktabelle(n) vorkommt und den gleichen Datentyp besitzt.

Bevor die Nachbildung der *JOIN*-Operationen mit ACCESS erfolgt,

- werden alle sechs Tabellen der Beispieldatenbank vorgestellt,
- wird das Schema der Beispieldatenbank sowohl in MySQL als auch in ACCESS grafisch dargestellt.

In der Praxis häufig vorkommende SQL-JOIN-Vorgänge werden in Anhang mit Mengendiagrammen visuell erklärt.

2 Beispieldatenbank

2.1 Tabellen

Die Beispieldatenbank (Quelle: [1]) umfasst 6 Tabellen, davon sind 5 relational miteinander verbunden:

- *Bestellungen_Oktober* (Aliasname in ACCESS*: Bestellungen*)
- *Kreditkarten*
- *Kunden*
- Positionen
- *Vorteilsclub*

Die 6. Tabelle '*Teams*' ist eigenständig, also nicht anderen Tabellen verknüpft. Die jeweilige Struktur der übernommenen Tabellen sowie deren Inhalte werden im Folgenden gezeigt:

```
DROP TABLE IF EXISTS `Bestellungen_Oktober`;
CREATE TABLE `Bestellungen_Oktober` (
  `KndNr` INT(11) NOT NULL,
  `BestellungsNr` INT(11) NOT NULL AUTO_INCREMENT,
  `Datum` DATE NOT NULL,
  PRIMARY KEY (`BestellungsNr`),
  KEY `KndNr` (`KndNr`),
  KEY `Datum` (`Datum`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin;
LOCK TABLES `Bestellungen_Oktober` WRITE;
INSERT INTO `Bestellungen_Oktober` VALUES
(123456, 987654, '2014-10-15'),
(123456, 987655, '2014-10-16'),
(123457, 987656, '2014-10-16');
UNLOCK TABLES;
```

Tabelle 1: Bestellungen_Oktober

[1] Die Abkürzung **SQL** bedeutet: **S**ystem **Q**uery **L**anguage (dt. Strukturierte Abfragesprache).

```
DROP TABLE IF EXISTS `Kreditkarten`;
CREATE TABLE `Kreditkarten` (
  `KartenNr` BIGINT(20) NOT NULL AUTO_INCREMENT,
  `Firma` tinytext COLLATE utf8_bin NOT NULL,
  `KndNr` BIGINT(20) NOT NULL,
  `Ablaufdatum` DATE NOT NULL,
  PRIMARY KEY (`KartenNr`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin;
LOCK TABLES `Kreditkarten` WRITE;
INSERT INTO `Kreditkarten` VALUES
(12345, 'VISA', 123457, '2019-05-01'),
(12346, 'Mastercard', 123459, '2020-01-01'),
(12348, 'American Express', 123459, '2019-05-01'),
(12349, 'Diners Club', 123458, '2022-02-01'),
(12350, 'VISA', 123458, '2017-03-01');
UNLOCK TABLES;
```

Tabelle 2: Kreditkarten

```
DROP TABLE IF EXISTS `Kunden`;
CREATE TABLE `Kunden` (
  `KndNr` BIGINT(20) NOT NULL,
  `Nachname` tinytext COLLATE utf8_bin NOT NULL,
  `Vorname` tinytext COLLATE utf8_bin NOT NULL,
  `Strasse` tinytext COLLATE utf8_bin NOT NULL,
  `PLZ` text COLLATE utf8_bin NOT NULL,
  `Ort` tinytext COLLATE utf8_bin NOT NULL,
  PRIMARY KEY (`KndNr`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin;
LOCK TABLES `Kunden` WRITE;
INSERT INTO `Kunden` VALUES
(123456, 'Mustermann', 'Max', 'Musterweg 1', '12345', 'Musterstadt'),
(123457, 'Musterfrau', 'Katrin', 'Musterstraße 7', '12345', 'Musterstadt'),
(123458, 'Müller', 'Lieschen', 'Beispielweg 3', '23987', 'Irgendwo'),
(123459, 'Schmidt', 'Hans', 'Hauptstraße 2', '98765', 'Anderswo'),
(123460, 'Becker', 'Heinz', 'Mustergasse 4', '12543', 'Musterdorf');
UNLOCK TABLES;
```

Tabelle 3: Kunden

```
DROP TABLE IF EXISTS `Positionen`;
CREATE TABLE `Positionen` (
  `PositionsNr` BIGINT(20) NOT NULL AUTO_INCREMENT,
  `BestellungsNr` BIGINT(20) NOT NULL,
  `Artikel` tinytext COLLATE utf8_bin NOT NULL,
  `Anzahl` INT(11) NOT NULL,
  `Preis` DECIMAL(10,2) NOT NULL,
  PRIMARY KEY (`PositionsNr`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin;
LOCK TABLES `Positionen` WRITE;
INSERT INTO `Positionen` VALUES
(10241, 987654, 'CD-Player', 2, 49.95),
(10242, 987654, 'DVD-Player', 3, 59.95),
(10243, 987654, 'CD xyz', 10, 15.95),
(10244, 987654, 'DVD abc', 5, 9.95),
(10245, 987655, 'CD-Player', 1, 51.20),
(10246, 987655, 'CD xyz extra   ', 20, 16.25),
(10247, 987656, 'DVD-Player', 1, 64.95);
UNLOCK TABLES;
```

Tabelle 4: Positionen

```
DROP TABLE IF EXISTS `Vorteilsclub`;
CREATE TABLE `Vorteilsclub` (
  `KndNr` BIGINT(20) NOT NULL,
  `ClubNr` BIGINT(20) NOT NULL AUTO_INCREMENT,
  `Kategorie` tinyint(4) NOT NULL,
  PRIMARY KEY (`ClubNr`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 COLLATE=utf8_bin;
LOCK TABLES `Vorteilsclub` WRITE;
INSERT INTO `Vorteilsclub` VALUES
(123458, 1414, 3),
(123456, 1415, 1),
(123460, 1416, 1);
UNLOCK TABLES;
```

Tabelle 5: Vorteilsclub

```
DROP TABLE IF EXISTS `Teams`;
CREATE TABLE `Teams` (
  `T_ID` INT(3) NOT NULL AUTO_INCREMENT,
  `Team` tinytext COLLATE utf8_bin NOT NULL,
  PRIMARY KEY (`T_ID`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 COLLATE=utf8_bin;
LOCK TABLES `Teams` WRITE;
INSERT INTO `Teams` VALUES
(1, 'Hamburg'),
(2, 'München'),
(3, 'Berlin'),
(4, 'Köln');
UNLOCK TABLES;
```

Tabelle 6: Teams

2.2 Besonderheiten der Tabellen:

„Nicht jeder Kunde muss über eine Kreditkarte verfügen, aber Kunden können Kreditkarten mehrerer Institute angeben. Entsprechend muss nicht von jedem Kunden im Oktober eine Bestellung vorliegen, nicht jeder Kunde ist Mitglied im Vorteilsclub.“ (s. [1]).

2.3 Speichersubsystem '*InnoDB*'

Alle 6 Tabellen beruhen auf dem Speichersubsystem *InnoDB* des Datenbankmanagementsystem *MySQL*, jeweils erkennbar an der SQL-Option *Engine=InnoDB*. Dieses Speichersubsystem unterstützt sowohl Fremdschlüsselbeziehungen[2] als auch Transaktionssicherung[3].

2.4 Tabellenbeziehungen

2.4.1 Beziehungen der MySQL-Datenbanktabellen

Mit dem Werkzeug '*Designer*' lassen sich die Verknüpfungen zwischen den oben genannten sechs Tabellen in der MySQL-Datenbank mit dem Namen '*advancedjoins*' grafisch darstellen (s. Abb. 1). Das Werkzeug '*Designer*' ist über eine eigene Schaltfläche in *phpMyAdmin*[4] aufrufbar.

[2] Das Anlegen einer *Fremdschlüsselbeziehung* geschieht im Wesentlichen mit den SQL-Anweisungen *FOREIGN KEY* und *REFERENCES*. Mehr dazu später.

[3] Durch Transaktionssicherung ist es möglich, den Datenbestand auf den Zustand zurückzusetzen, der zum Zeitpunkt des Beginns einer Transaktion vorhanden war. Das bedeutet, dass Schreibvorgänge in der Datenbank zurückgesetzt werden können, wenn sie nicht abgeschlossen werden können.

[4] phpMyAdmin ist ein mit der Programmiersprache PHP erstelltes Werkzeug zur Administration von MySQL-Datenbanken mittels Internetbrowser.

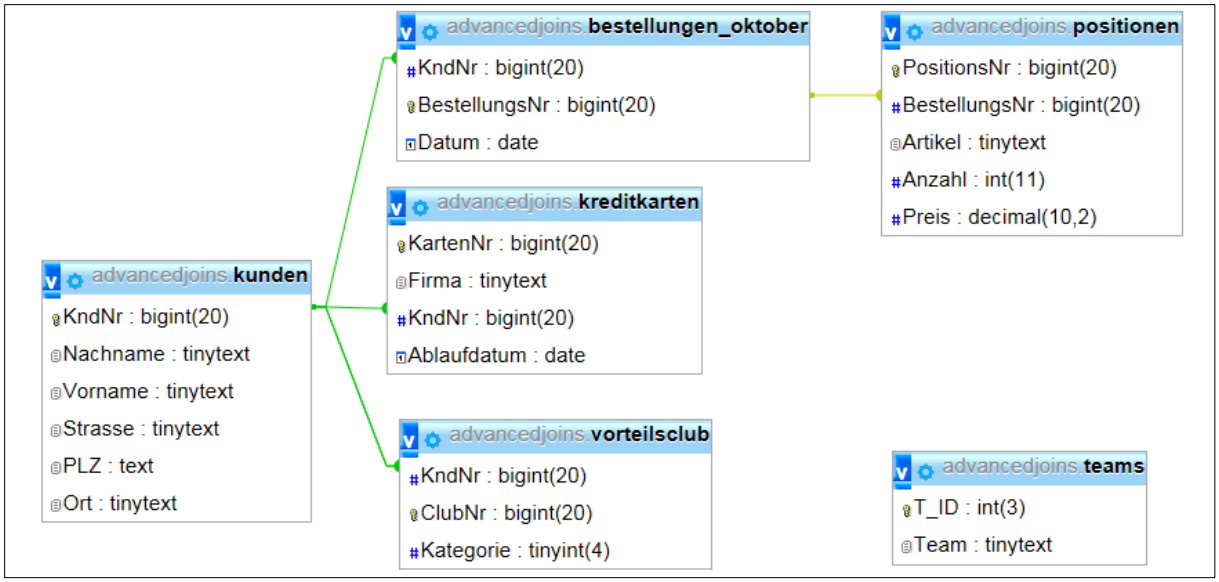

Abb. 1: Sechs Datenbanktabellen mit dem in phpMyAdmin eingebauten Werkzeug '*Designer*' darstellen

Durch die grafische Darstellung der Tabellen und Beziehungen (s. Abb. 1) ist das Schema der Datenbank mit dem Namen '*advancedjoins*' deutlich zu erkennen.

Die Erstellung einer 1:N-Beziehung[5] zwischen den Tabellen *Kunden* und *Kreditkarten* setzt beispielsweise voraus:

- Beide Tabellen beruhen auf den Speichersubsystem *InnoDB.*
- Die Spalte *KndNr* in der Tabelle *Kunden* muss als *Primärschlüssel* festgelegt sein (erkennbar durch das Schüsselsymbol vor dem Namen dieser Spalte).
- Die referenzierte Spalte *KndNr* in der Tabelle *Kreditkarten* muss indiziert[6] sein.
- Beide Spalten müssen genau den gleichen Datentyp besitzen, hier *bigint.*

Dadurch wird bestimmt, dass 1 bestimmter Kunde beliebig viele (also N) Kreditkarten besitzen darf. Alle übrigen Tabellenbeziehungen des Datenbankschemas sind analog festzulegen.

Dazu eignen sich die folgenden SQL-Anweisungen (s. Listing 1):

[5] Eine 1:N-Beziehung ist die Mutter aller Tabellenbeziehungen. Alle weiteren Tabellenbeziehungen sind Sonderfälle des genannten Typs.

[6] Ein Index in MySQL ist eine spezielle Datenstruktur, die das Auffinden von Datensätzen beschleunigt. Spalten die mittels *Index* gekennzeichnet werden, können doppelte Werte in der betreffenden Tabellenspalte enthalten.

```
# Spalte 'KndNr' in Tabelle 'Vorteilsclub' mit Index[7] versehen.
ALTER TABLE vorteilsclub
   ADD INDEX (KndNr);

# 1:N-Verknüpfung zwischen den Tabellen 'Kunden' und 'Vorteilsclub' herstellen[8].
ALTER TABLE vorteilsclub
   ADD FOREIGN KEY (KndNr)
   REFERENCES kunden(KndNr)
   ON DELETE RESTRICT
   ON UPDATE RESTRICT;

# Datentyp der Spalte 'KndNr' in der Tabelle 'Bestellungen_Oktober' ändern.
ALTER TABLE bestellungen_oktober
   MODIFY KndNr BIGINT
   NOT NULL;

# Spalte 'KndNr' in Tabelle 'Bestellungen_Oktober' mit Index versehen.
ALTER TABLE bestellungen_oktober
   ADD INDEX (KndNr);

# Datentyp der Spalte 'BestellungsNr' in Tabelle 'Bestellungen_Oktober' ändern
ALTER TABLE bestellungen_oktober
   MODIFY BestellungsNr BIGINT
   NOT NULL AUTO_INCREMENT;

# Spalte 'BestellungsNr' in Tabelle 'Positionen' mit Index versehen.
ALTER TABLE positionen
   ADD INDEX (BestellungsNr);

# 1:N-Verknüpfung zwischen den Tabellen 'Bestellungen_Oktober' und 'Positionen' herstellen.
ALTER TABLE positionen
   ADD FOREIGN KEY (BestellungsNr)
   REFERENCES bestellungen_oktober(BestellungsNr)
   ON DELETE RESTRICT
   ON UPDATE RESTRICT;

# Spalte 'KndNr' in Tabelle 'Kreditkarten' mit Index versehen.
ALTER TABLE kreditkarten
   ADD INDEX (KndNr);

# 1:N-Verknüpfung zwischen den Tabellen 'Kunden' und 'Kreditkarten' herstellen.
ALTER TABLE kreditkarten
   ADD FOREIGN KEY (KndNr)
   REFERENCES kunden(KndNr)
   ON DELETE RESTRICT
   ON UPDATE RESTRICT;
```

Listing 1: SQL-Anweisungen zur Verknüpfung und zur Darstellung der Tabellen mit dem Werkzeug '*Designer*'

Nicht übereinstimmende Datentypen in den Tabellenbeziehungen werden modifiziert (vgl. dazu die beiden *MODIFY*-Anweisungen in Listing 1 in Verbindung mit Tabelle 1 bzw. Tabelle 3).

[7] Index-Schlüssel (s. Listing 1) dienen nur zur Beschleunigung von Datenbankabfragen.

[8] Die Aktion *RESTRICT* ist Teil der *FOREIGN KEY*-Anweisung (s. Listing 1) und bestimmt das Verhalten der Detailtabelle sobald in der Haupttabelle der Primärschlüssel geändert wird. *RESTRICT* verweigert die Änderung auch in der Haupttabelle. Dadurch wird die referenzielle Integrität bei Anfüge- oder Löschabfragen bewahrt.

2.4.2 Beziehungen der ACCESS-Datenbanktabellen

Im Beziehungsfenster (s. Abb. 2) des Datenbanksystems ACCESS wird jede der sechs Tabellen in Form einer Feldliste angezeigt. Die Beziehungen werden durch Linien zwischen den Tabellen veranschaulicht:

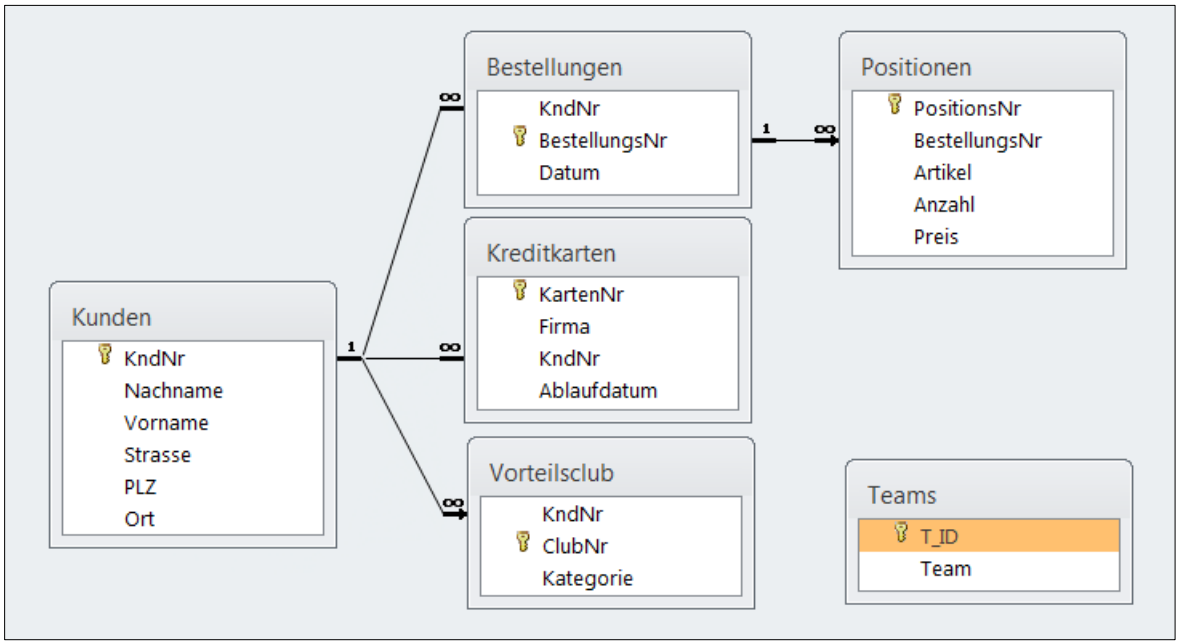

Abb. 2: Sechs ACCESS-Datenbanktabellen im Beziehungsfenster mit Darstellung der referenziellen Integrität

Der Typ der Beziehung zwischen zwei Tabellen wird oberhalb der jeweiligen Beziehungslinie angegeben, hier jeweils mit 1 und ∞. Es handelt sich also um 1:N-Beziehungen (engl.: *one-to-many relationships*), ausgehend von dem Primärschlüssel in der Spalte *KndNr* der Tabelle *Kunden* bzw. der Spalte *BestellungsNr* der Tabelle *Bestellungen*.

Zur Geltung kommt die referenzielle Integrität, wenn Datensätze in verknüpfte Datenbanktabellen hinzugefügt oder gelöscht werden sollen, also bei Anfüge- oder Löschabfragen.

3 Auswahlabfragen in ACCESS definieren und ausführen

3.1 Einfacher Self-Join

Die Tabelle *Kreditkarten* wird mit sich selbst verknüpft. Als Verknüpfungsfeld dient die Spalte *KndNr* der Tabelle *Kreditkarten*. *K1* und *K2* dienen als Aliasnamen für diese Tabelle.

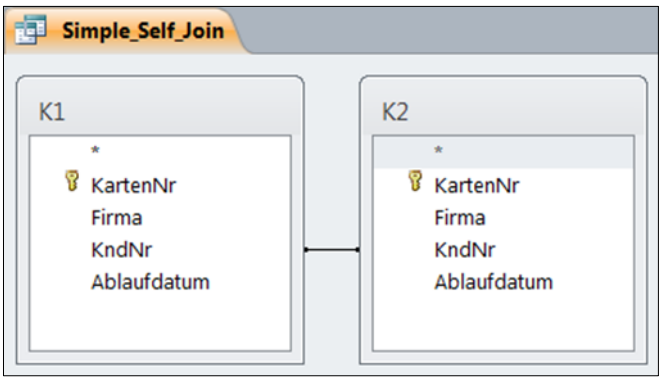

Abb. 3: Einfacher Self-JOIN der Tabelle *Kreditkarten*

Alle Kombinationen von Kundennummern und zugehörigen Kreditkarten-Firmen sollen ausgewählt werden.

Simple_Self_Join

```
SELECT
  K1.KndNr, K1.Firma,
  K2.KndNr, K2.Firma
FROM
  Kreditkarten AS K1
INNER JOIN
  Kreditkarten AS K2
ON
  K1.KndNr = K2.KndNr
ORDER BY
  K1.KndNr;
```

Abb. 4: SQL-Anweisungen für einfachen Self-JOIN

Ergebnis der Auswahlabfrage (9 Zeilen):

Simple_Self_Join

K1.Kndnr	K1.Firma	K2.Kndnr	K2.Firma
123457	VISA	123457	VISA
123458	VISA	123458	VISA
123458	VISA	123458	Diners Club
123458	Diners Club	123458	VISA
123458	Diners Club	123458	Diners Club
123459	American Express	123459	American Express
123459	American Express	123459	Mastercard
123459	Mastercard	123459	American Express
123459	Mastercard	123459	Mastercard

Abb. 5: Abfrageergebnis des einfachen Self-JOINs

3.2 Self-JOIN mit einschränkender WHERE-Klausel

Um zu ermitteln, welcher Kunde sowohl eine '*Mastercard*' als auch eine *'American Express*' besitzt, kann eine einschränkende WHERE-Klausel verwendet werden (s. Abb. 6):

Self_Join_Where

```
SELECT
  K1.KndNr, K1.Firma,
  K2.Kndnr, K2.Firma
FROM
  Kreditkarten AS K1
INNER JOIN
  Kreditkarten AS K2
ON
  K1.KndNr = K2.KndNr
WHERE
  K1.Firma = 'Mastercard' AND K2.Firma = 'American Express';
```

Abb. 6: Self-JOIN mit einschränkender WHERE-Klausel

Ergebnis der Auswahlabfrage (1 Zeile):

Self_Join_Where

K1.Kndnr	K1.Firma	K2.Kndnr	K2.Firma
123459	Mastercard	123459	American Express

Abb. 7: Ergebnis der vorstehenden Auswahlabfrage

4 Mehrere Tabellen mit JOIN verknüpfen - gleiche JOIN-Spalten

4.1 Mehrfachjoin verwenden

Im vorhergehenden Abschnitt wurden die Kundennummern von Kunden ermittelt, die sowohl über eine Kreditkarte der Firma '*Mastercard*' als auch eine der Marke *'American Express*' verfügen. Nunmehr soll zusätzlich herausgefunden werden, um welche(n) Kunden es sich handelt, d.h. Nachname, Vorname, Straße, PLZ und Ort.

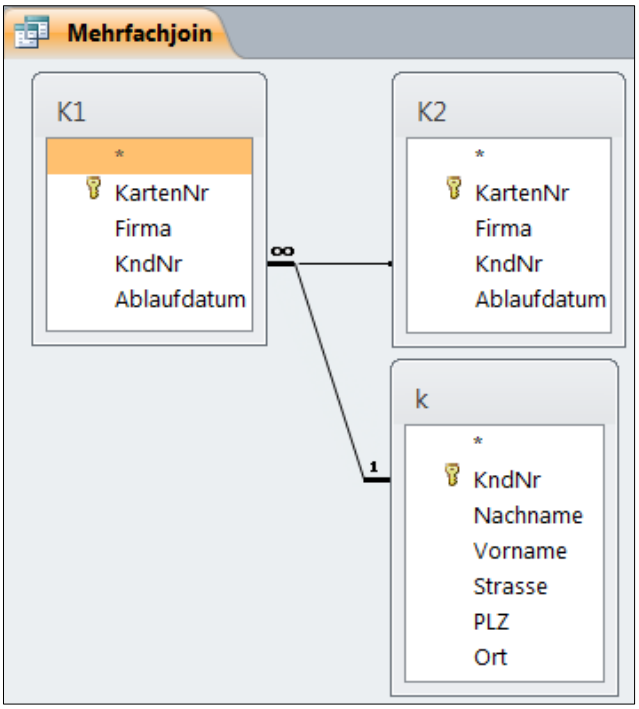

Abb. 8: Entwurfsansicht des Mehrfachjoins

Mehrfachjoin

```
SELECT
  K1.KndNr,
  k.Nachname, k.Vorname, k.Strasse, k.PLZ, k.Ort
FROM (
  Kreditkarten AS K1
INNER JOIN
  Kreditkarten AS K2
ON
  K1.KndNr = K2.KndNr
)
INNER JOIN
  Kunden AS k
ON
  k.KndNr = K1.KndNr
WHERE
  (((K1.Firma)='Mastercard') AND ((K2.Firma)='American Express'));
```

Abb. 9: Mehrfachjoin mit einschränkender WHERE-Klausel

Die Klammern (hinter *FROM* und nach der ersten *ON*-Klausel) legen die Reihenfolge der JOIN-Operationen fest. Wird darauf verzichtet, so entscheidet das Datenbankmanagementsystem in welcher Reihenfolge die JOINs ausgeführt werden.

Ergebnis der Auswahlabfrage (1 Zeile):

Mehrfachjoin

KndNr	Nachname	Vorname	Strasse	PLZ	Ort
123459	Schmidt	Hans	Hauptstraße 2	98765	Anderswo

Abb. 10: Abfrageergebnis des vorstehenden Mehrfachjoins

4.2 Komplexere Auswahlabfragen

4.2.1 Kreditkartendaten der Mitglieder des Vorteilsclubs ermitteln

Kreditkartendaten der Clubmitglieder

```
SELECT
  c.KndNr, c.Firma, c.KartenNr, c.Ablaufdatum
FROM
  Kreditkarten AS c
INNER JOIN
  Vorteilsclub AS v
ON
  c.KndNr = v.KndNr;
```

Abb. 11: Kreditkartendaten der Clubmitglieder mit SQL ermitteln

Ergebnis dieser Auswahlabfrage (2 Zeilen):

Erster_Zwischenschritt

KndNr	Firma	KartenNr	Ablaufdatum
123458	Diners Club	12349	01.02.2022
123458	VISA	12350	01.03.2017

Abb. 12: Ergebnis der vorstehenden Auswahlabfrage

Der INNER JOIN[9] der Tabelle *Kreditkarten* mit der Tabelle *Vorteilsclub* mittels JOIN-Spalte *KndNr* bewirkt die Auswahl der Kreditkarteninformationen der Vorteilsclubmitglieder. In diesem Fall ist es der Kunde mit der Nummer 123458, also *Lieschen Müller* (s. Tabelle 3). Sie besitzt zwei Kreditkarten (vgl. Tabelle 2) der Firmen *Diners Clu*b und *VISA.*

4.2.2 Abfrageergebnis mit der Tabelle *Kunden* verknüpfen

Das Ergebnis der vorstehenden Auswahlabfrage soll nun auf geeignete Weise mit der Tabelle *Kunden* verbunden werden. Die Detaildaten der Tabelle *Kunden* werden benötigt, auch wenn keine Entsprechungen in der Menge der Kreditkarteninformationen (der Vorteilsclubmitglieder) vorliegen.

Diese Aufgabe kann z. B. mit einem LEFT JOIN der Tabelle *Kunden* und dem Abfrageergebnis (s. Abb. 12) gelöst werden:

[9] Mit der SQL-Anweisung *INNER JOIN* (als sog. *Vergleichsabfrage*) können zwei Datenbanktabellen miteinander verglichen werden, die über ein identisches Feld verknüpft sind. Dieser Typ ist der wichtigste SQL-JOIN-Vorgang.

Complex_Left_Join

```
SELECT
  k.KndNr, k.Nachname, k.Vorname, k.Strasse, k.PLZ, k.Ort,
  c.Firma, c.KartenNr, c.Ablaufdatum
FROM (
  Kunden AS k
LEFT JOIN
  Kreditkarten AS c
ON
  k.KndNr = c.KndNr
)
INNER JOIN
  Vorteilsclub AS v
ON
  k.KndNr = v.KndNr;
```

Abb. 13: Kunden- und Kreditkarteninformationen der Vorteilsclubmitglieder ermitteln

Der INNER JOIN der Tabellen *Kreditkarten* (Alias **c**) und *Vorteilsclub* (Alias **v**) erzeugt eine Tabelle mit den Kundennummern (*KndNr*) der Kunden, die sowohl Kreditkarten besitzen als auch dem Vorteilsclub angehören. Der eingeklammerte äußere LEFT JOIN bewirkt, dass nur die Detaildaten der gewünschten Kunden ausgegeben werden.

Ergebnis dieser Auswahlabfrage (4 Zeilen):

Complex_Left_Join

KndNr	Nachname	Vorname	Strasse	PLZ	Ort	Firma	KartenNr	Ablaufdatum
123456	Mustermann	Max	Musterweg 1	12345	Musterstadt			
123458	Müller	Lieschen	Beispielweg 3	23987	Irgendwo	Diners Club	12349	01.02.2022
123458	Müller	Lieschen	Beispielweg 3	23987	Irgendwo	VISA	12350	01.03.2017
123460	Becker	Heinz	Mustergasse 4	12543	Musterdorf			

Abb. 14: Ergebnis der vorstehenden Auswahlabfrage

4.3 Variation der Reihenfolge und JOIN-Spalte

In dem bereits mehrmals erwähnten Beitrag [1] wird demonstriert, dass …

- die Reihenfolge der Tabellen sowie
- die Auswahl der JOIN-Spalten

sich auf das Abfrageergebnis auswirken, sobald ein LEFT JOIN oder RIGHT JOIN angewandt wird. Zum Beweis dieser Tatsache werden in [1] Variationen der gleichen JOIN-Operationen präsentiert, die alle *nicht* das gewünschte Abfrageergebnis ergeben.

5 Mehrere Tabellen verknüpfen - unterschiedliche JOIN-Spalten

Eine Übersicht über die *Bestellungen* mit Detailinformationen der *Kunden* wird benötigt, die diese Bestellungen getätigt haben. Die benötigten Daten befinden sich in folgenden 3 Tabellen:

- *Kunden*
- *Bestellungen* (Alias: *Bestellungen_Oktober*) und
- *Position*

Die Verknüpfung dieser Tabellen wurde weiter oben bereits dargestellt (s. Abb. 2). Folglich gibt es zwei mögliche Reihenfolgen:

- Erst die Tabellen *Kunden* und *Bestellungen* miteinander verknüpfen, anschließend das Ergebnis mit der Tabelle *Positionen*,
- Erst die Tabellen *Positionen* und *Bestellungen* verknüpfen, dann das Resultat mit der Tabelle *Kunden* verbinden.

5.1 Bestellungen mit Kundendetails

Bestelldaten abfragen

```
SELECT
   p.Artikel,  p.Anzahl,  p.Preis,
   b.Datum,
   k.Nachname, k.Vorname, k.Strasse, k.PLZ, k.Ort
FROM
   Positionen AS p
INNER JOIN (
   Bestellungen AS b
INNER JOIN
   Kunden As k
ON
   b.KndNr = k.KndNr
)
ON
   p.BestellungsNr = b.BestellungsNr;
```

Abb. 15: Bestelldaten abfragen mit zwei INNER JOIN

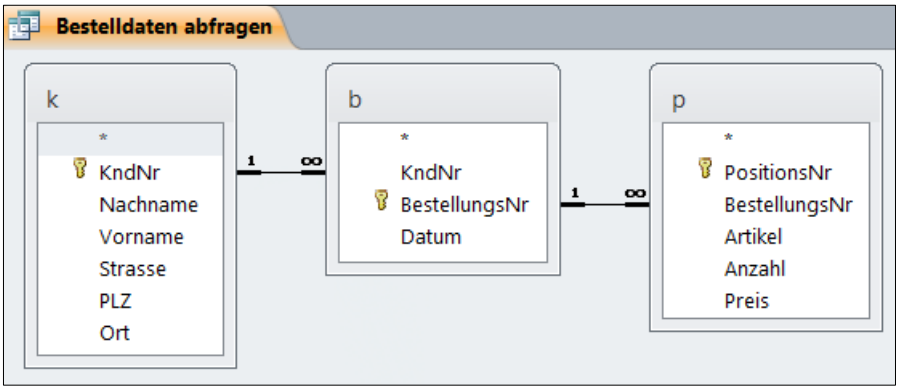

Abb. 16: Entwurfsansicht für vorstehende Auswahlabfrage

Ergebnis dieser Auswahlabfrage (7 Zeilen):

Bestelldaten abfragen

Artikel	Anzahl	Preis	Datum	Nachname	Vorname	Strasse	PLZ	Ort
CD-Player	2	49.95	15.10.2014	Mustermann	Max	Musterweg 1	12345	Musterstadt
DVD-Player	3	59.95	15.10.2014	Mustermann	Max	Musterweg 1	12345	Musterstadt
CD xyz	10	15.95	15.10.2014	Mustermann	Max	Musterweg 1	12345	Musterstadt
DVD abc	5	9.95	15.10.2014	Mustermann	Max	Musterweg 1	12345	Musterstadt
CD-Payer	1	51.20	16.10.2014	Mustermann	Max	Musterweg 1	12345	Musterstadt
CD xyz extra	20	16.25	16.10.2014	Mustermann	Max	Musterweg 1	12345	Musterstadt
DVD-Player	1	64.95	16.10.2014	Musterfrau	Katrin	Musterstraße 7	12345	Musterstadt

Abb. 17: Bestellungen mit Kundendetails

- Bestellungen ohne Bestellpositionen sind uninteressant.
- Bestellungen ohne Besteller sind ebenfalls uninteressant

Bei einer Auswahlabfrage, die nur INNER JOINs verwendet (s. Abb. 15), spielt die Reihenfolge der JOIN-Operationen für das Abfrageergebnis *keine* Rolle.[10]

5.2 Kunden mit und ohne Bestellungen abfragen

Die vorige Aufgabenstellung wird erweitert. Nunmehr sollen alle Kunden mit *Nachname* und *Vorname* ausgewählt werden, egal, ob Bestellungen registriert wurden oder nicht:

Kunden mit und ohne Bestellungen

```
SELECT
  k.Nachname, k.Vorname,
  p.Artikel, p.Anzahl, p.Preis,
  b.Datum
FROM
  Kunden AS k
LEFT JOIN (
  Bestellungen AS b
LEFT JOIN
  Positionen AS p
ON
  p.BestellungsNr = b.BestellungsNr
)
ON
  k.KndNr = b.KndNr;
```

Abb. 18: Kunden mit und ohne Bestellungen abfragen

Im Original-Beitrag [1] beinhaltet der zweite JOIN-Ausdruck einen *INNER JOIN*[11] (vgl. Listing 2). Dieser JOIN-Typ löst jedoch in ACCESS einen Fehler aus (s. Abb. 19):

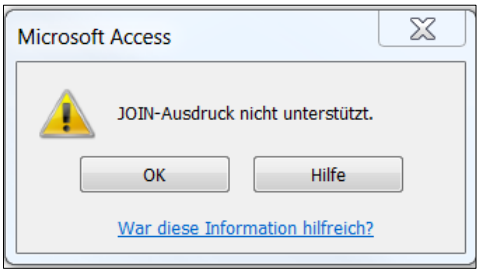

Abb. 19: Fehlermeldung in ACCESS

Wird stattdessen ein *LEFT JOIN* angewandt, läuft die Auswahlabfrage (s. Abb. 18) fehlerfrei ab und erzeugt das erwartete Abfrageergebnis (s. Abb. 20 und vgl. auch Abb. 21).

Ergebnis dieser Auswahlabfrage (10 Zeilen):

Kunden mit und ohne Bestellungen

Nachname	Vorname	Artikel	Anzahl	Preis	Datum
Mustermann	Max	CD-Player	2	49.95	15.10.2014
Mustermann	Max	DVD-Player	3	59.95	15.10.2014
Mustermann	Max	CD xyz	10	15.95	15.10.2014
Mustermann	Max	DVD abc	5	9.95	15.10.2014
Mustermann	Max	CD-Payer	1	51.20	16.10.2014
Mustermann	Max	CD xyz extra	20	16.25	16.10.2014
Musterfrau	Katrin	DVD-Player	1	64.95	16.10.2014
Müller	Lieschen				
Schmidt	Hans				
Becker	Heinz				

Abb. 20: Alle Kunden mit und ohne Bestellungen in ACCESS abfragen

[10] Anders als bei INNER JOINs ist bei OUTER JOINs die Reihenfolge der Datenbanktabellen in SQL-Anweisungen zu beachten.

[11] „Eine LEFT JOIN- oder RIGHT JOIN-Operation kann in einer INNER JOIN-Operation geschachtelt werden, wohingegen eine INNER JOIN-Operation *nicht* in einer LEFT JOIN- oder RIGHT JOIN-Operation geschachtelt werden kann.“ (Quelle: [3]). Vgl. dazu Listing 2.

Nachname	Vorname	Artikel	Anzahl	Preis	Datum
Mustermann	Max	CD-Player	2	49.95	2014-10-15
Mustermann	Max	DVD-Player	3	59.95	2014-10-15
Mustermann	Max	CD xyz	10	15.95	2014-10-15
Mustermann	Max	DVD abc	5	9.95	2014-10-15
Mustermann	Max	CD-Player	1	51.20	2014-10-16
Mustermann	Max	CD xyz extra	20	16.25	2014-10-16
Musterfrau	Katrin	DVD-Player	1	64.95	2014-10-16
Müller	Lieschen	*NULL*	*NULL*	*NULL*	*NULL*
Schmidt	Hans	*NULL*	*NULL*	*NULL*	*NULL*
Becker	Heinz	*NULL*	*NULL*	*NULL*	*NULL*

Abb. 21: Alle Kunden mit und ohne Bestellungen in MySQL abfragen

```
SELECT Nachname, Vorname, Artikel, Anzahl, Preis, Datum
FROM
   Kunden
LEFT JOIN (
   Bestellungen_Oktober
INNER JOIN
   Positionen
ON
   Positionen.BestellungsNr = Bestellungen_Oktober.BestellungsNr
)
ON
   Kunden.KndNr = Bestellungen_Oktober.KndNr;
```

Listing 2: SQL-Auswahlabfrage für vorstehendes Ergebnis (Quelle: [1])

6 Theta-JOIN

Ein *Theta-JOIN* ermöglicht beliebige Vergleichsbeziehungen. Der *Equi-JOIN* (Bedingung: Gleichheit) ist ein Spezialfall des *Theta-JOIN*.

6.1 Einfachrunde

Angenommen, in der Datenbanktabelle *Teams* befinden sich die Namen von Mannschaften, die in einer Einfachrunde gegeneinander antreten sollen. Das Heimrecht wird ignoriert. Ein *Theta-JOIN* liefert die Begegnungen der Einfachrunde:

Einfachrunde

```
SELECT
  h.Team AS Heim,
  g.Team AS Gast
FROM
  Teams AS h
INNER JOIN
  Teams AS g
ON
  h.T_ID > g.T_ID;
```

Abb. 22: Auswahlabfrage der Einfachrunde

Ergebnis der vorstehenden Auswahlabfrage (6 Zeilen):

Einfachrunde

Heim	Gast
München	Hamburg
Berlin	Hamburg
Köln	Hamburg
Berlin	München
Köln	München
Köln	Berlin

Abb. 23: Begegnungen der Einfachrunde

6.2 Doppelrunde

Für eine Doppelrunde, also Hin- und Rückspiel, wird der Ungleichheitsoperator <> verwendet:

Doppelrunde

```
SELECT
  h.Team AS Heim,
  g.Team AS Gast
FROM
  Teams AS h
INNER JOIN
  Teams AS g
ON
  h.T_ID <> g.T_ID;
```

Abb. 24: Auswahlabfrage der Doppelrunde

Ergebnis der vorstehenden Auswahlabfrage (12 Zeilen):

Doppelrunde

Heim	Gast
München	Hamburg
Berlin	Hamburg
Köln	Hamburg
Hamburg	München
Berlin	München
Köln	München
Hamburg	Berlin
München	Berlin
Köln	Berlin
Hamburg	Köln
München	Köln
Berlin	Köln

Abb. 25: Begegnungen der Doppelrunde

7 Ergebnis des Tests auf Kompatibilität

Wenn die Regeln (vgl. Seite 4 f) zur Verknüpfung von Datenbanktabellen eingehalten werden, führen SQL-Auswahlabfragen in MySQL und ACCESS zum gleichen Ergebnis. Das entspricht den Erwartungen. Nur eine einzige Unverträglichkeit (s. Abb. 19, S. 14) wurde entdeckt.

8 Anhang: SQL-JOIN-Vorgänge

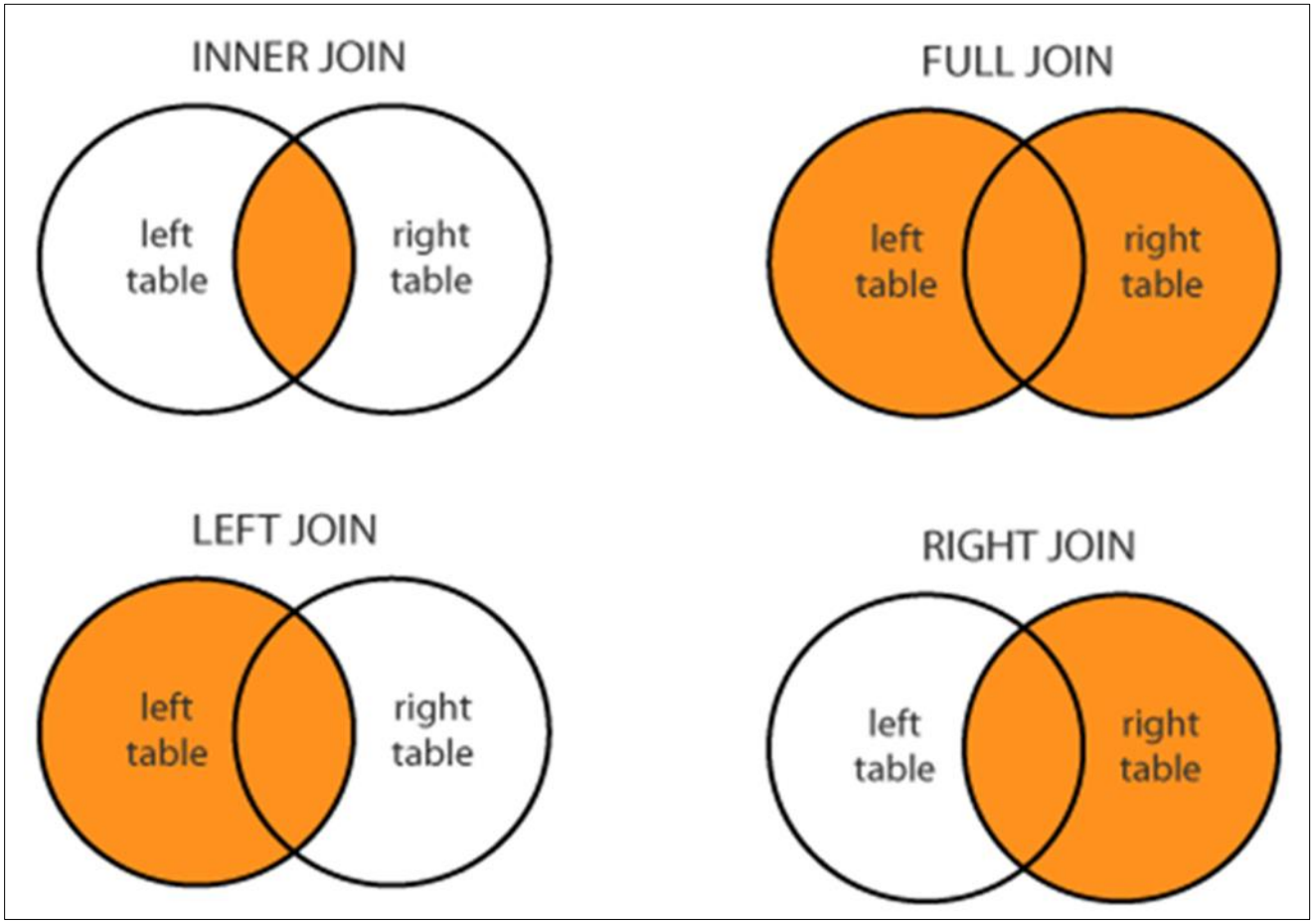

Abb. 26: Visuelle Erklärung von SQL-JOIN-Vorgängen mit Mengendiagrammen

- ***INNER JOIN***: Erzeugt nur diejenigen Datensätze, die in den Tabellen **left_table** und **right_table** übereinstimmen.
- ***FULL (OUTER) JOIN***: Erzeugt die Menge aller Datensätze in den Tabellen **left_table** und **right_table** mit übereinstimmenden Datensätzen von beiden Seiten, sofern vorhanden. Wenn keine Übereinstimmung vorhanden ist, enthält die fehlende Seite ***NULL***.
- ***LEFT (OUTER) JOIN***: Erzeugt alle Datensätzen aus Tabelle **left_table** mit passenden Datensätzen (soweit vorhanden) in Tabelle **right_table**. Wenn keine Übereinstimmung vorhanden ist, enthält die rechte Seite ***NULL***.

  Um nur die Datensätze in der Tabelle ***left_table*** auszuwählen, aber nicht in Tabelle ***right_table***, wird der gleiche ***LEFT (OUTER) JOIN*** ausgeführt. Die nicht benötigten Datensätze der Tabelle ***right_table*** werden aber über eine ***WHERE***-Klausel ausgeschlossen.

- ***RIGHT (OUTER) JOIN***: Erzeugt alle Datensätzen aus Tabelle ***right_table*** mit passenden Datensätzen (soweit vorhanden) in Tabelle ***left_table***. Wenn keine Übereinstimmung vorhanden ist, enthält die linke Seite ***NULL***.

  Um nur die Datensätze in der Tabelle ***right_table*** auszuwählen, aber nicht in Tabelle ***left_table***, wird der gleiche ***RIGHT (OUTER) JOIN*** ausgeführt. Die nicht benötigten Datensätze der Tabelle ***left_table*** werden aber über eine ***WHERE***-Klausel ausgeschlossen.

Abbildungen

Listings

Quellen- und Literaturverzeichnis

[1] o. V., „Datenbank/Fortgeschrittene Jointechniken,“ selfhtml.org, 30 01 2018. [Online]. Available: https://wiki.selfhtml.org/wiki/Datenbank/Fortgeschrittene_Jointechniken. [Zugriff am 12 09 2018].

[2] o. V., „Datenbank/Einführung in Joins,“ selfhtml.org, 08 05 2018. [Online]. Available: https://wiki.selfhtml.org/wiki/Datenbank/Einf%C3%BChrung_in_Joinss. [Zugriff am 12 09 2018].

[3] o. V., „INNER JOIN-Vorgang,“ Microsoft, 2018. [Online]. Available: https://support.office.com/de-de/article/inner-join-vorgang-b9e73ab6-884a-403e-9f22-cb502feae36a. [Zugriff am 10 10 2018].