

Um den **Beziehungstyp zu bestimmen**, geht man wie folgt vor:

- 1) **Erstens** Als Ausgangslage braucht es zwei Entitätsmengen und eine Beschreibung der Beziehung (Verb, Substantiv oder kurzer Satz). Beispiel: Entitätsmenge 1 = Kunde, Entitätsmenge 2 = Ort, Beziehung = wohnt bzw. wohnen
- 2) **Zweitens** Um die Kardinalität auf der rechten Seite (Ort) zu bestimmen, geht man von der Menge aller Kunden aus: Jeder Kunde wohnt an genau einem Ort -> Kardinalität = 1.
- 3) **Drittens** Um die Kardinalität auf der linken Seite (Kunde) zu bestimmen, geht man von der Menge aller Orte aus: An jedem Ort wohnen 0, 1 oder mehrere Kunden -> Kardinalität = mc.
- 4) **Demnach** ist die Beziehung mc:1

ANOMALIEN sind: **Update-Anomalie**, wenn gleiche Daten unterschiedlich benannt werden und dadurch bei Updates nur Teile der DB verändert werden. **Delete-Anomalie**, wenn beim Löschen einer Entität ungewollt auch andere Informationen verloren gehen. **Insert-Anomalie**, wenn Attribute fehlen, mehrere Primärschlüssel dieselbe Information tragen oder ein Objekt mehrere Schlüssel trägt.

ABHÄNGIGKEITEN

| RNr | PNr | Datum | Name | KNr | Adresse | Artikel | ANr | Anz | Preis |
|-----|-----|------------|--------------|-----|-----------------------|-----------|-------|-----|-------|
| 123 | 1 | 29.01.2020 | Marc Muster | 11 | Talweg 11, 3000 Bern | Monitor | 2-023 | 10 | 200 |
| 123 | 2 | 29.01.2020 | Marc Muster | 11 | Talweg 11, 3000 Bern | Maus | 4-023 | 12 | 25 C |
| 123 | 3 | 29.01.2020 | Marc Muster | 11 | Talweg 11, 3000 Bern | Bürostuhl | 5-023 | 1 | 450 |
| 124 | 1 | 30.01.2020 | Erika Muster | 12 | Bergweg 21, 3000 Bern | Notebook | 1-023 | 2 | 1200 |
| 124 | 2 | 30.01.2020 | Erika Muster | 12 | Bergweg 21, 3000 Bern | Maus | 4-023 | 2 | 25 C |

- 1) **funktionale Abhängigkeit** liegt dann vor, wenn ein Nichtschlüsselattribut nur von einem Teil der Attribute eines zusammengesetzten Schlüsselkandidaten funktional abhängig ist. Beispiel: Das Datum ist funktional abhängig von der Rechnungsnummer *RNr*, nicht aber vom gesamten Schlüssel *RNr* und *PNr*.
- 2) **vollständig funktionale (volle) Abhängigkeit** liegt dann vor, wenn ein Nichtschlüsselattribut nicht nur von einem Teil der Attribute eines zusammengesetzten Schlüsselkandidaten funktional abhängig ist, sondern von allen Teilen des Primärschlüssels (bzw von mehreren Nichtschlüsselattributen) . Die vollständig funktionale Abhängigkeit wird mit der zweiten Normalform erreicht. Beispiel: Der Artikel ist vollständig funktional abhängig vom gesamten Primärschlüssel *RNr* und *PNr*.
- 3) **transitive Abhängigkeit** liegt dann vor, wenn ein Nichtschlüsselattribut funktional abhängig ist von einem andern Nichtschlüsselattribut. Es ist transitiv oder indirekt abhängig vom Primärschlüssel. Beispiel: Name und Adresse sind abhängig von der Kundennummer *KNr*. Somit sind diese Attribute transitiv abhängig vom Schlüsselkandidaten.

NORMALFORMEN- Definition Nullte Normalform: Eine Tabelle befindet sich in der nullten Normalform, wenn alle Datenelemente der realen Welt darin zusammengefasst und aufgelistet sind. **erste Normalform:** wenn alle Daten atomar vorliegen und alle Tabellenspalten gleichartige Werte enthalten, also muss jedes Attribut atomar sein.

Definition zweite Normalform: wenn sie sich in der ersten Normalform befindet und zudem jedes einzelne Nichtschlüsselattribut vom Primärschlüssel voll funktional abhängig ist. Also darf kein Attribut voll abhängig sein!

Definition dritte Normalform: wenn er sich in der zweiten Normalform befindet und es zudem *keine transitiven Abhängigkeiten* gibt.

- **Nullte Normalform:** Eine Tabelle ist in der nullten Normalform, wenn alle Daten der realen Welt darin unstrukturiert aufgefasst sind. In so einer Tabelle können in einem Attributwert auch mehrere Informationen gelistet sein (was in einer DB nicht vorkommen sollte (atomarität)).
- **Erste Normalform:** Wenn alle Daten atomar vorliegen (pro Attributwert nur ein Wert), ist die Tabelle in der ersten Normalform. Man trennt also alle Informationen in die kleinst mögliche Einheit auf (die Sinn ergibt) und gibt ihr eine eigene Spalte (macht sie zu einem eigenen Attribut)
- **Zweite Normalform:** Wenn die Attribute einer Tabelle nur von einem Primärschlüssel abhängig sind, ist sie automatisch bereits in der zweiten Normalform. Der Unterschied zur ersten Normalform besteht darin, dass jedes Attribut von den Schlüsseln abhängig ist. Es können jedoch immernoch Abhängigkeiten von Attributen untereinander bestehen wie zb. die Abhängigkeit von Name zu Kundennummer. Um eine Tabelle in die zweite normalform umzuwandeln, muss man alle Nichtschlüsselattribute, die nicht vollständig vom Primärschlüssel abhängig sind, in sperate Tabellen umlagern
- **Dritte Normalform:** In Tabellen der dritten Normalform sind alle Attribute der Tabelle nur noch von ihrem Primärschlüssel und nicht mehr von anderen Attributen abhängig. Es werden sperate Tabellen gemacht, die nur dafür da sind, die Schlüssel verschiedener Tabellen (Einkäufe, Kunden) zu verbinden.

SQL-BEFEHLE: LIMIT<Zahl>-Wie viele Werte zurückgegeben werden einschränken. DISCTINCT-zeige jeden Wert nur einmal an. <attribut> | <attribut> -gibt die beiden Attribute innerhalb einer Spalte aus.

```

SELECT Strasse, PLZ, Ort FROM Lernende l
JOIN Orte o ON l.ort_id = o.id
UNION SELECT Strasse, PLZ, Ort FROM Firmen f
JOIN Orte o ON f.ort_id = o.id
ORDER BY Ort, Strasse;

```

Aggregatsfunktionen, die normalerweise einen einzelnen Wert zurückgeben, kann man mit GROUP BY diese Werte pro Attribut xyz ausgeben lassen (zb sum(id) pro Ort).

Konzeptionelles Modell-Attribute, Entitätsmengen, Beziehungen. Datenobjekte identifizieren->Merkmale (Attribute) der Datenobjekte definieren->Von Datenobjekten zu Tabellen->Beziehungen zwischen Tabellen festlegen

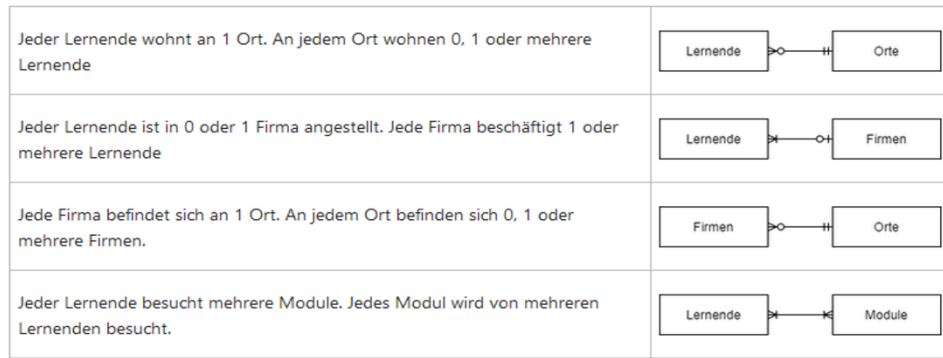
Logisches Modell->Schlüssel, Datentypen, Zwischentabellen.

JOIN-IMMER ON VERWENDEN- Inner Join/Join: Gibt nur das zurück, was in beiden Datensätzen vorkommt. Das passiert nur, wenn nicht alle Datensätze die ON/WHERE Klausel erfüllen. Sonst wird alles retourniert. LEFT JOIN: Gibt alle Inhalte der linken Tabelle. Muss nur verwendet werden, wenn Inhalte der linken

und der rechten Tabelle nicht übereinstimmen (fehlende Fremdschlüssel können sich so trotzdem anzeigen lassen). UNION – Kombiniert zwei Tabellen, deren Attribute identisch sind. Wird wie ein weiteres Select nach dem ersten Select

verwendet, mit Angabe der Attribute und allem.

SUM(),AVG(),MIN(),MAX(), ORDER BY <attribut>, ASC/DESC. WHERE-Befehl zum Filtern. BETWEEN <wert1> AND <wert2>. LIKE-lässt einen Platzhalter wie «_» (ein Zeichen) oder «%» (mehrere Zeichen – vorname like «b%»- findet vornamen, die mit b beginnen) verwenden. GROUP BY-Bei



```

SELECT l.Vorname, l.Nachname, l.Strasse, f.Firma
FROM Lernende AS l
LEFT JOIN Firmen AS f
ON l.firma_id = f.id

```

Füge Firmennamen zu den Lernenden hinzu, aber so, dass auch lernende ohne Firma angezeigt werden (left join)

```

SELECT count(*), jahr from lernende group by jahr
order by count(*) desc

```

Zähle die Anzahl Personen pro Jahr und sortiere nach Jahren absteigend

SELECT Query

```

SELECT col1, col2
FROM table
JOIN table2 ON table1.col = table2.col
WHERE condition
GROUP BY column_name
HAVING condition
ORDER BY col1 ASC|DESC;

```

SELECT Keywords

| | |
|---------------------------------------------------------------|----------------------------------------------------------------------------------|
| DISTINCT: Removes duplicate results | SELECT DISTINCT product_name FROM product; |
| BETWEEN: Matches a value between two other values (inclusive) | SELECT product_name FROM product WHERE price BETWEEN 50 AND 100; |
| IN: Matches to any of the values in a list | SELECT product_name FROM product WHERE category IN ('Electronics', 'Furniture'); |
| LIKE: Performs wildcard matches using _ or % | SELECT product_name FROM product WHERE product_name LIKE 'Desk%'; |

Joins

```

SELECT t1.*, t2.*
FROM t1
join_type t2 ON t1.col = t2.col;

```

| Table 1 | Table 2 |
|---------|---------|
| A | A |
| B | B |
| C | |
| | D |

INNER JOIN: show all matching records in both tables.

| | |
|---|---|
| A | A |
| B | B |

LEFT JOIN: show all records from left table, and any matching records from right table.

| | |
|---|---|
| A | A |
| B | B |
| C | |

RIGHT JOIN: show all records from right table, and any matching records from left table.

| | |
|---|---|
| A | A |
| B | B |
| | D |

FULL JOIN: show all records from both tables, whether there is a match or not.

| | |
|---|---|
| A | A |
| B | B |
| C | |
| | D |

| Commands / Clauses | Joins | Examples | | | | | | | | |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|------------------------------------------------------------|----------------------------------------------|--|--|
| SELECT Select data from database FROM Specify table we're pulling from WHERE Filter query to match a condition AS Rename column or table with alias JOIN Combine rows from 2 or more tables AND Combine query conditions. All must be met OR Combine query conditions. One must be met LIMIT Limit rows returned. See also FETCH & TOP IN Specify multiple values when using WHERE CASE Return value on a specified condition IS NULL Return only rows with a NULL value LIKE Search for patterns in column COMMIT Write transaction to database ROLLBACK Undo a transaction block ALTER TABLE Add/Remove columns from table UPDATE Update table data CREATE Create TABLE, DATABASE, INDEX or VIEW DELETE Delete rows from table INSERT Add single row to table DROP Delete TABLE, DATABASE, or INDEX GROUP BY Group data into logical sets ORDER BY Set order of result. Use DESC to reverse order HAVING Same as WHERE but filters groups COUNT Count number of rows SUM Return sum of column AVG Return average of column MIN Return min value of column MAX Return max value of column | <p>a INNER JOIN b</p> <p>a LEFT JOIN b</p> <p>a RIGHT JOIN b</p> <p>a FULL OUTER JOIN b</p> | <p>Select all columns with filter applied</p> <pre> SELECT * FROM tbl WHERE col > 5; </pre> <p>Select first 10 rows for two columns</p> <pre> SELECT col1, col2 FROM tbl LIMIT 10; </pre> <p>Select all columns with multiple filters</p> <pre> SELECT * FROM tbl WHERE col1 > 5 OR col2 < 2; </pre> <p>Select all rows from col1 & col2 ordering by col1</p> <pre> SELECT col1, col2 FROM tbl ORDER BY 1; </pre> <p>Return count of rows in table</p> <pre> SELECT COUNT(*) FROM tbl; </pre> <p>Return sum of col1</p> <pre> SELECT SUM(col1) FROM tbl; </pre> <p>Return max value for col1</p> <pre> SELECT MAX(col1) FROM tbl; </pre> <p>Compute summary stats by grouping col2</p> <pre> SELECT AVG(col1) FROM tbl GROUP BY col2; </pre> <p>Combine data from 2 tables using left join</p> <pre> SELECT * FROM tbl1 AS t1 LEFT JOIN tbl2 AS t2 ON t2.col1 = t1.col1; </pre> <p>Aggregate and filter result</p> <pre> SELECT col1, COUNT(*) AS total FROM tbl GROUP BY col1 HAVING COUNT(*) > 10; </pre> <p>Implementation of CASE statement</p> <pre> SELECT col1, CASE WHEN col1 > 10 THEN 'more than 10' WHEN col1 < 10 THEN 'less than 10' ELSE '10' END AS NewColumnName FROM tbl; </pre> | | | | | | | | |
| <h3>Data Definition Language</h3> <table border="1"> <tr> <th>CREATE</th> <th>ALTER</th> </tr> <tr> <td> <pre> CREATE DATABASE MyDatabase; CREATE TABLE MyTable (id int, name varchar(10)); CREATE INDEX IndexName ON TableName(col1); </pre> </td> <td> <pre> ALTER TABLE MyTable DROP COLUMN col5; ALTER TABLE MyTable ADD col5 int; DROP DATABASE MyDatabase; DROP TABLE MyTable; </pre> </td> </tr> </table> | CREATE | ALTER | <pre> CREATE DATABASE MyDatabase; CREATE TABLE MyTable (id int, name varchar(10)); CREATE INDEX IndexName ON TableName(col1); </pre> | <pre> ALTER TABLE MyTable DROP COLUMN col5; ALTER TABLE MyTable ADD col5 int; DROP DATABASE MyDatabase; DROP TABLE MyTable; </pre> | <h3>Order Of Execution</h3> <ol style="list-style-type: none"> FROM WHERE GROUP BY HAVING SELECT ORDER BY LIMIT | | | | | |
| CREATE | ALTER | | | | | | | | | |
| <pre> CREATE DATABASE MyDatabase; CREATE TABLE MyTable (id int, name varchar(10)); CREATE INDEX IndexName ON TableName(col1); </pre> | <pre> ALTER TABLE MyTable DROP COLUMN col5; ALTER TABLE MyTable ADD col5 int; DROP DATABASE MyDatabase; DROP TABLE MyTable; </pre> | | | | | | | | | |
| <h3>Data Manipulation Language</h3> <table border="1"> <tr> <th>UPDATE</th> <th>INSERT</th> </tr> <tr> <td> <pre> UPDATE MyTable SET col1 = 50 WHERE col2 = 'something'; </pre> </td> <td> <pre> INSERT INTO MyTable (col1, col2) VALUES ('value1', 'value2'); </pre> </td> </tr> <tr> <th>DELETE</th> <th>SELECT</th> </tr> <tr> <td> <pre> DELETE FROM MyTable WHERE col1 = 'something'; </pre> </td> <td> <pre> SELECT col1, col2 FROM MyTable; </pre> </td> </tr> </table> | UPDATE | INSERT | <pre> UPDATE MyTable SET col1 = 50 WHERE col2 = 'something'; </pre> | <pre> INSERT INTO MyTable (col1, col2) VALUES ('value1', 'value2'); </pre> | DELETE | SELECT | <pre> DELETE FROM MyTable WHERE col1 = 'something'; </pre> | <pre> SELECT col1, col2 FROM MyTable; </pre> | | |
| UPDATE | INSERT | | | | | | | | | |
| <pre> UPDATE MyTable SET col1 = 50 WHERE col2 = 'something'; </pre> | <pre> INSERT INTO MyTable (col1, col2) VALUES ('value1', 'value2'); </pre> | | | | | | | | | |
| DELETE | SELECT | | | | | | | | | |
| <pre> DELETE FROM MyTable WHERE col1 = 'something'; </pre> | <pre> SELECT col1, col2 FROM MyTable; </pre> | | | | | | | | | |

>, <, = und => können auch bei Buchstaben verwendet werden.

