Programmierung von Systemen – 12 – SQL 1

Matthias Tichy & Stefan Götz | SoSe 2020

Ziele

- Entwicklung von SQL kennen
- Unterschiede zwischen dem Erlernen von Sprachen und Programmiersprachen
- Umsetzung der Relationenalgebra in SQL erkennen
- Komplexe SQL-Anfragen verstehen und selbst bauen können
- Vorteile von SQL als Abfragesprache erkennen

SQL

- Vorgänger:
  SEQUEL = Structured English Query Language
- oft bezeichnet als: Structured Query Language
- de-facto Standard bei Datenbank-abfragesprachen
- Trotz Standardisierung (leicht) abweichende Syntax bei verschiedenen DBMS

Standardisierungen

- SQL-86
  - Zuerst nur ANSI-Standard, ab 1987 ISO-Standard
  - Umfang: 105 Seiten
  - Basis-Features:
    - Anfragesprache auf Tabellen (SELECT …)
    - Datendefinitionssprache (CREATE …)
    - Datenmanipulationssprache (UPDATE …)
    - Zugriffsrechte (GRANT …)
    - Primitive Datentypen (CHAR, INT, FLOAT, …)
    - Einbettung in Programmiersprachen (C, Cobol usw.)

Standardisierungen

SQL-89

- Umfang: 120 Seiten
- Einzige wesentliche Neuerung: referentielle Integrität (z. B. Fremdschlüssel)

Standardisierungen

SQL-92

- Umfang: 607 Seiten
- Substanzielle Neuerungen
  - Neue Datentypen (z.B. DATE, TIME, INTERVAL)
  - Outer Joins
  - Domains, Assertions
  - Referentielle Aktionen (… ON UPDATE CASCADE)
  - Schema-Manipulationssprache (ALTER TABLE…)
  - Dynamisches SQL

Standardisierungen

SQL:1999

- Umfang: 1984 Seiten
- Aufspaltung des Standards in fünf Teile:
  1. SQL/Framework
     - Übersicht über den kompletten Standard
     - Für alle Teile gültige Definitionen und Konzepte
  2. SQL/Foundation
     - Hauptteil des Standards (allein 1121 Seiten)
     - Enthält alle Elemente der SQL-92-Anfragesprache (außer eingebettetem und dynamischem SQL)
     - … und umfangreiche Erweiterungen (s. u.)

Standardisierungen

3. **SQL/CLI (Call Level Interface)**
   - Standardisierte Aufrufschnittstelle für SQL-Datenbanken
   - Bekannteste Implementierung: ODBC

4. **SQL/PSM (Persistent Stored Modules)**
   - Erweiterung von SQL um imperative Sprachkonstrukte
   - Gespeicherte Prozeduren und Funktionen

5. **SQL/Bindings (Host Language Bindings)**
   - Eingebettetes und dynamisches SQL
   - Unterstützung zahlreicher Programmiersprachen (Ada, C, Cobol, Fortran, Mumps, Pascal, PL/1)

Standardisierungen

- **Erweiterung der klassischen RDBMS-Funktionalität**
  - Trigger
  - Rekursive Anfragen
  - Änderungen über Sichten (mit UNION, JOIN)
  - Savepoints für Transaktionen
- **Objektrelationale Erweiterungen**
  - Benutzerdefinierte Datentypen mit Methoden („Objekte“)
  - Typisierte Tabellen und Sichten („Objekttabellen/-sichten“)
  - Objektidentität und Referenzen
  - Einfache Vererbung

Standardisierungen

SQL:2003

- Umfang: ca. 3250 Seiten
- 5 neue Teile:

1. SQL/MED (Management of External Data)
   - Verarbeitung von Daten außerhalb der SQL-Datenbank
   - Applikationen erhalten SQL-Interface zu externen Daten
   - Synchronisation zwischen externen und Datenbank-Daten

2. SQL/OLB (Object Language Bindings)
   - Einbettung von statischen SQL-Befehlen in Java („SQLJ“)

Standardisierungen

3. SQL/Schemata
   - Standardisierte Sichten auf die Metadaten (Schema) einer SQL-Datenbank

4. SQL/JRT (SQL Routines and Types Using the Java Programming Language)
   - Aufruf von Java-Routinen aus SQL
   - Abbildung zwischen SQL-Typen und Java-Klassen

5. SQL/XML
   - Unterstützung von XML in SQL

Standardisierungen

- Einige Neuerungen in SQL/Foundation
  - Reintegration von SQL/Bindings (damit also insgesamt 9 Teile)
  - Beseitigung vieler Fehler/Ungenauigkeiten aus SQL:1999
  - Neues DML-Statement: MERGE
  - Erzeugung einer Tabelle aus einer Anfrage
  - MULTISET-Datentyp

Vorlesungsinhalt

- SQL92 ist bereits eine sehr umfangreiche DB-Sprache
- Durch SQL:1999 und SQL:2003 kamen insbesondere folgende Funktionen hinzu
  - Rekursion
  - Trigger
  - benutzerdefinierten Datentypen und Operationen
  - Objektorientierung
- Im Folgenden schwerpunktmäßig SQL92
  (und selbst das wird immer noch nicht von allen DBS voll unterstützt)

Vorlesungsinhalt

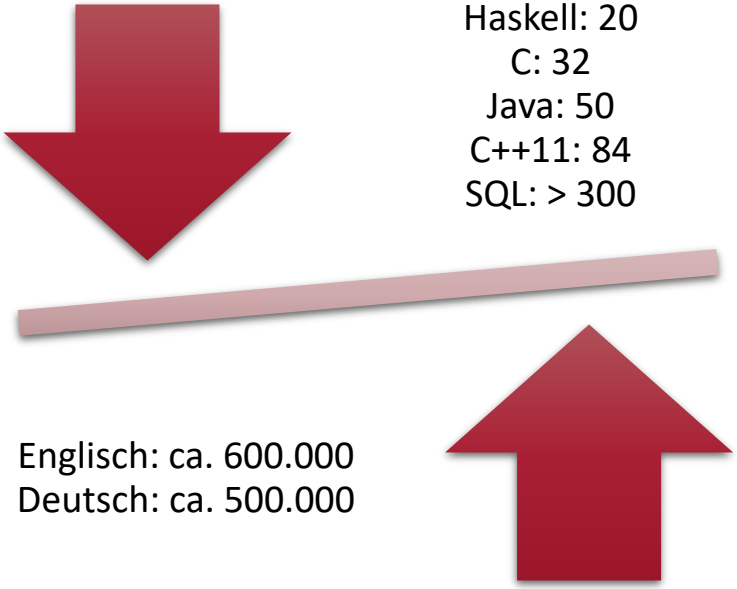
- Wie üblich, vollständige Lehre nicht möglich ➔ Selbststudium
- Ziele:
  - grundlegende Begriffe und Möglichkeiten vermitteln
  - Einstieg in Dokumentation erleichtern
- Klausurstoff: alles was Thema der Folien/Videos/Vorlesungen/Übungen ist

Sprachen lernen

natürliche Sprache	Programmiersprache
Vokabeln	Schlüsselwörter
Grammatik	Syntax
Bedeutung	Semantik
typische Floskeln, Redewendungen	Best practices, Styleguides

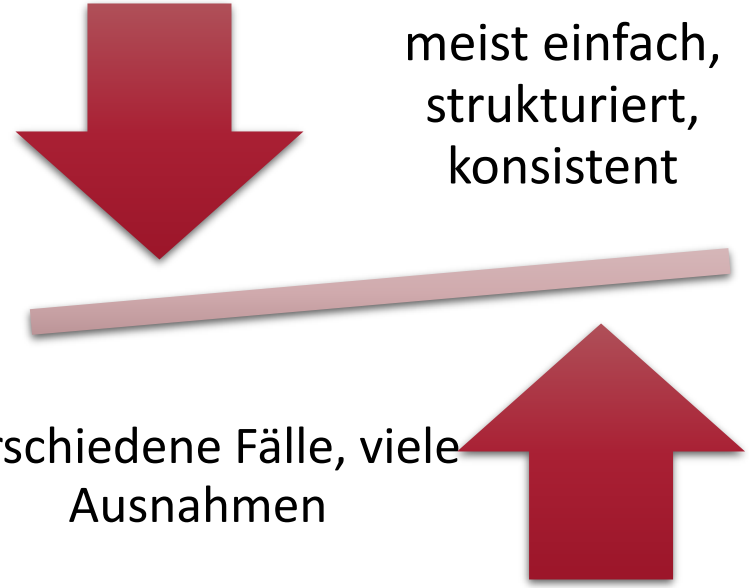
Sprachen lernen

Wortschatz:



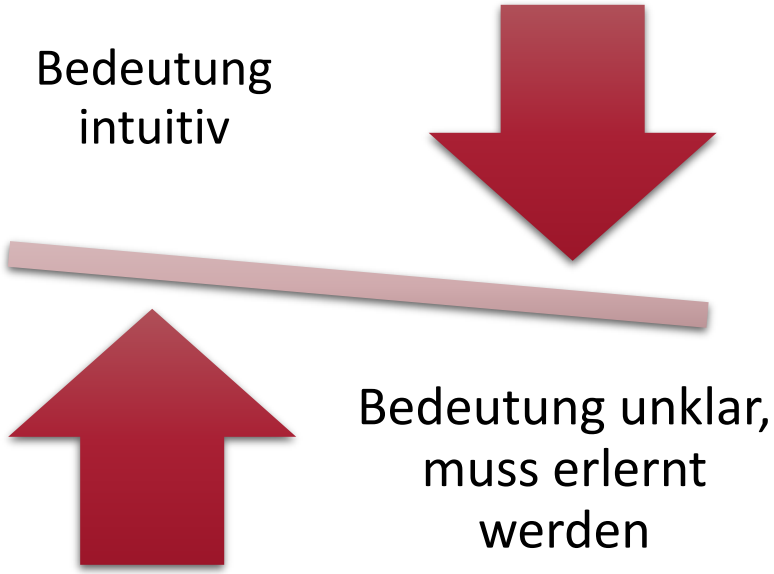
Sprachen lernen

Grammatik:



Sprachen lernen

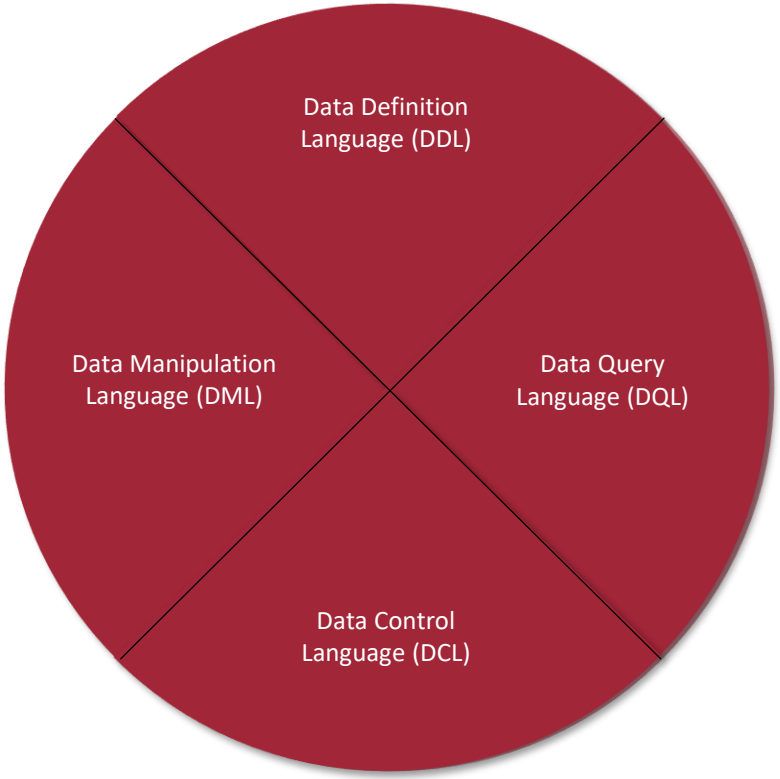
Semantik:



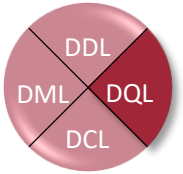
HyperSQL

- http://hsqldb.org
- komplettes ZIP-Paket im Moodle abrufbar
- Idee:
  - 'runterladen, entpacken, starten, 'rumspielen
  - falls etwas nicht mehr tut:
    'runterladen, entpacken, starten, 'rumspielen
- Demo

Vier Bereiche



Unterschiede zur Relationenalgebra

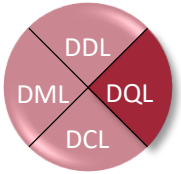


1. SQL-Relationen sind Multimengen (Bags) und daher im allgemeinen nicht duplikatfrei

2. Selektion, Projektion, Umbenennung und Join keine Einzel-Operatoren, sondern eingebettet in die SELECT-FROM-WHERE-Anweisung (SFW)

3. Sortierung der Tupel möglich, aber *nur für die Ausgabe*

SELECT-FROM-WHERE



SELECT [**ALL** | **DISTINCT**] *Attributliste*

AttrName bzw. R_i.*AttrName*
oder * bzw. R_i.* (= „alle“)
(Eindeutigkeit gefordert!)

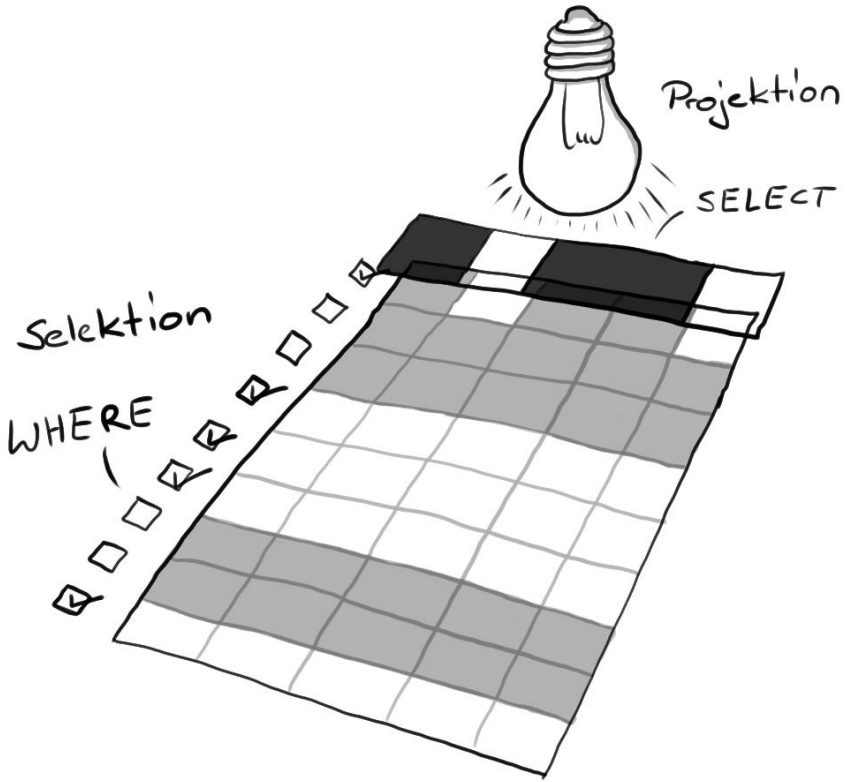
FROM $R_1, R_2, \ldots, R_n$

[**WHERE** *Prädikat*]

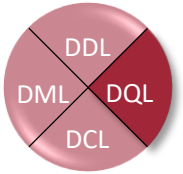
- Bedeutung SFW: $\pi_{Attributliste}\,(\sigma_{Prädikat}\ (R_1 \times \ldots \times R_n))$
- Auswertungsreihenfolge (logisch):

  FROM-Klausel ➔ WHERE-Klausel ➔ SELECT-Klausel

Vorsicht



SFW-Beispiele



S1: *„Gib die Lieferanten-Relation komplett aus“*

```
SELECT  *
FROM    Lieferanten
```

ohne Duplikateliminierung

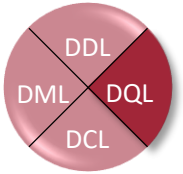
```
SELECT  DISTINCT *
FROM    Lieferanten
```

mit Duplikateliminierung

S2: *„Gib die Lieferanten-Relation komplett aus, aber in der Attribut-Reihenfolge Bewertung, LiefName, LiefStadt, LiefNr“*

```
SELECT  Bewertung, LiefName, LiefStadt, LiefNr
FROM    Lieferanten
```

SFW-Beispiele



S3: *„Gib alle Kunden aus, die aus 'Ulm' kommen“*

```
SELECT  KdNr, KdName, KdStadt, Bonitaet
FROM    Kunden
WHERE   KdStadt = 'Ulm'
```

explizite Attributangabe

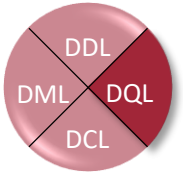
```
SELECT  *
FROM    Kunden
WHERE   KdStadt = 'Ulm'
```

Kurzform

S4: *„Gib alle Lieferanten mit Lieferantennummer ≥ 560 aus“*

```
SELECT  *
FROM    Lieferanten
WHERE   LiefNr >= 560
```

Vergleichsoperatoren



- Verknüpfung mit **AND** und **OR** (**AND** bindet stärker als **OR**)

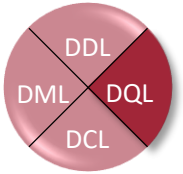
  ... LiefNr > 100 **AND** TnNR < 600
  ... LiefStadt = 'Ulm' **OR** LiefStadt = 'Neu-Ulm',
  ... (LiefStadt = 'Ulm' **OR** LiefStadt = 'Neu-Ulm') **AND** LiefNr > 600

- weitere Vergleichsoperatoren

  … LiefNr **BETWEEN** 400 **AND** 600 (≙ … LiefNr >= 400 **AND** LiefNr <= 600)
  ... LiefNr **NOT BETWEEN** 400 **AND** 600
  ... LiefNr **IN** (100, 200, 130, 400)
  ... LiefNr **NOT IN** (100, 200, 130, 400)
  … **NOT (** LiefNr **IN** (100, 200, 130, 400) **)**

- Anmerkungen zu **IN**:
  - Nach **IN** kommt immer eine Menge atomarer Werte (kann auch einelementig oder leer sein)
  - Die „**IN**-Menge“ kann auch durch ein Subquery erzeugt werden

Tabellenausdrücke

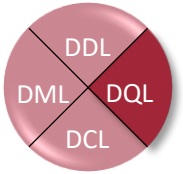


- Allgemeinere Form von SFW:

```
SELECT …
FROM   Tabellenausdruck
WHERE  …
```

- Ein Tabellenausdruck kann sein
  - ein Relationsname
  - kartesisches Produkt von Relationen
  - ein SFW-Ausdruck
  - **VALUES**-Klausel ≙ „konstante“ Tabelle

Tabellenausdrücke



Einfacher Fall:

```
SELECT   LiefNr, LiefStadt
FROM     Lieferanten
WHERE    LiefNr > 500
```

Ebenfalls möglich:

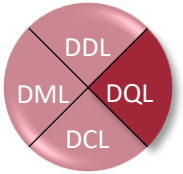
```
SELECT   *
FROM     ( SELECT   LiefNr, LiefName
           FROM     Lieferanten
           WHERE    LiefNr > 500)
```

```
SELECT   lf.LiefNr, lf.LiefName
FROM     ( SELECT   *
           FROM     Lieferanten
           WHERE    LiefNr > 500) AS lf
```

```
SELECT   lf.LiefNr, lf.LiefName
FROM     ( SELECT   *
           FROM     Lieferanten ) AS lf
WHERE    lf.LiefNr > 500
```

```
WITH  lf  AS (   SELECT   *
                 FROM     Lieferanten
                 WHERE    LiefNr > 500)
SELECT   lf.LiefNr, lf.LiefName
FROM     lf
```

konstante Relationen



- Tabellenkonstruktor: **VALUES**

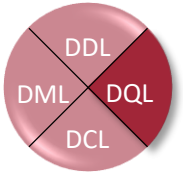
```
SELECT * FROM (VALUES (11,12,13), (21,22,23)) AS t
```

- Resultat:

```
C1 C2 C3
-- -- --
11 12 13
21 22 23
```

- Anmerkungen:
  - Nützlich in Verbindung mit UNION- oder EXCEPT-Operationen
  - Die Elemente einer VALUES-Klausel müssen keine Konstanten sein, sondern können auch Ausdrücke sein

Arithmetik

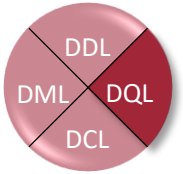


- Operatoren: +, -, *, / mit der üblichen Semantik
- bei den meisten DBMS auch:
  `mod(x,y), power(x,y), sin(x), cos(x), sqr(x), ...`

S5: *„Gib aus, bei welchen Teilen der Mindeststand unterschritten wurde und welche Kosten für ein Auffüllen auf den Mindestbestand jeweils entstehen würden“*

```
SELECT  TeileNr, Bestand, MinBestand, Farbe,
        (MinBestand - Bestand) * KalkKosten AS Auffüllkosten
FROM    Teile
WHERE   Bestand < MinBestand
```

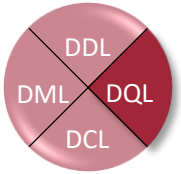
String-Funktionen



- einige im SQL-Standard definierte String-Funktionen

LCASE(*string***)**	*string* in Kleinbuchstaben
UCASE(*string***)**	*string* in Großbuchstaben
LENGTH(*string***)**	Länge von *string*
LOCATE(*suchstr***, *string***)**	Position von *suchstr* in *string*
LEFT(*string***, *anz***), RIGHT(***string***, *anz***)**	Begrenzt *string* von links bzw. rechts
SUBSTR(*string***, *start* [, *länge*])**	Liefert einen Teil von *string*
LTRIM(*string***), RTRIM(***string***), TRIM(***string***)**	Entfernen von Blanks
CHAR(*numerischerWert***)**	**CHAR**(123) → '123' (als String)

Substring-Vergleiche



S6: *„Gib alle Daten zu Lieferant 'Koch' aus“*

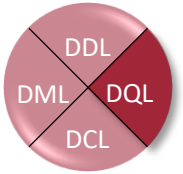
```
SELECT  *
FROM    Lieferanten
WHERE   LiefName = 'Koch'
```

Schreibweise genau so in DB?
Falls nicht, kein Treffer!

Lösung:

```
SELECT  *
FROM    Lieferanten
WHERE   LiefName LIKE 'Koch%'
```

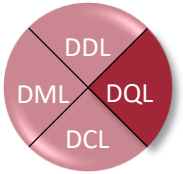
Substring-Vergleiche



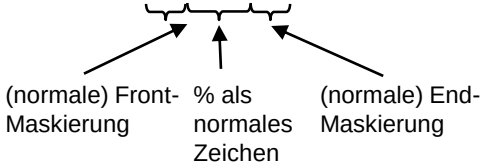
LIKE-Prädikat

- Spezieller Vergleichsoperator für Text-Attribute
- kein regulärer Ausdruck
- Maskierungszeichen:
  - **%** : beliebig viele zu ignorierende Zeichen (auch 0)
  - **_** : genau ein Zeichen wird ignoriert
- keine Wortgrenzen: gesamter Attributwert wird als eine Zeichenkette aufgefasst, einschließlich aller darin auftretenden Leerzeichen, Zahlen und Sonderzeichen

Substring-Vergleiche



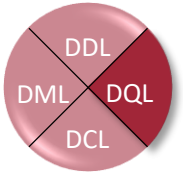
- Falls "%" oder "_" in einem Text gesucht, so können sie durch ein frei definierbares, vorangestelltes „Escape“-Zeichen maskiert.
- ... **WHERE** A **LIKE** '%**\%**%' **ESCAPE '\'**



findet alle Tupel, die (irgendwo) im Attributwert von A ein %-Zeichen haben

- Viele DBMS bieten Zusatzfunktionen an, die sehr viel mehr Funktionalität bieten (z.B. reguläre Ausdrücke)

Vergleichsbeispiele



S7: *„Gib alle Lieferanten aus, die aus einer Stadt kommen, die auf 'burg' endet.“*

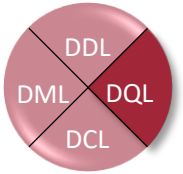
```
SELECT *
FROM   Lieferanten
WHERE LiefStadt  LIKE  '%burg'
```

S8: *„Wie S6, aber die Stadt soll außerdem mit 'H' beginnen.“*

...

```
WHERE  LiefStadt  LIKE  'H%burg'
```

Vergleichsbeispiele



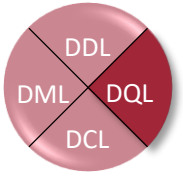
S9: *„Gib alle Lieferanten aus, die 'GmbH' im Firmennamen haben"*

```
SELECT *
FROM   Lieferanten
WHERE LiefName LIKE '%GmbH%'
```

S10: *„Gib alle Lieferanten aus, die 'Maier', 'Meier', 'Mayer' usw. im Namen tragen“*

```
...
WHERE  LiefName  LIKE  '%M_ _er%'
```

Datumsprädikate



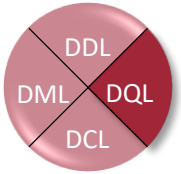
S11: *„Gib alle Bestellungen aus, die nach dem 01.03.2013 ausgestellt wurden“*

```
SELECT  *
FROM    Bestellungen
WHERE   BestDatum > '2013-03-01'
```

S12: *„Gib alle Bestellungen aus, die vom 01.-15.02.2013 erstellt wurden“*

```
SELECT  *
FROM    Bestellungen
WHERE   BestDatum BETWEEN '2013-02-01' AND
        '2013-02-15'
```

Datumsfunktionen

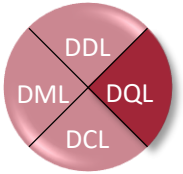


- Extraktionsfunktionen:
  year(*date***)**, **month(***date***)**, **day(***date***)**

S13: *„Gib die Relation Bestellungen mit separaten Spalten für Tag, Monat und Jahr aus.“*

```
SELECT  BestNr, LiefNr,
        day(BestDatum) AS Tag,
        month(BestDatum) AS Monat,
        year(BestDatum) AS Jahr
FROM    Bestellungen
```

Datumsfunktionen



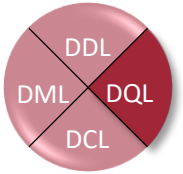
- Berechnung von Zeitabständen:

  ± **day, month, year**

S14: *„Gib Bestelldatum und Datum für die erste Mahnung (+ 2 Monate nach Bestelldatum) für alle Bestellungen aus“*

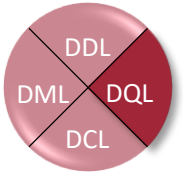
```
SELECT  BestNr, LiefNr, BestDatum AS Bestelldatum,
        BestDatum + 2 month AS Mahndatum
FROM    Bestellungen
```

Ausgabe von Werten



- SQL-Standard legt Attributnamen von berechneten Werten nicht fest
  ➔ jedes DBMS unterschiedlich
- Deshalb: Solche Attributwerte mittels **AS** immer explizit benennen
- Datumsformat nach ISO 8601: "YYYY-MM-DD"
- Ausgabe nicht definiert ➔ Doku

Fallunterscheidung

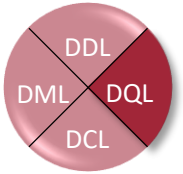


- CASE-Klausel
  Fallunterscheidung, um Tupelwerte in der Ergebnismenge attributwert-abhängig zu ändern

S15: *„Gib die Lieferantentupel mit der Bewertung im Klartext aus, und zwar wie folgt: -2 = sehr schlecht, -1 = schlecht, 0 = neutral, +1 = gut, +2 = sehr gut“*

```
SELECT  LiefNr, LiefName, LiefStadt,
        CASE
           WHEN Bewertung =  -2  THEN  'sehr schlecht'
        WHEN Bewertung =     -1  THEN  'schlecht'
           WHEN Bewertung =   0  THEN  'neutral'
           WHEN Bewertung = +1  THEN  'gut'
        WHEN Bewertung =     +2  THEN  'sehr gut'
           ELSE  '????'
         END  AS Bewertung
FROM    Lieferanten
```

Fallunterscheidung

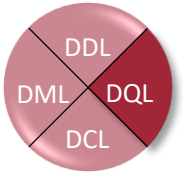


- Abkürzende Schreibweise

```
SELECT LiefNr, LiefName, LiefStadt, CASE Bewertung
            WHEN  -2  THEN 'sehr schlecht'
            WHEN  -1  THEN 'schlecht'
            WHEN   0  THEN 'neutral'
            WHEN  +1  THEN 'gut'
            WHEN  +2  THEN 'sehr gut'
            END AS Bewertung
FROM    Lieferanten
```

- Wichtig:
  - Erster **WHEN**-Fall legt den Datentyp der Ausgabe fest
  - Alle folgenden Fälle müssen gleichen Typ haben

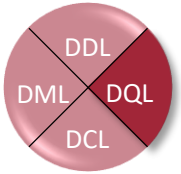
Fallunterscheidung



S16: *„Gib für alle Platten-Teile mit aus, wie hoch die Auffüllkosten auf den Mindestbestand sind“*

```
SELECT TeileNr, Bestand, MinBestand,
        CASE
            WHEN  Bestand < MinBestand
            THEN (MinBestand - Bestand) * KalkKosten
            ELSE  0
        END AS Auffüllkosten
FROM    Teile WHERE TeileNr LIKE 'P%'
```

Sortierung der Ausgabe



- Erweiterung der SFW-Klausel um **ORDER BY**:

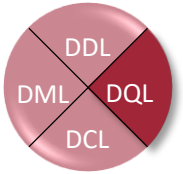
  …
  WHERE …
  ORDER BY

  $Attr_1 \left(\{ \underline{\textbf{ASC}} \mid \textbf{DESC} \} \right) \left(, Attr_2 \left(\{ \underline{\textbf{ASC}} \mid \textbf{DESC} \} \right) , \ldots\ldots \right)$

S17: *„Gib die Relation Bestellungen (Ausgabe: LiefNr, BestDatum, BestNr) aus und zwar sortiert nach LiefNr (aufsteigend) und BestDatum (absteigend)“*

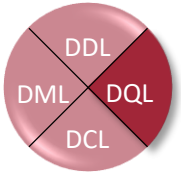
```
SELECT LiefNr, BestDatum, BestNr
FROM   Bestellungen
ORDER BY LiefNr, BestDatum DESC
```

Sortierung der Ausgabe



- Anmerkungen
  - **ORDER BY**-Klausel kann in einem Anfrageausdruck nur einmal ganz am Schluss auftreten
  - Sortierung nur der Ergebnistabelle, nicht des darunterliegenden Datenbestands
  - Attribute werden über ihren Namen angesprochen

Join



syntaktische Grundstruktur aus SQL86/89

SELECT [{ **ALL** | **DISTINCT** }] *Attributliste*

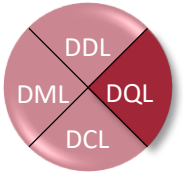
FROM $Rel_1, Rel_2, \ldots, Rel_n$

WHERE $Rel_i.Attributname = Rel_j.Attributname$ **AND**
$Rel_k.Attributname = Rel_l.Attributname$ **AND**
…
$Rel_y.Attributname = Rel_z.Attributname,$ mit i, j, k, l, …, z ∈ {1, 2, …, n}

wobei beliebige Vergleichsoperationen sowie Vergleiche zwischen Attributwerten und Konstanten zugelassen sind

vergleiche Join-Definition: Kartesisches Produkt + Selektion

Join Beispiel



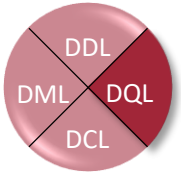
S18: *„Gib die Preisliste, ergänzt um die Teilenamen sowie die Farbcodes im Klartext aus“*

86/89er Form:

```
SELECT  p.TeileNr, t.TeileName, p.Farbe,
        f.FarbeText, p.Preis
FROM    Preisliste p, Teiletypen t, Farbcodes f
WHERE   p.TeileNr = t.TeileNr AND p.Farbe = f.Farbcode
```

- Mit SQL92 wurden explizite Join-Operatoren eingeführt

Condition Join

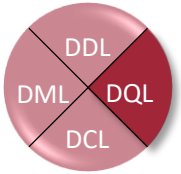



```
SELECT   select-list
FROM     table1 [ AS t1 ]
         JOIN table2 [ AS t2 ] ON join-condition1
         JOIN table3 [ AS t3 ] ON join-condition2
         JOIN …
( WHERE  … )
```

Anmerkungen

- Interpretation von links nach rechts
- d.h. jede *join-condition* kann nur Attribute aus den vorherigen oder unmittelbar beteiligten Tabellen referenzieren
- Dieses Join macht insbesondere dann Sinn, wenn andere Join-Bedingungen als "=" ausgedrückt werden sollen

Natural Join




```
SELECT    select-list
FROM      table1 NATURAL JOIN table2
```

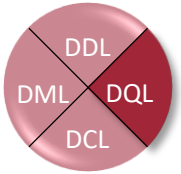
Anmerkung:

- Spaltennamen dürfen nicht mehr durch ihren Source-Tabellen-Namen qualifiziert werden ➔ Warum?

- Beispiel:

```
SELECT    BestNr, LiefNr, LiefName
FROM      Bestellungen NATURAL JOIN Lieferanten
WHERE     LiefNr > 109
```

Column Name Join

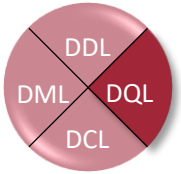



```
SELECT   select-list
FROM     table1 JOIN table2 USING (attribute-list)
```

Anmerkungen

- Spezialfall des Natural Join
- Beschränkung auf die in *attribute-list* angegebenen Spaltennamen
- In der USING-Klausel angegebene Spaltennamen dürfen nicht mehr mit einem Tabellen-Namen qualifiziert werden

Cross Join

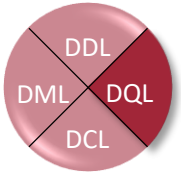



```
SELECT    select-list
FROM      table1 CROSS JOIN table2
```

- Entspricht einem Join der bisherigen Form ohne WHERE-Klausel:

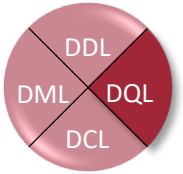
```
SELECT  select-list
FROM    table1, table2
```

Outer Join




```
SELECT  select-list
FROM    table1 LEFT OUTER JOIN table2 ON ...
SELECT  select-list
FROM    table1 RIGHT OUTER JOIN table2 ON ...
SELECT  select-list
FROM    table1 FULL OUTER JOIN table2 ON ...
```

Beispiel zu Outer Join



S19: *„Gib alle Teiletypen mit ihren Preisen laut Preisliste (soweit vorhanden) aus“*

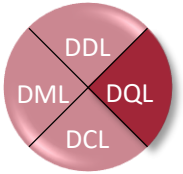
- Formuliert mittels **LEFT OUTER JOIN**:

```
SELECT  t.TeileNr, t.TeileName, p.Farbe, p.Preis
FROM    Teiletypen AS t LEFT OUTER JOIN
        Preisliste AS p ON t.TeileNr = p.TeileNr
```

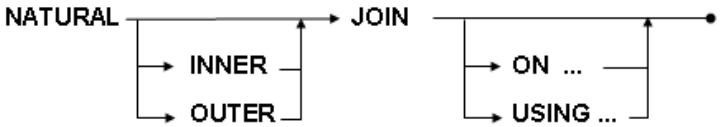
- Formuliert mittels **RIGHT OUTER JOIN**:

```
SELECT  t.TeileNr, t.TeileName, p.Farbe, p.Preis
FROM    Preisliste AS p RIGHT OUTER JOIN
        Teiletypen AS t ON t.TeileNr = p.TeileNr
```

Zusammenfassung Joins

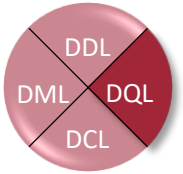


- Ziel: Verbesserte Lesbarkeit
- Mehr Fehlerprüfung durch DBMS (etwa zur Vermeidung unbeabsichtigter (partieller) Kreuzprodukte)
- Zur besseren Unterscheidung kann den „inner Joins“ das Schlüsselwort **`INNER`** vorangestellt werden
- Inner Joins und Outer Joins können zudem in der **`NATURAL`**-Join-Variante auftreten:



- Aber: alte Join-Syntax nach wie vor gültig!

Beispiele zu Joins



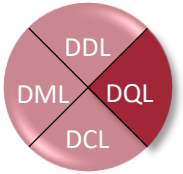
S20: *„Gib alle Mitarbeiter mit PersNr, Name, Vorname und Abteilung (AbtName) aus“*

```
SELECT  m.PersNr, m.Name, m.Vorname, a.AbtName
FROM    Mitarbeiter AS m JOIN
        Abteilungen AS a ON m.AbtNr = a.AbtNr
```

Anmerkungen:

- Die (frei gewählten) Bezeichner m und a in der FROM-Klausel sind sog. Korrelationsvariablen, die man dazu verwenden kann, um die Attribute jeweils eindeutig zu identifizieren.

Beispiele zu Joins

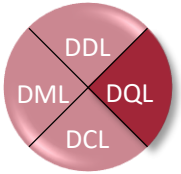


S21: *„Gib aus, welche Teile (TeileNr, Farbe) mit TeileName = „Klotz xxx“ von welchen Lieferanten (LiefNr) geliefert werden“*

Gewünschtes Resultat:

TEILENR	FARBE	TEILENAME	LIEFNR
K14	0	Klotz 1 x 4	506
K14	0	Klotz 1 x 4	522
K14	1	Klotz 1 x 4	506
K14	1	Klotz 1 x 4	522
K14	2	Klotz 1 x 4	522
K16	0	Klotz 1 x 6	506
K16	0	Klotz 1 x 6	522
K16	1	Klotz 1 x 6	506
K16	1	Klotz 1 x 6	522
K16	2	Klotz 1 x 6	522
K18	1	Klotz 1 x 8	506
K18	1	Klotz 1 x 8	522
K18	2	Klotz 1 x 8	522
KR24	0	Klotz 2 x 4 für Rad	540
K28	0	Klotz 2 x 8	506
K28	0	Klotz 2 x 8	527

Beispiele zu Joins



S21: *„Gib aus, welche Teile (TeileNr, Farbe) mit TeileName = „Klotz xxx“ von welchen Lieferanten (LiefNr) geliefert werden“*

Systematische Konstruktion des Joins:

1. Ausgabe- + WHERE-Klausel-Attribute bestimmen
2. Potenziell relevante Relationen bestimmen
3. Geeignete Join-Pfade mit Join-Attributen bestimmen
4. Optional: „Abwahl“ der nicht benötigten Relationen

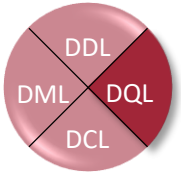
Zu 1: TeileNr, Farbe, TeileName, LiefNr

Zu 2: Teile, TeileTypen, Lieferanten

Zu 3: TeileTypen $\xrightarrow{\text{TeileNr}}$ Teile $\xrightarrow{\text{(TeileNr,Farbe)}}$ Liefert $\xrightarrow{\text{LiefNr}}$ ~~Lieferant~~

Zu 4: Lieferant

Beispiele zu Joins



S21: *„Gib aus, welche Teile (TeileNr, Farbe) mit TeileName = „Klotz xxx“ von welchen Lieferanten (LiefNr) geliefert werden“*

Zu 1: TeileNr, Farbe, TeileName, LiefNr

Zu 2: Teile, TeileTypen, Lieferanten

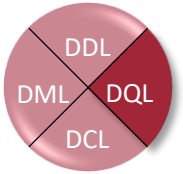
Zu 3: TeileTypen $\xrightarrow{\text{TeileNr}}$ Teile $\xrightarrow{\text{(TeileNr,Farbe)}}$ Liefert $\xrightarrow{\text{LiefNr}}$ ~~Lieferant~~

Zu 4: Lieferant

SQL-Anfrage:

```
SELECT ty.TeileNr, t.Farbe, ty.TeileName, lf.LiefNr
FROM   TeileTypen AS ty
       JOIN Teile AS t ON ty.TeileNr = t.TeileNr
       JOIN Liefert AS lf ON (t.TeileNr, t.Farbe) = (lf.TeileNr,lf.Farbe)
WHERE  ty.TeileName LIKE 'Klotz%'
```

Beispiele zu Joins



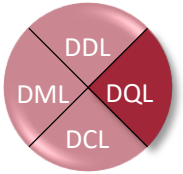
SQL-Anfrage:

```
SELECT ty.TeileNr, t.Farbe, ty.TeileName, lf.LiefNr
FROM   TeileTypen AS ty
       JOIN Teile AS t ON ty.TeileNr = t.TeileNr
       JOIN Liefert AS lf ON (t.TeileNr, t.Farbe) = (lf.TeileNr,lf.Farbe)
WHERE  ty.TeileName LIKE 'Klotz%'
```

Kompakter (wartbarer?):

```
SELECT TeileNr, Farbe, TeileName, LiefNr
FROM   TeileTypen
       JOIN Teile USING (TeileNr)
       JOIN Liefert USING (TeileNr, Farbe)
WHERE  TeileName LIKE 'Klotz%'
```

Beispiele zu Joins



S22: *„Gib alle Aufträge mit Auftragsnummer, Kundennummer, Kundenname, Auftragsdatum, den Auftragspositionen mit Teilenummer, Teilebezeichnung, Farbcode und bestellter Menge aller Kunden aus Augsburg und Senden aus“*

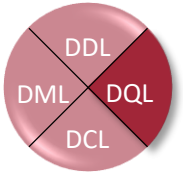
1. Ausgabe- + WHERE-Klausel-Attribute bestimmen

2. Potenziell relevante Relationen bestimmen

3. Geeignete Join-Pfade mit Join-Attributen bestimmen

4. Optional: „Abwahl“ der nicht benötigten Relationen

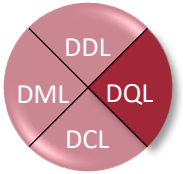
Beispiele zu Joins



SQL-Anfrage:

```
SELECT    AuftrNr, KdNr, KdName, AuftrDatum, Pos,
          TeileNr, TeileName, Farbe, Anzahl, KdStadt
FROM      Kunden
          NATURAL JOIN Auftraege
          NATURAL JOIN AuftragsPos
          NATURAL JOIN TeileTypen
WHERE     KdStadt IN ('Augsburg','Senden')
```

Beispiele zu Joins



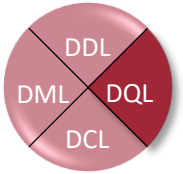
- Beim Join einer Relation mit sich selbst, werden dieser zur Unterscheidung mehrere Korrelationsvariablen zugeordnet:

  SELECT … FROM Rel1 AS r1 JOIN Rel1 AS r2 ON …

S23: *„Gib alle Mitarbeiter aus, die weniger Gehalt als ihre Kollegen erhalten. Gib PersNr und Gehalt des weniger und des mehr verdienenden Mitarbeiters sowie die Gehaltsdifferenz aus, und zwar absteigend sortiert nach Gehaltsdifferenzen“*

```
SELECT   m1.PersNr AS PersNrWeniger, m1.Gehalt AS GehaltWeniger,
         m2.PersNr AS PersNrMehr, m2.Gehalt AS GehaltMehr,
         m2.Gehalt - m1.Gehalt AS Differenz
FROM     Mitarbeiter AS m1
         JOIN Mitarbeiter AS m2 ON m1.Gehalt < m2.Gehalt
ORDER BY Differenz DESC
```

Beispiele zu Joins

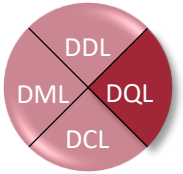


S24: *„Gib alle Teile mit TeileNr, Farbe, KalkKosten und ggf. Preis aus“*

```
SELECT   t.TeileNr, t.Farbe, t.KalkKosten, p.Preis
FROM     Teile AS t LEFT OUTER JOIN Preisliste AS p
         ON t.TeileNr = p.TeileNr AND t.Farbe = p.Farbe
```

Kompakter?

Beispiele zu Joins

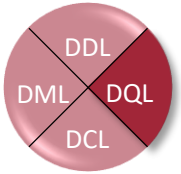


S25: *„Ermittle alle Teile (Ausgabe TeileNr), zu denen es keinen (externen) Preis gibt“*

Frage:

- Hätten wir in der obigen WHERE-Klausel auch p.TeileNr **`IS NULL`** sagen können?

(letztes) Beispiel zu Joins

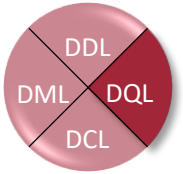


S26: *„Gib für jedes Teil die kalkulatorischen Kosten der verschiedenen Farben (soweit vorhanden) sowie die Information aus, welche Farbe (Farbcode) jeweils am teuersten ist bzw. NULL, wenn keine Kostenunterschiede bestehen"*

Zunächst der FROM-Teil:

```
FROM    Teile AS t
        LEFT OUTER JOIN Teile AS neutral
           ON t.TeileNr = neutral.TeileNr AND neutral.Farbe = 0
        LEFT OUTER JOIN Teile AS rot
           ON t.TeileNr = rot.TeileNr AND rot.Farbe = 1
        LEFT OUTER JOIN Teile AS blau
           ON t.TeileNr = blau.TeileNr AND blau.Farbe = 2
```

(letztes) Beispiel zu Joins



S26: *„Gib für jedes Teil die kalkulatorischen Kosten der verschiedenen Farben (soweit vorhanden) sowie die Information aus, welche Farbe (Farbcode) jeweils am teuersten ist bzw. NULL, wenn keine Kostenunterschiede bestehen"*

```
SELECT DISTINCT  t.TeileNr, neutral.KalkKosten AS Kosten_neutral,
        rot.KalkKosten AS Kosten_rot, blau.KalkKosten AS Kosten_blau,
        CASE WHEN   neutral.KalkKosten > rot.KalkKosten AND
                          neutral.KalkKosten > blau.KalkKosten THEN 0
             WHEN   rot.KalkKosten > neutral.KalkKosten AND
                    rot.KalkKosten > blau.KalkKosten THEN 1
             WHEN   blau.KalkKosten > neutral.KalkKosten AND
                    blau.KalkKosten > rot.KalkKosten THEN 2
             ELSE   NULL
        END AS  max_Kosten
```

Ziele

- Entwicklung von SQL kennen
- Unterschiede zwischen dem Erlernen von Sprachen und Programmiersprachen
- Umsetzung der Relationenalgebra in SQL erkennen
- Komplexe SQL-Anfragen verstehen und selbst bauen können
- Vorteile von SQL als Abfragesprache erkennen