Merkblatt SQL

Grundbausteine:

SELECT <Attributliste> (= Schema der Ergebnisrelation; * für alle Attribute)
FROM <Relationenliste> (Relationen aus denen die Tupel stammen)
WHERE <Bedingungen>**;** (Bedingungen an die Daten; Verknüpfung über Schlüsselwort **AND**)

SELECT:

- Entspricht Projektion π der relationalen Algebra
- Umbenennung: SELECT Titel AS Filmtitel
- Arithmetik: SELECT Länge * 3.1415 AS LängeMalPi
- Konstanten: SELECT ‚Herr' AS Titel
- Duplikateliminierung δ: SELECT DISTINCT Titel

FROM:

- Entspricht Kreuzprodukt × der relationalen Algebra (falls mehrere Relationen gewählt)

WHERE:

- Entspricht Selektion σ der relationalen Algebra
- WHERE-Teil der Anfrage ist optional
- Operatoren: =, <>, >, <, <=, >=, LIKE, NOT, ANY, ALL, EXISTS, IN, BETWEEN (für Intervalle)
  LIKE '%ei_' (Wildcards, % beliebig viele Zeichen, _ ein Zeichen)
- Kann Kreuzprodukt des FROM-Teils zum Join machen:
  WHERE Person.ID = Mensch.ID (Natürlicher Join $\bowtie$)
  WHERE Person.Name = Mensch.Vorname (Theta Join $\bowtie_\theta$)

Nützliche Funktionen:

```
UPPER, LOWER, INITCAP (alle Buchstaben groß, klein, ersten groß – übrige
klein schreiben)
LENGTH(„spalte") – Länge des Textes bestimmen
CONCAT(string1,string2) oder (string1 || string2) fügt Strings zusammen
WHERE „spalte" IS NULL – nach „nicht gefüllten" Feldern fragen
EXTRACT(YEAR (oder DAY, MONTH, HOUR) FROM „spalte") – einzelne
Datumsbestandteile extrahieren oder TO_CHAR(datum, 'YYYY'), (DD-MON-YYYY)
(NVL(LENGTH(vname),0) – NULL-Werte bei Berechnungen durch 0 ersetzen
```

Komplexe Anfragen:

SELECT <Attributliste>
FROM <Relationenliste>
WHERE <Bedingungen>
GROUP BY <Gruppierungsattribute>
HAVING <Bedingungen auf Gruppierungsattribute>
ORDER BY <Attributliste>**;**

GROUP BY … HAVING:

- Entspricht Gruppierung γ der relationalen Algebra

- Aggregationsoperatoren für SELECT-Statement: AVG(<Attribut>)
  COUNT(<Attribut>)
  SUM(<Attribut>)
- HAVING entspricht einer Selektion nach der Gruppierung
- Anwendung DISTINCT bei Aggregatfunktion direkt vor dem Spaltennamen
  SELECT COUNT(DISTINCT nname) AS anz_namen
  FROM personal
-

ORDER BY:
- Entspricht Sortierung τ der relationalen Algebra
- Sortiert das Ergebnis der Anfrage entsprechend der Attributliste
- Aufsteigend: ORDER BY Vorname, Nachname ASC
- Absteigend: ORDER BY Vorname, Nachname DESC

Datendefinition: Data Definition Language (DDL)

CREATE TABLE <Tabellenname>**(**<Attributliste mit Datentypen>**);**
- Aufgabe: neue Tabelle erstellen
- Datentypen mit Länge n bzw. m: CHAR(n), VARCHAR(n), BIT(n), DECIMAL(n,m)
- Datentypen mit impliziter Länge: INT, FLOAT
- Datentypen für Objekte: CLOB, BLOB (Character and Binary large objects (jpeg, gif))
- Datentypen für Zeiten: TIME, DATE, TIMESTAMP
  Beispiele: DATE 'YYYY-MM-DD' – DATE '1948-05-14', TIMESTAMP '1948-05-14 15:00:02.2'
- Bsp.: CREATE TABLE Schauspieler (
  Name CHAR(30),
  Adresse VARCHAR(255),
  Geschlecht CHAR(1),
  Geburtstag DATE);
- Nebenbedingungen:
  - Primärschlüssel: PRIMARY KEY
  - Eindeutigkeit: UNIQUE
  - Default-Werte: DEFAULT <Defaultwert>
  - Nicht-Null: NOT NULL
  - Fremdschlüssel: FOREIGN KEY (<Attributliste>) REFERENCES <Tabellenname>(<Attributliste>)
  - Weitere: CHECK, CREATE TRIGGER, CREATE ASSERTION, …

  Bsp.: CREATE TABLE Schauspieler (
  SchauspielerNummer INT PRIMARY KEY,
  Name CHAR(30) NOT NULL,
  Adresse VARCHAR(255) NOT NULL UNIQUE,
  Geschlecht CHAR(1),
  Geburtstag DATE DEFAULT DATE ‚0000-00-00',
  FOREIGN KEY (Adresse) REFERENCES Haus(Adresse));

DROP TABLE <Tabellenname>**;** ***(Achtung! Abhängigkeiten und aktuelle Zugriffe beachten)***
- Aufgabe: bestehende Tabelle löschen
- Bsp.: DROP TABLE Schauspieler;

ALTER TABLE <Tabellenname> <Aktion>**;**

- Aufgabe: bestehende Tabelle ändern
- ADD:
  - Attribut hinzufügen
    - Bsp.: ALTER TABLE Schauspieler ADD Telefon CHAR(16)
- DROP:
  - Attribut löschen
    - Bsp.: ALTER TABLE Schauspieler DROP Geburtstag

CREATE INDEX <Indexname> **ON** <Tabellenname>**(**<Attributliste des neuen Index>**);**
- Aufgabe: Index erstellen

DROP INDEX <Indexname>**;**
- Aufgabe: Index löschen

Datenbearbeitung: Data Modelling Language (DML)

INSERT INTO <Tabellenname>**(**<Attributliste>**) VALUES (**<Attributliste>**);**
- Aufgabe: Tupel einfügen
- Bsp.: INSERT INTO Studio(Name, Nummer) VALUES ('Pixa', 34);
- Ergebnis einer Anfrage für Einfügen nutzen:

  Bsp.:
```
INSERT INTO Studios(Name)
SELECT DISTINCT StudioName
FROM  Film
WHERE StudioName NOT IN
        (SELECT Name
         FROM Studios);
```

DELETE FROM <Tabellenname> **WHERE** <Bedingung>**;**
- Aufgabe: Tupel löschen
- Bsp.: DELETE FROM Studio WHERE Name='Pixa';

UPDATE <Tabellenname> **SET** <Zuweisung> **WHERE** <Bedingung>**;**
- Aufgabe: Attributwerte ändern
- Bsp.: UPDATE Studio SET Name='Pixa' WHERE Name='Pi';

Bulk insert: IMPORT, LOAD, … (→ DBMS spezifisch)

Fallunterscheidungen

```
UPDATE Artikel
SET ArtMarkierung =
(CASE WHEN ARtMwStSatz = 7
 THEN 1
 ELSE 0
 END)
```

Allgemein:
```
CASE WHEN <Bedingung> THEN <Ergebnis>
   [WHEN <Bedingung> THEN <Ergebnis>
    ...]
   [ELSE <Ergebnis>]
END
```

Mengenoperationen

(<Anfrage>) **UNION** (<Anfrage>) (Liefert Vereinigung „∪“ der beiden Ergebnismengen)

(<Anfrage>) **EXCEPT** (<Anfrage>) (Liefert Differenz „–“der beiden Ergebnismengen)

(<Anfrage>) **INTERSECT** (<Anfrage>) (Liefert Schnittmenge „∩“ der beiden Ergebnismengen)

- UNION, EXCEPT und INTERSECT nutzen Mengensemantik (→eliminieren Duplikate)
- UNION ALL, EXCEPT ALL und INTERSECT ALL nutzen Multimengensemantik (→erhalten Duplikate)

Join-Varianten

1.) Kreuzprodukt mit Bedingung:

```
SELECT *
FROM <Join-Relation1>, <Join-Relation2>
WHERE <Join-Attribut1> = <Join-Attribut2>;
```

2.) Schlüsselwort:

<Tabellenname> **CROSS JOIN** <Tabellenname>
<Tabellenname> **NATURAL JOIN** <Tabellenname>
<Tabellenname> **NATURAL INNER JOIN** <Tabellenname>
<Tabellenname> **NATURAL LEFT OUTER JOIN** <Tabellenname>
<Tabellenname> **NATURAL RIGHT OUTER JOIN** <Tabellenname>
<Tabellenname> **NATURAL FULL OUTER JOIN** <Tabellenname>

Beispiel INNER JOIN:

```
SELECT k.name
FROM klasse k
INNER JOIN auszubildender a
ON k.kid = a.kid
```

Beispiel Workaround FULL OUTER JOIN in MySQL:

```
SELECT k.name
FROM klasse k
LEFT JOIN auszubildender a
ON k.kid = a.kid
UNION
SELECT k.name
FROM klasse k
RIGHT JOIN auszubildender a
ON k.kid = a.kid
```

Beispiel JOIN mit 3 Tabellen:

```
SELECT p.persnr, p.nname, p.vname, a.abtname
FROM personal p
JOIN projekt pr
ON pr.projnr = p.projnr
JOIN abteilung a
ON pr.abtnr = a.abtnr
ORDER BY p.persnr
```

Beispiel SELF JOIN (Wer hat eine Mastercard und eine American Express-Karte?):

```
SELECT
   KK1.KndNr,
   KK1.Firma,
   KK2.KndNr,
   KK2.Firma
FROM
   Kreditkarten KK1
INNER JOIN
   Kreditkarten KK2
ON
   KK1.KndNr = KK2.KndNr
WHERE
   KK1.Firma = 'Mastercard' AND KK2.Firma = 'American Express'
```

Sichten

CREATE VIEW <Sichtname> **AS** <Anfrage>;

- Aufgabe: Erstelle eine Sicht für die gegebene SQL-Anfrage

Unterabfragen

Unterabfragen (*)

Standorte:

SELECT <Attributliste> (*) (= Schema der Ergebnisrelation; * für alle Attribute)
FROM <Relationenliste> (*) (Relationen aus denen die Tupel stammen)
WHERE <Bedingungen> (*)**;** (Bedingungen an die Daten; Verknüpfung über Schlüsselwort **AND**)

Subanfrage im FROM

```
SELECT M.Name
FROM Manager M, (SELECT ProduzentID AS ID
                 FROM Film, spielt_in
                 WHERE Titel = FilmTitel
                 AND Jahr = FilmJahr
                 AND  Schauspieler = ‘Harrison Ford‘) Produzent

WHERE M.ManagerID = Produzent.ID;
```

Subanfrage im SELECT:

```
SELECT a.AbtID, a.Name, (SELECT MAX(Gehalt)
                         FROM   Personal p)
WHERE a.AbtID = p.AbtID) AS maxGehalt
FROM   Abteilung a
WHERE  a.Ort= ‘Potsdam‘
```

Arten von Unterabfragen:

1.) **Skalare Unterabfragen: Ergebnis ist ein Wert (Skalar)**
Operatoren: <,<=,>,>=,=,<>

```
SELECT Name
FROM   Manager
WHERE  ManagerID = ( SELECT ProduzentID
                     FROM   Film
                     WHERE  Titel = 'Star Wars' AND Jahr = '1977' )
```

2.) Sonstigen Unterabfragen: Ergebnis ist ein Tupel (Spalte, Tabelle)
IN, NOT IN, EXISTS, ANY, ALL

EXISTS:

```
SELECT ISBN
FROM   BuchExemplar
WHERE  EXISTS
       (SELECT *
        FROM  Ausleihe
        WHERE Ausleihe.Inventarnr = BuchExemplar.Inventarnr)
```

IN:

```
SELECT Matrikel
FROM   Prüft
WHERE  Prüfer IN ( SELECT Prüfer
                   FROM Prüft
                   WHERE Matrikel = '123456')
```

ALL:

```
SELECT Note
FROM   Prüft
WHERE  Matrikel = '123456'
AND    Note >= ALL (SELECT Note
                    FROM   Prüft
                    WHERE  Matrikel = '123456)
```

ANY:

```
SELECT Name, Matrikel
FROM   Student
WHERE  Matrikel = ANY (SELECT Matrikel
                       FROM   Prüft)
```

Unkorreliert:

Name und Gehalt aller Mitarbeiter in Potsdam

```
SELECT Name, Gehalt
FROM Personal p
WHERE AbtID IN
     (SELECT AbtID
      FROM   Abteilung
      WHERE  Ort = 'Potsdam')
```

Korreliert:

Name und Gehalt aller Mitarbeiter, deren Gehalt höher als 10% des Abteilungsbudgets ist.

```
SELECT Name, Gehalt
FROM   Personal p
WHERE  Gehalt >
     (SELECT 0.1*Budget
      FROM Abteilung a
      WHERE a.AbtID = p.AbtID)
```

‘