SQL Anweisungen/Anfragen

Transitive Hülle

```
SELECT U,B
      FROM R
      START WITH O=2
      CONNECT BY PRIOR U=O;
```

Extract-Funktion

`EXTRACT (YEAR FROM start_date):` Zeigt nur das Jahr an

Tabelle anlegen

```
CREATE TABLE person
      (person_id SMALLINT AUTO_INCREMENT, …, CONSTRAINT pk_person
      PRIMARY KEY (person_id));
```

Tabellenbeschreibung

```
DESC person;
```

In Tabelle einfügen

```
INSERT INTO person
      (person_id, fname, …)
      VALUES (null, ‚William‘,…);
```

Check-Constraint für nicht erlaubte Eingaben

```
gender CHAR(1) (gender IN (‚M‘, ‚F‘));
gender ENUM (‚M‘,‘F‘);
```

Primärschlüssel, Fremdschlüssel

`CONSTRAINT pk_fav_food PRIMARY KEY (person_id, food);` [2 Spaltiger Primärschlüssel]
`CONSTRAINT fk_fav_food FOREIGN KEY (person_id) REFERENCES person (person_id);` [Fremdschlüssel mit Verweis auf die „person“-Tabelle, direktes Ändern der Zeile nicht möglich, wenn die Person mit der person_id nicht bereits in der person Tabelle existiert]

Abfragen ordnen

```
SELECT person_id, fname, lname
FROM person
WHERE lname = ‚Turner‘
ORDER BY fname;
```

Daten ändern

```
UPDATE person
SET address = ‚1234 Irgendwas‘, city="blbla", … [keine Klammern!]
WHERE person_id = 1;
```

Ändern mehreren Spalten z.b. durch: …`WHERE person_id < 20;`

Daten löschen

```
DELETE FROM person
WHERE person_id = 2;
```

Fehlerbeschreibung

```
SHOW WARNINGS;
```

Tabellen löschen

```
DROP TABLE person;
```

Stringumwandlungen

`UPPER(lname):` Nachnamen in Großbuchstaben
`LEFT(lname,1):` der erste Buchstabe eines Strings
`Wildcard-Zeichen:`
`,_‘` genau ein Zeichen; `,%‘` beliebig viele Zeichen (einschließlich null)
Bedinungen werden mit `LIKE` formuliert:

```
WHERE lname LIKE ,_a%e%‘;
```

`C LIKE ,d%‘:` Spalte C erster Buchstabe d, danach mind. 0 andere Zeichen
`SUBSTR (C,1,2):` Wählt aus String in Spalte C den ersten und zweiten Buchstaben (Anfangsbuchstabe 1, Länge 2)

Abfrageklauseln

`SELECT`	Legt fest, welche Spalten in die Ergebnismenge kommen
`FROM`	Woher kommen die Daten, wie sollen Tab. verbunden werden
`WHERE`	Beschränkt die Zeilenanzahl der Ergebnismenge
`GROUP BY`	Fasst Zeilen nach gemeinsamen Spaltenwerten zusammen
`HAVING`	Beschränkt die Zeilenanzahl der Ergebnismenge mit Hilfe von gruppierten Daten
`ORDER BY`	Sortiert die Ergebnismenge nach einer oder mehreren Spalten Aufsteigend [Default]: `ORDER BY xxx ASC` Absteigend: `ORDER BY xxx DESC` String-Sortierungen: Nach den letzten 3 Buchstaben in fed_id: `ORDER BY RIGHT (fed_id, 3)` Nach numerischen Platzhaltern sortieren: `ORDER BY 2, 5;` sortiert das Suchergebnis nach der 2. und 5. Rückgabespalte

Spalten-Aliase

```
SELECT emp_id,
      ,ACTIVE‘ status,  [Spaltenname ‚ACTIVE‘ wird als „status“ ausgegeben]
      emp_id * 3.14159 empid_mal_pi  [Berechnung emp_id*pi wird als empid_mal_pi
      ausgegeben]
FROM employee
```

Duplikate entfernen

```
SELECT DISTINCT cust_id FROM account;
```

[Zeigt die Duplikate für die cust_id Zeilen nicht an, in der Tabelle account kann ein Kunde mehrere Konten haben, daher kommt es zu mehreren Anzeigen einer einzelnen cust_id]

Views

Eine virtuelle Tabelle anlegen, auf die normal zugegriffen werden kann. Die Tabelle enthält keine Daten, sondern dient nur als Verweis auf die angegebenen Daten. Dadurch können z.B. bestimmte Inhalte vor Benutzern verborgen werden oder der Zugriff vereinfacht werden:

```
CREATE VIEW employee_vw AS
      SELECT emp_id, fname, lname, YEAR(start_date) start_year
      FROM employee;
```

➔ Zugriff wie auf normale Tabelle:

```
SELECT emp_id FROM employee_vw;
```

Referential Actions

Beim Erstellen einer Tabelle, um die Integrität zu erhalten (Was machen die zugehörigen Fremdschlüssel, wenn…)

```
CREATE TABLE… (… ON UPDATE CASCADE, ON DELETE SET NULL);
```

RESTRICT	Operation beschränkt auf den Fall, dass keine zugehörigen Fremdschlüssel vorhanden sind
CASCADE	Operation „kaskadiert“ zu allen zugehörigen Sätzen
SET NULL	FS wird in zugehörigen Sätzen zu „null“ gesetzt
SET DEFAULT	FS wird auf benutzerdefinierten Wert gesetzt
NO ACTION	Keine bestimmte Aktion

Prüfzeitpunkt der referentiellen Integrität: `IMMEDIATE|DEFERRED`

Aggregat-Funktionen

```
SELECT
      COUNT (*) „Anzahl Angestellte“,
      MAX (PGEHALT) „Spitzengehalt“,
      AVG (PGEHALT/12) „mittleres Monatsgehalt“,
      SUM (PGEHALT+PBONUS) „Summe Bezüge“
      COUNT (DISTINCT PGEHALT) „Gehaltsstufen“ [jedes Gehalt nur 1x]
FROM personal
```

Aggregatfunktionen dürfen niemals in der WHERE-Klausel stehen, nur in SELECT oder HAVING
(Ergebnis kann erst berechnet werden, wenn auch eines da ist, und nicht schon vorher)

Vergleiche

```
ANY: entspricht „SOME“ und „IN“: sämtliche Werte in der Subquery
werden berücksichtigt.
ALL:
      SELECT A,B,C FROM T WHERE B = ALL (SELECT B FROM T WHERE A<5);
```

Bei „=“ werden Tabellen miteinander über kreuz verglichen. In diesem Fall wird in der Subquery herausgefunden, wo A<5 ist. Das Ergebnis liefert mehrere B-Zeilen die mit dem ersten B verglichen werden müssen. D.h. Zeile1 mit Zeile1, Zeile1 mit Zeile2 usw., falls bei dem Vergleich irgendwo ein FALSE auftaucht, erscheint im Ergebnis eine leere Tabelle.
Generell: Wenn bei in der Subquery SELECT B FROM T WHERE A <5 unterschiedliche Werte für B herauskommen, dann ist die Ergebnistabelle bei einem „B=“ durch das ‚ALL‘ leer. Wenn jedoch nur ein Wert für B herauskommt, kann dieser auch in verschiedenen Zeilen vorkommen. Das Ergebnis für das B in der Subquery wird dann mit der Tabelle T in der Spalte B verglichen, als ob ANY darinstehen würde.
ALL-Vergleiche mit >= oder <=: Standardvergleich wie bei ANY, die Kriterien müssen erfüllt sein. Wobei ALL bei mehreren Ergebniszeilen der Subquery wie UND fungiert (ALLE Zeilen müssen ALLE Kriterien erfüllen), und ANY wie ODER.

Filter

Ober- und Untergrenze in Where:

```
…WHERE start_date BETWEEN ‚2002-01-01‘ AND ‚2002-12-31‘;
```

Mehrere Bedingungen in Where:

```
…WHERE product_cd IN (‚CHK‘,‘SAV‘,‘CD‘);
```

entspricht: `WHERE product_cd = ‚CHK‘ OR product_cd = ‚SAV‘ OR …`

Joins

Kartesisches Produkt/Cross Join

```
SELECT e.fname, e.lname, d.name
      FROM employee e JOIN department d;
```

Gibt das kartesische Produkt der beiden Tabellen aus, e hat 18 Zeilen, d hat 3 Zeilen, für jede Zeile aus e wird eine Zeile von d ausgegeben, also insgesamt 18*3 Zeilen; → sinnlos, wird auch nie verwendet

Inner Joins

Verbindung der zwei Spalten muss klar sein: `…JOIN…ON e.dept_id = d.dept_id;` wobei dept_id in der Tabelle d der Primärschlüssel ist und in der Tabelle e ein Fremdschlüssel. Der Server sucht findet also in ‚e‘ den Hinweis auf die Tabelle ‚d‘ und sucht dort den erfragten Tabelleneintrag (d.name).

Falls, wie im Beispiel die Tabellennamen gleich sind, kann auch `…USING (dept_id);` geschrieben werden.

Outer Joins

Left- und Right Outer Join:

```
SELECT a.account_id, a.cust_id, b.name
      FROM account a LEFT OUTER JOIN business b
      ON a.cust_id = b.cust_id;
```

Left: Wählt alle gewählten Zeilen der Linken Tabelle und vergleicht die rechte Tabelle damit. Right: Umgekehrt. Beispiel: Tabelle account hat 12 Zeilen, Tabelle business 5 -> Left Outer Join nimmt 12 Zeilen in die Ergebnismenge auf, Right Outer Join 5.