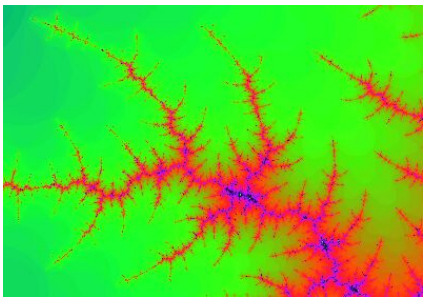
Datenbanken

2. Einfache SQL Abfragen



Fachhochschule Darmstadt
Fachbereich Informatik

Dr. Peter Nevermann

Rückblick

- **Begriffe**
  - Datenbank
  - Datenbank Management System (DBMS)
  - Datenbanksystem
  - Miniwelt
  - Datenmodell ➔ Schema ➔ Daten
  - Katalog
- **Eigenschaften & Nutzen von Datenbanksystemen**
  - Datenunabhängigkeit
  - Paralleler Zugriff
  - Datenintegrität
  - Zugriffskontrolle
  - Wiederanlauf nach Fehlersituationen

Einfache SQL Anfrage

- **Alle Informationen über Frau Noll?**

```
SELECT *
FROM mitarbeiter
WHERE nachname = 'Noll';
```

nachname	vorname	pnr	gebdatum	adresse	geschlecht	gehalt	vpnr	abtnr
Noll	Inga	8888	1967-09-28	Heinemannstr. 6, Frankfurt	W	70000	NULL	1

SELECT <attribut-liste>
FROM <tabelle>
[**WHERE** <bedingung>];

- Statt alle Spalten einer Tabelle in SELECT aufzulisten kann man auch * schreiben, wie das Beispiel auf der Folie zeigt.
- Lässt man die WHERE-Bedingung weg, so bekommt man in der Ergebnistabelle genau so viele Zeilen wie die Ausgangstabelle hat.

Berechnete Werte

- **Frau Noll's Gehalt in US-Dollar**

```
SELECT nachname, gehalt * 1.25
FROM mitarbeiter
WHERE nachname = 'Noll';
```

nachname	gehalt * 1.25
Noll	87500

Datenbanken I | 4 | SS 05, 22.04.2005 | FH Darmstadt, FB Informatik, Dr. P. Nevermann

Operatoren:

•Arithmetik
Addition (+), Subtraktion (-), Multiplikation (*), Division (/)

•Zeichenketten (Strings)
Verkettung (||)

Achtung: Der || Operator für String-Verkettung funktioniert weder in MySQL noch in Access. In Access funktioniert stattdessen '+' und in MySQL gibt es die CONCAT() Funktion.

Funktionen:

•Math-Funktionen

- •ABS(n), EXP(n), LN(n), MOD(n, m), ROUND(n), ...

•String-Funktionen

- •CONCAT(s1, s2) [nicht in Access!!]
- •SUBSTRING(...), INSTR(...), LOCATE(...), REPLACE(...), TRIM(s), UPPER(s), LOWER(s)

•Datumsfunktionen

- •CURRENT_DATE(), CURRENT_TIME(), NOW(), DATE_ADD(...), DATE_SUB(...), YEAR(...), MONTH(...), DAY(...)

Operatoren und Funktionen können in SELECT und WHERE verwendet werden.

Vergleichsoperatoren in WHERE

- **Alle Mitarbeiter mit mindestens EUR 60.000 Gehalt**

```
SELECT nachname, vorname, gehalt
FROM mitarbeiter
WHERE gehalt >= 60000;
```

nachname	vorname	gehalt
Mayer	Johannes	60000
Grenz	Pauline	65000
Bach	Robert	60000
Noll	Inga	70000

Datenbanken I | 5 | SS 05, 22.04.2005 | FH Darmstadt, FB Informatik, Dr. P. Nevermann

SQL kennt folgende Vergleichsoperatoren:

•gleich (**=**), größer (**>**), kleiner (**<**), größer-gleich (**>=**), kleiner-gleich (**<=**), ungleich (**<>**)

Diese Vergleichsoperatoren können nicht nur auf numerische Attribute, sondern auch auf Attribute der Datentypen Zeichenkette (String) oder Datum angewendet werden.

Beispiele:

•Alle Mitarbeiter Jahrgang 1980 oder jünger

```
SELECT nachname, gebdatum
FROM mitarbeiter
WHERE gebdatum >= '1980-01-01';
```

•Alle Mitarbeiter in der alphabetischen Sortierung nach 'N'

```
SELECT nachname, vorname
FROM mitarbeiter
WHERE nachname >= 'N';
```

Vergleichsoperator LIKE

- **Mitarbeiter die im Juni Geburtstag haben?**

```
SELECT nachname, vorname, gebdatum
FROM mitarbeiter
WHERE gebdatum LIKE '____-06-__';
```

nachname	vorname	gebdatum
Schneider	Werner	1977-06-06

Datenbanken I | 6 | SS 05, 22.04.2005 | FH Darmstadt, FB Informatik, Dr. P. Nevermann

Hauptsächlich für Zeichenketten (Strings) gibt es den mächtigen Vergleichoperator LIKE, der es erlaubt auf Teile des Wertes zu vergleichen. Dafür werden zwei besondere Platzhalter-Zeichen (Wildcards) verwendet:

- '%' beliebig-viele-Zeichen
- '_' genau-ein-Zeichen

Beispiele:

- Alle Mitarbeiter mit Wohnort in Darmstadt

```
SELECT nachname, vorname, adresse
FROM mitarbeiter
WHERE adresse LIKE '%Darmstadt%';
```

Frage: Hat die Spalte "adresse" in unserer Beispiel-Datenbank die geeignete Form für obige Abfrage? Was wäre besser?

Sortierung der Ausgabe

- **Nachname und Vorname der Mitarbeiter in alphabetischer Reihenfolge?**

```
SELECT nachname, vorname
FROM mitarbeiter
ORDER BY nachname, vorname
```

nachname	vorname
Bach	Robert
Clausen	Achim
Grenz	Pauline
Mayer	Johannes
Noll	Inga
Schneider	Werner
Schumann	Ingrid
Wegner	Walter

Datenbanken I | 7 | SS 05, 22.04.2005 | FH Darmstadt, FB Informatik, Dr. P. Nevermann

Mit ORDER BY kann die Ergebnistabelle sortiert werden.

Man kann die Richtung der Sortierung mit den Schlüsselwörtern **ASC** (ascending, aufsteigend) und **DESC** (descending, absteigend) vorgeben. Ohne zusätzliche Angaben wird aufsteigend sortiert, d.h. man kann das Schlüsselwort ASC weglassen.

Beispiele:

•Liste der Mitarbeiter in absteigender Sortierung
SELECT nachname, vorname
FROM mitarbeiter
ORDER BY nachname DESC, vorname DESC

Logische Operatoren in WHERE

- **Mitarbeiter des Jahrgangs 1970 oder älter die weniger als EUR 60.000 verdienen?**

```
SELECT nachname, vorname, gebdatum, gehalt
FROM mitarbeiter
WHERE gebdatum <= '1970-12-31' AND gehalt < 60000;
```

nachname	vorname	gebdatum	gehalt
Wegner	Walter	1970-12-15	55000

Datenbanken I | 8 | SS 05, 22.04.2005 | FH Darmstadt, FB Informatik, Dr. P. Nevermann

Logische Operatoren in SQL:

•**AND** (und), **OR** (oder) und **NOT** (nicht)

Logische Ausdrücke in WHERE können mit '(' und ')' geklammert werden. Die Operatoren haben aber auch eine Vorangsordung, nämlich NOT > AND > OR (d.h. NOT bindet am stärksten, OR am schwächsten). So sind die beiden folgende Ausdrücke Äquivalent:

•NOT a OR b AND NOT c

•(NOT a) OR (b AND (NOT c))

Der Operator OR stellt das *inklusive oder* dar, d.h. mit der Bedingung (a OR b) werden alle Tupel selektiert, welche a, b oder a^b genügen.

Frage: Wie muß die Bedingung für das *exclusive oder* formuliert werden?

•Beispiele:
Alle Mitarbeiter dessen Name mit "Sch" beginnt
SELECT nachname, vorname
FROM mitarbeiter
WHERE nachname >= 'Sch' AND nachname < 'Sci';

•Alle weiblichen Mitarbeiter mit Wohnort in Frankfurt oder Mainz
SELECT nachname, vorname, adresse
FROM mitarbeiter
WHERE geschlecht = 'W' AND (adresse LIKE %Frankfurt% OR adresse LIKE %Mainz%);

Die Bedingung "value >= A and value <= B" kann übrigens auch mit dem Operator **BETWEEN** ausgedrückt werden.

Beispiele:

•Alle Mitarbeiter der Jahrgänge 1971 bis 1980
SELECT nachname, gebdatum
FROM mitarbeiter
WHERE YEAR(gebdatum) BETWEEN 1971 AND 1980;

Aufgabe:

Wie muß eine SQL Query lauten, die alle männlichen Mitarbeiter die im 2. Halbjahr Geburtstag haben findet?

Duplikate in Abfrage-Ergebnissen

Alle Gehälter (mit Duplikaten)

```
SELECT gehalt
FROM mitarbeiter;
```

gehalt
60000
55000
65000
45000
50000
55000
60000
70000

Alle Gehälter (ohne Duplikate)

```
SELECT DISTINCT gehalt
FROM mitarbeiter;
```

gehalt
60000
55000
65000
45000
50000
70000

Datenbanken I | 10 | SS 05, 22.04.2005 | FH Darmstadt, FB Informatik, Dr. P. Nevermann

SQL eliminiert Duplikate aus Abfrage-Ergebnissen nicht automatisch!

Gründe hierfür sind:

- Duplikat-Bereinigung ist rechenaufwendig
- Duplikate können erwünscht sein
- Aggregat-Funktionen (kommen später!) würden nicht funktionieren

Mit dem Schlüsselwort **DISTINCT** bestimmt man, daß Duplikate in der Ergebnistabelle eliminiert werden sollen. Daneben gibt es noch das Schlüsselwort **ALL** mit der gegensätzlichen Bedeutung (welches man aber weglassen kann, da 'SELECT' und 'SELECT ALL' äquivalent sind).

Microsoft Access kennt zusätzlich noch das Schlüsselwort **DISTINCTROW**, mit dem Duplikate basierend auf allen Spalten einer verknüpften Abfrage (kommt später!) eliminiert werden. Bei einfachen Abfragen auf einer Tabelle, wie wir sie momentan hier betrachten, ist DISTINCTROW äquivalent mit ALL weil eine Grundtabelle in Access keine Duplikate haben darf.

Selbstverständlich kann sich DISTINCT auf mehrere Spalten beziehen.

Beispiel:

- Alle (Geschlecht, Gehalt)-Tupel ohne Duplikate

```
SELECT DISTINCT geschlecht, gehalt
FROM mitarbeiter;
```

Mengen-Operationen

- **Menge aller (Gehalt, Geschlecht)-Tupel zu männlichen Mitarbeitern vereinigt mit der entsprechenden Tupel-Menge von weiblichen Mitarbeitern?**

```
SELECT gehalt, geschlecht
FROM mitarbeiter
WHERE geschlecht = 'M'
UNION
SELECT gehalt, geschlecht
FROM mitarbeiter
WHERE geschlecht = 'W';
```

gehalt	geschlecht
60000	M
55000	M
45000	M
50000	M
65000	W
55000	W
70000	W

Datenbanken I | 11 | SS 05, 22.04.2005 | FH Darmstadt, FB Informatik, Dr. P. Nevermann

Mit dem **UNION** Mengen-Operator können tupel-kompatible Ergebnis-Mengen vereinigt werden.

Dabei werden Duplikate automatisch eliminiert, d.h. das Schlüsselwort DISTINCT kann aus den beteiligten Abfragen weggelassen werden. Verwendet man allerdings den Operator **UNION ALL**, dann bleiben Duplikate erhalten.

In SQL2 werden noch zwei weitere Mengen-Operatoren beschrieben: mit **INTERSECT** bzw. **INTERSECT ALL** kann der Durchschnitt tupel-kompatibler Ergebnismengen gebildet werden, mit **EXCEPT** bzw. **EXCEPT ALL** kann die Differenz tupel-kompatibler Ergebnismengen gebildet werden.

Achtung: Weder Access noch MySQL unterstützen INTERSECT oder EXCEPT.

NULL-Werte

- **Wer ist in diesem Laden eigentlich der Boss?**

```
SELECT nachname, vorname
FROM mitarbeiter
WHERE vpnr IS NULL;
```

nachname	vorname
Noll	Inga

Ein Attribut-Wert kann **NULL** sein, mit der Bedeutung, daß der Wert *unbekannt* bzw. *unbestimmt* ist. Dabei ist NULL nicht dasselbe wie der numerische Wert 0 oder die leere Zeichenkette "".

In SQL wird allerdings NULL nicht als zusätzlicher *ausgezeichneter* Wert betrachtet, sondern NULL-Werte sind in SQL zu jedem anderen Wert unvergleichbar (also auch zu anderen NULL-Werten!!). Alle Vergleichsoperatoren ergeben zusammen mit NULL-Werten immer das Ergebnis "falsch", d.h. NULL ist weder = noch <>, weder > noch <= im Vergleich mit "Sahnetorte", 1999-04-06 oder 4711.

Deshalb wird auch nicht der Versgleichsoperator = verwendet um auf NULL oder nicht NULL zu testen, sondern NULL bzw. nicht NULL muß immer explizit über das Schlüsselwort **IS NULL** bzw. **IS NOT NULL** abgefragt werden.

Aggregatfunktionen

Wieviele Mitarbeiter verdienen mehr als EUR 55.000?

```
SELECT COUNT(*)
FROM mitarbeiter
WHERE gehalt > 55000;
```

COUNT(*)
4

Was ist das Durchschnittsgehalt?

```
SELECT AVG(gehalt)
FROM mitarbeiter;
```

AVG(gehalt)
57500.0000

Datenbanken I | 13 | SS 05, 22.04.2005 | FH Darmstadt, FB Informatik, Dr. P. Nevermann

Aggregatfunktionen in SQL:

•**COUNT**: Anzahl der selektierten Tupel

•**SUM**, **MAX**, **MIN**, **AVG**: Summe, Maximal-, Minimal- bzw. Durchschnitts-Wert einer Ergebnismenge von numerischen Werten.

Die Funktionen MAX und MIN können auch auf nicht-numerische Attribute (mit linearen Ordnung) angewendet werden.

Beispiele:

•Höchstes Gehalt unter den männlichen Mitarbeitern
SELECT MAX(gehalt)
FROM mitarbeiter
WHERE geschlecht = 'M';

•Erster und letzter Nachname in der Sortierreihenfolge
SELECT MIN(nachname), MAX(nachname)
FROM mitarbeiter;

•Frühestes und spätestes Geburtsdatum
SELECT MIN(gebdatum), MAX(gebdatum)
FROM mitarbeiter;

Regel:

•Mit einer WHERE-Bedingung kann die Menge der Werte, auf die eine Aggregatsfunktion angewendet wird, vorab eingeschränkt werden.

Das Schlüsselwort **DISTINCT** kann in Kombination mit COUNT verwendet werden, wenn man Anzahl der unterschiedlichen selektierten Tupel wissen möchte [... geht aber leider nicht mit Access ☹].

Beispiel:

•Anzahl unterschiedlicher Gehälter

```
SELECT COUNT(DISTINCT gehalt)
FROM mitarbeiter;
```

•Anzahl unterschiedlicher (Geschlecht, Gehalt)-Kombinationen

```
SELECT COUNT(DISTINCT geschlecht, gehalt)
FROM mitarbeiter;
```

Aufgabe:

Durchschnittsgehalt pro Geschlecht: formulieren Sie eine SQL Query derart, daß die Ergebnistabelle aus zwei Spalten besteht und die folgenden zwei Tupel als Zeilen hat:

•("M", Durchschnittsgehalt-Männer)

•("W", Durchschnittsgehalt-Frauen).

Weitere Aggregatsfunktionen:

Access

•FIRST, LAST (Spaltenwert der *ersten* bzw. *letzten* Zeile des Ergebnisses)

•STDEV, VAR (*Standardabweichung* bzw. *Varianz* der Spaltenwerte)

MySQL

•STDDEV (*Standardabweichung* der Spaltenwerte)

Gruppierungen

Was ist das Durchschnittsgehalt *pro Abteilung*?

```
SELECT abtnr, AVG(gehalt)
FROM mitarbeiter
GROUP BY abtnr;
```

abtnr	AVG(gehalt)
1	67500.0000
2	53333.3333
3	55000.0000

Was ist das Durchschnittsgehalt der *männlichen Mitarbeiter pro Abteilung*?

```
SELECT abtnr, AVG(gehalt)
FROM mitarbeiter
WHERE geschlecht = 'M'
GROUP BY abtnr;
```

abtnr	AVG(gehalt)
2	53333.3333
3	55000.0000

Datenbanken I | 15 | SS 05, 22.04.2005 | FH Darmstadt, FB Informatik, Dr. P. Nevermann

Beispiel (vorige Aufgabe):

•Durchschnittsgehalt pro Geschlecht (**ohne** Gruppierung)?

```
SELECT 'M', AVG(gehalt)
FROM mitarbeiter
WHERE geschlecht = 'M'
UNION
SELECT 'W', AVG(gehalt)
FROM mitarbeiter
WHERE geschlecht = 'W'
```

•Durchschnittsgehalt pro Geschlecht (**mit** Gruppierung)?

```
SELECT geschlecht, AVG(gehalt)
FROM mitarbeiter
GROUP BY geschlecht;
```

Regel:

•Verwendet man GROUP BY so müssen alle unter SELECT gelisteten Attribute entweder auch unter GROUP BY gelistet sein, oder als Argument einer Aggregatsfunktion vorkommen.

•Mit einer WHERE-Bedingung kann die Menge der Tupel vor der Gruppierung eingeschränkt werden.

Gruppierungen

Was ist das Durchschnitts-gehalt *pro Abteilung mit mehr als zwei Mitarbeitern*?

```
SELECT abtnr, AVG(gehalt)
FROM mitarbeiter
GROUP BY abtnr
HAVING count(*) > 2;
```

abtnt	AVG(gehalt)
2	53333.3333
3	55000.0000

Was ist das Durchschnitts-gehalt der *männlichen Mitarbeiter pro Abteilung mit mehr als zwei Mitarbeitern*?

```
SELECT abtnr, AVG(gehalt)
FROM mitarbeiter
WHERE geschlecht = 'M'
GROUP BY abtnr
HAVING count(*) > 2;
```

abtnt	AVG(gehalt)
2	53333.3333

Datenbanken I | 16 | SS 05, 22.04.2005 | FH Darmstadt, FB Informatik, Dr. P. Nevermann

Regel:

•Mit einer WHERE-Bedingung kann die Menge der Tupel vor der Gruppierung eingeschränkt werden.

•Mit einer HAVING-Bedingung kann die Menge der Gruppen (d.h. Ergebnisse nach der Gruppierung) eingeschränkt werden.

Die zweite Abfrage auf der Folie ist falsch!

Warum?

Wir wollten das Durchschnittsgehalt der männlichen Mitarbeiter in allen Abteilungen mit mehr als 2 Mitarbeitern ... obige Abfrage liefert das Ergebnis aber nur für Abteilungen mit mehr als zwei männlichen Mitarbeitern.

Aufgabe:

Wie muß die zweite Abfrage auf der Folie richtig lauten?

Ausblick

- **Komplexere SQL Abfragen**
  - Unterabfragen (nested queries)
  - Abfragen die mehr als eine Tabelle einbeziehen (join)
  - Datenmanipulation (INSERT, UPDATE, DELETE)

Anhang: Tabellen

projekt

projname	pnummer	port	abtnummer
Saturn	66	Darmstadt	1
Jupiter	55	Mainz	2
Mars	44	Frankfurt	2
Venus	33	Darmstadt	2
Mercur	22	Mainz	2
Pluto	11	Frankfurt	3

abteilung

abtname	anr	mgrpnr	mgrstartdat
Headquarters	1	3333	1996-11-25
Forschung	2	1111	1990-02-25
Verwaltung	3	7777	1999-03-03

Die hier verwendete Beispieldatenbank orientiert sich an Beispielen aus: R. Elmasri – S. Navathe, *Fundamentals of Database Systems,* Addison Wesley, 2003

Sie können die Beispieldatenbank "Firma" für Access und MySQL unter der Adresse http://www.fbi.fh-darmstadt.de/~pnevermann/ss05/Download/V2.ZIP herunterladen.

Anhang: Tabellen

mitarbeiter

nachname	vorname	pnr	gebdatum	adresse	geschlecht	gehalt	vpnr	abtnr
Mayer	Johannes	1111	1965-07-23	Annastr.3, Frankfurt	M	60000	3333	2
Wegner	Walter	2222	1970-12-15	Bertaweg 4, Aschaffenbug	M	55000	1111	2
Grenz	Pauline	3333	1966-03-01	Clarastr. 3, Darmstadt	W	65000	8888	1
Clausen	Achim	4444	1978-01-14	Daumallee 7, Mainz	M	45000	1111	2
Schneider	Werner	5555	1977-06-06	Emilstr. 8, Darmstadt	M	50000	7777	3
Schumann	Ingrid	6666	1982-02-27	Frankenplatz 1, Frankfurt	W	55000	7777	3
Bach	Robert	7777	1959-08-08	Gustav-Allee 2, Mainz	M	60000	3333	3
Noll	Inga	8888	1967-09-28	Heinemannstr. 6, Frankfurt	W	70000	NULL	1

Anhang: Tabellen

angehoeriger

pnr	vorname	geschlecht	gebdatum	beziehung
2222	Else	W	1972-11-11	P
2222	Jan	M	1997-05-04	K
2222	Julia	W	1999-06-07	K
4444	Ingrid	W	1981-01-06	P
6666	Thomas	M	1982-11-08	P
8888	Udo	M	1963-12-09	P
8888	Kai	M	1988-03-09	K
8888	Klaus	M	1989-09-14	K
8888	Kristel	W	1991-05-05	K
7777	Maya	W	1960-12-07	P
7777	Lukas	M	1985-01-07	K

Anhang: Tabellen

arbeitet_in

mapnr	projectnr	stunden
1111	22	10
2222	33	10
4444	44	20
1111	55	10
2222	22	10
4444	33	20
1111	44	10
2222	55	10
3333	66	20
3333	22	10
3333	44	10
5555	11	40
6666	11	40
7777	11	20
7777	66	20
1111	66	10

standort

abtnummer	stadt
1	Darmstadt
2	Darmstadt
2	Mainz
2	Frankfurt
3	Darmstadt