## SQL (Structured Query Language)

= relationale DB-Sprache zur Definition, Abfrage und Manipulation von Daten in relationalen Datenbanken.

→ 1986 wurde der erste SQL-Standard geschaffen
→ 2008 wurde zuletzt der Standard geändert
→ alle aktuellen DBMS halten sich im Wesentlichen an diesen Standard

**1. Umbenennung**
**SELECT** FName as Fakultät
**FROM** FAKULTÄTEN;

**O2. Durchschnitt** → nicht in allen DBMS verfügbar
**SELECT** * **FROM** STUDENT_Master (* = alle Spalten)
INTERSECT
**SELECT** * **FROM** STUDENT_Bafög;

**O2. Vereinigung** → nicht in allen DBMS verfügbar
**SELECT** SName **FROM** STUDENT_Master
UNION
**SELECT** SName **FROM** STUDENT_Bafög;

**O2. Differenz** → nicht in allen DBMS verfügbar
**SELECT** * **FROM** STUDENT_Master
EXCEPT
**SELECT** * **FROM** STUDENT_Bafög;

**O2. Produkt**
**SELECT** * **FROM** STUDENT_Bafög, FAKULTÄTEN;

**RO3. Projektion**
$P_{MName,MVorname}$ (Mitarbeiter)
**SELECT** MName, MVorname **FROM** MITARBEITER;

**RO3. Selection**
$S_{Wohnort=\text{“Dresden“}}$(MITARBEITER)
**SELECT** * **FROM** MITARBEITER **WHERE** Wohnort=“Dresden“;

**RO3. Projektion mit Selection**
$P_{MName,MVorname}(S_{Wohnort=\text{“Dresden“}}(MITARBEITER))$
**SELECT** MName, MVorname **FROM** MITARBEITER **WHERE** Wohnort=“Dresden“;

© A. Kaden

→ besondere Operatoren in der WHERE-Klausel:

between → ermittelt Werte zwischen festgelegten Grenzen
**SELECT** FNr **FROM** FAKULTÄTEN **WHERE** FNr between 1 and 3

like → prüft, ob eine Teilzeichenkette in einer größeren enthalten ist
**SELECT** SName **FROM** STUDENT_Master **WHERE** SName like "Sch*"

→ Sortierungen
aufsteigend: **SELECT** * **FROM** STUDENT_Master ORDER by SName;
absteigend: **SELECT** * **FROM** STUDENT_Master ORDER by SName desc; → *IMMER ANS ENDE SETZEN!*

→ Ausblenden von Duplikaten mit distinct
**SELECT** distinct Wohnort **FROM** MITARBEITER;

**RO3. Join**
→ als einfachster Join kann das Kreuzprodukt betrachtet werden
→ wird hier nicht weiter untersucht, da für praktischen Einsatz unbrauchbar

→ Inner Join (auch Natural Join)
Bsp.:
$J_{INr}$(INSTITUTE, MITARBEITER)

```
SELECT *
FROM INSTITUTE
INNER JOIN MITARBEITER ON  INSTITUTE.INr = MITARBEITER.INr;
```

*auch:*

```
SELECT *
FROM INSTITUTE, MITARBEITER
WHERE INSTITUTE.INr = MITARBEITER.INr;
```

→ Left Join (auch Left Outer Join)
Bsp.:
$J_{INr}$(INSTITUTE, MITARBEITER)

```
SELECT *
FROM MITARBEITER
LEFT JOIN INSTITUTE ON  MITARBEITER.INr = INSTITUTE.INr;
```

→ Right Join (auch Right Outer Join)
Bsp.:
$J_{INr}$(INSTITUTE, MITARBEITER)

```
SELECT *
FROM MITARBEITER
RIGHT JOIN INSTITUTE ON  MITARBEITER.INr = INSTITUTE.INr;
```

→ Full Outer Join

© A. Kaden

Bsp.:
$J_{INr}$(INSTITUTE, MITARBEITER)

**SELECT** *
**FROM** MITARBEITER
FULL OUTER JOIN INSTITUTE ON MITARBEITER.INr = INSTITUTE.INr; geht nicht in Access

## 4. Weiter Operationen, die nur sehr schwer mit der Relationenalgebra darstellbar sind

→ Join über Subqueries
Erläuterung: Verschachtelung von Einzelabfragen über Schlüsselfelder

Bsp.: In welchem Institut arbeitet Willi Qualle?
**SELECT** IName
**FROM** INSTITUTE
**WHERE** INr **IN** (**SELECT** INr **FROM** MITARBEITER **WHERE**
MName=”Qualle” and MVorname=”Willi”);

→ Gruppenfunktionen (Aggregatfunktionen)

### 1. MAX/MIN

→ sucht den größten / kleinsten Wert eines Datenfeldes
Bsp.: Die größte Anzahl von Instituten.
**SELECT** max(Inst_anz) as Anzahl **FROM** FAKULTÄTEN;

→ in Verbindung mit einem SELF-JOIN
Bsp.: Welche Fakultät hat die meisten Institute?
**SELECT** FName
**FROM** FAKULTÄTEN
**WHERE** Inst_anz **IN** (**SELECT** max(Inst_anz) **FROM** FAKULTÄTEN);

### 2. AVG

→ Ermittelt den Mittelwert eines Datenfeldes
Bsp.:
**SELECT** avg(Inst_anz) as Durchschnitt **FROM** FAKULTÄTEN;

### 3. SUM

→ Bildet die Summe von Werten eines Datenfeldes
Bsp.: Wie viel Institute hat die TU Freiberg?
**SELECT** sum(Inst_anz) as Summe **FROM** FAKULTÄTEN;

### 4. COUNT

→ Zählt die Werte eines Datenfeldes.
Bsp.: Wie viele Mitarbeiter wohnen in Dresden?
**SELECT** count(*) as Anzahl
**FROM** MITARBEITER
**WHERE** Wohnort=”Dresden”;

© A. Kaden

## 5. Aggregatfunktionen lassen sich auch auf Datensatzgruppen einzeln anwenden (Gruppierungen)

→ GROUP by
Bsp.: Anzahl der Mitarbeiter aus den einzelnen Orten:
**SELECT** count(*) as Anzahl
**FROM** MITARBEITER
**GROUP by** Wohnort;
→ Die Mitarbeiter werden nach Wohnorten gruppiert und dann gezählt.

→ HAVING
Gruppen können auch von Bedingungen abhängen
Bsp:: In welchen Orten wohnen zwei Mitarbeiter?
**SELECT** Wohnort
**FROM** MITARBEITER
**GROUP by** Wohnort
**HAVING** count(*)=2; immer mit "GROUP by"

### Zusammenfassung – Reihenfolge der SQL-Anweisungen

**SELECT (Distinct)**
**FROM**
**WHERE**
**GROUP by**
**HAVING**
**ORDER by**

### Formuliere folgende Abfragen in SQL

1. Gesucht sind die Vornamen von Masterstudenten aufsteigend sortiert.
2. Gesucht sind alle Bafög-Studenten mit einem „a" im Vornamen.
3. Wie viele Bafög-Studenten sind erfasst?
4. Wie viele Masterstudenten beziehen Bafög?
5. Angabe aller Fakultätsinfos mit zugehörigen Institutsinformationen.
6. Angabe aller Institutsinfos mit zugehörigen Fakultätsinformationen.
7. ~~Angabe aller Informationen zu Fakultäten und Institute.~~
8. An welchem Institut studieren Anna und Fritz.
9. Ausgabe der Anzahl gleicher Vornamen bei den Masterstudenten.

© A. Kaden