

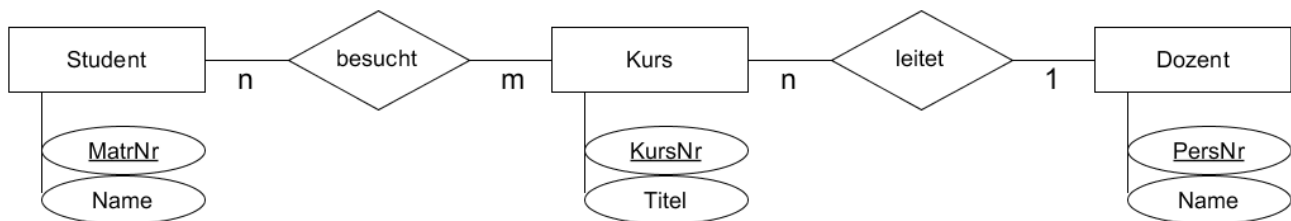
## Structured Query Language (SQL)

SQL ist eine Datenbanksprache, mit der alle notwendigen Operationen zum Arbeiten mit einer relationalen Datenbank ausgedrückt werden können. Dazu gehört die Definition von Tabellen, das Einfügen, Verändern und Löschen von Datensätzen und die Abfrage von Daten.

Die Syntax von SQL ist relativ einfach aufgebaut und an die englische Umgangssprache angelehnt. Ein gemeinsames Gremium von ISO und IEC standardisiert die Sprache unter Mitwirkung nationaler Gremien wie ANSI oder DIN. Fast alle gängigen Datenbanksysteme unterstützen SQL, allerdings in unterschiedlichem Umfang und mit leicht voneinander abweichenden „Dialekten“.

Wir beschäftigen uns im Unterricht (auch im Hinblick auf das Abitur) hauptsächlich mit **Abfragen** (engl. **queries**), das heißt mit SQL-Ausdrücken, mit denen man Daten aus einer Datenbank herausuchen kann. Während die SQL-Ausdrücke zum Erstellen von Tabellen oder Einfügen von Datensätzen relativ einfach sind, können Abfragen durchaus komplex sein.

Betrachten wir als Beispiel die Datenbank einer Universität:



Einfache Abfragen betreffen lediglich Daten aus einer Tabelle, zum Beispiel

*„Suche die Matrikelnummern aller Studenten, die nach dem 01.01.2000 geboren sind.“*

Komplexe Abfragen suchen nach Daten, für die Informationen aus mehreren Tabellen miteinander verknüpft werden müssen, zum Beispiel

*„Suche die Namen der Studenten, die der Dozent „Wirth“ in seinen Kursen unterrichtet.“*

Für die Erläuterungen zu Abfragen werden als Beispiel die folgenden Daten verwendet:

**Student**

<u>MatrNr</u>	Name
1011	Schmitz
1012	Müller
1013	Jansen
1014	Meier

**besucht**

<u>MatrNr</u>	<u>KursNr</u>
1012	K01
1012	K03
1013	K02
1014	K03

**Kurs**

<u>KursNr</u>	Titel	PersNr
K01	Java	D02
K02	HTML	D02
K03	Datenbank	D01

**Dozent**

<u>PersNr</u>	Name
D01	Wirth
D02	Dijkstra
D03	Knuth

## Abfragen mit SELECT über eine Tabelle

Eine SELECT-Abfrage gibt an, aus welcher Tabelle bzw. welchen Tabellen und welchen Spalten Daten gesucht werden. Das Ergebnis einer SELECT-Abfrage ist selbst auch wieder eine Tabelle mit Spaltenüberschriften und Daten.

*Allgemeine Form einer SELECT-Abfrage*

**SELECT** [Spalte(n)] **FROM** [Tabelle(n)] **WHERE** [Suchausdruck] ;

*Abfrage aller Daten einer Tabelle*

**SELECT** \* **FROM** Dozent;

\* bedeutet hier „alle Spalten der Tabelle“

*Ergebnis →*

PersNr	Name
D01	Wirth
D02	Dijkstra
D03	Knuth

*Einschränkung auf bestimmte Spalten („Projektion“)*

**SELECT** KursNr, Titel **FROM** Kurs;

KursNr	Titel
K01	Java
K02	HTML
K03	Datenbank

*Einschränkung auf bestimmte Zeilen („Selektion“)*

Um die Suche auf bestimmte Zeilen einzuschränken verwendet man die „**WHERE-Klausel**“. Man kann mithilfe von Bedingungen sehr fein bestimmen, welche Zeilen ausgewählt werden sollen.

*Operatoren für die where-Klausel*

= (gleich)

<> (ungleich)

>, >= (größer bzw. größer gleich)

<, <= (kleiner bzw. kleiner gleich)

BETWEEN a AND b (zwischen den Zahlen a und b; Alternative zu  $x \geq a \text{ AND } x \leq b$ )

*Logische Verknüpfungen*

AND

OR

NOT

*Beispiel für Abfrage mit WHERE-Klausel*

**SELECT** PersNr **FROM** Dozent **WHERE** Name = 'Knuth';

PersNr
D03

*Beispiel für WHERE-Klausel mit mehreren Bedingungen*

**SELECT** Titel, PersNr **FROM** Kurs  
**WHERE** Titel = 'Datenbank' **OR** Titel = 'HTML';

Titel	PersNr
HTML	D02
Datenbank	D01

Man kann in der WHERE-Klausel nach **Textmustern**, z.B. einzelnen Wörtern oder Anfangsbuchstaben suchen, mithilfe des Operators LIKE.

Dabei kann man sogenannte **Wildcards** verwenden:

% bedeutet „eine beliebige Anzahl von Buchstaben“, also 0, 1, oder mehr

\_ bedeutet „genau ein Buchstabe“.

*Beispiel mit Textmustersuche: Namen mit Anfangsbuchstabe M*

```
SELECT * FROM Student
WHERE Name LIKE 'M%';
```

MatrNr	Name
1011	Meier
1012	Müller

Schließlich gibt es noch die Möglichkeit, nach einer **Menge** oder Liste von Suchbegriffen zu suchen mithilfe des Operators IN:

*Beispiel für WHERE-Klausel mit IN*

```
SELECT KursNr, Titel FROM Kurs
WHERE Titel IN ('HTML', 'Java');
```

KursNr	Titel
K01	Java
K02	HTML

## Abfragen mit Datum

Um abzufragen, ob ein Datensatz mit einem Datum vor / nach einem bestimmten Datum oder innerhalb eines Zeitraums liegt, kannst du die Operatoren >, <, =, BETWEEN usw. verwenden.

Vor und hinter jedes Datum musst du ein # setzen (z.B. #1/1/1996#).

Beachte: anders als bei uns ist das Datumsformat in den USA Monat / Tag / Jahr.

*Beispiel: Dozenten, die vor 1935 geboren sind  
(angenommen, die Dozenten-Tabelle hat die Spalte „Geburtsdatum“)*

```
SELECT Name, Geburtsdatum FROM Dozent
WHERE Geburtsdatum < #1/1/1935#;
```

Name	Geburtsdt.
Dijkstra	5/11/1930
Wirth	2/15/1934

## Reihenfolge der Ergebnisse ordnen

Bei Ergebnissen mit vielen Daten hilft es der Übersicht, die Ergebnisse in einer bestimmten Reihenfolge anzuzeigen. Die Tabelle Dozent ist nach Personalnummern geordnet. Wenn man eine Liste der Dozenten erzeugen möchte, ist es unter Umständen sinnvoller, sie alphabetisch nach den Namen zu ordnen.

*Allgemeine Form einer SELECT-Abfrage mit Sortierung*

```
SELECT [Spalte(n)] FROM [Tabelle(n)] WHERE [Suchausdruck]  
ORDER BY [Spalte(n)] ASC | DESC ;
```

ASC steht dabei für „ascending“, also aufsteigende Reihenfolge (für Namen also A., B., C., usw.)  
Das ist die Standardreihenfolge und kann auch einfach weggelassen werden.  
DESC steht für „descending“, also die absteigende Reihenfolge (Z., Y., X., usw.)

*Beispiel: Ergebnisse in alphabetischer Reihenfolge*

```
SELECT Name FROM Dozent  
ORDER BY Name;
```

Name
Dijkstra
Knuth
Wirth

## UNION: Ergebnisse zweier SELECT-Abfragen verbinden

In manchen Fällen (die allerdings nicht so häufig vorkommen) möchte man Daten aus zwei Tabellen miteinander verbinden. In unserem Beispiel könnte man alle Personen der Universität, also Studenten und Dozenten auflisten.

*Allgemeine Form der UNION-Abfrage*

```
SELECT ... FROM ... WHERE ...  
UNION  
SELECT ... FROM ... WHERE ... ;
```

*Beispiel: Abfrage aller Namen aus Student und Dozent*

```
SELECT Name FROM Student UNION  
SELECT Name FROM Dozent  
ORDER BY Name;
```

Hinweis:

Die ausgewählten Spalten müssen zueinander passen.

Es würde z.B. keinen Sinn machen, Ergebnisse von zwei SELECT-Abfragen zu verbinden, die unterschiedlich viele Spalten oder verschiedene Datentypen haben.

Name
Dijkstra
Jansen
Knuth
Meier
Müller
Schmitz
Wirth

## Leere Datenfelder und mehrfach vorhandene Werte

Es kommt vor, dass in einer Tabellenzelle kein Wert steht. Man hat vielleicht vergessen, ihn einzutragen, oder er ist nicht bekannt. An dieser Stelle steht dann „NULL“.

Ebenso gibt es häufig Werte, die sich in einer Spalte wiederholen, insbesondere in Beziehungstabellen.

In manchen Fällen möchte man Zeilen, in denen NULL-Werte stehen, oder in denen mehrmals die gleichen Werte vorkommen, vom Ergebnis einer Abfrage aussortieren.

**Kurs**

<u>KursNr</u>	Titel	PersNr
K01	Java	D02
K02	HTML	D02
K03	Datenbank	D01
K04	Netzwerke	NULL

Mit `SELECT DISTINCT` werden im Ergebnis nur Zeilen mit **verschiedenen Werten** angezeigt.

Mit ... `WHERE [Spalte] IS NOT NULL` kann man Zeilen mit **NULL-Werten** aussortieren.

*Beispiel: Alle Dozenten, die (mindestens) einem Kurs zugeordnet sind*

```
SELECT DISTINCT PersNr FROM Kurs  
WHERE PersNr IS NOT NULL;
```

PersNr
D02
D01

Hinweis: NULL ist kein Wert wie die Zahl 0, daher greifen Vergleiche wie `=`, `>=` usw. nicht.