

**Regeln** sind kausale Beziehungen zwischen komplexen Bedingungen und einer Konklusion. Die Auswertung erfolgt nach der so genannten Horn Regel.

- Wenn der Auftraggeber bekannt und seine Bonität gesichert ist, dann liefere Ware gegen Rechnung.
- ▲ Wissensbasierte Sprachen sind ein Teilgebiet der **Künstlichen Intelligenz (KI)**.
- Zu den Vertretern der wissensbasierten Sprachen zählen: PROLOG (Programmiersprache zur Unterstützung der regelorientierten Programmierung), LISP, SCHEME (stellen spezielle Prozeduren zur Verarbeitung von „Listen“ zur Verfügung).

## 6.7 Datenbankmanagement auf Basis der relationalen Sprache SQL

**SQL (Structured Query Language)** ist eine Standardsprache für relationale Datenbanken.

- ▲ Eine Normierung von SQL erfolgte durch **ISO (International Standardization Organization)** und **ANSI (American National Standards Institute)**.

SQL bildet die Basis aller gängigen **Relationalen Datenbankmanagementsysteme (RDBMS)**. Die Verfügbarkeit von SQL ist entscheidendes Merkmal der „Güte“ eines RDBMS.

### a) SQL als Data Definition Language (DDL)

Unter einer **Data Definition Language (DDL)** (synonym: **Data Description Language**) wird eine Sprache zur Generierung des **intensionalen Relationenschemas** verstanden.

SQL als DDL beinhaltet drei grundlegende Befehle:

- **Anlegen einer Relation (create):**

```
CREATE TABLE <tablename> (<attributname_1> <datentyp_1>,
<attributname_2> <datentyp_2>, ...<attributname_n>
datentyp_n)
```

- Anlegen einer Relation Auftrag, die aus den Attributen Auftragsnummer, Kundennummer, Datum und Mitarbeiternummer des Kundenbetreuers besteht.

```
SQL> CREATE TABLE AUFTRAG ( auftrag_no number(4) not null,
kunden_no number(4), datum date,
mitarbeiter_no number(4));
```

- **Ändern des Schemas einer Relation (alter)**, Hinzufügen oder Ändern eines Attributes:

```
ALTER TABLE <tablename> ADD/MODIFY (<attributname_1>
<datentyp_1>, <attributname_2> <datentyp_2>, ...,
<attributname_n> <datentyp_n>)
```

- Hinzufügen des Attributes Position der Relation MITARBEITER

```
SQL> ALTER TABLE MITARBEITER ADD (position char (20)
CHECK position in('Techniker', 'Kundenbetreuer',
'Rechnungswesen'));
```

- **Löschen einer Relation (drop):**

```
DROP TABLE <tablename>
```

- Löschen der Relation MITARBEITER:

```
SQL> DROP TABLE MITARBEITER;
```

**Allgemeine Datentypen (datenbankabhängig, hier ORACLE):**

number(x)	ganzzahliger numerischer x-stelliger Wert
number(x,y)	numerischer Wert mit x – y Vorkomma- und y Nachkommastellen
char(x)	alphanumerische Zeichenkette mit x Zeichen
date	Datum

**Spezielle Zusätze für Datentypen zur Integritätssicherung:**

not null	Attribut muss für jeden Tupel einen Wert haben
unique	Attribut darf für verschiedene Tupel keine identischen Werte haben
primary key	not null und unique, sowie Primärschlüssel der Tabelle
references	sichert Integrität bei Fremdschlüsselbeziehungen
check	definiert Wertebereiche einer Domäne

## ■ Beispiele:

```
SQL> CREATE TABLE BANKLEITZAHL (blz number(8) primary key,
bankname char (20));
SQL> CREATE TABLE MITARBEITER (pers_nr number(5) primary key,
nachname char(20), vorname char(15),
geschlecht number(1), strasse char(20), plz number(5),
wohnort char(20), blz number(8) references BANKLEITZAHL,
abt_nr number(2), gehalt number(7)
check (gehalt between 2000 and 1000000));
```

**6.7.1 SQL als Data Manipulation Language (DML)**

Eine **Data Manipulation Language (DML)** dient der Veränderung der **extensionalen Datensicht** und beinhaltet drei grundlegende Befehle:

● **Einfügen von Tupeln (INSERT)**

```
INSERT INTO <tablename> VALUES (<wert_1>, <wert_2><, ..., <wert_n>)
```

## ■ Einfügen von Tupeln in die Relation MITARBEITER

```
SQL> INSERT INTO MITARBEITER values (79, 'Silcher', 'Angela',
2, 'Heiligenstr. 82', 60054, 'Frankfurt', 6, 4900, 'Kundenbetreuer');
```

● **Ändern von Tupeln (UPDATE)**

```
UPDATE <tablename> SET <attributname> = <arithm. Ausdruck>
WHERE <bedingung>
```

## ■ Aufgrund der guten Leistungen der Abteilung 3 (Debitorenbuchhaltung) erhalten die Mitarbeiter dieser Abteilung 5% mehr Lohn

```
SQL > UPDATE MITARBEITER SET gehalt = gehalt * 1.05
WHERE abt_nr=3;
```

● **Löschen von Tupeln (DELETE)**

```
SQL> DELETE FROM <tablename> WHERE <bedingung>
```

## ■ Der Mitarbeiter mit der Personalnummer 19 scheidet aus dem Unternehmen aus und wird aus der Relation Mitarbeiter entfernt:

```
SQL> DELETE FROM MITARBEITER WHERE pers_nr=19;
```

## 6.7.2 SQL als Query Language (QL)

Eine **Query Language (QL)** ist eine Abfragesprache zur Extraktion von Daten bzw. Informationen aus der Datenbank. Grundlage ist der **Select**-Befehl, wobei der einfachste Select-Befehl die vollständige Anzeige aller Tupel einer Relation ist:

```
SQL> SELECT * FROM <tablename>
```

### Grundfunktionen:

- **Projektion:** Auswahl einzelner Spalten (Attribute) aus den Tabellen.

Die Projektion generiert aus einer Relation B eine Relation A, indem eine Teilmenge der Attribute der Relation B in A eingehen.

```
SELECT <attributname_1>, <attributname_2>, ..., <attributname_n>
FROM <tablename>
```

- Auswahl der Attribute pers\_nr, vorname, nachname und gehalt aus der Relation MITARBEITER

```
SQL > SELECT pers_nr, vorname, nachname, gehalt
FROM MITARBEITER;
```

- ▲ Werden durch eine Projektion mehrere identische Tupel generiert, so sind die identischen Tupel nach der Relationenlehre zu entfernen. Diese Forderung wird von den meisten RDBMS jedoch nicht unterstützt.

```
SQL> SELECT position FROM MITARBEITER;
```

Falls ein RDBMS nicht automatisch Duplikate entfernt, ist, um wirklich nur die unterschiedlichen Berufe zu erhalten, folgender SQL-Befehl auszuführen:

```
SQL> SELECT DISTINCT(position) FROM MITARBEITER;
```

- **Selektion:** Auswahl einzelner Zeilen (Datentupel) aus den Tabellen nach gewissen Kriterien. Die Selektion generiert aus einer Relation B eine Relation A durch Bildung einer Teilmenge der Tupel, die einer bestimmten Bedingung genügen:

```
SELECT * FROM <tablename> WHERE <bedingung>
```

- Auswahl aller Mitarbeiter der Abteilung 3

```
SQL > SELECT * FROM MITARBEITER WHERE abt_nr=3;
```

### Weitere Selektionsoperatoren:

- Sortieren
- Mathematische Funktionsoperatoren
- Gruppierung
- Logische Operatoren: AND, OR, NOT
- Arithmetische Operatoren: =, >, <, >=, <=, !=

- Beispiel für die Anwendung logischer und arithmetischer Operatoren:

- Auswahl aller Mitarbeiter, die in Abteilung 3 arbeiten und weiblich (geschlecht=2) sind

```
SQL> SELECT * FROM MITARBEITER WHERE abt_nr=3 AND
geschlecht=2;
```

- Auswahl aller Mitarbeiter, die entweder in Abteilung 3 oder Abteilung 4 arbeiten

```
SQL> SELECT * FROM MITARBEITER WHERE abt_nr=3
OR abt_nr=4;
```

- Auswahl aller Mitarbeiter, die ein Gehalt zwischen 4000 und 5000 beziehen und weiblich sind, oder die ein Gehalt zwischen 5000 und 6000 beziehen und männlich sind

```
SQL> SELECT * FROM MITARBEITER WHERE (gehalt>4000 AND
gehalt<5000 AND geschlecht=2) OR (gehalt between 5000
and 6000 AND geschlecht=1);
```

- **Join:** Der Join-Operator verbindet zwei oder mehr Relationen über Attribute zu einer neuen Relation. Ein Join ist eine **Abfrage**, die Daten aus mehr als einer Relation zusammensucht und in einer Relation ausgibt:

```
SELECT <attributname_1>, <attributname_2>, ..., <attributname_n>
FROM <tablename_1>, <tablename_2>, ..., <tablename_m>
WHERE <Join-Bedingung>
```

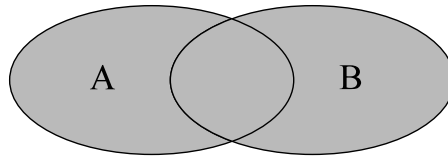
- ▲ Existieren in verschiedenen zu verbindenden Relationen Attribute mit identischem Namen, ist diesen Attributen der Name der zugehörigen Relation (gefolgt von einem Punkt) voranzustellen.

- Zu den Attributen pers\_nr, nachname, vorname, abteilungs\_no, gehalt, konto\_nr und bankleitzahl sind zusätzlich die Attribute bankname und abteilungsname zu selektieren.

```
SQL> SELECT m.pers_nr, m.nachname, m.vorname, m.abt_nr,
a.bezeichnung, m.gehalt, m.konto_nr, m.blz, b.bankname
FROM MITARBEITER m, ABTEILUNG a, BANK b
WHERE m.abt_nr=a.abt_nr AND blz=b.blz;
```

- ▲ Bei den Bezeichnern a,b und m handelt es sich um so genannte Alias-Namen, unter denen in diesem Beispiel die Relationen Mitarbeiter, Abteilung und Bank angesprochen werden.

- **Vereinigung:** Aus zwei oder mehr Relationen wird eine Relation gebildet, die die **Vereinigungsmenge** dieser Relationen bildet.



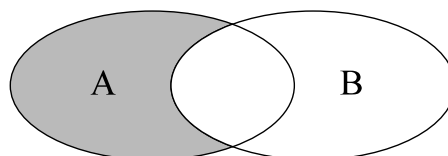
```
SELECT <liste von attributwerten> FROM <tablename_1> UNION
SELECT <liste von attributwerten> FROM <tablename_2>
```

- ▲ Die Operation Union ist nur für Projektionen von Relationen durchführbar, deren Attribute identische Domänen aufweisen.

- Neben der uns bereits bekannten Relation Lieferant existiert auch eine Relation Kunde. Zur Einführung eines EDI-Systems werden von allen Geschäftspartnern (Kunden und Lieferanten) die Daten Name, Strasse, PLZ und Ort benötigt.

```
SQL > SELECT lieferantename, strasse, plz, ort
FROM LIEFERANT UNION SELECT kundenname, strasse, plz, ort
FROM KUNDE;
```

- **Differenz:** **Differenzmenge** aus einer Menge von Relationen. Diese Verknüpfung zweier Relationen wird in einer neuen Relation dargestellt, die alle Zeilen (Tupel) der einen Relation enthält, ohne diejenigen, die auch in der anderen vorhanden sind.



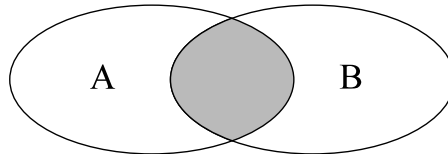
```
SELECT <liste von attributwerten> FROM <tablename_1> MINUS
SELECT <liste von attributwerten> FROM <tablename_2>
```

▲ Operation Differenz ist nur für Projektionen von Relationen durchführbar, deren Attribute identische Domänen aufweisen.

■ Selektion aller Lieferanten, die nicht auch Kunden sind.

```
SQL> SELECT lieferantename, strasse, plz, ort FROM
LIEFERANT MINUS SELECT kundename, strasse, plz, ort
FROM KUNDE;
```

- **Schnittmenge:** Verknüpfung zweier Relationen in einer neuen Relation, die alle Zeilen (Tupel), die sowohl in der einen Relation wie auch in der anderen vorhanden sind, enthält.



```
SELECT <liste von attributwerten> FROM <tablename_1> INTERSECT
SELECT <liste von attributwerten> FROM <tablename_2>
```

▲ Die Operation Schnittmenge ist nur für Projektionen von Relationen durchführbar, deren Attribute identische Domänen aufweisen.

■ Die Finanzbuchhaltung benötigt die Daten aller Lieferanten, die auch Kunden sind, um bspw. eine Verrechnung von Soll- und Habenpositionen vorzunehmen:

```
SQL> SELECT lieferantename, strasse, plz, ort
FROM LIEFERANT INTERSECT SELECT kundename, strasse, plz,
ort FROM KUNDE;
```

### Spezielle Operatoren:

- Sortieren von Datensätzen

Am Ende des SELECT-Befehls steht der Zusatz ORDER BY <column> [DESC]

■ Absteigendes Sortieren der Relation Kunde nach dem Namen:

```
SQL > SELECT * FROM KUNDE ORDER BY name DESC;
```

Der Zusatz DESC bewirkt eine Sortierung in absteigender Reihenfolge.

- Mathematische Operatoren

– Zählen: SELECT COUNT(\*) FROM <tablename> WHERE <condition>

■ Anzahl aller Mitarbeiter in der Abteilung mit der Nummer 4:

```
SQL> SELECT COUNT(*) FROM MITARBEITER WHERE abt_nr=4;
```

– Addieren: SELECT SUM(<column>) FROM <tablename> WHERE...

■ Gehaltssumme aller Mitarbeiter:

```
SQL> SELECT SUM(gehalt) FROM MITARBEITER;
```

– Minimum: SELECT MIN(<column>) FROM <tablename> WHERE...

■ Niedrigstes Gehalt aller Mitarbeiter:

```
SQL> SELECT MIN(gehalt) FROM MITARBEITER;
```

– Maximum: SELECT MAX(<column>) FROM <tablename> WHERE...

■ Höchstes Gehalt aller Mitarbeiter:

```
SQL> SELECT MAX(gehalt) FROM MITARBEITER;
```

– Mittelwert: `SELECT AVG(<column>) FROM <tablename> WHERE...`

■ Durchschnittsgehalt aller Mitarbeiter:

```
SQL> SELECT AVG(gehalt) FROM MITARBEITER;
```

– Standardabweichung: `SELECT STDDEV(<column>) FROM <tablename> WHERE...`

■ Standardabweichung der Gehälter aller Mitarbeiter:

```
SQL> SELECT STDDEV(gehalt) FROM MITARBEITER;
```

### Gruppierung von Datensätzen:

**Group by:** Datensätze lassen sich mithilfe der Anweisung `GROUP BY` nach einem zu spezifizierendem Kriterium gruppieren.

```
SELECT <attributname_1, attributname_2...> FROM <tablename>
GROUP BY <column>;
```

■ Beispielsweise kann eine Aufstellung der Anzahl gestellter Rechnungen pro Datumstag vorgenommen werden.

```
SQL > SELECT datum, COUNT(*) FROM RECHNUNG GROUP BY datum;
```

### Spezielle Datenbankoperatoren:

#### Having

■ Ermittlung der durchschnittlichen Gehälter der Berufsgruppen. Aus statistischen Gründen sollen nur Gruppen, die mindestens vier Tupel beinhalten, ausgewertet werden.

```
SQL> SELECT position, AVG(gehalt) FROM MITARBEITER GROUP BY
position HAVING COUNT(*) > 4;
```

#### In, geschachtelte Select-Befehle:

■ Ermittlung der Lieferantenummer und der Namen aller Lieferanten, die das Teil No. 3 bereits geliefert haben.

```
SQL> SELECT lieferanten_no, name FROM LIEFERANT
WHERE lieferanten_no IN (SELECT lieferanten_no FROM BESTELLUNG
WHERE teil_no=3);
```

▲ Die Verschachtelung über die In-Klausel kann beliebig tief erfolgen.

#### Exists:

■ Ermittle die Namen der Lieferanten, die alle Teile liefern.

```
SQL> SELECT lieferantenname FROM LIEFERANT l WHERE NOT EXISTS
(SELECT * FROM TEILE t WHERE NOT EXISTS (SELECT * FROM
BESTELLUNG b WHERE lieferanten_no=l.lieferanten_no AND
teile_no=t.teile_no));
```

### Anlegen einer Relation aus bereits in der Datenbank existierenden Daten:

Das `SQL`-Kommando `CREATE` kann auch zum Anlegen einer Relation aus bereits in der Datenbank existierenden Daten verwendet werden.

```
CREATE TABLE <table> AS SELECT ...
```

▲ Der `Select`-Teil des `Create`-Kommandos kann dabei alle Basisoperationen wie Projektion, Selektion, Join, Union usw. beinhalten.