Fachbereich Automatisierung und Informatik	HOCHSCHULE harz Hochschule für angewandte Wissenschaften (FH) Wernigerode

„SQL-Nachschlagewerk“

Grundlagen

Firebird / Interbase / Oracle

Dipl. Inf., Dipl.-Ing. (FH) Michael Wilhelm
Friedrichstraße 57 - 59
38855 Wernigerode

Raum: **2.202**
Tel.: **03943/659-338**
Fax: **03943/659-399**
Email: **mwilhelm@hs-harz.de**

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung in SQL

Dieses Script bietet einen Einstieg in die Benutzung von Datenbanken mittels SQL-Befehlen. Als Grundlage wird die Firebird / Interbase Datenbank verwendet. Die Installation ist sehr einfach und mit IBOConsole steht ein guter DBS-Client zur Verfügung. Die Datenbank ist frei erhältlich und sowohl als lokale Datenbank als auch als Server-Datenbank einsetzbar.

Eine relationale Datenbank kann über SQL-Anweisungen gelesen, beschrieben und verändert werden. Dieses Script führt in die SQL-Sprache ein. Das erste Kapitel zeigt den Aufbau eines SQL-Statements, bschreibt die Unterschiede zwischen einer Projektion, einer Selektion und einem Join. Im zweiten, dritten und vierten Kapitel werden SQL-Befehle vorgestellt und die Kenntnisse mit Beispielen vertieft. Jedes dieser Kapitel hat einen zusätzlichen Übungsteil, deren Lösungen im Kapitel 10 erläutert werden.

Die letzten Kapitel beschreiben die verwendeten Datenbanktabellen. Im Kapitel 8.1 werden die wichtigsten Tabellen angezeigt und deren Attribute beschrieben. Kapitel 8.2 beinhaltet die Anweisungen für die Erzeugung der Tabellen nebst den Eingabebedingungen.

1.1 Aufbau der SQL-Sprache

Datenbanken werden über über standardisierte Sprache angesprochen. Diese Sprache ist leicht zu erlernen und ist weniger kompliziert als eine Programmiersprache. SQL ist in folgende Teile aufgebaut:

- Datenabfrage
  - SELECT
- Data Manipulation Language - DML
  - INSERT — Einfügen
  - UPDATE — Ändern
  - DELETE — Löschen
  - MERGE — Zusammenfügen
- Data Definition Language - DDL
  - CREATE — Erzeugen einer Tabelle
  - ALTER — Ändern einer Tabelle
  - DROP — Löschen einer Tabelle
  - RENAME — Umbenennen einer Tabelle
  - TRUNCATE — Abschneiden einer Tabelle
- Transaktionssteuerung
  - COMMIT — Änderungen speichern
  - ROLLBACK — Änderungen rücknehmen
  - SAVEPOINT — Sicherheitspunkt
- Data Control Language - DCL
  - GRANT — Rechte erlauben
  - REVOKE — Rechter verbieten

1.2 Fähigkeiten der SQL-Anweisung SELECT

Eine Relationale Datenbank kann man sich als Exceltabelle vorstellen.

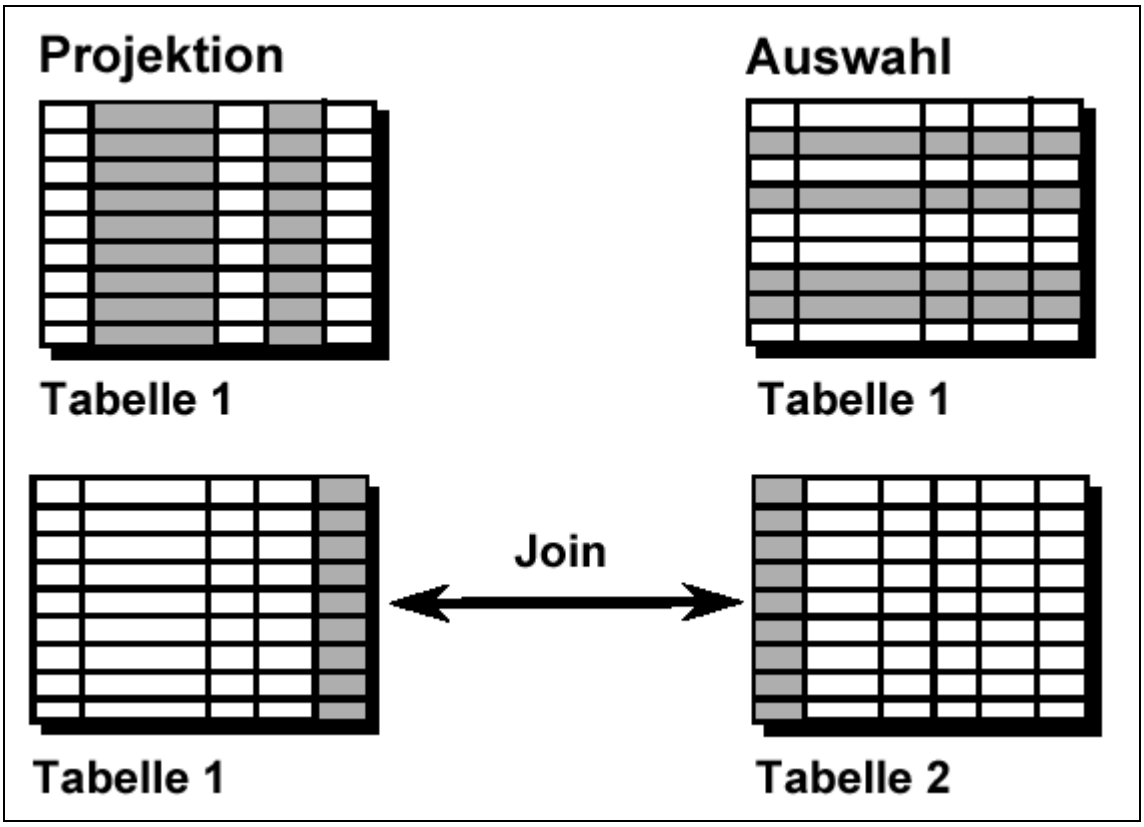

Abbildung 1 Fähigkeiten der SQL-Anweisung SELECT

Eine SELECT-Anweisung ruft Informationen aus der Datenbank ab. Folgende Aktionen können ausgeführt werden:

- Projektion
  Mit dieser Fähigkeit können Sie festlegen, welche Spalten einer oder mehrerer Tabellen Ihre Abfrage zurückgibt. Sie können beliebig viele Spalten angeben.
- Auswahl
  Mit dieser Fähigkeit können Sie festlegen, welche Zeilen einer Tabellen Ihre Abfrage zurückgibt. Sie können verschiedene Kriterien angeben, um die angezeigten Zeilen zu beschränken (Where-Klausel).
- Join
  Mit dieser Fähigkeit können Sie Daten mehrerer Tabellen zu einer gesamten Tabellen vereinigen.

1.3 SELECT-Anweisung

1. Beispiel:

```
SELECT * FROM emp;
```

Erläuterung:

Anzeige aller Spalten, Felder, der Tabelle EMP

Ergebnis (Auszug):

EMP_NO	FIRST_NAME	LAST_NAME	PHONE_EXT	HIRE_DATE	DEPT_NO	JOB_CODE	JOB_GRADE	JOB_COUNTRY	SALARY
2	Robert	Nelson	250	28.12.1988	600	VP	2	USA	105900
4	Bruce	Young	233	28.12.1988	621	Eng	2	USA	97500
5	Kim	Lambert	22	06.02.1989	130	Eng	2	USA	102750
8	Leslie	Johnson	410	05.04.1989	180	Mktg	3	USA	64635
9	Phil	Forest	229	17.04.1989	622	Mngr	3	USA	75060
11	K. J.	Weston	34	17.01.1990	130	SRep	4	USA	86292.93
12	Terri	Lee	256	01.05.1990	0	Admin	4	USA	53793
14	Stewart	Hall	227	04.06.1990	900	Finan	3	USA	69482.62
15	Katherine	Young	231	14.06.1990	623	Mngr	3	USA	67241.25
20	Chris	Papadopoulos	887	01.01.1990	671	Mngr	3	USA	89655
24	Pete	Fisher	888	12.09.1990	671	Eng	3	USA	81810.18
28	Ann	Bennet	5	01.02.1991	120	Admin	5	England	22935
29	Roger	De Souza	288	18.02.1991	623	Eng	3	USA	69482.62
34	Janet	Baldwin	2	21.03.1991	110	Sales	3	USA	61637.81

2. Beispiel:

SELECT EMP_NO, FIRST_NAME, LAST_NAME, SALARY FROM emp;

Erläuterung:

Anzeige aller Spalten, Felder der Tabelle EMP

Ergebnis (Auszug):

EMP_NO	**FIRST_NAME**	**LAST_NAME**	**SALARY**
2	Robert	Nelson	105900
4	Bruce	Young	97500
5	Kim	Lambert	102750
8	Leslie	Johnson	64635
9	Phil	Forest	75060
11	K. J.	Weston	86292.93
12	Terri	Lee	53793
14	Stewart	Hall	69482.62
15	Katherine	Young	67241.25
20	Chris	Papadopoulos	89655
24	Pete	Fisher	81810.18
28	Ann	Bennet	22935
29	Roger	De Souza	69482.62
34	Janet	Baldwin	61637.81

1.3.1 Allgemeine Syntax

SELECT [DISTINCT| DISTINCTROW | TOP [n PERCENT] |ALL]{*, column [alias], expr }

```
FROM      table, …
[WHERE    condition(s)]
[GROUP BY      expr [, expr] …]
[HAVING   condition(s)]
[ORDER BY      {column, expr, alias} [ASC|DESC]];
```

Abbildung 10.: Syntax der SELECT-Anweisung

Syntax:

SELECT	Liste mit einer oder mehreren Spalten
DISTINCT	Schlüsselwort, um vorkommende Zeilen auszuschließen
DISTINCTROW	Schlüsselwort, um vorkommende Zeilen auszuschließen
TOP	
*	Auswahl aller Spalten aller in der FROM-Klausel aufgeführten Tabellen, Views oder Snapshots
column	Auswahl der benannten Spalte
alias	Ausgewählte Spalten erhalten andere Überschriften
expr	Zeichenkette oder errechneter Ausdruck
FROM table	Schlüsselwort zur Angabe der Tabelle, View, oder des Snapshot mit den Spalten
WHERE	Klausel zur Einschränkung der auszuwählenden Zeilen entsprechend einer Bedingung
GROUP BY	zur Gruppierung ausgewählter Zeilen und zur Rückgabe einer zusammenfassenden Zeile
HAVING	gibt an, welche durch die GROUP-BY-Klausel definierten Zeilengruppen von der Abfrage zurückgegeben werden
ORDER BY	Reihenfolge zur Anzeige der abgerufenen Zeilen.
ASC:	aufsteigend (Standard),
DESC:	absteigend

Folgende einfache Regeln und Richtlinien müssen einhalten werden, damit gültige Anweisungen konstruiert werden, die leicht lesbar und einfach zu editieren sind:

- SQL-Anweisungen unterscheiden keine Groß- und Kleinbuchstaben.
- Klauseln stehen gewöhnlich zur leichteren Lesbarkeit und Bearbeitung in separaten Zeilen.
- Schlüsselwörter werden gewöhnlich in Großbuchstaben angegeben; alle anderen Wörter, wie z.B. Tabellennamen und Spalten, werden in Kleinbuchstaben geschrieben.

In der “Where-Klausel” wird die Bedingung mittels folgender Operatoren bestimmt:

```
AND     OR      NOT
<       <=      >=    >
=       <>
IN
BETWEEN         AND
```

1.4 Beispiele

Dieses Kapitel zeigt Beispiele mit einfachen Abfragen. Als erstes wird die Projektion, danach die Selection behandelt. Der Join wird in einem späteren Kapitel behandelt (siehe Kapitel 5, Seite 73).

1.4.1 Beispiele einer Projektion:

- Gesucht die Liste aller Mitarbeiter der Tabelle „employee“

```
SELECT *
FROM employee
```

Ergebnis siehe Tabelle Seite 96.

- Gesucht die Liste der Namen aller Mitarbeiter in der Tabelle „employee“ auf.

```
SELECT first_name, last_name
FROM employee
```

Ergebnis (Auszug):

FIRST_NAME	**LAST_NAME**
Robert	Nelson
Bruce	Young
Kim	Lambert
Leslie	Johnson
Phil	Forest
K. J.	Weston
Terri	Lee
Stewart	Hall
Katherine	Young
Chris	Papadopoulos
Pete	Fisher
Ann	Bennet
Roger	De Souza
Janet	Baldwin
Roger	Reeves
Willie	Stansbury
Leslie	Phong
Ashok	Ramanathan
Walter	Steadman
Carol	Nordstrom
Luke	Leung
Sue Anne	O'Brien
Jennifer M.	Burbank
Claudia	Sutherland
Dana	Bishop
Mary S.	MacDonald

- Gesucht die Liste der Namen und Mitarbeiternummern aller Mitarbeiter der Tabelle „employee“ auf. Sortiert wird nach der Nummer.

```
SELECT first_name, last_name, emp_no
FROM employee
ORDER BY emp_no
```

Ergebnis:

FIRST_NAME	LAST_NAME	EMP_NO
Robert	Nelson	2
Bruce	Young	4
Kim	Lambert	5
Leslie	Johnson	8
Phil	Forest	9
K. J.	Weston	11
Terri	Lee	12
Stewart	Hall	14
Katherine	Young	15
Chris	Papadopoulos	20
Pete	Fisher	24
Ann	Bennet	28
Roger	De Souza	29
Janet	Baldwin	34
Roger	Reeves	36
Willie	Stansbury	37
Leslie	Phong	44
Ashok	Ramanathan	45
Walter	Steadman	46
Carol	Nordstrom	52
Luke	Leung	61
Sue Anne	O'Brien	65
Jennifer M.	Burbank	71
Claudia	Sutherland	72
Dana	Bishop	83
Mary S.	MacDonald	85
Randy	Williams	94
Oliver H.	Bender	105
Kevin	Cook	107
Kelly	Brown	109
Yuki	Ichida	110
Mary	Page	113
Bill	Parker	114
Takashi	Yamamoto	118
Roberto	Ferrari	121
Michael	Yanowski	127
Jacques	Glon	134
Scott	Johnson	136
T.J.	Green	138
Pierre	Osborne	141
John	Montgomery	144
Mark	Guckenheimer	145

- Gesucht die Liste der Mitarbeiternummern und der Gehälter aller Mitarbeiter der Tabelle „emp“ auf. Es soll sowohl das normale Gehalt, als auch ein um 10% erhöhte Gehalt angezeigt werden.

```
SELECT emp_no, Salary AS Gehalt, (Salary*1.1) AS Gehalt_10Prozent
FROM emp;
```

Ergebnis (Auszug):

emp_no	Gehalt	Gehalt_10Prozent
2	105900	116490
4	97500	107250
5	102750	113025
8	64635	71098.5
9	75060	82566
11	86292.93	94922.23
12	53793	59172.3
14	69482.62	76430.88
15	67241.25	73965.37
20	89655	98620.5

Diese Anweisung hat zwei Neuerungen:

1) Alias-Namen

Mit der Anweisung AS können Spalten umbenannt werden. Spalten bzw. Attribute werden häufig mit verkürzten Namen gespeichert. Mit dem Alias „AS“ kann die Bezeichnung in eine beliebige Sprache umgewandelt werden. Hinter „AS“ wird der neue Name eingetragen. Dieser darf keine Leerstellen haben.

1) Mathematischer Ausdruck

Die Anweisung „(Salary*1.1)” berechnet einen neuen Wert aus den feldern der Datenbank. Möglich wäre auch:

```
SELECT (13+4)
FROM emp;
```

Diese Anweisung liefert den Wert 17. leider mehrfach. Mit der Anweisung DISTINCT werden doppelte Zeilen aus der Ergebnistabelle entfernt. Damit liefert die nächste Anweisung genau einmal den Wert 17.

```
SELECT DISTINCT (13+4)
FROM emp;
```

1.4.2 Beispiele einer Selektion

Dieses Beispiel zeigt Abfragebeispiele. Hier werden einzelne Zeilen in der Tabelle ausgewählt. Dazu benutzt man die WHERE-Klausel.

- Gesucht die Liste aller Attribute des Mitarbeiters mit der Nummer 15 der Tabelle „employee“.

```
SELECT *
FROM employee
WHERE EMP_NO = 15
```

Ergebnis (Auszug der Attribute):

EMP#	FIRSTNAME	LAST_NAME	PHONE	HIREDATE	DEPT#	FULL_NAME

15	Katherine	Young	231	14.06.1990	623	Young, Katherine

- Gesucht die Liste der Namen aller Mitarbeiter in der Abteilung 623.

```
SELECT first_name, last_name
FROM employee
WHERE DEPT_NO = 623
```

Ergebnis:

FIRST_NAME	**LAST_NAME**
Katherine	Young
Roger	De Souza
Leslie	Phong
Bill	Parker
Scott	Johnson

Gesucht die Liste der Namen und Mitarbeiternummern aller Mitarbeiter in der Tabelle „employee“, die ein Gehalt größer 100000,00€ haben. Sortiert wird nach dem Gehalt.

```
SELECT full_name, emp_no, salary
FROM employee
WHERE salary>100000
ORDER BY salary
```

Ergebnis:

FULL_NAME	**EMP_NO**	**SALARY**
Sutherland, Claudia	72	100914
Lambert, Kim	5	102750
Nelson, Robert	2	105900
Osborne, Pierre	141	110000
Cook, Kevin	107	111262,5
MacDonald, Mary S.	85	111262,5
Steadman, Walter	46	116100
Bender, Oliver H.	105	212850
Glon, Jacques	134	390500
Ichida, Yuki	110	6000000
Yamamoto, Takashi	118	7480000
Ferrari, Roberto	121	99000000

Die Sortierreihenfolge kann auch mit folgender Anweisung erfolgen:

```
SELECT full_name, emp_no, salary
FROM employee
WHERE salary>100000
ORDER BY 3
```

Mit „Order by Nummer“ erleichtert man sich die Angabe des Attributes. Bei einer Änderung in der Liste, muss diese Nummer aber mitgeändert werden!!!

- Gesucht die Liste aller Abteilungsnummer aller Mitarbeiter in der Tabelle „employee“.

```
SELECT dept_no
FROM employee
ORDER BY dept_no
```

Ergebnis:

DEPT_NO	DEPT_NO	DEPT_NO
000	000	100
100	110	110
115	115	120
120	120	121
123	125	130
130	140	180
180	600	600
621	621	621
621	622	622
622	623	623
623	623	623
670	670	671
671	671	672
672	900	900

Die Daten sind um Platz zu sparen, in drei Spalten dargestellt. Das Beispiel zeigt, dass doppelte Werte als Ergebnis berechnet werden. Mit der Anweisung DISTINCT werden doppelte Zeilen aus der Ergebnistabelle entfernt.

```
SELECT DISTINCT dept_no
FROM employee
ORDER BY dept_no
```

Ergebnis:

DEPT_NO
000
100
110
115
120
121
123
125
130
140
180
600
621
622
623
670

671
672
900

1.4.3 Beispiele eines Joins

Ein Join ist ein kartesisches Produkt zweier Tabellen (A und B). Jeder Datensatz der Tabelle A wird mit jedem Datensatz der Tabelle B verknüpft. Damit ist das Ergebnis unter Umständen eine sehr Tabelle. Mit Hilfe der WHERE-Klausel kann die Datenmenge eingeschränkt werden.

SELECT *
FROM employee, department

Listet die Kombination aller Zeilen der Tabellen „employee“ und „department“ auf. Die Tabelle „employee“ hat 42 Einträge. Die Tabelle „Department“ 21 Abteilungen. Als Ergebnis erhält man eine Tabelle mit 42×21=882 Einträgen. Alle Attribute der Tabelle „employee“ und der Tabelle „department“ sind als Attribute in der Tabelle vorhanden.

Ergebnis (Auszug):

EMP	EMP	EMP	EMP	DEPT	DEPT	DEPT	DEPT
EMP#	DEPT#	JOB CODE	FULL NAME	DEPT_NO_1	DEPARTMENT	LOCATION	PHONE NO
2	600	VP	Nelson, Robert	100	Sales and Marketing	San Francisco	(415) 555-1234
2	600	VP	Nelson, Robert	600	Engineering	Monterey	(408) 555-1234
2	600	VP	Nelson, Robert	900	Finance	Monterey	(408) 555-1234
2	600	VP	Nelson, Robert	130	Field Office: East Coast	Boston	(617) 555-1234
2	600	VP	Nelson, Robert	140	Field Office: Canada	Toronto	(416) 677-1000
2	600	VP	Nelson, Robert	670	Consumer Electronics Div.	Burlington, VT	(802) 555-1234
2	600	VP	Nelson, Robert	671	Research and Development	Burlington, VT	(802) 555-1234
2	600	VP	Nelson, Robert	622	Quality Assurance	Monterey	(408) 555-1234
2	600	VP	Nelson, Robert	672	Customer Services	Burlington, VT	(802) 555-1234
2	600	VP	Nelson, Robert	623	Customer Support	Monterey	(408) 555-1234
2	600	VP	Nelson, Robert	620	Software Products Div.	Monterey	(408) 555-1234
2	600	VP	Nelson, Robert	180	Marketing	San Francisco	(415) 555-1234
2	600	VP	Nelson, Robert	621	Software Development	Monterey	(408) 555-1234
2	600	VP	Nelson, Robert	000	Corporate Headquarters	Monterey	(408) 555-1234
2	600	VP	Nelson, Robert	110	Pacific Rim Headquarters	Kuaui	(808) 555-1234

In der ersten Zeile wurde noch eine Zusatzinformation gespeichert. Sie zeigt an, aus welcher Tabelle die Information stammt. EMP bedeutet, dass dieses Attribut aus der Tabelle „Employee“ stammt. Bedeutsam ist, dass die „Dept_no“ zweimal als Attribut auftauscht. SQL fügt dazu in der Überschrift eine „1“ als Unterscheidung hinzu.

Von den 821 Einträgen interessieren uns aber nur die Einträge, in der die beiden Abteilungsnummern identisch sind. Alle anderen Zeilen können ignoriert werden. Diese Forderung kann nun mit Hilfe der WHERE-Klausel eingefügt werden.

```
SELECT *
FROM employee, department
WHERE employee.dept_no = department.dept_no
```

Das Ergebnis liefert nun nur 42 Zeilen!

Ergebnis (Auszug):

EMP	EMP	EMP	EMP	DEPT	DEPT	DEPT
EMP#	DEPT_NO	SALARY	FULL_NAME	DEPT_NO_1	DEPARTMENT	HEAD_DEPT
12	000	53793	Lee, Terri	000	Corporate Headquarters	
105	000	212850	Bender, Oliver H.	000	Corporate Headquarters	
85	100	111262,5	MacDonald, Mary S.	100	Sales and Marketing	000
127	100	44000	Yanowski, Michael	100	Sales and Marketing	000
2	600	105900	Nelson, Robert	600	Engineering	000
109	600	27000	Brown, Kelly	600	Engineering	000
14	900	69482,625	Hall, Stewart	900	Finance	000
46	900	116100	Steadman, Walter	900	Finance	000
8	180	64635	Johnson, Leslie	180	Marketing	100
52	180	42742,5	Nordstrom, Carol	180	Marketing	100
4	621	97500	Young, Bruce	621	Software Development	620
45	621	80689,5	Ramanathan, Ashok	621	Software Development	620
83	621	62550	Bishop, Dana	621	Software Development	620
138	621	36000	Green, T.J.	621	Software Development	620
9	622	75060	Forest, Phil	622	Quality Assurance	620
71	622	53167,5	Burbank, Jennifer M.	622	Quality Assurance	620
145	622	32000	Guckenheimer, Mark	622	Quality Assurance	620
15	623	67241,25	Young, Katherine	623	Customer Support	620
29	623	69482,625	De Souza, Roger	623	Customer Support	620
44	623	56034,375	Phong, Leslie	623	Customer Support	620
114	623	35000	Parker, Bill	623	Customer Support	620
136	623	60000	Johnson, Scott	623	Customer Support	620
65	670	31275	O'Brien, Sue Anne	670	Consumer Electronics Div.	600
107	670	111262,5	Cook, Kevin	670	Consumer	600

					Electronics Div.	
20	671	89655	Papadopoulos, Chris	671	Research and Development	670
24	671	81810,1875	Fisher, Pete	671	Research and Development	670
113	671	48000	Page, Mary	671	Research and Development	670
94	672	56295	Williams, Randy	672	Customer Services	670
144	672	35000	Montgomery, John	672	Customer Services	670
5	130	102750	Lambert, Kim	130	Field Office: East Coast	100
11	130	86292,9375	Weston, K. J.	130	Field Office: East Coast	100
72	140	100914	Sutherland, Claudia	140	Field Office: Canada	100
34	110	61637,8125	Baldwin, Janet	110	Pacific Rim Headquarters	100
61	110	68805	Leung, Luke	110	Pacific Rim Headquarters	100
110	115	6000000	Ichida, Yuki	115	Field Office: Japan	110
118	115	7480000	Yamamoto, Takashi	115	Field Office: Japan	110
28	120	22935	Bennet, Ann	120	European Headquarters	100
36	120	33620,625	Reeves, Roger	120	European Headquarters	100
37	120	39224,0625	Stansbury, Willie	120	European Headquarters	100
141	121	110000	Osborne, Pierre	121	Field Office: Switzerland	120
134	123	390500	Glon, Jacques	123	Field Office: France	120
121	125	99000000	Ferrari, Roberto	125	Field Office: Italy	120

Die erste Zeile zeigt wieder die Quelle der Tabellen. Die Attribute „Dept_no“ und „Dept_no_1“ sind in jeder Zeile identisch!

1.5 Übungen

Dieses Kapitel bietet Übungen, die auf die vorherigen Aufgaben aufbauen. Im Kapitel 10.1, Seite 113 sind die Lösungen aufgelistet und kommentiert.

Vorgehensweise für die Lösungen:

1) Bestimmen der Tabelle(n)

2) Bestimmen der Attribute

3) Test der kompletten SELECT

```
SELECT full_name, dept_no
FROM employee
```

4) Einbau der WHERE-Bedingung (optional)

```
SELECT full_name, dept_no
FROM employee
WHERE emp_no < 600
```

5) Einbau der Sortierreihenfolge (optional)

```
SELECT full_name, dept_no
FROM employee
WHERE emp_no < 600
ORDER BY emp_no
```

1.5.1 Anzeige aller eindeutigen Abteilungsnummern

Geben Sie eine Liste aller eindeutigen Abteilungsnummern aus. Ausgabe absteigend sortiert.

Ergebnis:

JOB_CODE
VP
Sales
SRep
PRel
Mngr
Mktg
Finan
Eng
Doc
Dir
CFO
CEO
Admin

1.5.2 Anzeige aller Mitarbeiter in den USA

Geben Sie eine Liste aller Mitarbeiter in den USA aus. Die Ausgabe soll absteigend nach dem Gehalt sortiert werden.

Ergebnis:

FULL_NAME	SALARY
Bender, Oliver H.	212850
Steadman, Walter	116100
Cook, Kevin	111262,5
MacDonald, Mary S.	111262,5
Nelson, Robert	105900
Lambert, Kim	102750
Young, Bruce	97500
Papadopoulos, Chris	89655
Weston, K. J.	86292,9375

Fisher, Pete	81810,1875
Ramanathan, Ashok	80689,5
Forest, Phil	75060
De Souza, Roger	69482,625
Hall, Stewart	69482,625
Leung, Luke	68805
Young, Katherine	67241,25
Johnson, Leslie	64635
Bishop, Dana	62550
Baldwin, Janet	61637,8125
Johnson, Scott	60000
Williams, Randy	56295
Phong, Leslie	56034,375
Lee, Terri	53793
Burbank, Jennifer M.	53167,5
Page, Mary	48000
Yanowski, Michael	44000
Nordstrom, Carol	42742,5
Green, T.J.	36000
Parker, Bill	35000
Montgomery, John	35000
Guckenheimer, Mark	32000
O'Brien, Sue Anne	31275
Brown, Kelly	27000

1.5.3 Nachnamen und Gehälter

Zeigen die Nachnamen und Gehälter aller Angestellten an. Sortieren Sie das Ergebnis nach aufsteigenden Abteilungsnummer und dann absteigend nach dem Gehalt.

Ergebnis:

LAST_NAME	**SALARY**
Nelson	105900
Young	97500
Lambert	102750
Johnson	64635
Forest	75060
Weston	86292,9375
Lee	53793
Hall	69482,625
Young	67241,25
Papadopoulos	89655
Fisher	81810,1875
Bennet	22935
De Souza	69482,625
Baldwin	61637,8125
Reeves	33620,625
Stansbury	39224,0625
Phong	56034,375
Ramanathan	80689,5
Steadman	116100

Nordstrom	42742,5
Leung	68805
O'Brien	31275
Burbank	53167,5
Sutherland	100914
Bishop	62550
MacDonald	111262,5
Williams	56295
Bender	212850

2 Erweiterte SQL-Befehle

2.1 Alias-Namen

Die Attributnamen sind häufig Abkürzungen von allgemeinen Bezeichnungen. Möchte man diese Namen ändern – z. B. in eine andere Sprache – so kann man einen Alias-Namen angeben. Dieser darf keine Leerzeichen enthalten und muss mit einem Buchstaben beginnen. Man schreibt den Alias direkt hinter dem Attribut in der Select-Anweisung.

Beispiel:

```
SELECT full_name  Name, emp_no  MitarbeiterNr
FROM employee
```

Ergebnis (Ausschnitt):

NAME	MITARBEITERNR
Nelson, Robert	2
Young, Bruce	4
Lambert, Kim	5
Johnson, Leslie	8
Forest, Phil	9
Weston, K. J.	11
Lee, Terri	12
Hall, Stewart	14
Young, Katherine	15
Papadopoulos, Chris	20
Fisher, Pete	24
Bennet, Ann	28
De Souza, Roger	29
Baldwin, Janet	34
Reeves, Roger	36

Substantive eines Alias kann man mit dem Zeichen „_“ verbinden.

Beispiele: Vor_Nachname
PLZ_Ort
Monatliches_Gehalt
Gehalt_Plus_Provision

2.2 Mathematische Operationen

Mit Hilfe der mathematischen Grundoperationen (+,–,*,/) und weiteren Attributen können neue Attribute errechnet werden. Aus Jahresgehältern können monatliche Grundgehälter oder Gehaltserhöhungen ermittelt werden.

Beispiel:

Alle Mitarbeiter mit der Stufe 1 sollen eine 10% Gehaltserhöhung bekommen. Folgende Abfrage ist dann möglich:

```
SELECT    full_name, salary, (salary*1.10), (salary*1.10-salary)
FROM      employee
WHERE JOB_GRADE=1
```

Im Ergebnis werden die Namen, das alte Gehalt, das neue Gehalt und die Gehaltserhöhung aufgelistet. Die mathematischen Ausdrücken sollten grundsätzlich in Klammern gesetzt werden.

Ideal wäre noch die Summe der zusätzlichen Gehaltserhöhungen. Dafür existiert die Funktion

Ergebnis:

FULL_NAME	SALARY	COLUMN2	COLUMN3
Steadman, Walter	116100	127710	11610
Bender, Oliver H.	212850	234135	21285

Mit einem Alias-Namen kann man die Tabelle noch verbessern.

```
SELECT full_name, salary Gehalt, (salary*1.10) Neues_Gehalt, (salary*1.10-salary) Gehaltserhöhung
FROM    employee
WHERE JOB_GRADE=1
```

Ergebnis:

FULL_NAME	GEHALT	NEUES_GEHALT	GEHALTSERHÖHUNG
Steadman, Walter	116100	127710	11610
Bender, Oliver H.	212850	234135	21285

2.3 Boolsche Operatoren

Dieses Kapitel zeigt nur die Wahrheitstabellen, da Kenntnisse in Programmiersprache vorausgesetzt werden. Die Besonderheit ist der Zustand NULL. Hat das Attribut Null-Werte sollte man die Abfragen genau programmieren. Mit der WHERE-Bedingung „IS NULL“ kann man den Inhalt überprüfen.

Hinweis:

Ein NULL-Wert ist nicht dasselbe wie die Zahl Null oder ein Leerzeichen

2.3.1 UND-Operator

	WAHR	FALSCH	NULL
WAHR	WAHR	FALSCH	NULL
FALSCH	FALSCH	FALSCH	FALSCH
NULL	NULL	FALSCH	NULL

2.3.2 ODER-Operator

	WAHR	**FALSCH**	**NULL**
WAHR	WAHR	WAHR	WAHR
FALSCH	WAHR	FALSCH	NULL
NULL	WAHR	NULL	NULL

2.3.3 NOT-Operator

	WAHR	**FALSCH**	**NULL**
NOT	FALSCH	WAHR	NULL

2.4 BETWEEN, IN, IS, LIKE

2.4.1 BETWEEN / AND

Mit der Anweisung „BETWEEN“ kann man in der WHERE-Bedingung Bereiche angeben. Die Syntax lautet dabei:

BETWEEN untereGrenze AND obereGrenze

Beispiel:

Suche alle Mitarbeiter, deren Gehalt zwischen 40.000,00 und 60.000,00 liegt.

```
SELECT full_name, salary
FROM employee
WHERE salary BETWEEN 40000.00 AND 60000.00
```

Zur Kontrolle sollte immer die abgefragte Größe mit ausgegeben werden.

Ergebnis:

FULL_NAME	**SALARY**
Lee, Terri	53793
Phong, Leslie	56034,375
Nordstrom, Carol	42742,5
Burbank, Jennifer M.	53167,5
Williams, Randy	56295
Page, Mary	48000
Yanowski, Michael	44000
Johnson, Scott	60000

2.4.2 IN

Komplexere Abfragen mit AND-Bedingungen lassen sich mit dem Befehl „IN“ einfacher schreiben. Für Sub-Select-Anweisungen ist nur diese Variante möglich.

Beispiel:

Gesucht sind die Mitarbeiter mit den Nummer 107, 109 und 110. Mit Hilfe mehrerer Bedingungen kann diese Abfrage realisiert werden.

```
SELECT full_name, emp_no
FROM employee
WHERE (emp_no=107) AND (emp_no=109) AND (emp_no=110)
```

Diese Abfrage liefert als Ergebnis die leere Menge. Die Verknüpfung mit der AND-Bedingung ist hier nicht richtig. Korrekt wäre eine Oder-Abfrage.

```
SELECT full_name, emp_no
FROM employee
WHERE (emp_no=107) OR (emp_no=109) OR (emp_no=110)
```

Ergebnis:

FULL_NAME	**EMP_NO**
Cook, Kevin	107
Brown, Kelly	109
Ichida, Yuki	110

Für größere Mengen ist diese Technik sehr fehlerträchtig. Abhilfe schafft die IN-Bedingung. Hier wird geprüft, ob der Tupel – das Attribut des Datensatzes – innerhalb dieser Menge ist. Um zu prüfen, ob die Telefonnummer in der Liste (216, 210, 477 und 420) reicht folgende Anweisung

```
WHERE phone_ext IN (216, 210, 477,420)
```

Die obige Anweisung lautet mit der IN-Anweisung:

```
SELECT full_name, emp_no
FROM employee
WHERE emp_no IN (107,109,110)
```

Hinweis:

In der Oberfläche kann man im Register „Plan“ sich die SQL-Befehle nach dem Parser ansehen. Für beide Variante gibt das Programm den identischen Code aus.

PLAN:

```
PLAN (EMPLOYEE INDEX (RDB$PRIMARY7,RDB$PRIMARY7,RDB$PRIMARY7))
```

2.4.3 LIKE

Um Zahlen zu suchen verwendet man die Vergleichsoperatoren oder die Anweisung „BETWEEN“. Bei Zeichenketten besteht häufig das Problem, dass man nicht den exakten Eintrag kennt. Zudem man auch nicht immer sicher ist, ob der Text in Klein- oder Großbuchstaben eingetragen wurde. Als Abhilfe bietet SQL den LIKE-Operator. Dieser Operator funktioniert nach ähnlichen Prinzip der Suche mit dem GREP-Befehl unter UNIX bzw. dem DIR-Befehl unter DR-DOS.

- Verwenden Sie den LIKE-Operator, um eine Platzhaltersuche nach gültigen Zeichenfolgenwerten durchzuführen.
- Die Suchkriterien können entweder literale Zeichen oder Zahlen enthalten.

Suchzeichen:
% steht für kein, ein, oder beliebige Zeichen (entspricht dem *)
_ steht für genau ein Zeichen (entspricht dem ?)

Beispiel:

Gesucht sind alle Mitarbeiter, deren Namen mit dem Buchstaben „B“ anfängt.

```
Select Full_name, emp_no, salary
From employee
Where last_name LIKE "B%";
```

Ergebnis:

FULL_NAME	**EMP_NO**	**SALARY**
Bennet, Ann	28	22935
Baldwin, Janet	34	61637,8125
Burbank, Jennifer M.	71	53167,5
Bishop, Dana	83	62550
Bender, Oliver H.	105	212850
Brown, Kelly	109	27000

Im obigen Beispiel müssen alle Nachnamen mit einem Großbuchstaben anfangen. Später werden String-Operationen behandelt, mit denen man dieses Problem lösen kann.

2.5 Priortätenregel

Die folgende Tabelle zeigt die Reihenfolge der Operatoren. Für die Übersicht ist es aber immer sinnvoll, die mathematischen und bool'schen Ausdrücke in Klammern zu setzen.

Auswertungsreihenfolge	**Operator**
1	Arithmetische Operationen
2	Verkettungsoperator
3	Vergleichsoperator
4	IS [NOT] NULL, LIKE, [NOT] IN
5	[NOT] BETWEEN
6	Logischer Operator NOT
7	Logischer Operator AND
8	Logischer Operator OR

2.6 Verkettungsoperator

Mit Hilfe des Verkettungsoperator kann man Spalten oder Zeichenketten mit Spalten verbinden.

Ein Verkettungsoperator
- verkettet Spalten oder Zeichenfolgen mit anderen Spalten. Ergebnis ist **eine** Spalte!
- wird durch zwei senkrechte Striche (||) dargestellt.
- erstellt eine Ergebnisspalte, die einen Zeichenausdruck enthält.

Mit dem Verkettungsoperator können Sie Spalten mit anderen Spalten, arithmetischen Ausdrücken oder konstanten Werten zu einem Zeichenausdruck verknüpfen. Die Spalten auf beiden Seiten des Operators werden zu einer einzigen Ausgabespalte kombiniert.

Beispiel:

- Gesucht ist eine Tabelle, in der der Mitarbeiter mit dem Job-Code angezeigt wird.

```
SELECT last_name, job_code
From employee
```

Ergebnis (Auszug):

LAST_NAME	**JOB_CODE**
Nelson	VP
Young	Eng
Lambert	Eng
Johnson	Mktg
Forest	Mngr
Weston	SRep
Lee	Admin
Hall	Finan
Young	Mngr
Papadopoulos	Mngr

Mit Hilfe des Verkettungsoperators können die Spalten zusammengefügt werden.

```
Select last_name || job_code Mitarbeiter_Funktionen, emp_no
From employee
```

Ergebnis (Auszug):

MITARBEITER_FUNKTIONEN	EMP_NO
NelsonVP	2
YoungEng	4
LambertEng	5
JohnsonMktg	8
ForestMngr	9
WestonSRep	11
LeeAdmin	12
HallFinan	14
YoungMngr	15
PapadopoulosMngr	20
FisherEng	24
BennetAdmin	28
De SouzaEng	29
BaldwinSales	34
ReevesSales	36
StansburyEng	37
PhongEng	44
RamanathanEng	45
SteadmanCFO	46
NordstromPRel	52

Die Texte sind in der Tabelle direkt zusammengefügt. Mit Hilfe von zusätzlichen Zeichenliteralen ist eine bessere Darstellung möglich.

```
Select last_name || " hat den Job " || job_code Mitarbeiter_Funktionen, emp_no
From employee
```

Ergebnis (Auszug):

MITARBEITER_FUNKTIONEN	EMP_NO
Nelson hat den Job VP	2
Young hat den Job Eng	4
Lambert hat den Job Eng	5
Johnson hat den Job Mktg	8
Forest hat den Job Mngr	9
Weston hat den Job Srep	11
Lee hat den Job Admin	12
Hall hat den Job Finan	14
Young hat den Job Mngr	15
Papadopoulos hat den Job Mngr	20
Fisher hat den Job Eng	24
Bennet hat den Job Admin	28
De Souza hat den Job Eng	29
Baldwin hat den Job Sales	34
Reeves hat den Job Sales	36
Stansbury hat den Job Eng	37

2.7 Beispiele

Dieses Kapitel beschreibt ausführlich mehrere Beispiele und erläutert diese detailliert.

2.7.1 Suche Mitarbeiter (71→100)

Gesucht:

Suche Mitarbeiter mit den Nummern zwischen 71 und 100. Für die Abfrage wird die BETWEEN-Bedingung benutzt.

Abfrage:

```
SELECT full_name, emp_no
FROM employee
WHERE emp_no  BETWEEN 71 AND 100
```

Ergebnis:

FULL_NAME	EMP_NO
Burbank, Jennifer M.	71
Sutherland, Claudia	72
Bishop, Dana	83
MacDonald, Mary S.	85
Williams, Randy	94

2.7.2 Suche Mitarbeiter (Abteilung 610 und 623)

Gesucht:

Suche das Gehalt der Mitarbeiter in den Abteilungen 610 und 623. Für die Abfrage wird die IN-Bedingung benutzt.

Abfrage:

```
SELECT emp_no, full_name, salary
FROM employee
WHERE dept_no IN (610,623)
```

Ergebnis:

EMP_NO	FULL_NAME	SALARY
15	Young, Katherine	67241,25
29	De Souza, Roger	69482,625
44	Phong, Leslie	56034,375
114	Parker, Bill	35000
136	Johnson, Scott	60000

Denkbar wäre auch folgende Variante:

```
SELECT emp_no, full_name, salary
FROM employee
WHERE (dept_no =610)  OR  (dept_no =623)
```

2.7.3 Suche die Mitarbeiter Green und Bishop

Gesucht:

Suche das Gehalt der Mitarbeiter Green und Bishop. Für die Abfrage wird die IN-Bedingung benutzt.

Abfrage:

```
Select Full_name, emp_no, salary
From employee
Where last_name IN ("Green", "Bishop")
```

Ergebnis:

FULL_NAME	EMP_NO	SALARY
Bishop, Dana	83	62550
Green, T.J.	138	36000

2.7.4 Full-Name

Das Attribut „Full_Name“ ist nur virtuell definiert. Es wird automatisch beim Anzeigen erzeugt. Ziel der nächsten Abfrage ist eine Tabelle, die dieses Attribut erzeugt.

Abfrage:

```
Select last_name || ", " || first_name, emp_no
From employee
```

Ergebnis (Auszug):

COLUMN0	EMP_NO
Nelson, Robert	2
Young, Bruce	4
Lambert, Kim	5
Johnson, Leslie	8
Forest, Phil	9
Weston, K. J.	11
Lee, Terri	12
Hall, Stewart	14
Young, Katherine	15
Papadopoulos, Chris	20
Fisher, Pete	24
Bennet, Ann	28
De Souza, Roger	29
Baldwin, Janet	34
Reeves, Roger	36
Stansbury, Willie	37
Phong, Leslie	44
Ramanathan, Ashok	45
Steadman, Walter	46

2.7.5 Welchen Job hat welcher Mitarbeiter

Die nächste Abfrage zielt darauf, eine Tabelle zu erhalten, in dem der Job_Code mit dem Nachnamen des Mitarbeiters in einer Spalte eingetragen wurde. Also „Meier ist ein Admin“. Zusätzlich soll dieses Spalte die Überschrift „Hat_Job“ erhalten und nur Mitarbeiter der Abteilungsnummer 621 und 623 betreffen.

Abfrage:

```
Select last_name || " ist ein  " || job_code Hat_Job, dept_no
From employee
where (dept_no=621) OR (dept_no=623)
```

Ergebnis:

HAT_JOB	DEPT_NO
Young ist ein Eng	621
Young ist ein Mngr	623
De Souza ist ein Eng	623
Phong ist ein Eng	623
Ramanathan ist ein Eng	621
Bishop ist ein Eng	621
Parker ist ein Eng	623
Johnson ist ein Doc	623
Green ist ein Eng	621

2.7.6 Mitarbeiter aus dem Jahr 1994

Gesucht:
Alle Mitarbeiter, die im Jahr 1994 angefangen haben. Die Datumsabfragen kommen in einem späteren Kapitel, aber mit Hilfe des LIKE-Operators ist diese Abfrage realisierbar.

Abfrage:

```
SELECT Full_name, emp_no, salary, hire_date
FROM employee
WHERE hire_date LIKE "%1994%"
```

Ergebnis:

FULL_NAME	EMP_NO	SALARY	HIRE_DATE
Osborne, Pierre	141	110000	03.01.1994
Montgomery, John	144	35000	30.03.1994
Guckenheimer, Mark	145	32000	02.05.1994

Das Prozentzeichen bedeutet eine beliebige Anzahl von Zeichen.

2.8 Übungen

Dieses Kapitel bietet Übungen, die auf die vorherigen Aufgaben aufbauen. Im Kapitel 10.2, Seite 115 sind die Lösungen aufgelistet und kommentiert..

2.8.1 Full-Name mit Alias Namen

Das Attribut „Full_Name“ ist nur virtuell definiert. Es wird automatisch beim Anzeigen erzeugt. Ziel der nächsten Abfrage ist eine Tabelle, die dieses Attribut erzeugt. Betroffen sind aber nur Mitarbeiter, die in den USA arbeiten. Die Überschrift der ersten Spalte sollte Name betragen. Sortiert wird als erstes nach den Abteilungen, dann nach den Namen.

Ergebnis:

NAME	EMP_NO
Nelson, Robert	2
Young, Bruce	4
Lambert, Kim	5
Johnson, Leslie	8
Forest, Phil	9
Weston, K. J.	11
Lee, Terri	12
Hall, Stewart	14
Young, Katherine	15
Papadopoulos, Chris	20
Fisher, Pete	24
De Souza, Roger	29
Baldwin, Janet	34
Phong, Leslie	44
Ramanathan, Ashok	45

Steadman, Walter	46
Nordstrom, Carol	52
Leung, Luke	61
O'Brien, Sue Anne	65
Burbank, Jennifer M.	71
Bishop, Dana	83
MacDonald, Mary S.	85
Williams, Randy	94
Bender, Oliver H.	105
Cook, Kevin	107
Brown, Kelly	109
Page, Mary	113
Parker, Bill	114
Yanowski, Michael	127
Johnson, Scott	136
Green, T.J.	138
Montgomery, John	144
Guckenheimer, Mark	145

2.8.2 Welcher Mitarbeiter heisst Robert am Anfang

Gesucht sind die Mitarbeiter, die mit Anfangsvornamen Robert heißen. Ausgegeben werden sollen der Vor- und der Nachname. Als Überschriften dienen die Bezeichnungen „Vorname“ und „Nachname“.

Ergebnis:

VORNAME	NACHNAME
Robert	Nelson
Roberto	Ferrari

2.8.3 Mitarbeiter aus dem Jahr 1992 und 1994

Gesucht:
Alle Mitarbeiter, die im Jahr 1992 und 1994 angefangen haben und ein Gehalt größer € 60000,00 haben. Sortiert aufsteigend nach Gehalt.

Ergebnis:

FULL_NAME	EMP_NO	SALARY	HIRE_DATE
Bishop, Dana	83	62550	01.06.1992
Leung, Luke	61	68805	18.02.1992
Sutherland, Claudia	72	100914	20.04.1992
Osborne, Pierre	141	110000	03.01.1994
MacDonald, Mary S.	85	111262,5	01.06.1992
Bender, Oliver H.	105	212850	08.10.1992

2.8.4 Codierungsfehler

Die folgende SELECT-Anweisung enthält vier Fehler. Können Sie alle vier finden?

```
SELECT emp_no, last_name
salary x 12   ANNUAL SALARAY
from employee
```

3 Data Defintion Language

3.1 Die INSERT INTO Anweisung

Der Aufbau der INSERT INTO Anweisung lässt zwei Möglichkeiten des Hinzufügen von Daten in eine Tabelle zu. In der einfachsten Notationsform wird mit jedem Befehl genau ein Datensatz in eine Tabelle eingefügt.

```
INSERT INTO     table [(column [, column …])]
VALUES          (value [, value …]);
```

Abbildung 2.: Die INSERT INTO Anweisung

In der Syntax ist:

table der Name der Tabelle
column der Name der mit Daten zu füllenden Tabellenspalte
value der entsprechende Wert für die Spalte

Durch diesen Befehl wird immer nur eine Zeile in die Datenbanktabelle eingefügt. Sollen Daten aus anderen Tabellen eingefügt werden so muss folgende Befehlsnotation benutzt werden:

```
INSERT INTO     table [(column [, column …])]
   SELECT column [, column …])
FROM    table, …;
```

Abbildung 3.: Die INSERT INTO Anweisung mit SELECT

Hinweis:

Beispiele:

```
INSERT INTO KUNDE( empno, name )
VALUES (123,'Müller');

INSERT INTO KUNDE( name, empno )
VALUES ('Schmidt',125);
```

3.2 Die UPDATE Anweisung

Mit Hilfe der UPDATE-Anweisung können vorhandene Zeilen bearbeiten werden.

```
UPDATE      table
SET         column = value [, column = value …])
[WHERE      condition(s)];
```

Abbildung 4.: Die UPDATE Anweisung

Beispiel:

```
UPDATE KUNDE
SET SALDO=61615, KREDIT=81674
WHERE ( KUNDENNR = 1);
```

In der obigen Syntax ist:

table der Name der Tabelle,
column der Name der mit Daten zu füllenden Tabellenspalte,
value der entsprechende Wert oder die Unterabfrage für die Spalten,
conditions identifiziert die zu aktualisierenden Zeilen und setzt sich zusammen aus Spaltennamen, Ausdrücken, Konstanten, Unterabfragen und Vergleichsoperatoren

Anschließend sollte der Änderungsvorgang durch Abfragen der Tabelle überprüft werden, um die aktualisierten Zeilen anzuzeigen.

Generell sollten, falls vorhanden der Primärschlüssel verwendet werden, um eine einzelne Zeile zu identifizieren. Wenn Sie andere Spalten verwenden, könnte das zu unerwarteten Änderungen mehrerer Zeilen führen. Eine einzelne Zeile der Tabelle EMP beispielsweise über den Namen zu identifizieren, könnte gefährlich sein, da mehrere Mitarbeiter den gleichen Namen haben können.

3.3 Die DELETE FROM Anweisung

Mit Hilfe der DELETE-Anweisung können vorhandene Zeilen gelöscht werden.

```
DELETE [FROM] table
   [WHERE                condition(s)];
```

Abbildung 5.: Die DELETE [FROM] Anweisung

Syntax :

Syntax:

Table	der Tabellenname,
conditions	identifiziert die zu löschenden Zeilen und setzt sich zusammen aus Spaltennamen, Ausdrücken, Konstanten, Unterabfragen und Vergleichsoperatoren.

Beispiele:

Löschen aller Mitarbeiter der Abteilung 110;	DELETE FROM EMPLOYEE; WHERE deptno=110
Löschen der gesamten Tabelle:	DELETE FROM EMPLOYEE;

Das Löschen geht nur, wenn keine weitere Beziehung vorhanden ist.

3.4 Die Data Definition Language

Diese Teilsprache von SQL dient zum Erzeugen, Ändern und Löschen von Datenbankobjekten. Aus der Vielzahl möglicher Objekte seinen hier nur die wichtigsten behandelt. Es handelt sich um die Tabellen als Ort der Abspeicherung von Daten.

Das konzeptionelle Schema mündet in eine Vielzahl, miteinander über Beziehungen verbundener Tabellen – Relationen (für Entitätsmengen und Beziehungen).

Views dienen der Steuerung des Zugriffs verschiedener Benutzergruppen auf die Daten.

3.5 Die CREATE TABLE Anweisung

Es werden Tabellen zum Speichern von Daten erstellt, indem die Anweisung CREATE TABLE ausführt wird. Dies ist eine Anweisung der Data Definition Language (DDL). DDL-Anweisungen sind eine Untermenge von SQL-Anweisungen zum Erstellen, Ändern oder Entfernen von Datenbankstrukturen. Diese Anweisungen wirken sich unmittelbar auf die Datenbank aus und zeichnen auch Informationen im Data Dictionary auf.

Um eine Tabelle zu erstellen, benötigt ein Benutzer das Privileg CREATE TABLE sowie einen Speicherbereich, in dem er die Objekte erstellt. Der Datenbank-Administrator verwendet zum Vergeben von Privilegien Anweisungen der Data Control Language (DCL).

```
CREATE [GLOBAL TEMPORARY] TABLE      [schema.] table
    (column datatype [DEFAULT expr], …);
```

Abbildung 6.: Die CREATE TABLE Anweisung

Syntax:

GLOBAL TEMPORARY	gibt an, dass die Tabelle temporär ist und dass ihre Definition in allen Sessions sichtbar ist. Die Daten einer temporären Tabelle sind nur in der Session sichtbar, in welcher die Daten eingefügt wurden.
schema	entspricht dem Eigentümernamen.
table	ist der Name der Tabelle.
DEFAULT expr	gibt einen Standardwert an, falls die INSERT-Anweisung keinen Wert enthält.
Column	ist der Name der Spalte.
Datatype	ist der Datentyp und die Länge der Spalte.

Die Datentypen beschreiben den Wertebereich der Attribute und werden allgemein in Datenbanksystemen den folgende Möglichkeiten zugeordnet:

- Zahlen,
- Geldbeträgen,
- Datum und Zeitwerten und
- Zeichenketten.

Im Besonderen ermöglichen moderne Datenbankmanagementsysteme auch den Umgang mit großen binären Datenmengen, den BLOBs.

Beispiel für Tabelle EMP:

```
create table emp
(   empno          number(4),
    ename          varchar2(10),
    job            varchar2(9),
    mgr            number(4),
    hiredate       date          default sysdate,
    sal            number(7,2),
    comm           number(7,2),
    deptno         number(2),
  constraint   emp_pk primary key (empno),
  constraint       emp_fk_emp foreign key (mgr) references emp(empno),
    constraint     emp_fk_dept foreign key (deptno) references dept(deptno));
```

Beispiel für Tabelle EMP:

```
create table emp (
    empno          number(4),
    ename          varchar2(10),
    job            varchar2(9),
    mgr            number(4),
    hiredate       date          default sysdate,
    sal            number(7,2),
    comm           number(7,2),
    deptno         number(2),
  primary key (empno),
  foreign key (mgr) references emp(empno),
  foreign key (deptno) references dept(deptno));
```

3.6 Die CREATE VIEW Anweisung

Eine View kann erstellen, indem eine Unterabfrage in die Anweisung CREATE VIEW einbetten wird.

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW      view
 [(alias [, alias] …)]
as subquery
[WITH CHECK OPTION [CONSTRAINT constraint]]
[WITH READ ONLY];
```

Abbildung 7.: Die CREATE VIEW Anweisung

Syntax:

OR REPLACE	erstellt die View neu, falls sie bereits vorhanden ist.
FORCE	erstellt die View unabhängig davon, ob die Basistabelle existiert.
NOFORCE	erstellt die View nur, wenn die Basistabelle existiert. Dies ist der Standard.
View	ist der Name der View.
Alias	gibt Namen für die Ausdrücke an, die durch die View-Abfrage ausgewählt wurden. Die Anzahl der Aliasnamen muss mit der Anzahl der durch die View ausgewählten Ausdrücke übereinstimmen.
subquery	ist eine komplette SELECT-Anweisung. Für die Spalten in der SELECT-Liste können Sie Aliasnamen verwenden.
WITH CHECK OPTION	gibt an, dass nur solche Zeilen eingefügt oder aktualisiert werden können, die über die View zugänglich sind.
constraint	ist der dem Constraint CHECK OPTION zugewiesene Name.
WITH READ ONLY	stellt sicher, dass keine DML-Operationen auf diese View durchgeführt werden können.

3.7 Beispiele

Beispiele:

```
REM  Eintragen aller Daten eines Tupels
INSERT INTO KUNDE
VALUES(123,'Schmidt', 'Andreas', );

REM  Eintragen zweier Daten eines Tupels
INSERT INTO KUNDE    ( KNR, NAME )
VALUES(123,'Schmidt' );
```

Beispiel Update:

```
update Kunde
  set Ort = 'WR'
where KNR = 1;
```

4 Funktionen

Dieses Kapitel befasst sich mit SQL-Funktionen. Dazu gehören Funktionen zur Manipulation von Zeichenketten, Datumsfunktionen, Konvertierungsfunktionen und allgemeine Funktionen.

4.1 String-Manipulation

Die Schreibweise von Texten in Attributen ist nicht immer eindeutig. So kann ein Mitarbeiter die Namen in Kleinbuchstaben oder komplett in Großbuchstaben eingeben. Eine lexikalische Suche ist dann nicht mehr möglich. SQL bietet hier mit String-Funktionen Abhilfe.

Folgende Funktionen sind möglich:

Oracle-Funktionen:

Funktion	**Zweck**
LOWER(column | Expression)	Konvertiert alphanumerische Zeichenwerte in Kleinbuchstaben
UPPER(column | Expression)	Konvertiert alphanumerische Zeichenwerte in Großbuchstaben
INITCAP(column | Expression)	Konvertiert alphanumerische Zeichenwerte in Groß- und Kleinbuchstaben. Der erste Buchstabe wird großgeschrieben, alle anderen Buchstaben des Wortes werden kleingeschrieben. Ähnlich ist die Namensgebung bei vielen Programmiersprachen.
CONCAT(column1 | Expression1, column2 | Expression2)	Verkettet den ersten Zeichenwert mit dem zweiten. Entspricht dem Verkettungsoperator (||).
SUBSTR(column | Expression,m, [,n])	Gibt bestimmte Zeichen aus einem Zeichenwert (String) zurück. Die zurückgegebene Teilfolge beginnt bei der Position m und ist *n* Zeichen lang. Ist *m* negativ, so wird am Ende des Strings mit der Zählung begonnen. Wird *n* nicht angegeben, so werden alle Zeichen bis zum Ende bzw. Anfang des Strings zurückgegeben.
LENGTH(column | Expression)	Gibt die Anzahl der Zeichen in String zurück.
INSTR(column | Expression, 'STRING' [, m] [,n])	Gibt die numerische Position einer genannten Zeichenfolge zurück. Optional können Sie mit *m* die Position angeben, an der die Suche beginnt, und mit *n* festlegen, das wievielte Auftreten des Strings Sie anzeigen möchten. Standardmäßig ist *m* und *n* gleich 1. Das bedeutet, die Suche beginnt am Anfang und das erste Vorkommen des Suchfolge wird zurückgegeben. Bei erfolgloser Suche wird eine leere Zeichenkette zurückgegeben (NULL).
LPAD(column | Expression, n 'STRING')	Füllt den Zeichenwert rechtsbündig bis zur Zeichenposition *n* mit Leerzeichen auf.
RPAD(column | Expression, n 'STRING')	Füllt den Zeichenwert linksbündig bis zur Zeichenposition *n* mit Leerzeichen auf.
TRIM(leading | trailing | both, *trim_character* FROM *trim_source*)	Ermöglicht es, Zeichen am Anfang und / oder am Ende einer Zeichenfolge abzuschneiden. Wenn *trim_character* oder *trim_source* ein Zeichenliteral ist, müssen Sie es in Hochkommatas setzen. Diese Eigenschaft ist ab Oracle 8i implementiert.
REPLACE(text, search_string, replace_string)	Sucht einen Textausdruck in einer Zeichenfolge und ersetzt diesen durch *replace_string*. Das Ergebnis wird als Funktionswert ausgegeben.

Firebird-Funktionen:

Funktion	
ASCII_CHAR(column | Expression)	Liefert das Charakterzeichen das als Parameter übergeben wurde.
ASCII_VAL	Liefert den Charakterwert (Zahl) des übergebenen Parameters.
LOWER(column | Expression)	Konvertiert alphanumerische Zeichenwerte in Klein-buchstaben
LTRIM(column | Expression)	Löscht alle Leerzeichen am Anfang der Zeichenfolge.
RTRIM(column | Expression)	Löscht alle Leerzeichen am Ende der Zeichenfolge.
SUBSTR(column | Expression,m,[,n])	Gibt bestimmte Zeichen aus einem Zeichenwert (String) zurück. Die zurückgegebene Teilfolge beginnt bei der Position m und endet bei der Position n. Sie ist (n-m+1) Zeichen lang.. **Hinweis:** Wenn n größer der Länge wird der Originalstring zurückgegeben!
SUBSTRLEN (column | Expression,m, len)	Gibt bestimmte Zeichen aus einem Zeichenwert (String) zurück. Die zurückgegebene Teilfolge beginnt bei der Position m und ist *len* Zeichen lang. Ist *m* negativ, so wird am Ende des Strings mit der Zählung begonnen. Wird *len* nicht angegeben, so werden alle Zeichen bis zum Ende bzw. Anfang des Strings zurückgegeben. Hinweis: Wenn *len* größer der Länge wird der Originalstring zurückgegeben!
STRLEN(column | Expression)	Gibt die Anzahl der Zeichen in String zurück.
UPPER(column | Expression)	Konvertiert alphanumerische Zeichenwerte in Groß-buchstaben

Beispiele (Oracle):

Funktion	**Ergebnis**
CONCAT('Hello', 'World')	HelloWorld
SUBSTR('HelloWorld',2,5)	ElloW
LENGTH('HelloWorld')	10
INSTR('HelloWorld', 'W')	6
LPAD(salary,10,'*')	*****24000
RPAD(salary, 10, '*')	24000*****
TRIM('H' FROM 'HelloWorld')	elloWorld
LOWER('Hallo Welt')	hallo welt

- Mit CONCAT werden String zusammengefügt. Der Verkettungsoperator hat dieselbe Funktion.
- Der Befehl SUBSTR gibt ab Position 2 fünf Zeichen aus. Als Ergebnis erhält man „elloW“.
- Mit LENGTH kann man die Länge einer Zeichenkette bestimmen. Dient auch zur Tabellierung von Spalten.
- Das Suchen des Buchstaben „W“ erreicht man mit INSTR. Als Ergebnis erhält man die Position des Buchstabens „W“.

- Die Funktionen LPAD und RPAD dienen dem Füllen beim Ausdrucken von Zahlen.
- Die Funktion TRIM löscht beliebige Zeichen an den Enden eines Textes bzw. eines Attributes.
- LOWER und UPPER werden zur genauen Suche nach Begriffen verwandt.

Beispiele (Firebird):

SUBSTRLEN entspricht dem SUBSTR aus Oracle.
Statt INSTR sollte man LIKE verwenden.

Funktion	**Ergebnis**	**Bemerkung**
'Hello' || 'World'	HelloWorld	
SUBSTRLEN ('HelloWorld',2,5)	ElloW	
SUBSTR ('HelloWorld',2,6)	ElloW	
STRLEN('HelloWorld')	10	
'HelloWorld' LIKE '%W'	6	
LPAD(salary,10,'*')	*****24000	Externe Funktion
RPAD(salary, 10, '*')	24000*****	Externe Funktion
LTRIM(' HelloWorld')	HelloWorld	
RTRIM('HelloWorld ')	HelloWorld	
LOWER('Hallo Welt')	hallo welt	
UPPER('Hallo Welt')	HALLO WELT	

- Mit dem Verkettungsoperator werden die beiden String verbunden.
- Der Befehl SUBSTRLEN gibt ab Position 2 fünf Zeichen aus. Als Ergebnis erhält man „elloW“.
- Der Befehl SUBSTR gibt ab Position 2 bis Position 6 fünf Zeichen aus. Als Ergebnis erhält man „elloW“.
- Mit STRLEN kann man die Länge einer Zeichenkette bestimmen. Dient auch zur Tabellierung von Spalten.
- Das Suchen des Buchstaben „W“ erreicht man mit INSTR. Als Ergebnis erhält man die Position des Buchstabens „W“. ?
- Die Funktionen LPAD und RPAD dienen dem Füllen beim Ausdrucken von Zahlen. Sie sind in Firebird nicht verfügbar und müssen über externe Funktionen implementiert werden.
- Die Funktionen LTRIM und RTRIM löschen Leerzeichen an den Enden von Zeichenketten. Dabei kann man diese auch kombinieren (LTRIM(RTRIM(" Hallo Welt ")) ergibt "Hallo Welt".
- LOWER und UPPER werden zur genauen Suche nach Begriffen verwandt.

Beispiele:

- **Gesucht werden alle Nachnamen der Mitarbeiter in Großbuchstaben.**

Abfrage:

SELECT UPPER(last_name)
FROM employee

UPPER
NELSON
YOUNG
LAMBERT
JOHNSON
FOREST

WESTON
LEE
HALL
YOUNG
PAPADOPOULOS
FISHER
BENNET
DE SOUZA
BALDWIN
REEVES
STANSBURY
PHONG
RAMANATHAN
STEADMAN
NORDSTROM
LEUNG
O'BRIEN
BURBANK
SUTHERLAND
BISHOP
MACDONALD
WILLIAMS
BENDER
COOK
BROWN
ICHIDA
PAGE
PARKER
YAMAMOTO
FERRARI
YANOWSKI
GLON
JOHNSON
GREEN
OSBORNE
MONTGOMERY
GUCKENHEIMER

- **Gesucht werden alle Nachnamen der Mitarbeiter die ein „on“ im Buchstaben haben.**

Abfrage (Oracle):

```
SELECT last_name
FROM employee
WHERE SUBSTR(last_name,"on") = "on"
```

Abfrage (Firebird):

```
SELECT last_name
FROM employee
WHERE last_name LIKE "%on"
```

Ergebnis:

LAST_NAME
Nelson
Johnson
Weston
Glon
Johnson

- **Gesucht werden alle Mitarbeiter (full_name) die ein „mary“ im Vornamen haben. Sortiert werden soll nach den Mitarbeiternummern.**

Abfrage:

```
SELECT full_name
FROM employee
WHERE lower(first_name) LIKE "%mary"
ORDER BY emp_no
```

Ergebnis:

FULL_NAME
Page, Mary

Betrachtet man die Original-Tabelle, so fällt auf, dass es zwei „Mary´s“ gibt. Die zweite heisst „Mary S.“. Dementsprechend muss die Abfrage verfeinert werden:

Abfrage:

```
SELECT full_name
FROM employee
WHERE lower(first_name) LIKE "%mary%"
ORDER BY emp_no
```

Ergebnis:

FULL_NAME
MacDonald, Mary S.
Page, Mary

Die Abfrage mit LOWER ist hier sinnvoll, da nicht immer gewährleistet ist, dass nur der erste Buchstabe in Großbuchstaben eingetragen wurde. Vorhandene Eingabebedingungen helfen hier natürlich.

4.2 Datumsfunktionen

Ein Datum wird intern als binäre Bitfolge dargestellt. Meistens als Anzahl der Tage ab einem bestimmten Stichtag (z. B. 1.1.1970). Ein Datum vor diesem Tag ist dann nicht möglich. Jede Datenbank speichert die Datumswerte individuell. Deshalb existieren Funktionen, die die Tage, Monate und Jahre aus den binären Werten extrahieren.

4.2.1 Oracle-Datumsfunktionen

Name	**Beschreibung**	**Beispiel**
SYSDATE	Gibt das aktuelle Datum und die aktuelle Uhrzeit des Servers aus.	SELECT emp_no, sysdate from employee
MONTHS_BETWEEN	Anzahl der Monate zwischen zwei Datumswerten	SELECT emp_no FROM employee WHERE MONTHS_BETWEEN(d1,d2)>2
ADD_MONTHS	Kalendermonate zu einem Datum hinzuaddieren	ADD_MONTHS(d1,n)
NEXT_DAY	Datum des nächsten Wochentags	NEXT_DAY(date , 2)
LAST_DAY	Letzter Tag des Monats	LAST_DAY(date)
ROUND	Datumswert runden	ROUND(date [, fmt])
TRUNC	Datumswert abrunden	TRUNC(date [, fmt])

Beispiele:

MONTHS_BETWEEN('01-SEP-95', '11-JAN-94') ⇒ 19.6774194

ADD_MONTHS('11-JAN-94', 6) ⇒ '11-JUL-94'

NEXT_DAY('01-SEP-95', FRIDAY') ⇒ '08-SEP-95'

LAST_DAY('01-FEB-95') ⇒ '28-FEB-95'

4.2.2 FIREBIRD-Datumsfunktionen

Name	**Beschreibung**
CAST('NOW' AS TIMESTAMP)	Gibt das aktuelle Datum und die aktuelle Uhrzeit des Servers aus (In der SELECT-Anweisung bzw. bei der Zuweisung eines Datensatzes).
CAST('TODAY' AS TIMESTAMP)	Gibt das aktuelle Datum des Servers aus (In der SELECT-Anweisung bzw. bei der Zuweisung eines Datensatzes).
CAST('YESTERDAY' AS TIMESTAMP)	Gibt das gestrige Datum des Servers aus (In der

	SELECT-Anweisung bzw. bei der Zuweisung eines Datensatzes).
CAST('TOMORROW' AS TIMESTAMP)	Gibt das morgige Datum des Servers aus (In der SELECT-Anweisung bzw. bei der Zuweisung eines Datensatzes).
EXTRACT(day FROM column)	Liest aus dem Datum den Tag
EXTRACT(month FROM column)	Liest aus dem Datum den Monat
EXTRACT(year FROM column)	Liest aus dem Datum den Jahr
EXTRACT(hour FROM column)	Liest aus dem Datum die Stunde
EXTRACT(minute FROM column)	Liest aus dem Datum die Minute
EXTRACT(second FROM column)	Liest aus dem Datum die Sekunde

Anzeige des aktuellen Jahres:
Select DISTINCT Extract (YEAR FROM CAST('NOW' AS TIMESTAMP))
From employee;

Beispiele:

- Anzeige des aktuellen Datums und der aktuellen Zeit

Dazu verwendet man den Zeitstempel „now“ mit dem expliziten Typecasting auf einen TIMESTAMP.

Abfrage:

SELECT CAST('NOW' AS TIMESTAMP)
FROM employee

Ergebnis:

CAST
12.11.2003 17:33:49
12.11.2003 17:33:49
12.11.2003 17:33:49
12.11.2003 17:33:49
12.11.2003 17:33:49
12.11.2003 17:33:49
12.11.2003 17:33:49
12.11.2003 17:33:49
12.11.2003 17:33:49
12.11.2003 17:33:49
12.11.2003 17:33:49
12.11.2003 17:33:49
12.11.2003 17:33:49
12.11.2003 17:33:49
12.11.2003 17:33:49

- Gesucht die Mitarbeiternamen und der jeweilige Tag, Monat, Jahr des Einstellungstages. Diese numerischen Werte sollen in jeweils eine eigene Spalte eingetragen werden. Wichtig sind aber nur die Mitarbeiter, der Abteilung 632. Sinnvolle Spaltennamen sollten verwendet werden.

Dazu verwendet man das Attribut hire_date mit den drei EXTRACT-Methoden.

Abfrage:

```
SELECT    full_name Name, EXTRACT(day FROM hire_date) Tag,
          EXTRACT(month FROM hire_date) Monat, EXTRACT(year FROM hire_date) Jahr
FROM employee
WHERE dept_no = 623
```

Ergebnis:

Name	**Tag**	**Monat**	**Jahr**
Young, Katherine	14	6	1990
De Souza, Roger	18	2	1991
Phong, Leslie	3	6	1991
Parker, Bill	1	6	1993
Johnson, Scott	13	9	1993

4.2.3 Check-Bedingungen mit einem Datum

Für Firebird wäre das diese Syntax:

```
CAST('NOW' AS DATE)              liefert das aktuelle Datum
CAST('NOW' AS TIMESTAMP)         liefert das aktuelle Datum, Zeit
CAST('NOW' AS TIME)              liefert die aktuelle Zeit

EXTRACT(year FROM column)        Auslesen des Jahres
```

Bedingung GebDatum nur als Jahr:

```
CHECK (
  GEBDATUM   <  EXTRACT(year FROM  CAST('NOW' AS DATE))-17
)
```

Bedingung GebDatum als Datum:

```
CHECK (
  EXTRACT(year FROM GEBDATUM)  <  EXTRACT(year FROM  CAST('NOW' AS DATE))-17
)
```

oder mit Monaten etwas genauer

```
CHECK (
  EXTRACT(year FROM GEBDATUM)*12 + EXTRACT(month FROM GEBDATUM)

    <
          EXTRACT(year FROM  CAST('NOW' AS DATE))*12 +
          EXTRACT(month FROM  CAST('NOW' AS DATE))
)
```

```
select
current_date,   current_time    from rdb$database
```

4.2.4 Datumprüfung unter Firebird

4.2.4.1 Tabellendefinitionen

```
create table emp1 (
  empno integer not null,
  last_name varchar(30),
  birthday date,
  hire_date date,
  primary key(empno)
  );

create table emp2 (
  empno integer not null,
  last_name varchar(30),
  birthday date,
  hire_date date,
  primary key(empno)
  );
```

4.2.4.2 PL-SQL-Funktionen

Check mit NOW:

```
set term ^^ ;

create procedure CheckGebDateNow(birthday date, iAlter integer)
returns (iOk integer)
as
--
declare variable GebTAG integer;
declare variable GebMONAT integer;
declare variable GebJAHR integer;

declare variable TAG integer;
declare variable MONAT integer;
declare variable JAHR integer;

declare variable heute date;

begin
  heute = CAST('NOW' AS DATE);

  GebTAG = extract(day from birthday);
  GebMONAT = extract(month from birthday );
  GebJAHR = extract(year from birthday);

  TAG = extract(day from Heute);
  MONAT = extract(month from Heute);
  JAHR = extract(year from Heute);

    // pruefe, ob Mindestalter iAlter
```

```
  if (  (GebJAHR+iAlter) < JAHR ) then
  begin
    iOk=1; // true
  end
  else
  begin
    if (  (GebMonat+ iAlter) = JAHR ) then
    begin
      if ( (GebMonat*100+GebTag) <= (Monat*100+Tag) ) then
        iOk=1; // true
      else
         iOk=0; // false
    end
    else
    begin
      iOk=0; // false
    end
  end

  suspend;
end
^^

set term ; ^^
```

Check mit hire_Date:

```
set term ^^ ;

create procedure CheckDates(birthday date, hire_date date, iAlter integer)
returns (iOk integer)
as
--
declare variable GebTAG integer;
declare variable GebMONAT integer;
declare variable GebJAHR integer;

declare variable TAG integer;
declare variable MONAT integer;
declare variable JAHR integer;

begin

  GebTAG = extract(day from birthday);
  GebMONAT = extract(month from birthday );
  GebJAHR = extract(year from birthday);

  TAG = extract(day from hire_date);
  MONAT = extract(month from hire_date);
  JAHR = extract(year from hire_date );

    // pruefe, ob Mindestalter iAlter
  if (  (GebJAHR+iAlter) < JAHR ) then
  begin
    iOk=1; // true
```

```
  end
  else
  begin
    if (  (GebMonat+ iAlter) = JAHR ) then
    begin
      if ( (GebMonat*100+GebTag) <= (Monat*100+Tag) ) then
        iOk=1; // true
      else
         iOk=0; // false
    end
    else
    begin
      iOk=0; // false
    end
  end

  suspend;
end
^^

set term ; ^^
```

Tests:

```
select * from CheckGebDateNow ('11.10.1964', 16);
select * from CheckGebDateNow ('11.10.2004',16);

select * from CheckDates ('11.10.1986', '11.10.2004', 16);
select * from CheckDates ('11.10.1986',   CAST('NOW' AS DATE) , 16);
```

4.2.4.3 Checkbedingungen

```
ALTER TABLE emp1
  ADD CONSTRAINT chdate1 CHECK(
      ( birthday IS NULL)  OR
     ( hire_date IS NULL) OR

    (    (EXTRACT(year FROM birthday) +16)  < EXTRACT(year FROM hire_date)       )   OR

    (
       ( (EXTRACT(year FROM birthday) +16)  = EXTRACT(year FROM hire_date)  )   AND
       (
           (
             EXTRACT(month FROM birthday)*100+ EXTRACT(day FROM birthday)
           )  <=
           (
             EXTRACT(month FROM hire_date)*100+ EXTRACT(day FROM hire_date)
           )
      )
    )
   );
```

```
ALTER TABLE emp2
  ADD CONSTRAINT chdate2 CHECK(
    (
       (select iOk from CheckDates (new.birthday, new.hire_date,16) ) >0
     )
  );
```

4.2.4.4 Daten

```
insert into emp1 (empno, last_name)
values( 1, 'Müller');


insert into emp1 (empno, last_name, birthday)
values( 2, 'Schmidt', '12.12.2000');


insert into emp1 (empno, last_name, hire_date)
values( 3, 'Meyer', '12.06.2007');



insert into emp1 (empno, last_name, birthday, hire_date)
values( 4, 'Meier', '12.01.1986', '12.06.2007');



insert into emp1 (empno, last_name, birthday, hire_date)
values( 6, 'Brandt', '12.01.1996', '12.06.2007');




/* Fehler, NULL-Wert in Birthday  hire_date */
insert into emp2 (empno, last_name)
values( 1, 'Müller');

/* Fehler, NULL-Wert in hire_date */
insert into emp2 (empno, last_name, birthday)
values( 2, 'Schmidt', '12.12.2000');


/* Fehler, NULL-Wert in Birthday   */
insert into emp2 (empno, last_name, hire_date)
values( 3, 'Meyer', '12.06.2007');


/* okay */
insert into emp2 (empno, last_name, birthday, hire_date)
values( 4, 'Meier', '12.01.1986', '12.06.2007');


/* Fehler, zu jung */
```

```
insert into emp2 (empno, last_name, birthday, hire_date)
values( 6, 'Brandt', '12.01.1996', '12.06.2007');
```

4.3 Allgemeine Funktionen

4.3.1 CASE, Firebird

Ermöglichen bedingte Abfragen in der Form einer
IF-THEN-ELSE-Anweisung:

```
CASE expr WHEN comparison_expr1 THEN return_expr1
       [WHEN comparison_expr2 THEN return_expr2
        WHEN comparison_exprn THEN return_exprn
        ELSE else_expr]
END
```

Die Case-Anwesiung ist erst ab Version 1,5 implementiert.

Beispiele:

```
A) (simple)
 SELECT
   o.ID,
   o.Description,
   CASE o.Status
     WHEN 1 THEN 'confirmed'
     WHEN 2 THEN 'in production'
     WHEN 3 THEN 'ready'
     WHEN 4 THEN 'shipped'
     ELSE 'unknown status '' || o.Status || ''
   END
 FROM
   Orders o

B) (searched)
 SELECT
   o.ID,
   o.Description,
   CASE
     WHEN (o.Status IS NULL) THEN 'new'
     WHEN (o.Status = 1) THEN 'confirmed'
     WHEN (o.Status = 3) THEN 'in production'
     WHEN (o.Status = 4) THEN 'ready'
     WHEN (o.Status = 5) THEN 'shipped'
     ELSE 'unknown status '' || o.Status || ''
   END
 FROM
   Orders o
```

Die Case-Anwesiung ist erst ab Version 1,5 implementiert.

4.3.2 CASE, Oracle

Ermöglichen bedingte Abfragen in der Form einer
IF-THEN-ELSE-Anweisung:

```
CASE expr WHEN comparison_expr1 THEN return_expr1
      [WHEN comparison_expr2 THEN return_expr2
       WHEN comparison_exprn THEN return_exprn
       ELSE else_expr]
END
```

4.3.3 1. Beispiel:

```
SELECT    ID,    Description,      CASE Status
                                       WHEN 1 THEN 'confirmed'
                                       WHEN 2 THEN 'in production'
                                       WHEN 3 THEN 'ready'
                                       WHEN 4 THEN 'shipped'
                                       ELSE 'unknown status '" || o.Status || ""
                                  END
FROM Orders o;
```

4.3.4 2. Beispiel: searched

```
SELECT    o.ID,  o.Description,     CASE
                                       WHEN (o.Status IS NULL) THEN 'new'
                                       WHEN (o.Status = 1) THEN 'confirmed'
                                       WHEN (o.Status = 3) THEN 'in production'
                                       WHEN (o.Status = 4) THEN 'ready'
                                       WHEN (o.Status = 5) THEN 'shipped'
                                       ELSE 'unknown status '" || o.Status || ""
                                  END
FROM Orders o;
```

4.3.5 Oracle

4.3.5.1 Numerische Oracle Funktion

Funktion	Beschreibung

abs(a)	absoluter Wert von a
ceil(a)	kleinste ganze Zahl größer als a
floor(a)	größte ganze Zahl kleiner als a
mod(m,n)	m Modulo n (Rest von m geteilt durch n)
power(m,n)	m hoch n round(n[,m]) n auf m Stellen gerundet
sign(a)	Vorzeichen von a (0, 1 oder -1)
sin(a)	Sinus von a (weitere trigonometrische Funktionen verfügbar)
sqrt(a)	Wurzel aus a trunc(a[,m]) a auf m Stellen abgeschnitten
exp(n)	liefert e hoch n (e=2,17828...)
ln(n)	natürlicher Algorithmus von zu e log(m,n) Logarithmus von n zu Basis m

4.3.5.2 DECODE, Oracle

Ermöglicht bedingte Abfragen in der Form einer CASE oder IF-THEN-ELSE-Anweisung:

```
DECODE(col|expression, search1, result1
                [, search2, result2,...,]
                [, default])
```

```
SELECT last_name, job_id, salary,
    DECODE(job_id, 'IT_PROG',  1.10*salary,
                'ST_CLERK', 1.15*salary,
                'SA_REP',   1.20*salary,
            salary)
    REVISED_SALARY
FROM   employees;
```

Zeigen Sie den anwendbaren Steuersatz für jeden Angestellten der Abteilung 80 an.

```
SELECT last_name, salary,
    DECODE (TRUNC(salary/2000, 0),
                0, 0.00,
                1, 0.09,
                2, 0.20,
                3, 0.30,
                4, 0.40,
                5, 0.42,
                6, 0.44,
                  0.45) TAX_RATE
FROM   employees
WHERE  department_id = 80;
```

4.3.5.3 DESCRIBE, Oracle

Mit Hilfe des Befehls DESCRIBE kann
- die Spaltendefinition einer Tabelle oder einer View (auch über ein Synonym)
- die Definition einer PL/SQL-Prozedur oder -Funktion

angezeigt werden.

Beispiel:

SQL> describe emp

Name	NOT NULL ?	Typ
EMPNO	NOT NULL	INTEGER
ENAME		CHAR(50)
JOB		CHAR(50)
MGR		INTEGER
HIREDATE		DATE
SAL		NUMERIC(7,2)
COMM		NUMERIC(7,2)
DEPTNO	NOT NULL	INTEGER

4.3.6 Firebird

Die folgende Tabelle gibt einen Überblick über alle vorhandenen Funktionen. Diese sind aber meistens in einer DLL implementiert. Diese muss dann über ein SQL-Statement installiert werden.

Vorgehensweise:

- Im Fester den Inhalt der Datei ib_udf.sql bzw. udf.sql laden (Zwischenablage)
- Starten mit STRG+E

4.3.6.1 Numerische Firebird-Funktionen:

Funktion	Beschreibung
abs	Liefert den Absolutwert eines Zahlenwertes
acos	Liefert den inversen Consinus eines Zahlenwertes
asin	Liefert den inversen Sinus eines Zahlenwertes
atan	Liefert den inversen Tangens eines Zahlenwertes
atan2	Liefert den Tangens eines Zahlenwertes
bin_and	Binäre Verknüpfung zweier ganzer Zahlen mit der AND-Operation (Bitweise)
bin_or	Binäre Verknüpfung zweier ganzer Zahlen mit der OR-Operation (Bitweise)
bin_xor	Binäre Verknüpfung zweier ganzer Zahlen mit der XOR-Operation (Bitweise)
ceiling	Liefert eine ganze Zahl größer gleich dem Parameter. Returnwert ist eine Fließkommazahl

cos	Liefert den Cosinus eines Zahlenwertes
cosh	Liefert den Hyperbolic Cosinus eines Zahlenwertes
cot	Liefert den 1/Tangens eines Zahlenwertes
div	Liefert den ganzzahligen Anteil einer Division
floor	Liefert eine ganze Zahl kleiner gleich dem Parameter. Returnwert ist eine Fließkommazahl
ln	Liefert den natürlichen Logarithmus eines Zahlenwertes
log	Liefert den Logarithmus eines Zahlenwertes zur Basis n
log10	Liefert den dekadischen Logarithmus eines Zahlenwertes
mod	Berechnet den Rest einer Division
pi	Liefert die Kreiszahl Pi
rand	Liefert eine Zufallszahl zwischen 0 und 1
sign	Liefert das Vorzeichen eines Zahlenwertes
sin	Liefert den Sinus eines Zahlenwertes
sinh	Liefert den Hyperbolic Sinus eines Zahlenwertes
sqrt	Liefert die Quadratwurzel eines Zahlenwertes
tan	Liefert den Tangens eines Zahlenwertes
tanh	Liefert den Hyperbolic Tangens eines Zahlenwertes

Hinweis:

Eingaben für Trigonometrische Funktionen sind im Bogenmaß einzugeben.

Beispiele:

- abs

```
SELECT abs(-33.44)
FROM employee
```

liefert den Wert 33.44

- acos

```
SELECT acos(0.454)
FROM employee
```

liefert den Wert 1,099546766

- asin

```
SELECT asin(0.454)
FROM employee
```

liefert den Wert 0,47124956

- atan

SELECT atan(0.454)
FROM employee

liefert den Wert 0,426175345

- atan2

SELECT atan2(0.454, 3)
FROM employee

liefert den Wert 0,150193685

- bin_and

SELECT bin_and(14,22)
FROM employee

liefert den Wert 6.

Hier wird bitweise die AND-Operation ausgeführt.

Dezimale Zahl	**Binäre Darstellung**
14	0 1110
22	1 0110
14 AND 22	0 0110

- bin_or

SELECT bin_or(14,22)
FROM employee

liefert den Wert 30.

Hier wird bitweise die AND-Operation ausgeführt.

Dezimale Zahl	**Binäre Darstellung**
14	0 1110
22	1 0110
14 OR 22	1 1110

- bin_xor

SELECT bin_xor(14,22)
FROM employee

liefert den Wert 24.

Hier wird bitweise die AND-Operation ausgeführt.

Dezimale Zahl	**Binäre Darstellung**
14	0 1110
22	1 0110
14 XOR 22	1 1000

Überall wo sich die Bits unterscheiden, wird eine Eins eingetragen.

- ceiling

Abrunden auf die nächste größere Ganze Zahl.

SELECT ceiling(12.454)
FROM employee

liefert den Wert 13

SELECT ceiling(-12.454)
FROM employee

liefert den Wert -12

- cos

SELECT cos(12.454)
FROM employee

liefert den Wert 0,993693063

- cosh

SELECT cosh(1.454)
FROM employee

liefert den Wert 2,256917502

- cot

SELECT cot(1.454)
FROM employee

liefert den Wert 0,117330329

- div

Liefert den ganzzahligen Anteil einer Division.

```
SELECT div(15,4)
FROM employee
```

liefert den Wert 3

```
SELECT div(15,6)
FROM employee
```

liefert den Wert 2

```
SELECT div(-15,6)
FROM employee
```

liefert den Wert -2

- floor

Abrunden auf die nächste kleinere Ganze Zahl.

```
SELECT floor(12.454)
FROM employee
```

liefert den Wert 12

```
SELECT floor(-12.454)
FROM employee
```

liefert den Wert -13

- ln

```
SELECT ln(134.66)
FROM employee
```

liefert den Wert 4,902753083

- log

```
SELECT log(100,10)
FROM employee
```

liefert den Wert 2, da $10^2 = 100$

```
SELECT log(8,2)
FROM employee
```

liefert den Wert 3

```
SELECT log(123.456,10)
FROM employee
```

liefert den Wert 2,091512202

- log10

```
SELECT log10(123.456)
FROM employee
```

liefert den Wert 2,091512202

- mod

Liefert den Rest einer Division.

```
SELECT mod(15,4)
FROM employee
```

liefert den Wert 3

```
SELECT mod(-15,4)
FROM employee
```

liefert den Wert -3

- pi

```
SELECT pi()
FROM employee
```

liefert den Wert 3,141592654

- rand

```
SELECT rand()
FROM employee
```

liefert den Wert 0,872005371

Dieser „Zufallswert“ wird aus der aktuellen Zeit ermittelt. Sehr unzuverlässig!

- sign

Liefert 1 bei positiven Zahlen
Liefert 0 bei Null
Liefert -1 bei negativen Zahlen

SELECT sign(2.345)
FROM employee

liefert den Wert 1

SELECT sign(-2.345)
FROM employee

liefert den Wert -1

- sin

SELECT sin(92.345)
FROM employee

liefert den Wert -0,945397815

SELECT sin(90)
FROM employee

liefert den Wert 0,893996664

SELECT sin(90*PI()/180)
FROM employee

liefert den Wert 1

Bitte beachten: Eingaben sind im Bogenmaß einzugeben.

- sinh

SELECT sinh(90*PI()/180)
FROM employee

liefert den Wert 2,301298902

SELECT sinh(90)
FROM employee

liefert den Wert 6,10202E+38

Bitte beachten: Eingaben sind im Bogenmaß einzugeben.

- sqrt

SELECT sqrt(123)
FROM employee

liefert den Wert 11,0905365064094

- tan

SELECT tan(0.566)
FROM employee

liefert den Wert 0,635339591

- tanh

SELECT tanh(0.566)
FROM employee

liefert den Wert 0,512415606

4.4 Konvertierungsfunktionen in Oracle

4.4.1 Implizite Datentypkonvertierung

Beim Auswerten von Ausdrücken kann der Oracle-Server automatisch folgende Datentypen konvertieren:

Von	**Nach**
VARCHAR2	NUMBER
VARCHAR2	DATE
CHAR	DATE

Dabei muss aber sichergestellt sein, dass die zu konvertierenden Zeichen einen gültigen Ausdruck darstellen.

4.4.2 Explizite Datentypkonvertierung

SQL stellt drei Funktionen bereit, um einen Wert von einem Datentyp in einem anderen zu konvertieren.

FUNKTION	**ZWECK**
TO_CHAR(number | date [, *format*] [, *nlsparams*]	Konvertiert einen numerischen Wert oder Datumswert in eine VARCHAR2-Zeichenfolge nach der Formatmaske *format.* **Konvertierung von numerischen Werten:** Der Parameter *nlsparams* gibt folgende nähere Spezifikation für die Konvertierung: • Dezimalzeichen • Gruppentrennzeichen • Lokales Währungssymbol • Internationales Währungssymbol Wird *nlsparams* nicht übergeben, wird der Default-Wert benutzt. **Konvertierung von Datumswerten:** Der Parameter *nlsparams* legt die Sprache fest, in der die Namen der Monate und Tage sowie deren Abkürzungen zurückgegeben werden. Wird *nlsparams* nicht übergeben, wird der Default-Wert des Systems benutzt.
TO_NUMBER(char [, *format*] [, *nlsparams*]	Konvertiert eine Zeichenfolge, die aus gültigen Ziffern besteht, in eine Zahl im Format, das durch die optionale Formatmaske format festgelegt wurde. Der Parameter *nlsparams* gibt folgende nähere Spezifikation für die Konvertierung: • Dezimalzeichen • Gruppentrennzeichen • Lokales Währungssymbol • Internationales Währungssymbol Wird *nlsparams* nicht übergeben, wird der Default-Wert benutzt.
TO_DATE(char [, *format*] [, *nlsparams*]	Konvertiert eine Zeichenfolge, die ein gültiges Datum enthält in einen Datumswert im Format der Formatmaske *format*. Wird *format* nicht angegeben, ist das Format DD-MM-YY.

Richtlinien:

- Die Formatmaske muss in Hochkommata angegeben werden. Groß- und Kleinschreibung wird beachtet.
- Die Formatmaske kann alle gültigen Elemente des Datumsformats enthalten. Trennen Sie den Datumswert durch ein Komma von der Formatmaske.
- Die Namen der Tage und Monate in der Ausgabe werden automatisch mit Leerzeichen aufgefüllt.

- Um die füllenden Leerzeichen zu löschen odervorgestellte Nullen zu unterdrücken, verwenden Sie das Füllmodus-Elemente *fm*.
- Sie können das Ergebniszeichenfeld mit dem Befehl COLUMN formatieren.

Beispiele:

```
SELECT emp_no, TO_CHAR(hire_date, "MM/YY") Anfangsmonat
FROM employee
WHERE last_name = "Yamamoto"
```

EMP_NO	ANFANGSMONAT
118	07/93

Elemente der Datumsformatmaske:

Abkürzung	**Beschreibung**
YYYY	Komplettes Jahr als Zahl
YEAR	Das Jahr in Worten
MM	Zweistelliger Zahlenwert für den Monat
MONTH	Vollständiger Name des Monats
DY	Abkürzung aus drei Buchstaben für den Wochentag
DAY	Vollständiger Name des Tags
DD	Tag des Monats als numerischer Wert

Beispiele gültiger Datumsformate:

Element	**Beschreibung**
SCC oder CC	Jahrhundert; der Server verwendet las Präfix für vorchristliche Datumswerte ein Minuszeichen (-)
Jahre in Datumswerte YYYY oder SYYYY	Jahr; der Server verwendet las Präfix für vorchristliche Datumswerte ein Minuszeichen (-).
YYY oder YY oder Y	Die letzten drei oder zwei Ziffernbzw. Die letzte Ziffer der Jahreszahl
Y,YYY	Jahreszahl mit einem Komma an dieser Stelle
IYYY, IYY, IY, I	Vier, drei, zwei oder eine Ziffer(n), basierend auf dem ISO-Standard.
SYEAR oder YEAR	Jahr in Worten; der Server verwendet als Präfix für vorchristliche Datumswerte ein Minuszeichen (-).
BC oder AD	Angabe von BC (v. Chr.) AD (n. Chr)
B.C. oder A.D.	Angabe von B.C. (v. Chr.) A.D. (n. Chr) mit Punkten
Q	Jahresquartal
MM	Zweistellige Monatszahl (eventuell mit Nullen)
MONTH	Name des Monats, bis zur Länge von neun Zeichen mit Leerzeichen aufgefüllt.
MON	Name des Monats, Abkürzung mit drei Buchstaben
RM	Monat als römische Zahl
WW oder W	Kalenderwoche des Jahres oder Woche des Monats
DDD oder DD oder D	Tag des Jahres, des Monats oder der Woche als Zahlenwert
DAY	Name des Tages, bis zur Länge von neun Zeichen mit Leerzeichen aufgefüllt.
DY	Name des Tages, Abkürzung mit drei Buchstaben
J	Tag nach dem julianischen Kalender; die Anzahl seit dem 31. Dezember 4713 v. Chr.

Elemente der Uhrzeitformate:

Abkürzung	Beschreibung
AM oder PM	Vormittag, Nachmittag
A.M. oder P.M.	Vormittag, Nachmittag (mit Punkten)
HH, HH12 oder HH24	Stunde des Tages, Stunde(1-12), Stunde(0-23)
MI	Minute (0-59)
SS	Sekunde (0-59)
SSSS	Sekunden seit Mitternacht (0-86399)

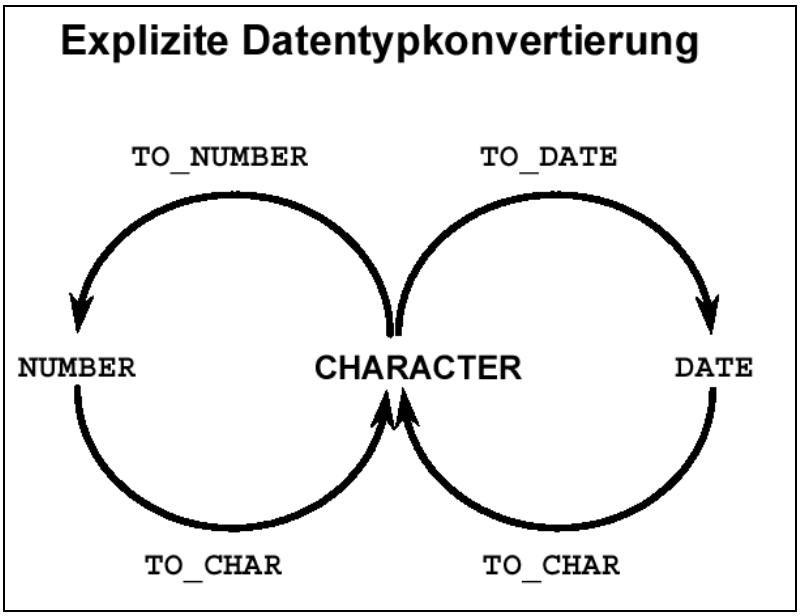

Abbildung 8 Explizite Datentypkonvertierung

4.5 Verwendung von Nullwerten in Oracle

Bei Verwendung von Nullwerten bietet Oracle folgende Funktionen zur Unterstützung an:

- NVL(expr1, expr2)
- NVL2(expr1, expr2, expr3)
- NULLIF(expr1, expr2)
- COALESCE(expr1, expr2, expr3, ... exprn)

Abkürzung	Beschreibung
NVL	Konvertiert einen Nullwert in einen konkreten, vorgegeben Wert
NVL2	Wenn der Ausdruck *expr1* kein Nullwert ist, gibt NVL2 *expr2* zurück. Wenn *expr1* ein Nullwert ist, gibt NVL2 *expr3* zurück. Das Argument *expr1* kann einen beliebigen Datentyp enthalten.
NULLIF	Vergleicht zwei Ausdrücke und gibt einen NULL-WERT zurück, wenn sie übereinstimmen, bzw. den ersten Ausdruck, wenn sie nicht übereinstimmen.
COALESCE	Gibt den ersten Wert der Ausdruckliste zurück, der kein NULL-Wert ist.

Beispiele:

- Gib alle Mitarbeiternummer und Abteilungsnummer aus. Wenn ein Wert NULL ist, dann setze die Abteilungsnummer auf Null.

```
SELECT emp_no, NVL(dept_no,0)
FROM employee
```

- Funktion NVL2

```
SELECT last_name, salary, commision_pct,
        NVL2(commision_pct, 'SAL-COMM', 'SAL') Einkommen
FROM    employee
WHERE department_id IN (50,80);
```

- Funktion NULLIF

```
SELECT first_name, LENGTH(first_name) "expr1",
        last_name, LENGTH(last_name) "expr2",
        NULLIF(LENGTH(first_name), LENGTH(last_name)) result
FROM    employees;
```

Im dargestellten Beispiel wird die Job-Kennung in der Tabelle EMPLOYEES mit der Job-Kennung in der Tabelle JOB_HISTORY für alle Angestellten verglichen, die in beiden Tabellen enthalten sind. Die Ausgabe zeigt die aktuelle Job-Kennung des Angestellten. Wenn der Angestellte mehrere Male aufgelistet ist, waren dem Angestellten in der Vergangenheit mindestens zwei andere Job-Kennungen zugeordnet.

Der Vorteil der Funktion COALESCE gegenüber der Funktion NVL besteht darin, dass Sie mit der Funktion COALESCE mehrere alternative Werte angeben können.
Wenn der erste Ausdruck kein NULL-Wert ist, wird dieser Ausdruck zurückgegeben. Ansonsten werden die übrigen Ausdrücke mit COALESCE zusammengeführt.

```
SELECT   last_name,
      COALESCE(commission_pct, salary, 10) comm
FROM     employees
ORDER BY commission_pct;
```

4.6 Verwendung von Nullwerten in Firebird

Bei Verwendung von Nullwerten bietet Firebird ab der Version 1,5 folgende Funktionen zur Unterstützung an:

- COALESCE(expr1, Ersetzung)

Beispiele:

- Gib alle Mitarbeiternummer und Abteilungsnummer aus. Wenn ein Wert NULL ist, dann setze die Abteilungsnummer auf Null.

```
SELECT emp_no, COALESCE(dept_no,0)
FROM employee
```

4.7 Zusatzfunktionen

Converts a column from one datatype to another. Available in SQL, DSQL, and isql.

Syntax CAST (value AS datatype)

Argument Description
val A column, constant, or expression; in SQL, val can also be a host-language variable, function, or UDF
datatype Datatype to which to convert

Beispiel:

```
select CAST(plz AS INTEGER)
from emp
```

Description CAST() allows mixing of numerics and characters in a single expression by converting val to a specified datatype.

Normally, only similar datatypes can be compared in search conditions. CAST() can be used in search conditions to translate one datatype into another for comparison purposes.
Datatypes can be converted as shown in the following table:

TABLE 2 Compatible datatypes for CAST()

From datatype class	To datatype class
Numeric	character, varying character, date, time, timestamp
Character, varying character	numeric, date, time, timestamp
Date	character, varying character, timestamp
Time	character, varying character, timestamp
Timestamp	character, varying character, date, time
Blob, arrays	—

An error results if a given datatype cannot be converted into the datatype specified in CAST().

Example In the following WHERE clause, CAST() is used to translate a CHARACTER datatype, INTERVIEW_DATE, to a DATE datatype to compare against a DATE datatype, HIRE_DATE:

```
. . .
    WHERE HIRE_DATE = CAST (INTERVIEW_DATE AS DATE);
```

4.8 CASE-Anweisung

```
SELECT last_name, job_code, salary,
       CASE job_code
          WHEN "VP"   THEN  (1.10*salary)
          WHEN "Enf"   THEN  (2.10*salary)
       ELSE salary
       END
FROM employee
```

4.9 Übungen

4.9.1 Teuerste Rechnung

Gesucht ist der größte Betrag aller offenen Rechnungen des Kunden mit der Nummer 1005.

Das Ergebnis lautet: 9000,00

4.9.2 Durchschnittsverdienst in den USA

Gesucht ist der Durchschnittsverdienst aller Mitarbeiter in den USA. Dabei sind Sie sich nicht sicher, dass in allen Einträgen der Text USA steht. Eventuell existieren Varianten der Groß- und Kleinschreibung.

Das Ergebnis lautet: 71068,9185606061

4.9.3 Namen der Manager

Gesucht sind die Namen der Manager.

Ergebnis:

FULL_NAME
Forest, Phil
Young, Katherine
Papadopoulos, Chris
Williams, Randy

4.9.4 Anzahl der Manager

Gesucht ist die Anzahl der Manager.

Das Ergebnis lautet: 4

4.9.5 Welcher Mitarbeiter wurde im Mai eingestellt

Gesucht sind die Mitarbeiter, die im Mai eingestellt wurden. Ausgegeben werden sollen die Namen und das Einstellungsdatum.

Ergebnis:

FULL_NAME	**HIRE_DATE**
Lee, Terri	01.05.1990
Guckenheimer, Mark	02.05.1994

4.9.6 Liste mit neuem Datumsformat

Gesucht ist eine Liste, in der alle Mitarbeiter mit ihren Einstellungsdaten eingetragen sind. Folgendes Datumsformat soll verwendet werden: YYYY:MM:TT. Als Überschriften sollen die Bezeichnungen „Name“ und „Einstellungsdatum“ dienen. Sortiert werden soll nach der Mitarbeiternummer.

Ergebnis:

NAME	**EINSTELLUNGSDATUM**
Nelson, Robert	1988:12:28
Young, Bruce	1988:12:28
Lambert, Kim	1989:2:6
Johnson, Leslie	1989:4:5
Forest, Phil	1989:4:17
Weston, K. J.	1990:1:17
Lee, Terri	1990:5:1
Hall, Stewart	1990:6:4
Young, Katherine	1990:6:14
Papadopoulos, Chris	1990:1:1
Fisher, Pete	1990:9:12
Bennet, Ann	1991:2:1
De Souza, Roger	1991:2:18
Baldwin, Janet	1991:3:21
Reeves, Roger	1991:4:25
Stansbury, Willie	1991:4:25
Phong, Leslie	1991:6:3
Ramanathan, Ashok	1991:8:1
Steadman, Walter	1991:8:9
Nordstrom, Carol	1991:10:2
Leung, Luke	1992:2:18

O'Brien, Sue Anne	1992:3:23
Burbank, Jennifer M.	1992:4:15
Sutherland, Claudia	1992:4:20
Bishop, Dana	1992:6:1
MacDonald, Mary S.	1992:6:1
Williams, Randy	1992:8:8
Bender, Oliver H.	1992:10:8
Cook, Kevin	1993:2:1
Brown, Kelly	1993:2:4
Ichida, Yuki	1993:2:4
Page, Mary	1993:4:12
Parker, Bill	1993:6:1
Yamamoto, Takashi	1993:7:1
Ferrari, Roberto	1993:7:12
Yanowski, Michael	1993:8:9
Glon, Jacques	1993:8:23
Johnson, Scott	1993:9:13
Green, T.J.	1993:11:1
Osborne, Pierre	1994:1:3
Montgomery, John	1994:3:30
Guckenheimer, Mark	1994:5:2

- Gesucht die Mitarbeiternamen und der jeweilige Tag, Monat, Jahr des Einstellungstages. Diese numerischen Werte sollen in jeweils eine eigene Spalte eingetragen werden. Wichtig sind aber nur die Mitarbeiter, die in Jahren 1992 und 1994 angefangen haben.

Dazu verwendet man das Attribut hire_date mit den drei EXTRACT-Methoden.

Abfrage:

```
SELECT    full_name Name, EXTRACT(day FROM hire_date) Tag,
          EXTRACT(month FROM hire_date) Monat,
          EXTRACT(year FROM hire_date) Jahr,
          hire_date
FROM employee
```

Ergebnis:

NAME	Tag	MONAT	JAHR	HIRE_DATE
Nelson, Robert	28	12	1988	28.12.1988
Young, Bruce	28	12	1988	28.12.1988
Lambert, Kim	6	2	1989	06.02.1989
Johnson, Leslie	5	4	1989	05.04.1989
Forest, Phil	17	4	1989	17.04.1989
Weston, K. J.	17	1	1990	17.01.1990
Lee, Terri	1	5	1990	01.05.1990
Hall, Stewart	4	6	1990	04.06.1990
Young, Katherine	14	6	1990	14.06.1990
Papadopoulos, Chris	1	1	1990	01.01.1990
Fisher, Pete	12	9	1990	12.09.1990
Bennet, Ann	1	2	1991	01.02.1991

De Souza, Roger	18	2	1991	18.02.1991
Baldwin, Janet	21	3	1991	21.03.1991
Reeves, Roger	25	4	1991	25.04.1991
Stansbury, Willie	25	4	1991	25.04.1991
Phong, Leslie	3	6	1991	03.06.1991
Ramanathan, Ashok	1	8	1991	01.08.1991
Steadman, Walter	9	8	1991	09.08.1991
Nordstrom, Carol	2	10	1991	02.10.1991
Leung, Luke	18	2	1992	18.02.1992
O'Brien, Sue Anne	23	3	1992	23.03.1992
Burbank, Jennifer M.	15	4	1992	15.04.1992
Sutherland, Claudia	20	4	1992	20.04.1992
Bishop, Dana	1	6	1992	01.06.1992
MacDonald, Mary S.	1	6	1992	01.06.1992
Williams, Randy	8	8	1992	08.08.1992
Bender, Oliver H.	8	10	1992	08.10.1992
Cook, Kevin	1	2	1993	01.02.1993
Brown, Kelly	4	2	1993	04.02.1993
Ichida, Yuki	4	2	1993	04.02.1993
Page, Mary	12	4	1993	12.04.1993
Parker, Bill	1	6	1993	01.06.1993
Yamamoto, Takashi	1	7	1993	01.07.1993
Ferrari, Roberto	12	7	1993	12.07.1993
Yanowski, Michael	9	8	1993	09.08.1993
Glon, Jacques	23	8	1993	23.08.1993
Johnson, Scott	13	9	1993	13.09.1993
Green, T.J.	1	11	1993	01.11.1993
Osborne, Pierre	3	1	1994	03.01.1994
Montgomery, John	30	3	1994	30.03.1994
Guckenheimer, Mark	2	5	1994	02.05.1994

Diese Aufgabe ist noch nicht vollständig gelöst. Die WHERE-Bedingung muss noch eingearbeitet werden. Dazu kann aber die EXTRACT-Funktion verwendet werden. In der Tabelle kann überprüfen, ob diese Funktion ihren Zweck erfüllt, da das Attribut hire_date mit ausgegeben wurde.

Neue Abfrage:

```
SELECT    full_name Name, EXTRACT(day FROM hire_date) Tag,
          EXTRACT(month FROM hire_date) Monat,
          EXTRACT(year FROM hire_date) Jahr,
          hire_date
FROM employee
WHERE EXTRACT(year FROM hire_date) IN (1992,1994)
```

Ergebnis:

NAME	**TAG**	**MONAT**	**JAHR**	**HIRE_DATE**
Leung, Luke	18	2	1992	18.02.1992
O'Brien, Sue Anne	23	3	1992	23.03.1992
Burbank, Jennifer M.	15	4	1992	15.04.1992
Sutherland, Claudia	20	4	1992	20.04.1992
Bishop, Dana	1	6	1992	01.06.1992
MacDonald, Mary S.	1	6	1992	01.06.1992

Williams, Randy	8	8	1992	08.08.1992
Bender, Oliver H.	8	10	1992	08.10.1992
Osborne, Pierre	3	1	1994	03.01.1994
Montgomery, John	30	3	1994	30.03.1994
Guckenheimer, Mark	2	5	1994	02.05.1994

5 Gruppenfunktionen

Was sind Gruppenfunktionen?

Gruppenfunktionen werden auf Gruppen von Zeilen angewendet und geben ein Ergebnis pro Gruppe zurück.

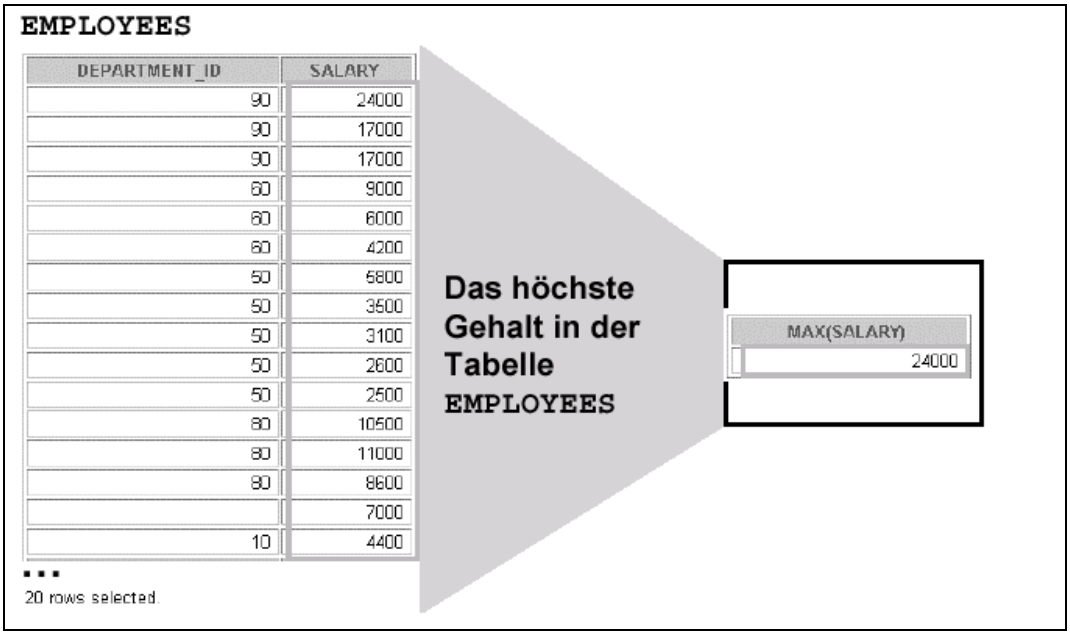

Abbildung 9 Gruppenfunktionen mit der Tabelle Employee

Die Gruppenfunktionen arbeiten mit Gruppen von Zeilen und geben ein Ergebnis pro Gruppe zurück. Bei diesen Gruppen kann es sich um ganze die Tabelle oder einzelne Gruppen der Tabelle handeln.

Funktion	Beschreibung
AVG([DISTINCT | ALL] n)	Berechnet den Durchschnittswert von n-Zahlen. Eventuelle NULL-Werte werden ignoriert.
COUNT({ * | [DISTINCT | ALL] Ausdruck)	Anzahl der Zeilen, für die Ausdruck auf einen anderen Wert als einen NULL-Wert ausgewertet wird. Alle mit * ausgewählten Zeilen zählen, einschließlich mehrfach vorhandener Zeilen und Zeilen mit NULL-Werten.
MAX([DISTINCT | ALL] n)	Größter Wert von n Werten. NULL-Werte werden ignoriert.
MIN([DISTINCT | ALL] n)	Kleinster Wert von n Werten. NULL-Werte werden ignoriert.
STDDEV [DISTINCT | ALL] n)	Standardabweichung von n Werten. NULL-Werte werden ignoriert.
SUM([DISTINCT | ALL] n)	Summe von n Werten. NULL-Werte werden ignoriert.
VARIANCE([DISTINCT | ALL] n)	Varianz von n Werten. NULL-Werte werden ignoriert.

5.1.1 Syntax der GROUP BY-Klausel

```
SELECT      column, group_function(column)
FROM        table
[ WHERE     Condition ]
[ GROUP BY group_by_expression ]
[ ORDER BY column  ]
```

Man kann die GROUP BY-Klausel verwenden, um die Zeilen einer Tabelle in Gruppen aufzuteilen. Anschließend kann man die normalen Gruppenfunktionen verwenden, um Aggregatinformationen für jede Gruppe zurückzugeben.

Für die Syntax gilt:

group_by_expression gibt die Spalten, deren aktuellen Werte die Basis für die Gruppierung der Zeilen darstellen.

5.1.2 Richtlinien für die Verwendung von Gruppenfunktionen

- DISTINCT bewirkt, dass die Funktion keine doppelten Werte berücksichtigt.
- ALL bewirkt, dass alle Werte, einschließlich der doppelten Werte, berücksichtigt werden. Der Daultwert ist ALL und muss daher nicht angegeben werden.
- Die Datentypen für Funktionen mit einem expr-Argument können CHAR-VARCHAR2, NUMBER oder DATE sein.
- Alle Gruppenfunktionen ignorieren NULL-Werte. Um NULL-Werte durch einen Wert zu erstzen, verwenden man die Funktioen NVL, NVL2, oder COALESCE.
- Wenn man eine GROUP BY-Klausel verwendet, sortiert der Datenbank-Server die Ergebnismenge implizit in aufsteigender Reihenfolge. Um diese Default-Sortierung außer Kraft zu setzen, verwenden Sie DESC in einer ORDER BY-Klausel.
- Wenn Sie eine Gruppenfunktion in einer SELECT-Klausel angeben, können Sie nicht gleichzeitig einzelne Ergebnisse auswählen, es sei denn, die einzelne Spalte wird in der GROUP BY-Klausel angegeben. Man erhält eine Fehlermeldung, wenn die Spaltenliste nicht in der GROUP BY-Klausel angegeben wurde.
- Mit einer WHERE-Klausel kann man Zeilen ausschließen, bevor die übrigen Zeilen in Gruppen aufgeteilt werden.
- Man muss die Spalten in der GROUP BY-Klausel angeben.
- Man kann dabei keine Spalten-Aliasnamen verwenden.
- Standardmäßig werden die Zeilen in aufsteigender Reihenfolge nach der in der GROUP BY-Liste angegebenen Spalten sortiert. Diese Sortierung kann man mit der ORDER BY-Klausel verändern.

Richtlinien:

Bei Verwendung der GROUP BY-Klausel, muss man sicher stellen, dass alle Spalten in der SELECT-Liste, die nicht in Gruppenfunktionen (z. B. AVG) enthalten sind, in der GROUP BY-Klausel angegeben werden.

Beispiel:

```
SELECT dept_no, AVG(salary)
FROM employee
GROUP BY dept_no
```

Ergebnis:

DEPT_NO	AVG
000	133321,5
100	77631,25
110	65221,40625
115	6740000
120	31926,5625
121	110000
123	390500
125	99000000
130	94521,46875
140	100914
180	53688,75
600	66450
621	69184,875
622	53409,16667
623	57551,65
670	71268,75
671	73155,0625
672	45647,5
900	92791,3125

Das Beispiel zeigt die Abteilungsnummer und das Durchschnittsgehalt für jede Abteilung an. Diese SELECT-Anweisung, die eine GROUP BY-Klausel enthält, wird folgender maßen ausgewertet.

- Die SELECT-Klausel gibt die abzurufenden Spalten an:
  - Die Spalte Dept_no aus der Tabelle
  - Den Durchschnitt aller Gehälter für die in der GROUP BY-Klausel angegebene Gruppe
- Die FROM-Klausel gibt die Tabellen an, auf die die Datenbank zugreift, die Tabelle employee
- Die WHERE-Klausel gibt die abzurufenden Zeilen an. Da in diesem Beispiel keine WHERE-Klausel existiert, werden standardmäßig alle Zeilen abgerufen.
- Die GROUP BY-Klausel gibt an, wie die Zeilen gruppiert werden. Die Zeilen werden nach Abteilungsnummer gruppiert, so dass die uaf die Spalte salary angewandte Funktion AVG das *durchschnittliche Gehalt* für jede Abteilung berechnet.

5.1.3 SUM-Funktion

Die SUM-Funktion berechnet die Summe eines oder mehrerer Attribute.

- Gesucht ist die Summe aller Gehälter

```
SELECT sum(salary)
FROM employee
```

Ergebnis:115522468

Das nächste Beispiel zeigt die Benutzung mehrerer Attribute:

- Gesucht ist die Summe aller Rechnungen, die Summe der Rabatte und deren Differenz.

```
SELECT    SUM(total_value), SUM (total_value*DISCOUNT)
          SUM (total_value- total_value*DISCOUNT),

FROM sales
```

Die Summe berechnet die Summe aller Rechungen. In der zweiten Summe wird der Gesamtrabatt berechnet, der in der dritten von der ersten Summe abgezogen wird.

SUM	**SUM_1**	**SUM_2**
2250591,03	182994,9654	2067596,065

5.1.4 AVG-Funktion

Die AVG-Funktion berechnet den Durchschnitt von Werten.

- Gesucht ist der durchschnittliche Verkaufsbetrag

```
SELECT AVG(total_value)
FROM sales
```

liefert den durchschnittlichen Verkaufsbetrag: 68199,73 €

```
SELECT    AVG(salary)
FROM employee
WHERE dept_no=623
```

liefert das durchschnittlichen Gehalt in der Abteilung 623: 57551,65

5.1.5 MIN-Funktion

Die MIN-Funktion sucht das Minimum einer Spalte. Hier können natürlich auch Bedingungen angegeben werden.

- Gesucht ist der minimale offene Rechnungsbetrag

```
SELECT    MIN(total_value)
FROM sales
WHERE order_status="open"
```

liefert den Wert 2693 (siehe Tabelle unten)

TOTAL_VALUE
450000,49
5980

9000
60000
3399,15
3999,99
16000
2693

5.1.6 MAX-Funktion

Die MAX-Funktion sucht das Maximum einer Spalte. Hier können natürlich auch Bedingungen angegeben werden.

- Gesucht ist der maximale offene Rechnungsbetrag

```
SELECT     MAX(total_value)
FROM sales
WHERE order_status="open"
```

liefert den Wert 450000,49 (siehe Tabelle oben)

5.1.7 COUNT-Funktion

Die COUNT-Funktion zählt die Tupel in einer Tabelle. Hier können natürlich auch Bedingungen angegeben werden, die die Anzahl einschänken.

- Bestimme die Anzahl aller Mitarbeiter

```
SELECT     COUNT(emp_no)
FROM employee
```

liefert den Wert 42

- Bestimme die Anzahl aller Mitarbeiter in den Abteilungen mit der Abteilungsnummer größer 600

```
SELECT     COUNT(emp_no)
FROM employee
WHERE dept_no > 600
```

liefert den Wert 21

- Bestimme die Anzahl aller Manager

```
SELECT     COUNT(emp_no)
FROM employee
WHERE job_code = "Mngr"
```

liefert den Wert 4

oder auch

```
SELECT      COUNT(emp_no)
FROM employee
WHERE lower(job_code) = "mngr"
```

Oder auch:

```
SELECT count(*)
FROM employee
WHERE lower(job_code) = "mngr"
```

Hier wird der Datenwert in Kleinbuchstaben umgewandelt.

5.1.8 STDDEV-Funktion (Oracle)

Die STDEV-Funktion berechnet die Standardabweichung von Werten. Dabei werden Null-Werte ignoriert.

- Gesucht ist die Standardabweichung der Gehälter

```
SELECT STDEV(salary) AS STDEV, STDEVP(salary) AS STDEVP
FROM emp;
```

Ergebnis:

STDEV	**STDEVP**
15282372,99	15099343,96

STDEV: $\sqrt{\frac{1}{n-1}(x_i - \bar{x})^2}$

STDEVP: $\sqrt{\frac{1}{n}(x_i - \bar{x})^2}$

Beispiel mit den Daten: 2 4 5 11

Mittelwert: AVG=5,5

Zahl	$(x_i - \bar{x})^2$
2	12,25
4	2,25
5	0,25
11	30,25
∑	45,00

STDEV: $\sqrt{\frac{1}{4-1}45} = \sqrt{15} = 3{,}87298$

STDEVP: $\sqrt{\frac{1}{4}45} = \sqrt{11{,}25} = 3{,}354$

5.1.9 VARIANCE-Funktion (Oracle)

Die VAR-Funktion berechnet die Varianz von Werten. Dabei werden Null-Werte ignoriert.

- Gesucht ist die Varianz der Gehälter

```
SELECT VAR(salary) AS VAR_Salary, VARP(salary) AS VARP_Salary
FROM emp;
```

Ergebnis:

VAR_Salary	**VARP_Salary**
233550924406763	227990188111364

VAR: $\frac{1}{\boldsymbol{n}-1}\left(\boldsymbol{x_i}-\bar{\boldsymbol{x}}\right)^2$

VARP: $\frac{1}{\boldsymbol{n}}\left(\boldsymbol{x_i}-\bar{\boldsymbol{x}}\right)^2$

Beispiel mit den Daten: 2 4 5 11

Mittelwert: AVG=5,5

Zahl	$\left(\boldsymbol{x_i}-\bar{\boldsymbol{x}}\right)^2$
2	12,25
4	2,25
5	0,25
11	30,25
∑	45,00

VAR: $\frac{1}{4-1}45 = 15$

VARP: $\frac{1}{4}45 = 11{,}25$

5.1.10 Beispiel: Oracle

HAVING-Klausel (Fortsetzung)
Im Beispiel auf der Folie werden die Job-Kennung und das monatliche Gesamtgehalt für jede Job-Kennung mit einer Gesamtgehaltsliste von über $ 1.000 angezeigt. Das Beispiel schließt alle Jobs mit einem L aus und sortiert die Liste nach dem monatlichen Gesamtgehalt.

Ausgeschlossene JOB-Gruppen:
- CLERK
- SALESMAN
- ANALYST

```
SELECT   job, SUM(sal) GEHALT
FROM     emp
WHERE    job NOT LIKE '%L%'
GROUP BY job
HAVING   SUM(sal) > 1000
ORDER BY SUM(sal);
```

5.2 Übungen

5.2.1 Extrema der Gehälter

Zeigen Sie das niedrigste Gehalt, das höchste Gehalt, die Summe der Gehälter und das Durchschnittsgehalt für alle Mitarbeiter an. Nennen Sie die Spalten „Minimum“, „Maximum“, „Summe“ und „Durchschnitt“.

Ergebnis:

MINIMUM	MAXIMUM	SUMME	DURCHSCHNITT
22935	99000000	115522468	2750534,952

5.2.2 Extrema der Gehälter pro Job_Gruppe

Zeigen Sie das niedrigste Gehalt, das höchste Gehalt, die Summe der Gehälter und das Durchschnittsgehalt für alle Mitarbeiter an. Nennen Sie die Spalten „Minimum“, „Maximum“, „Summe“ und „Durchschnitt“. Gruppieren Sie die Mitarbeiter nach den Job_Code.

Ergebnis:

JOB_CODE	MINIMUM	MAXIMUM	SUMME	DURCHSCHNITT
Admin	22935	53793	135003	33750,75
CEO	212850	212850	212850	212850
CFO	116100	116100	116100	116100
Dir	111262,5	111262,5	111262,5	111262,5
Doc	60000	60000	60000	60000
Eng	32000	6000000	6829208,25	455280,55
Finan	69482,625	69482,625	69482,625	69482,625
Mktg	64635	64635	64635	64635
Mngr	56295	89655	288251,25	72062,8125
PRel	42742,5	42742,5	42742,5	42742,5
SRep	44000	99000000	107280511,9	13410063,99
Sales	33620,625	61637,8125	95258,4375	47629,21875
VP	105900	111262,5	217162,5	108581,25

5.2.3 Anzahl der Ingenieure

Ermitteln Sie die Anzahl der Ingenieure, ohne alle in einer Liste anzuzeigen. Geben Sie nur eine Zeile mit der Anzahl aus. Als Überschrift wählen Sie „Ingenieure“

Ergebnis:

Ingenieure
15

5.2.4 Statistik der Abteilungen

Erstellen Sie eine Abfrage, um für jede Abteilung die Abteilungsnummer, die Mitarbeiterzahl und das jeweilige Durchschnittsgehalt anzuzeigen. Als Überschrift wählen Sie „Abt_Nr“, „Anz_Ang“ und „Durchschnitts_Gehalt“.Sortiert werden soll absteigend nach dem Gehalt.

Ergebnis:

ABT_NR	ANZ_ANG	DURCHSCHNITTS_GEHALT
125	1	99000000
115	2	6740000
123	1	390500
000	2	133321,5

121	1	110000
140	1	100914
130	2	94521,46875
900	2	92791,3125
100	2	77631,25
671	3	73155,0625
670	2	71268,75
621	4	69184,875
600	2	66450
110	2	65221,40625
623	5	57551,65
180	2	53688,75
622	3	53409,16667
672	2	45647,5
120	3	31926,5625

5.2.5 Welche Personen haben ein größeres Gehalt als das Durchschnittsgehalt?

Abfrageergebnis:

LAST_NAME	FIRST_NAME	SALARY
Ichida	Yuki	6000000
Yamamoto	Takashi	7480000
Ferrari	Roberto	99000000

5.2.6 Mitarbeiterabfrage

Welche Mitarbeiter sind in der gleichen Abteilung wie der Mitarbeiter mit dem größten Gehalt?

Abfrageergebnis:

LAST_NAME	JOB_CODE	SALARY
Ferrari	Srep	99000000

5.2.7 Welche Abteilungen liegen über den Durchschnittsgehalt?

Das Durchschnittsgehalt wird über alle Mitarbeiter bestimmt.

Abfrageergebnis:

DEPT_NO	AVG
115	6740000
125	99000000

6 Unterabfragen

Unterabfragen werden verwendet, wenn zwei Abfragen kombiniert werden müssen, damit das Ergebnis ermitteln werden kann.

Beispiel:

Gesucht sind die Mitarbeiter, die ein höheres Gehalt als der Angestellte Ramanathan Ashok (Nummer 45.

Lösung:

Es werden zwei Abfragen nacheinander ausgeführt.

1. Abfrage:

```
SELECT salary
FROM employee
WHERE emp_no = 45
```

Lösung: 80689,50 €

In der zweiten Abfrage werden nun alle Mitarbeiter gesucht, die ein höheres Gehalt als 80689,50 € beziehen.

2. Abfrage:

```
SELECT emp_no, full_name
FROM employee
WHERE salary > 80689.5
```

Ergebnis:

EMP_NO	FULL_NAME
2	Nelson, Robert
4	Young, Bruce
5	Lambert, Kim
11	Weston, K. J.
20	Papadopoulos, Chris
24	Fisher, Pete
46	Steadman, Walter
72	Sutherland, Claudia
85	MacDonald, Mary S.
105	Bender, Oliver H.
107	Cook, Kevin
110	Ichida, Yuki
118	Yamamoto, Takashi
121	Ferrari, Roberto
134	Glon, Jacques
141	Osborne, Pierre

Insgesamt sind es 16 Mitarbeiter.

Diese Hintereinanderschaltung ist für dieses Beispiel akzeptabel, für eine automatische Berechnung benötigt man eine einmalige Abfrage. Dazu werden beide Abfragen kombiniert. Die innere Abfrage oder Unterabfrage gibt einen Wert zurück, der von der äußeren Abfrage bzw. der Hauptabfrage in der WHERE-Klausel verwendet wird. Die Verwendung einer Unterabfrage entspricht dem Ausführen von zwei sequentiellen Abfragen, wobei das Ergebnis der ersten Abfrage als Suchwert in der zweiten Abfrage verwendet wird.

Wer bezieht ein höheres Gehalt als Ramanathan Ashok?	
Hauptabfrage: Welche Mitarbeiter beziehen ein höheres Gehalt als Ramanathan Ashok?	**Unterabfrage**: Wie hoch ist das Gehalt von Ramanathan Ashok?

Abbildung 10 Höheres Gehalt als Ramanathan Ashok

6.1 Syntax der Unterabfrage

Die Abbildung 11 zeigt die Syntax der beiden Abfragen. Die Kopplung findet in der WHERE-Klausel durch geeignete Operatoren statt.

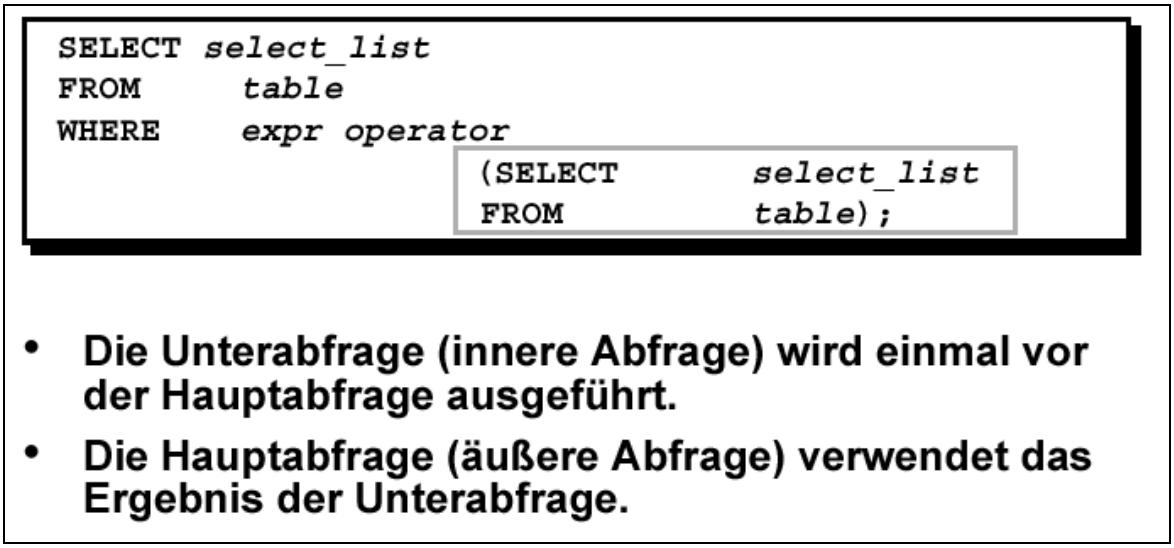

Abbildung 11 Syntax der Unterabfrage

Unterabfragen:

Eine Unterabfrage iste eine SELECT-Anweisung, die in eine Klausel einer anderen SELECT-Anweisung eingebettet ist. Mit Unterabfragen kann man einfache Anweisungen zu leistungsstarken Anweisungen kombinieren. Unterabfragen sind nützlich, wenn man Zeilen aus einer Tabelle mit einer Bedingung auswählen möchte, die von Daten in der Tabelle selbst abhängt.

Einfügen der Unterabfrage in verschiedene SQL-Klauseln:

- WHERE-Klauseln
- HAVING-Klauseln
- FROM-Klauseln

Einfach-Operatoren: (>, =, >=, <>, <=)
Mehrfach-Operatoren: (IN, ANY, ALL)

Die Unterabfrage wird generell zuerst ausgeführt, und ihre Ausgabe wird verwendet, um die Abfragekriterien für die hauptabfrage zu vervollständigen.

Richtlinien für die Verwednung von Unterabfragen

- Eine Unterabfrage muss in Klammern angegeben werden.
- Setzen Sie die Unterabfrage zur besseren lesbarkeit auf die rechte Seite des Vergleichsoperators.
- In Unterabfragen werden zwei Klassen von Vergleichsoperatoren verwendet:
  Einfach-Operatoren
  Mehrfach-Operatoren

6.2 Einfach-Operatoren

Für Einfachabfragen dienen die bekannten Vergleichs-Operatoren.

Beispiel:

Gesucht sind die Mitarbeiter, die dieselbe Jobkennung haben, wie der Mitarbeiter mit der Nummer 83 **UND** ein Gehalt größer als des Mitarbeiters mit der Nummer 94.

1. Abfrage:

Gesucht ist die Job-Kennung des Mitarbeiters mit der Nummer 83:

```
SELECT job_code
   FROM employee
   WHERE emp_no = 83;
```

Ergebnis: Eng

2. Abfrage:

Gesucht sind die Mitarbeiter, die die Jobkennung „Eng“ haben

```
SELECT    last_name, job_code, salary
   FROM   employee
   WHERE  job_code = 'Eng';
```

Ergebnis:

LAST_NAME	JOB_CODE	SALARY
Young	Eng	97500
Lambert	Eng	102750

Fisher	Eng	81810
De Souza	Eng	69483
Stansbury	Eng	39224
Phong	Eng	56034
Ramanathan	Eng	80690
Burbank	Eng	53168
Bishop	Eng	62550
Ichida	Eng	6000000
Page	Eng	48000
Parker	Eng	35000
Green	Eng	36000
Montgomery	Eng	35000
Guckenheimer	Eng	32000

3. Abfrage:

Gesucht ist das Gehalt des Mitarbeiters mit der Nummer 94.

```
SELECT salary
   FROM employee
   WHERE emp_no = 94
```

Ergebnis: 56295,00

4. Abfrage:

Gesucht sind alle Mitarbeiter die ein höheres Gehalt als 56295,00 haben

```
SELECT   last_name, job_code, salary
FROM     employee
WHERE    salary >  56295;
```

Ergebnis: 26 Mitarbeiter

5. Abfrage:

Verallgemeinerung der ersten Frage:

Gesucht sind die Mitarbeiter, die dieselbe Jobkennung haben, wie der Mitarbeiter mit der Nummer 83:

```
SELECT      last_name, job_code, salary
   FROM     employee
   WHERE    job_code = (
                        SELECT job_code
                        FROM employee
                        WHERE emp_no = 83;
                    );
```

6. Abfrage:

Verallgemeinerung der zweiten Frage:

Gesucht sind die Mitarbeiter, die ein Gehalt größer als des Mitarbeiters mit der Nummer 94 haben.

```
SELECT      last_name, job_code, salary
   FROM          employee
   WHERE         salary >  (
                           SELECT salary
                           FROM employee
                           WHERE emp_no = 94
                           )
```

6. Abfrage:

Zusammenfassung der beiden Kombi-Anfragen

```
SELECT    last_name, job_code, salary
FROM      employee
WHERE     job_code = (
                    SELECT job_code
                    FROM employee
                    WHERE emp_no = 83
                    )
AND
          salary >  (
                    SELECT salary
                    FROM employee
                    WHERE emp_no = 94
                    );
```

Ergebnis:

LAST_NAME	JOB_CODE	SALARY
Young	Eng	97500
Lambert	Eng	102750
Fisher	Eng	81810,1875
De Souza	Eng	69482,625
Ramanathan	Eng	80689,5
Bishop	Eng	62550
Ichida	Eng	6000000

6.3 Mehrfach-Operatoren

Für Mehrfachabfragen dienen folgende Operatoren:

- IN
- ANY
- ALL

6.3.1 Mehrfach-Operator IN

Der Mehrfach-Operator IN vergleicht in der Hauptabfrage, ob die Datensätze in der ermittelten Menge enthalten sind.

Beispiel:

Gesucht sind die Angestellten, deren Gehalt dem Mindestgehalt ihrer Abteilung entspricht.

```
SELECT    last_name, salary, dept_no
FROM      employee
WHERE     salary IN (
                    SELECT MIN(salary)
                    FROM employee
                    GROUP BY dept_no
                    )
```

Ergebnis:

LAST_NAME	SALARY	DEPT_NO
Weston	86292,9375	130
Lee	53793	000
Hall	69482,625	900
Bennet	22935	120
De Souza	69482,625	623
Baldwin	61637,8125	110
Nordstrom	42742,5	180
O'Brien	31275	670
Sutherland	100914	140
Brown	27000	600
Ichida	6000000	115
Page	48000	671
Parker	35000	623
Ferrari	99000000	125
Yanowski	44000	100
Glon	390500	123
Green	36000	621
Osborne	110000	121
Montgomery	35000	672
Guckenheimer	32000	622

6.3.2 Mehrfach-Operator ANY

Der Mehrfach-Operator ANY vergleicht einen Wert mit jedem von einer Unterabfrage zurückgegebenen Wert. in der Hauptabfrage, ob die Datensätze in der ermittelten Menge enthalten sind.

Beispiel:

Gesucht sind die Mitarbeiter, die keine „SRep“ sind, und ein geringeres Gehalt beziehen als einer der Mitarbeiter mit der Jobkennung „SRep“.

```
SELECT    emp_no, last_name, job_code, salary
FROM      employee
WHERE     salary < ANY
                    ( SELECT salary
                    FROM employee
                    WHERE job_code = "SRep"
                    )
AND       job_code <> "SRep"
```

< ANY bedeutet weniger als das Maximum
> ANY bedeutet mehr als das Minimum
= ANY hat dieselbe Bedeutung wie IN

6.3.3 Mehrfach-Operator ALL

Der Mehrfach-Operator ANY vergleicht einen Wert mit allen von einer Unterabfrage zurückgegebenen Werten.. in der Hauptabfrage, ob die Datensätze in der ermittelten Menge enthalten sind.

> ALL bedeutet mehr als das Maximum
< ALL bedeutet weniger als das Minimum

Beispiel:

Gesucht sind die Mitarbeiter, die keine „SRep“ sind, und deren Gehalt unter dem Gehalt aller Angestellten mit der Jobkennung „SRep“ liegt.

```
SELECT    emp_no, last_name, job_code, salary
FROM      employee
WHERE     salary < ALL
                    ( SELECT salary
                    FROM employee
                    WHERE job_code = "SRep"
                    )
AND       job_code <> "SRep"
```

Ergebnis:

EMP_NO	LAST_NAME	JOB_CODE	SALARY
28	Bennet	Admin	22935
36	Reeves	Sales	33620,625
37	Stansbury	Eng	39224,0625
52	Nordstrom	Prel	42742,5
65	O'Brien	Admin	31275
109	Brown	Admin	27000
114	Parker	Eng	35000
138	Green	Eng	36000
144	Montgomery	Eng	35000
145	Guckenheimer	Eng	32000

Hinweis:

Man kann den Operator NOT mit allen Mehrfach-Operatoren verknüpfen.

6.4 Trigger und Oracle

Trigger sind Prozeduren, die mit DML-Operationen verknüpft sind:

- Insert
- Update
- Delete

Jede Prozedure kann man vorher oder nachher definieren:

- Before Insert
- After Insert
- Before Update
- After Update
- Before Delete
- After Delete

Syntax:

```
CREATE OR REPLACE TRIGGER Trigger_Name
  BEFORE INSERT OR UPDATE ON Tablename
    FOR EACH ROW
    BEGIN
        -- Anweisung
    END;
```

Im Anweisungblock hat man mit der “Variablen “:NEW” Zugriff auf den aktuellen Datensatz

Beispiel:

```
CREATE OR REPLACE TRIGGER Trigger_Kunden
  BEFORE INSERT OR UPDATE ON Kunde
    FOR EACH ROW
    BEGIN
        :NOW.MitarbeiterNr := 123;
        :NOW.Datum := NOW;
    END;
```

7 Generator / Sequenz

Häufig benötigt man einen ganzzahligen Schlüssel für ein Feld. Nun kann man mit folgendem Code diesen bestimmen (Psudocode und PL-SQL):

```
MaxNr =   SELECT max(MatrNr)
          FROM Student

INSERT INTO Student (MatrNr, Name)
      VALUES (MaxNr +1, "meier”);
```

Diese Variante funktioniert leider nicht sicher in einem Netzwerk! Darum existieren Funktionen, die garantieren, dass ein Schlüssel immer nur einmalig vergeben wird.

7.1 Generator (Firebird)

Beispiel:

Anlegen eines Studenten mittels eines Generators

```
CREATE GENERATOR GEN_NUMBER;
SET GENERATOR  GEN_NUMBER   TO 999;

INSERT INTO EMP (empno, Name)
VALUES ( GEN_ID(GEN_NUMBER,1), 'Bates', ' Norman ');
```

7.2 Sequenz (Oracle)

Syntax:

```
CREATE SEQUENCE sequence
[INCREMENT BY n]
[START WITH n]
[{MAXVALUE n | NOMAXVALUE}]
[{MINVALUE n | NOMINVALUE}]
[{CYCLE | NOCYCLE}]
[{CACHE n | NOCACHE}];
```

Beispiel:

Anlegen eines Studenten mitttels Sequnez

```
CREATE SEQUENCE stud_matrnr
 INCREMENT BY 1
 START WITH 100
 MAXVALUE 100000
 NOCACHE
 NOCYCLE;
```

7.2.1 Ablauf Firebird:

- Tabelle erstellen (CREATE TABLE)

```
create table student (
    matrnr          integer,
    name      varchar(10)
)
```

- Generator erstellen CREATE GENERATOR GEN_MATRNR;
- Anfangswert bestimmen: SET GENERATOR GEN_MATRNR TO 0;

In Baumelement exitsiert nun der Generator „GEN_MATRNR.

Nun die Verknüpfung mit dem Trigger:

- Aufruf der Tabelle Student
- Anklicken des Register Trigger:

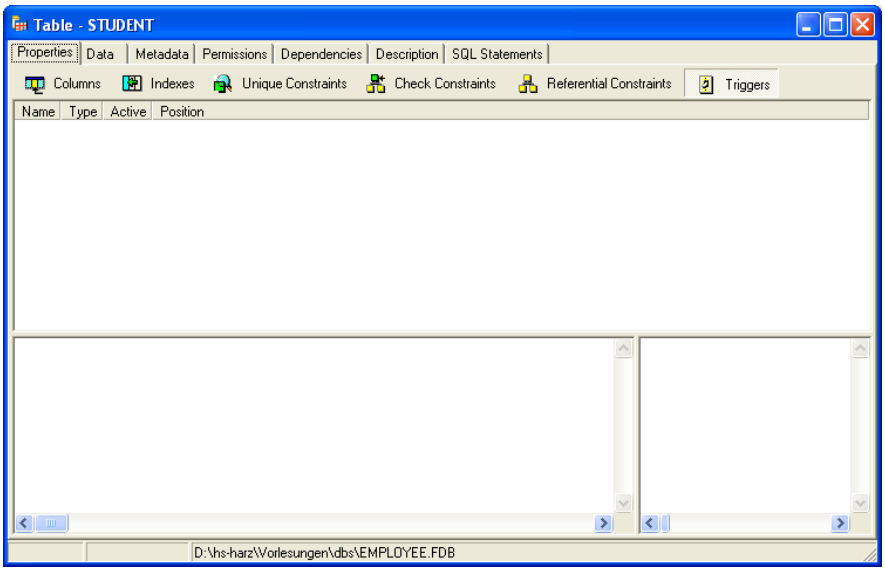

Abbildung 12 Trigger erstellen

- Rechte Maustaste, Eintrag „new“

```
AS
BEGIN
   new.matrnr = gen_id(gen_matrnr, 1);
END
```

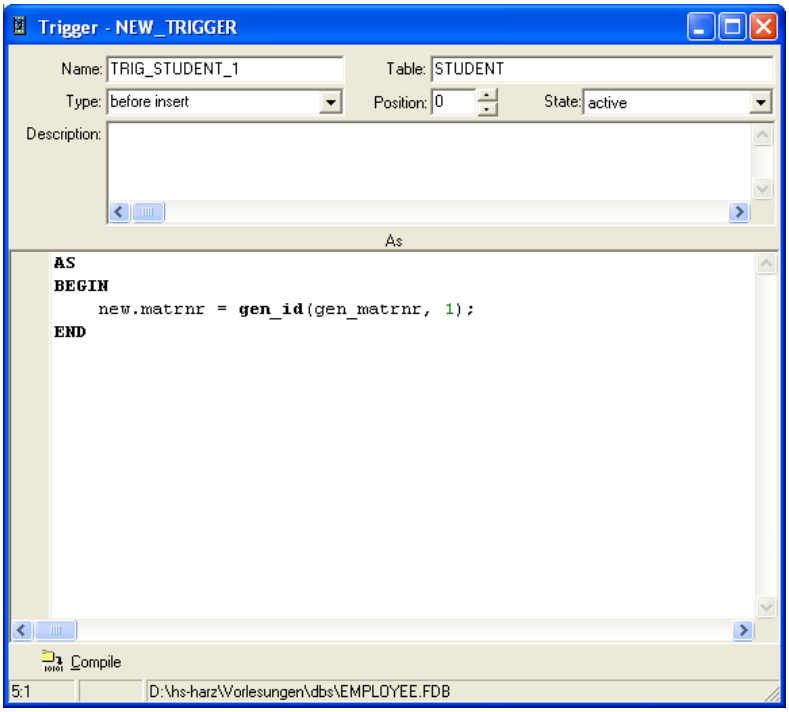

Abbildung 13 Genertaor im Trigger

- Übersetzen (Schalter Compile)

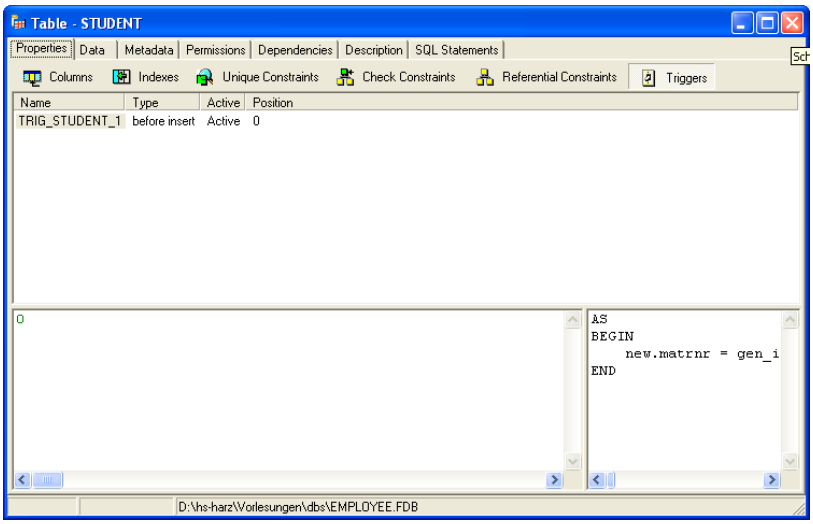

Abbildung 14 Fertiger Trigger

- Test mit Insert-Befehlen

```
INSERT INTO Student (matrnr, Name)
VALUES ( 22, 'Bates');

INSERT INTO Student (matrnr, Name)
VALUES ( 24, 'Müller');
```

Ergebnis:

Matrnr	Name
1	Bates
2	Müller

Oder auch:

```
INSERT INTO Student ( Name)
VALUES (  'Bates');

INSERT INTO Student ( Name)
VALUES (  'Müller');
```

8 Datenbank-Informationen

8.1 Firebird-Tabellen

Dieses Kapitel zeigt die Struktur der verwendeten Firebird-Datenbank mit den Entities (Country, Customer, Department, Employee, Employee_Projekt, Job, Proj_Dept_Budget, Project, Sales_History und Sales). Zusätzlichen werden alle verwendeten Eingabebedingungen erläutert. Die vorhandenen Trigger werden in einer späteren Vorlesung behandelt und sind deshalb hier nicht erwähnt.

8.1.1 Employee

Attribute	**Beschreibung**
EMP_NO	Mitarbeiter-Nummer
FIRST_NAME	Vorname
LAST_NAME	Nachname
PHONE_EXT	Telefonnummer
HIRE_DATE	Arbeitsbeginn in der Firma
DEPT_NO	Abteilungsnummer
JOB_CODE	Abkürzung die Arbeitsbezeichnung (z. B. Manager, Engineer)
JOB_GRADE	Bezeichnet die Hierarchiestufe in der Firma
JOB_COUNTRY	Land, in dem man für die Firma arbeitet
SALARY	Gehalt
FULL_NAME	Kompletter Name (Nachname, Vorname) REDUNDANT

8.1.2 Department

Attribute	**Beschreibung**
DEPT_NO	Abteilungsnummer
DEPARTMENT	Bezeichnung der Abteilung
HEAD_DEPT	Zugehörigkeit der Abteilung
MNGR_NO	Wer leitet die Abteilung
BUDGET	Jahresbugdet
LOCATION	Sitz der Abteilung
PHONE_NO	Telefonnumer des Managers

8.1.3 Employee_Project

Attribute	**Beschreibung**
EMP_NO	Mitarbeiternummer (siehe Tabelle Employee)
PROJ_ID	Projektnummer (siehe Tabelle Project)

8.1.4 Project

Attribute	Beschreibung
PROJ_ID	Projektnummer
PROJ_NAME	Projektbeschreibung (Memo)
PROJ_DESC	Projektbeschreibung (Memo-Feld)
TEAM_LEADER	Projektleiter
PRODUCT	Produkttyp (Software, Hardware)

8.1.5 Customer

Attribute	Beschreibung
CUST_NO	Kundennummer
CUSTOMER	Kundenname
CONTACT_FIRST	Erster Ansprechpartner beim Kunden
CONTACT_LAST	Letzter Ansprechpartner beim Kunden
PHONE_NO	Telefonnummer des aktuellen Ansprechpartners
ADDRESS_LINE1	1. Adresse
ADDRESS_LINE2	2. Adresse (Hotel, Suite)
CITY	Stadt
STATE_PROVINCE	Bundesstaat
COUNTRY	Land
POSTAL_CODE	Postleitzahl der Stadt
ON_HOLD	

8.1.6 Country

Attribute	Beschreibung
COUNTRY	Land
CURRENCY	Währung

8.1.7 Job

Attribute	Beschreibung
JOB_CODE	Abkürzung die Arbeitsbezeichnung (z. B. Manager, Engineer)
JOB_GRADE	Bezeichnet die Hierarchiestufe in der Firma
JOB_COUNTRY	Sitz der Abteilung
JOB_TITLE	Arbeitsbezeichnung (z. B. Manager, Engineer)
MIN_SALARY	Minimales Gehalt dieser Stufe
MAX_SALARY	Maximales Gehalt dieser Stufe
JOB_REQUIREMENT	Job-Anforderungen
LANGUAGE_REQ	Sprachkenntnisse

8.1.8 Proj_Dept_Budget

Beschreibt den Finanzrahmen der Abteilungen und der Projekte.

Attribute	**Beschreibung**
FISCAL_YEAR	Finazjahr (1994-1996)
PROJ_ID	Projekt-Nummer (VBASE, MAPDB, etc.)
DEPT_NO	Abteilungsnummer des Projektes
QUART_HEAD_CNT	
PROJECTED_BUDGET	Bugdet für den Zeitraum

8.1.9 Salary_History

Beschreibt die Gehaltserhöhungen.

Attribute	**Beschreibung**
EMP_NO	Mitarbeiter-Nummer
CHANGE_DATE	Änderungsdatum
UPDATER_ID	Wer hat diese Änderung eingetragen
OLD_SALARY	Altes Gehalt
PERCENT_CHANGE	Prozent der Steigerung
NEW_SALARY	Neues Gehalt

8.1.10 Sales

Attribute	**Beschreibung**
PO_NUMBER	Auftrags-Nummer (Jahreszahl,
CUST_NO	Kunden-Nummer
SALES_REP	Kunden-Report ?
ORDER_STATUS	Status des Auftrags (Offen, Wartend , Verschickt)
ORDER_DATE	Auftragsdatum
SHIP_DATE	Ausgangsdatum der Ware
DATE_NEEDED	Fester Zeitpunkt der Lieferung
PAID	Bezahlt (Ja / Nein)
QTY_ORDERED	Menge der Lieferung
TOTAL_VALUE	Gesamtbetrag
DISCOUNT	Rabatt in Prozent
ITEM_TYPE	Typ der gelieferten Ware (Software, Hardware)
AGED	Tage des Versands

8.2 Tabellen

8.2.1 Employee (Auszug)

Mitarbeitertabelle.

EMP_NO	FIRST_NAME	LAST_NAME	DEPT_NO	SALARY	FULL_NAME
2	Robert	Nelson	600	105900	Nelson, Robert
4	Bruce	Young	621	97500	Young, Bruce
5	Kim	Lambert	130	102750	Lambert, Kim
8	Leslie	Johnson	180	64635	Johnson, Leslie
9	Phil	Forest	622	75060	Forest, Phil
11	K. J.	Weston	130	86292,9375	Weston, K. J.
12	Terri	Lee	000	53793	Lee, Terri
14	Stewart	Hall	900	69482,625	Hall, Stewart
15	Katherine	Young	623	67241,25	Young, Katherine
20	Chris	Papadopoulos	671	89655	Papadopoulos, Chris
24	Pete	Fisher	671	81810,1875	Fisher, Pete
28	Ann	Bennet	120	22935	Bennet, Ann
29	Roger	De Souza	623	69482,625	De Souza, Roger
34	Janet	Baldwin	110	61637,8125	Baldwin, Janet
36	Roger	Reeves	120	33620,625	Reeves, Roger
37	Willie	Stansbury	120	39224,0625	Stansbury, Willie
44	Leslie	Phong	623	56034,375	Phong, Leslie
45	Ashok	Ramanathan	621	80689,5	Ramanathan, Ashok
46	Walter	Steadman	900	116100	Steadman, Walter
52	Carol	Nordstrom	180	42742,5	Nordstrom, Carol
61	Luke	Leung	110	68805	Leung, Luke
65	Sue Anne	O'Brien	670	31275	O'Brien, Sue Anne
71	Jennifer M.	Burbank	622	53167,5	Burbank, Jennifer M.
72	Claudia	Sutherland	140	100914	Sutherland, Claudia
83	Dana	Bishop	621	62550	Bishop, Dana
85	Mary S.	MacDonald	100	111262,5	MacDonald, Mary S.
94	Randy	Williams	672	56295	Williams, Randy
105	Oliver H.	Bender	000	212850	Bender, Oliver H.
107	Kevin	Cook	670	111262,5	Cook, Kevin
109	Kelly	Brown	600	27000	Brown, Kelly
110	Yuki	Ichida	115	6000000	Ichida, Yuki
113	Mary	Page	671	48000	Page, Mary
114	Bill	Parker	623	35000	Parker, Bill
118	Takashi	Yamamoto	115	7480000	Yamamoto, Takashi
121	Roberto	Ferrari	125	99000000	Ferrari, Roberto
127	Michael	Yanowski	100	44000	Yanowski, Michael
134	Jacques	Glon	123	390500	Glon, Jacques
136	Scott	Johnson	623	60000	Johnson, Scott
138	T.J.	Green	621	36000	Green, T.J.
141	Pierre	Osborne	121	110000	Osborne, Pierre
144	John	Montgomery	672	35000	Montgomery, John
145	Mark	Guckenheimer	622	32000	Guckenheimer, Mark

8.2.2 Department

Beschreibt die Abteilungen.

DEPT_NO	DEPARTMENT	HEAD_DEPT	MNGR_NO	BUDGET	LOCATION	PHONE_NO
000	Corporate Headquarters		105	1000000	Monterey	(408) 555-1234
100	Sales and Marketing	000	85	2000000	San Francisco	(415) 555-1234
110	Pacific Rim Headquarters	100	34	600000	Kuaui	(808) 555-1234
115	Field Office: Japan	110	118	500000	Tokyo	3 5350 0901
116	Field Office: Singapore	110		300000	Singapore	3 55 1234
120	European Headquarters	100	36	700000	London	71 235-4400
121	Field Office: Switzerland	120	141	500000	Zurich	1 211 7767
123	Field Office: France	120	134	400000	Cannes	58 68 11 12
125	Field Office: Italy	120	121	400000	Milan	2 430 39 39
130	Field Office: East Coast	100	11	500000	Boston	(617) 555-1234
140	Field Office: Canada	100	72	500000	Toronto	(416) 677-1000
180	Marketing	100		1500000	San Francisco	(415) 555-1234
600	Engineering	000	2	1100000	Monterey	(408) 555-1234
620	Software Products Div.	600		1200000	Monterey	(408) 555-1234
621	Software Development	620		400000	Monterey	(408) 555-1234
622	Quality Assurance	620	9	300000	Monterey	(408) 555-1234
623	Customer Support	620	15	650000	Monterey	(408) 555-1234
670	Consumer Electronics Div.	600	107	1150000	Burlington	(802) 555-1234
671	Research and Development	670	20	460000	Burlington	(802) 555-1234
672	Customer Services	670	94	850000	Burlington	(802) 555-1234
900	Finance	000	46	400000	Monterey	(408) 555-1234

8.2.3 Sales

Beschreibt die Verkäufe der Firma. Die Tabelle gibt nicht alle Attribute wider.
Atttribut QTY_ORDERED Menge der bestellten Ware
TOTAL_VALUE Rechnungsbetrag

PO_#	CUST#	ORDER_STATUS	ORDER_DATE	QTY_ORDERED	TOTAL VALUE	ITEM_TYPE
V91E0210	1004	shipped	04.03.1991	10	5000	hardware
V92E0340	1004	shipped	15.10.1992	7	70000	hardware
V92J1003	1010	shipped	26.07.1992	15	2985	software
V93J2004	1010	shipped	30.10.1993	3	210	software
V93J3100	1010	shipped	20.08.1993	16	18000,4	software
V92F3004	1012	shipped	15.10.1992	3	2000	software
V93F3088	1012	shipped	27.08.1993	10	10000	software
V93F2030	1012	open	12.12.1993	15	450000,49	hardware
V93F2051	1012	waiting	18.12.1993	1	999,98	software
V93H0030	1005	open	12.12.1993	20	5980	software
V94H0079	1005	open	13.02.1994	10	9000	software
V9324200	1001	shipped	09.08.1993	1000	560000	hardware
V9324320	1001	shipped	16.08.1993	1	0	software
V9320630	1001	open	12.12.1993	3	60000	hardware
V9420099	1001	open	17.01.1994	100	3399,15	software
V9427029	1001	shipped	07.02.1994	17	422210,97	hardware
V9333005	1002	shipped	03.02.1993	2	600,5	software
V9333006	1002	shipped	27.04.1993	5	20000	other
V9336100	1002	waiting	27.12.1993	150	14850	software

V9346200	1003	waiting	31.12.1993	3	0	software
V9345200	1003	shipped	11.11.1993	900	27000	software
V9345139	1003	shipped	09.09.1993	20	12582,12	software
V93C0120	1006	shipped	22.03.1993	1	47,5	other
V93C0990	1006	shipped	09.08.1993	40	399960,5	hardware
V9456220	1007	open	04.01.1994	1	3999,99	hardware
V93S4702	1011	shipped	27.10.1993	4	120000	hardware
V94S6400	1011	waiting	06.01.1994	20	1980,72	software
V93H3009	1008	shipped	01.08.1993	3	9000	software
V93H0500	1008	open	12.12.1993	3	16000	hardware
V93F0020	1009	shipped	10.10.1993	1	490,69	software
V93I4700	1013	open	27.10.1993	5	2693	hardware
V93B1002	1014	shipped	20.09.1993	1	100,02	software
V93N5822	1015	shipped	18.12.1993	2	1500	software

8.2.4 Project

Beschreibt die Projekte im Details.

PROJ_ID	**PROJ_NAME**	**PROJ_DESC**	**TEAM_LEADER**	**PRODUCT**
VBASE	Video Database	memo	45	software
DGPII	DigiPizza	memo	24	other
GUIDE	AutoMap	memo	20	hardware
MAPDB	MapBrowser port	memo	4	software
HWRII	Translator upgrade	memo	-	software
MKTPR	Marketing project 3	memo	85	N/A

Die Memo-Felder sind binäre Elemente, in denen Texte mit beliebiger Länge eingetragen werden können.

8.2.5 Employee_Project

Beschreibt die Beziehung zwischen Mitarbeiter und Projekt (m zu n).

EMP_NO	**PROJ_ID**
4	MAPDB
4	VBASE
8	GUIDE
8	MKTPR
8	VBASE
12	MKTPR
14	MKTPR
15	VBASE
20	GUIDE
24	DGPII
24	GUIDE
34	MKTPR
44	VBASE
45	VBASE
46	MKTPR
52	MKTPR
71	MAPDB
71	VBASE
83	VBASE
85	MKTPR
105	MKTPR

110	MKTPR
113	DGPII
113	GUIDE
136	VBASE
138	VBASE
144	DGPII
145	VBASE

8.3 Entity COUNTRY

8.3.1 Tabellendefinition

```
CREATE TABLE COUNTRY (
  COUNTRY COUNTRYNAME NOT NULL,
  CURRENCY VARCHAR(10) NOT NULL
)
```

8.4 Entity CUSTOMER

8.4.1 Tabellendefinition

```
CREATE TABLE CUSTOMER (
  CUST_NO CUSTNO NOT NULL,
  CUSTOMER VARCHAR(25) NOT NULL,
  CONTACT_FIRST FIRSTNAME,
  CONTACT_LAST LASTNAME,
  PHONE_NO PHONENUMBER,
  ADDRESS_LINE1 ADDRESSLINE,
  ADDRESS_LINE2 ADDRESSLINE,
  CITY VARCHAR(25),
  STATE_PROVINCE VARCHAR(15),
  COUNTRY COUNTRYNAME,
  POSTAL_CODE VARCHAR(12),
  ON_HOLD CHAR(1) DEFAULT NULL
)
```

8.4.2 Eingabebedingung

```
on_hold IS NULL OR on_hold = '*'
```

8.5 Entity DEPARTMENT

8.5.1 Tabellendefinition

```
CREATE TABLE DEPARTMENT (
  DEPT_NO DEPTNO NOT NULL,
  DEPARTMENT VARCHAR(25) NOT NULL,
  HEAD_DEPT DEPTNO,
  MNGR_NO EMPNO,
  BUDGET BUDGET,
  LOCATION VARCHAR(15),
  PHONE_NO PHONENUMBER DEFAULT '555-1234'
)
```

8.6 Entity EMPLOYEE

8.6.1 Tabellendefinition

```
CREATE TABLE EMPLOYEE (
  EMP_NO EMPNO NOT NULL,
  FIRST_NAME FIRSTNAME NOT NULL,
  LAST_NAME LASTNAME NOT NULL,
  PHONE_EXT VARCHAR(4),
  HIRE_DATE DATE DEFAULT 'NOW' NOT NULL,
  DEPT_NO DEPTNO NOT NULL,
  JOB_CODE JOBCODE NOT NULL,
  JOB_GRADE JOBGRADE NOT NULL,
  JOB_COUNTRY COUNTRYNAME NOT NULL,
  SALARY SALARY NOT NULL,
  FULL_NAME VARCHAR(37)
)
```

8.6.2 Eingabebedingung

```
salary >= (SELECT min_salary FROM job WHERE
                job.job_code = employee.job_code AND
                job.job_grade = employee.job_grade AND
                job.job_country = employee.job_country) AND
        salary <= (SELECT max_salary FROM job WHERE
                job.job_code = employee.job_code AND
                job.job_grade = employee.job_grade AND
                job.job_country = employee.job_country)
```

8.7 Entity EMPLOYEE_PROJECT

8.7.1 Tabellendefinition

```
CREATE TABLE EMPLOYEE_PROJECT (
  EMP_NO EMPNO NOT NULL,
  PROJ_ID PROJNO NOT NULL
)
```

8.8 Entity JOB

8.8.1 Tabellendefinition

```
CREATE TABLE JOB (
  JOB_CODE JOBCODE NOT NULL,
  JOB_GRADE JOBGRADE NOT NULL,
  JOB_COUNTRY COUNTRYNAME NOT NULL,
  JOB_TITLE VARCHAR(25) NOT NULL,
  MIN_SALARY SALARY NOT NULL,
  MAX_SALARY SALARY NOT NULL,
  JOB_REQUIREMENT BLOB SUB_TYPE 1 SEGMENT SIZE 400,
  LANGUAGE_REQ VARCHAR(15) [1:5]
)
```

8.8.2 Eingabebedingungen

```
min_salary < max_salary
```

8.9 Entity PROJ_DEPT_BUDGET

8.9.1 Tabellendefinition

```
CREATE TABLE PROJ_DEPT_BUDGET (
  YEAR INTEGER NOT NULL,
  PROJ_ID PROJNO NOT NULL,
  DEPT_NO DEPTNO NOT NULL,
  QUART_HEAD_CNT INTEGER [1:4],
  PROJECTED_BUDGET BUDGET
)
```

8.9.2 Eingabebedingungen

```
ALTER TABLE PROJ_DEPT_BUDGET
```

```
  ADD CONSTRAINT INTEG_43
  CHECK (YEAR >= 1993)
```

8.10 Entity PROJECT

8.10.1 Tabellendefinition

```
CREATE TABLE PROJECT (
  PROJ_ID PROJNO NOT NULL,
  PROJ_NAME VARCHAR(20) NOT NULL,
  PROJ_DESC BLOB SUB_TYPE 1 SEGMENT SIZE 800,
  TEAM_LEADER EMPNO,
  PRODUCT PRODTYPE
)
```

8.11 Entity SALARY_HISTORY

8.11.1 Tabellendefinition

```
CREATE TABLE SALARY_HISTORY (
  EMP_NO EMPNO NOT NULL,
  CHANGE_DATE DATE DEFAULT 'NOW' NOT NULL,
  UPDATER_ID VARCHAR(20) NOT NULL,
  OLD_SALARY SALARY NOT NULL,
  PERCENT_CHANGE DOUBLE PRECISION DEFAULT 0 NOT NULL,
  NEW_SALARY DOUBLE PRECISION
)
```

8.11.2 Eingabebedingungen

```
ALTER TABLE SALARY_HISTORY
  ADD CONSTRAINT INTEG_54
  CHECK (percent_change between -50 and 50)
```

8.12 Entity SALES

8.12.1 Tabellendefinition

```
CREATE TABLE SALES (
  PO_NUMBER PONUMBER NOT NULL,
  CUST_NO CUSTNO NOT NULL,
  SALES_REP EMPNO,
  ORDER_STATUS VARCHAR(7) DEFAULT 'new' NOT NULL,
  ORDER_DATE DATE DEFAULT 'now' NOT NULL,
  SHIP_DATE DATE,
  DATE_NEEDED DATE,
  PAID CHAR(1) DEFAULT 'n',
  QTY_ORDERED INTEGER DEFAULT 1 NOT NULL,
  TOTAL_VALUE NUMERIC(9, 2) NOT NULL,
  DISCOUNT FLOAT DEFAULT 0 NOT NULL,
  ITEM_TYPE PRODTYPE,
  AGED DOUBLE PRECISION
)
```

8.12.2 Eingabebedingungen

8.12.2.1 Integritätsbedingung 1

```
ALTER TABLE SALES
  ADD CONSTRAINT INTEG_65
  CHECK (order_status in
                ('new', 'open', 'shipped', 'waiting'))
```

8.12.2.2 Integritätsbedingung 2

```
ALTER TABLE SALES
  ADD CONSTRAINT INTEG_67
  CHECK (ship_date >= order_date OR ship_date IS NULL)
```

8.12.2.3 Integritätsbedingung 3

```
ALTER TABLE SALES
  ADD CONSTRAINT INTEG_68
  CHECK (date_needed > order_date OR date_needed IS NULL)
```

8.12.2.4 Integritätsbedingung 4

```
ALTER TABLE SALES
  ADD CONSTRAINT INTEG_69
  CHECK (paid in ('y', 'n'))
```

8.12.2.5 Integritätsbedingung 5

```
ALTER TABLE SALES
  ADD CONSTRAINT INTEG_71
  CHECK (qty_ordered >= 1)
```

8.12.2.6 Integritätsbedingung 1

```
ALTER TABLE SALES
  ADD CONSTRAINT INTEG_73
  CHECK (total_value >= 0)
```

8.12.2.7 Integritätsbedingung 6

```
ALTER TABLE SALES
  ADD CONSTRAINT INTEG_75
  CHECK (discount >= 0 AND discount <= 1)
```

8.12.2.8 Integritätsbedingung 7

```
ALTER TABLE SALES
  ADD CONSTRAINT INTEG_79
  CHECK (NOT (order_status = 'shipped' AND ship_date IS NULL))
```

8.12.2.9 Integritätsbedingung 8

```
ALTER TABLE SALES
  ADD CONSTRAINT INTEG_80
  CHECK (NOT (order_status = 'shipped' AND
         EXISTS (SELECT on_hold FROM customer
              WHERE customer.cust_no = sales.cust_no
              AND customer.on_hold = '*')))
```

9 Anhang

9.1 SELECT-SYNTAX

Die untere Syntax-Beschreibung zeigt die komplette Syntax mit allen Optionen. Je nach Datenabnk-Version können Unterschiede auftreten.

```
Syntax  SELECT [TRANSACTION transaction]
        [DISTINCT | ALL]
        {* | <val> [, <val> …]}
        [INTO :var [, :var …]]
        FROM <tableref> [, <tableref> …]
        [WHERE <search_condition>]
        [GROUP BY col [COLLATE collation] [, col [COLLATE collation
] …]
        [HAVING <search_condition>]
        [UNION <select_expr> [ALL]]
        [PLAN <plan_expr>]
        [ORDER BY <order_list>]
        [FOR UPDATE [OF col [, col …]]];

<val> = {
        col [<array_dim>] | :variable
        | <constant> | <expr> | <function>
        | udf ([<val> [, <val> …]])
        | NULL | USER | RDB$DB_KEY | ?
        } [COLLATE collation] [AS alias]
<array_dim> = [[x:]y [, [x:]y …]]
<constant> = num | 'string' | charsetname 'string'
<function> = COUNT (* | [ALL] <val> | DISTINCT <val>)
        | SUM ([ALL] <val> | DISTINCT <val>)
        | AVG ([ALL] <val> | DISTINCT <val>)
        | MAX ([ALL] <val> | DISTINCT <val>)
        | MIN ([ALL] <val> | DISTINCT <val>)
        | CAST (<val> AS <datatype>)
        | UPPER (<val>)
        | GEN_ID (generator, <val>)

<tableref> = <joined_table> | table | view | procedure
        [(<val> [, <val> …])] [alias]
<joined_table> = <tableref> <join_type> JOIN <tableref>
        ON <search_condition> | (<joined_table>)
<join_type> = [INNER] JOIN
        | {LEFT | RIGHT | FULL } [OUTER]} JOIN

<search_condition> = <val> <operator> {<val> | (<select_one>)}
        | <val> [NOT] BETWEEN <val> AND <val>
        | <val> [NOT] LIKE <val> [ESCAPE <val>]
        | <val> [NOT] IN (<val> [, <val> …] | <select_list>)
        | <val> IS [NOT] NULL
        | <val> {>= | <=}
        | <val> [NOT] {= | < | >}
        | {ALL | SOME | ANY} (<select_list>)
        | EXISTS (<select_expr>)
```

| **SINGULAR** (<select_expr>)
| <val> [NOT] CONTAINING <val>
| <val> [NOT] STARTING [WITH] <val>
| (<search_condition>)
| NOT <search_condition>
| <search_condition> OR <search_condition>

| <search_condition> AND <search_condition>
<operator> = {= | < | > | <= | >= | !< | !> | <> | !=}
<plan_expr> =
[JOIN | [SORT] [MERGE]] ({<plan_item> | <plan_expr>}
[, {<plan_item> | <plan_expr>} …])

<plan_item> = {table | alias}

{NATURAL | INDEX (<index> [, <index> …])| ORDER <index>}
<order_list> =
{col | int} [COLLATE collation]
[ASC[ENDING] | DESC[ENDING]]
[, <order_list> …]

9.2 Argumente einer SQL-Anweisung

Argument	**Description**
expr	A valid SQL expression that results in a single value
select_one	A SELECT on a single column that returns exactly one value
select_list	A SELECT on a single column that returns zero or more rows
select_expr	A SELECT on a list of values that returns zero or more rows

Notes on SELECT syntax

n When declaring arrays, you must include the outermost brackets, shown below in bold. For example, the following statement creates a 5 by 5 two-dimensional array of strings, each of which is 6 characters long:

my_array = varchar(6)[5,5]

Use the colon (:) to specify an array with a starting point other than 1. The following example creates an array of integers that begins at 10 and ends at 20:

my_array = integer[20:30]

n In SQL and isql, you cannot use val as a parameter placeholder (like "?").
n In DSQL and isql, val cannot be a variable.
n You cannot specify a COLLATE clause for Blob columns.

Important In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in isql, the semicolon is a terminating symbol for the statement, so it must be included.

Argument	**Description**
TRANSACTION transaction	Name of the transaction under control of which the statement is executed; SQL only

SELECT [DISTINCT | ALL]	Specifies data to retrieve. DISTINCT prevents duplicate values from being returned. ALL, the default, retrieves every value
{*|val [, val …]}	The asterisk (*) retrieves all columns for the specified tablesval [, val …] retrieves a list of specified columns, values, and expressions
INTO :var [, var …]	Singleton select in embedded SQL only; specifies a list of host-language variables into which to retrieve values
FROM tableref [, tableref …]	List of tables, views, and stored procedures from which to retrieve data; list can include joins and joins can be nested
table	Name of an existing table in a database
view	Name of an existing view in a database
procedure	Name of an existing stored procedure that functions like a SELECT statement
alias	Brief, alternate name for a table, view, or column; after declaration in tableref, alias can stand in for subsequent references to a table or view
joined_table	A table reference consisting of a JOIN
join_type	Type of join to perform. Default: INNER
WHERE search_condition	Specifies a condition that limits rows retrieved to a subset of all available rows
GROUP BY col [, col …]	Partitions the results of a query into groups containing all rows with identical values based on a column list
COLLATE collation	Specifies the collation order for the data retrieved by the query
HAVING search_condition	Used with GROUP BY; specifies a condition that limits grouped rows returned
UNION [ALL]	Combines two or more tables that are fully or partially identical in structure; the ALL option keeps identical rows separate instead of folding them together into one
PLAN plan_expr	Specifies the access plan for the Firebird optimizer to use during retrieval
plan_item	Specifies a table and index method for a plan
ORDER BY order_list	Specifies columns to order, either by column name or ordinal number in the query, and the order (ASC or DESC) in which rows to return the rows

Description SELECT retrieves data from tables, views, or stored procedures. Variations of the SELECT statement make it possible to:

n Retrieve a single row, or part of a row, from a table. This operation is referred to as a singleton select.

In embedded applications, all SELECT statements that occur outside the context of a cursor must be singleton selects.

n Retrieve multiple rows, or parts of rows, from a table.

In embedded applications, multiple row retrieval is accomplished by embedding a SELECT within a DECLARE CURSOR statement.
In isql, SELECT can be used directly to retrieve multiple rows.

n Retrieve related rows, or parts of rows, from a join of two or more tables.
n Retrieve all rows, or parts of rows, from union of two or more tables.

All SELECT statements consist of two required clauses (SELECT, FROM), and possibly others (INTO, WHERE, GROUP BY, HAVING, UNION, PLAN, ORDER BY). The following table explains the purpose of each clause, and when they are required:

TABLE 6 SELECT statement clauses

Clause	**Purpose**	**Singleton SELECT**	**Multi-row SELECT**
SELECT	Lists columns to retrieve	Required	Required
INTO	Lists host variables for storing retrieved columns	Required	Not allowed
FROM	Identifies the tables to search for values	Required	Required
WHERE	Specifies the search conditions used to restrict retrieved rows to a subset of all available rows; a WHERE clause can contain its own SELECT statement, referred to as a subquery	Optional	Optional
GROUP BY	Groups related rows based on common column values; used in conjunction with HAVING	Optional	Optional
HAVING	Restricts rows generated by GROUP BY to a subset of those rows	Optional	Optional
UNION	Combines the results of two or more SELECT statements to produce a single, dynamic table without duplicate rows	Optional	Optional
ORDER BY	Specifies which columns to order, either by column name or by ordinal number in the query, and the sort order of rows returned: ascending (ASC) [default] or descending (DESC)	Optional	Optional
PLAN	Specifies the query plan that should be used by the query optimizer instead of one it would normally choose	Optional	Optional
FOR UPDATE	Specifies columns listed after the SELECT clause of a DECLARE CURSOR statement that can be updated using a WHERE CURRENT OF clause	—	Optional

Because SELECT is such a ubiquitous and complex statement, a meaningful discussion lies outside the scope of this reference. To learn how to use SELECT in isql, see the Operations Guide. For a complete explanation of SELECT and its clauses, see the Embedded SQL Guide.

Examples The following isql statement selects columns from a table:

```
SELECT JOB_GRADE, JOB_CODE, JOB_COUNTRY, MAX_SALARY
FROM PROJECT;
```

The next isql statement uses the * wildcard to select all columns and rows from a table:

```
SELECT *
FROM COUNTRIES;
```

The following embedded SQL statement uses an aggregate function to count all rows in a table that satisfy a search condition specified in the WHERE clause:

```
EXEC SQL
    SELECT COUNT (*) INTO :cnt
    FROM COUNTRY
    WHERE POPULATION > 5000000;
```

The next isql statement establishes a table alias in the SELECT clause and uses it to identify a column in the WHERE clause:

```
SELECT C.CITY
FROM CITIES C
WHERE C.POPULATION < 1000000;
```

The following isql statement selects two columns and orders the rows retrieved by the second of those columns:

```
SELECT CITY, STATE
FROM CITIES
ORDER BY STATE;
```

The next isql statement performs a left join:

```
SELECT CITY, STATE_NAME
FROM CITIES C
    LEFT JOIN STATES S ON S.STATE = C.STATE
WHERE C.CITY STARTING WITH 'San';
```

The following isql statement specifies a query optimization plan for ordered retrieval, utilizing an index for ordering:

```
SELECT *
FROM CITIES
    PLAN (CITIES ORDER CITIES_1);
ORDER BY CITY
```

The next isql statement specifies a query optimization plan based on a three-way join with two indexed column equalities:

```
SELECT *
FROM CITIES C, STATES S, MAYORS M
    WHERE C.CITY = M.CITY AND C.STATE = M.STATE
    PLAN JOIN (STATE NATURAL, CITIES INDEX DUPE_CITY,
    MAYORS INDEX MAYORS_1);
```

The next example queries two of the system tables, RDB$CHARACTER_SETS and RDB$COLLATIONS to display all the available character sets, their ID numbers, number of bytes per character, and collations. Note the use of ordinal column numbers in the ORDER BY clause.

```
SELECT RDB$CHARACTER_SET_NAME, RDB$CHARACTER_SET_ID,
        RDB$BYTES_PER_CHARACTER, RDB$COLLATION_NAME
FROM RDB$CHARACTER_SETS JOIN RDB$COLLATIONS
    ON RDB$CHARACTER_SETS.RDB$CHARACTER_SET_ID =
        RDB%COLLATIONS.RDB$CHARACTER_SET_ID
    ORDER BY 1, 4;
```

10 Lösungen

10.1 Lösungen der 1. Übung

10.1.1 Anzeige aller eindeutigen Abteilungsnummern

Geben Sie eine Liste aller eindeutigen Abteilungsnummern aus. Ausgabe absteigend sortiert.

Abfrage:

```
SELECT DISTINCT job_code
FROM employee
ORDER BY job_code DESC
```

Bei dieser Anweisung muss der Zusatz DESC (Descendend, Absteigend) hinzugefügt werden.

Ergebnis:

JOB_CODE
VP
Sales
SRep
PRel
Mngr
Mktg
Finan
Eng
Doc
Dir
CFO
CEO
Admin

10.1.2 Anzeige aller Mitarbeiter in den USA

Geben Sie eine Liste aller Mitarbeiter in den USA aus. Die Ausgabe soll absteigend nach dem Gehalt sortiert werden.

Abfrage:

```
SELECT full_name, salary
FROM employee
WHERE job_country=“USA“
ORDER BY salary DESC
```

Bei dieser Anweisung muss der Zusatz DESC (Descendend, Absteigend hinzugefügt werden.

Ergebnis:

FULL_NAME	SALARY
Bender, Oliver H.	212850
Steadman, Walter	116100
Cook, Kevin	111262,5
MacDonald, Mary S.	111262,5
Nelson, Robert	105900
Lambert, Kim	102750
Young, Bruce	97500
Papadopoulos, Chris	89655
Weston, K. J.	86292,9375
Fisher, Pete	81810,1875
Ramanathan, Ashok	80689,5
Forest, Phil	75060
De Souza, Roger	69482,625
Hall, Stewart	69482,625
Leung, Luke	68805
Young, Katherine	67241,25
Johnson, Leslie	64635
Bishop, Dana	62550
Baldwin, Janet	61637,8125
Johnson, Scott	60000
Williams, Randy	56295
Phong, Leslie	56034,375
Lee, Terri	53793
Burbank, Jennifer M.	53167,5
Page, Mary	48000
Yanowski, Michael	44000
Nordstrom, Carol	42742,5
Green, T.J.	36000
Parker, Bill	35000
Montgomery, John	35000
Guckenheimer, Mark	32000
O'Brien, Sue Anne	31275
Brown, Kelly	27000

10.1.3 Nachnamen und Gehälter

Zeigen die Nachnamen und Gehälter aller Angestellten an. Sortieren Sie das Ergebnis nach aufsteigenden Abteilungsnummer und dann absteigend nach dem Gehalt.

Abfrage:

```
SELECT last_name, salary
FROM employee
ORDER BY emp_no, salary DESC
```

Bei dieser Anweisung wird nach der Abteilungsnummer sortiert, obwohl diese nicht mit angezeigt wird.

Ergebnis:

LAST_NAME	SALARY
Nelson	105900
Young	97500
Lambert	102750
Johnson	64635
Forest	75060
Weston	86292,9375
Lee	53793
Hall	69482,625
Young	67241,25
Papadopoulos	89655
Fisher	81810,1875
Bennet	22935
De Souza	69482,625
Baldwin	61637,8125
Reeves	33620,625
Stansbury	39224,0625
Phong	56034,375
Ramanathan	80689,5
Steadman	116100
Nordstrom	42742,5
Leung	68805
O'Brien	31275
Burbank	53167,5
Sutherland	100914
Bishop	62550
MacDonald	111262,5
Williams	56295
Bender	212850

10.2 Lösungen der 2. Übung

10.2.1 Full-Name mit Alias Namen

Das Attribut „Full_Name“ ist nur virtuell definiert. Es wird automatisch beim Anzeigen erzeugt. Ziel der nächsten Abfrage ist eine Tabelle, die dieses Attribut erzeugt. Betroffen sind aber nur Mitarbeiter, die in den USA arbeiten. Die Überschrift der ersten Spalte sollte Name betragen. Sortiert wird als erstes nach den Abteilungen, dann nach den Namen.

Abfrage:

```
SELECT last_name || ", " || first_name Name, emp_no
FROM employee
WHERE job_country = "USA"
ORDER BY emp_no, 1
```

Die WHERE-Klausel muss vor der ORDER-Klausel geschrieben werden. Folgende Varianten der Sortierung sind gültig:

- ORDER BY 2, 1
- ORDER BY emp_no, Name
- ORDER BY emp_no, 1

Ergebnis:

NAME	EMP_NO
Nelson, Robert	2
Young, Bruce	4
Lambert, Kim	5
Johnson, Leslie	8
Forest, Phil	9
Weston, K. J.	11
Lee, Terri	12
Hall, Stewart	14
Young, Katherine	15
Papadopoulos, Chris	20
Fisher, Pete	24
De Souza, Roger	29
Baldwin, Janet	34
Phong, Leslie	44
Ramanathan, Ashok	45
Steadman, Walter	46
Nordstrom, Carol	52
Leung, Luke	61
O'Brien, Sue Anne	65
Burbank, Jennifer M.	71
Bishop, Dana	83
MacDonald, Mary S.	85
Williams, Randy	94
Bender, Oliver H.	105
Cook, Kevin	107
Brown, Kelly	109
Page, Mary	113
Parker, Bill	114
Yanowski, Michael	127
Johnson, Scott	136
Green, T.J.	138
Montgomery, John	144
Guckenheimer, Mark	145

10.2.2 Welcher Mitarbeiter heisst Robert am Anfang

Gesucht sind die Mitarbeiter, als Anfangsvornamen Robert heißen. Ausgegeben werden sollen der Vor- und der Nachname. Als Überschriften dienen die Bezeichnungen „Vorname“ und „Nachname“.

Abfrage:

Select first_name Vorname, last_name Nachname

```
From employee
WHERE  first_name LIKE "Robert%"
```

Ergebnis:

VORNAME	NACHNAME
Robert	Nelson
Roberto	Ferrari

10.2.3 Mitarbeiter aus dem Jahr 1992 und 1994

Gesucht:
Alle Mitarbeiter, die im Jahr 1992 und 1994 angefangen haben und ein Gehalt größer € 60000,00 haben. Sortiert aufsteigend nach Gehalt.

Abfrage:

```
Select Full_name, emp_no, salary, hire_date
From employee
Where ((hire_date LIKE "%1992%") or (hire_date LIKE "%1994%") )  and (
            salary>60000
        )
ORDER BY 3
```

Ergebnis:

FULL_NAME	EMP_NO	SALARY	HIRE_DATE
Bishop, Dana	83	62550	01.06.1992
Leung, Luke	61	68805	18.02.1992
Sutherland, Claudia	72	100914	20.04.1992
Osborne, Pierre	141	110000	03.01.1994
MacDonald, Mary S.	85	111262,5	01.06.1992
Bender, Oliver H.	105	212850	08.10.1992

Ein weitere Variante wäre die Bestimmung des Monats aus dem Datum mittels der Funktion EXTRACT

```
Select Full_name, emp_no, salary, hire_date
From employee
Where    ( ( EXTRACT (year FROM hire_date )=1992 ) OR
           ( EXTRACT (year FROM hire_date )=1994 ) )
         and (
            salary>60000
         )
ORDER BY 3
```

Alle Bedingungen sind geklammert und zur Übersicht eingerückt.

10.2.4 Codierungsfehler

Die folgende SELECT-Anweisung enthält vier Fehler. Können Sie alle vier finden?

Falsche Abfrage:

```
SELECT emp_no, last_name
salary x 12   ANNUAL SALARAY
from employee
```

Folgende Fehler sind vorhanden:

1) Der Arithemetische Ausdruck muss in Klammern
2) Das x ist ein Muliplikationszeichen
3) Der Alias-Name muss mit einem Underline (_) verbunden werden.
4) Es fehlt ein Komma nach last_name

Korrekte Abfrage:

```
SELECT emp_no, last_name, (salary*12)   ANNUAL_SALARAY
from employee
```

10.3 Lösungen der 3. Übung

10.3.1 Teuerste Rechnung

Gesucht ist der größter Betrag aller offenen Rechnungen des Kunden mit der Nummer 1005.

Abfrage:

```
SELECT     MAX(total_value)
FROM sales
WHERE (order_status="open") and (cust_no=1005)
```

Das Ergebnis lautet: 9000,00

10.3.2 Durchschnittsverdienst in den USA

Gesucht ist der Durchschnittsverdient aller Mitarbeiter in den USA. dabei sind Sie sich nicht sicher, dass in allen Einträgen der Text USA steht. Eventuell existieren Varianten der Groß- und Kleinschreibung.

Abfrage:

```
SELECT     AVG(salary)
FROM employee
```

```
WHERE lower(job_country)="usa"
```

Das Ergebnis lautet: 71068,9185606061

10.3.3 Namen der Manager

Gesucht sind die Namen der Manager.

Abfrage:

```
SELECT     full_name
FROM employee
WHERE lower(job_code)="mngr"
```

Ergebnis:

FULL_NAME
Forest, Phil
Young, Katherine
Papadopoulos, Chris
Williams, Randy

10.3.4 Anzahl der Manager

Gesucht ist die Anzahl der Manager.

Abfrage:

```
SELECT COUNT(emp_no)
FROM employee
WHERE lower(job_code)="mngr"
```

Das Ergebnis lautet: 4

10.3.5 Welcher Mitarbeiter wurde im Mai eingestellt

Gesucht sind die Mitarbeiter, die im Mai eingestellt wurden.

Abfrage:

```
SELECT full_name, hire_date
FROM employee
WHERE extract(month FROM hire_date)=5
```

Ergebnis:

FULL_NAME	HIRE_DATE
Lee, Terri	01.05.1990
Guckenheimer, Mark	02.05.1994

10.3.6 Liste mit neuem Datumsformat

Gesucht ist eine Liste, in der alle Mitarbeiter mit ihren Einstellungsdaten eingetragen sind. Folgendes Datumsformat soll verwendet werden: YYYY:MM:TT. Als Überschriften sollen die Bezeichnungen „Name“ und „Einstellungsdatum“ dienen. Sortiert werden soll nach der Mitarbeiternummer.

Abfrage:

```
SELECT full_name Name, EXTRACT(year FROM hire_date) || ":" ||
        EXTRACT(month FROM hire_date) || ":" ||
        EXTRACT(day FROM hire_date) Einstellungsdatum
FROM employee
ORDER BY emp_no
```

Ergebnis:

NAME	EINSTELLUNGSDATUM
Nelson, Robert	1988:12:28
Young, Bruce	1988:12:28
Lambert, Kim	1989:2:6
Johnson, Leslie	1989:4:5
Forest, Phil	1989:4:17
Weston, K. J.	1990:1:17
Lee, Terri	1990:5:1
Hall, Stewart	1990:6:4
Young, Katherine	1990:6:14
Papadopoulos, Chris	1990:1:1
Fisher, Pete	1990:9:12
Bennet, Ann	1991:2:1
De Souza, Roger	1991:2:18
Baldwin, Janet	1991:3:21
Reeves, Roger	1991:4:25
Stansbury, Willie	1991:4:25
Phong, Leslie	1991:6:3
Ramanathan, Ashok	1991:8:1
Steadman, Walter	1991:8:9
Nordstrom, Carol	1991:10:2
Leung, Luke	1992:2:18
O'Brien, Sue Anne	1992:3:23
Burbank, Jennifer M.	1992:4:15
Sutherland, Claudia	1992:4:20
Bishop, Dana	1992:6:1

MacDonald, Mary S.	1992:6:1
Williams, Randy	1992:8:8
Bender, Oliver H.	1992:10:8
Cook, Kevin	1993:2:1
Brown, Kelly	1993:2:4
Ichida, Yuki	1993:2:4
Page, Mary	1993:4:12
Parker, Bill	1993:6:1
Yamamoto, Takashi	1993:7:1
Ferrari, Roberto	1993:7:12
Yanowski, Michael	1993:8:9
Glon, Jacques	1993:8:23
Johnson, Scott	1993:9:13
Green, T.J.	1993:11:1
Osborne, Pierre	1994:1:3
Montgomery, John	1994:3:30
Guckenheimer, Mark	1994:5:2

Obwohl die Mitarbeiternummer nicht ausgegeben wird, darf danach sortiert werden.

10.4 Lösungen der 4. Übung

10.4.1 Extrema der Gehälter

Zeigen Sie das niedrigste Gehalt, das höchste Gehalt, die Summe der Gehälter und das Durchschnittsgehalt für alle Mitarbeiter an. Nennen Sie die Spalten „Minimum“, „Maximum“, „Summe“ und „Durchschnitt“.

Abfrage:

```
SELECT    MIN(salary) Minimum, MAX(salary) Maximum,
          SUM(salary)Summe, AVG(salary) Durchschnitt
FROM employee
```

Ergebnis:

MINIMUM	**MAXIMUM**	**SUMME**	**DURCHSCHNITT**
22935	99000000	115522468	2750534,952

10.4.2 Extrema der Gehälter pro Job_Gruppe

Zeigen Sie das niedrigste Gehalt, das höchste Gehalt, die Summe der Gehälter und das Durchschnittsgehalt für alle Mitarbeiter an. Nennen Sie die Spalten „Minimum“, „Maximum“, „Summe“ und „Durchschnitt“. Gruppieren Sie die Mitarbeiter nach den Job_Code.

Abfrage:

```
SELECT    job_code ,
          MIN(salary) Minimum, MAX(salary) Maximum,
          SUM(salary)Summe, AVG(salary) Durchschnitt
FROM employee
GROUP BY job_code
```

Ergebnis:

JOB_CODE	**MINIMUM**	**MAXIMUM**	**SUMME**	**DURCHSCHNITT**
Admin	22935	53793	135003	33750,75
CEO	212850	212850	212850	212850
CFO	116100	116100	116100	116100
Dir	111262,5	111262,5	111262,5	111262,5
Doc	60000	60000	60000	60000
Eng	32000	6000000	6829208,25	455280,55
Finan	69482,625	69482,625	69482,625	69482,625
Mktg	64635	64635	64635	64635
Mngr	56295	89655	288251,25	72062,8125
PRel	42742,5	42742,5	42742,5	42742,5
SRep	44000	99000000	107280511,9	13410063,99
Sales	33620,625	61637,8125	95258,4375	47629,21875
VP	105900	111262,5	217162,5	108581,25

10.4.3 Anzahl der Ingenieure

Ermitteln Sie die Anzahl der Ingenieure, ohne alle in einer Liste anzuzeugen. Geben Sie nur eine Zeile mit der Anzahl aus. Als Überschrift wählen Sie „Ingenieure“

Abfrage:

```
SELECT    Count(job_code) Ingenieur
FROM employee
WHERE job_code="Eng"
```

Ergebnis:

INGENIEURE
15

10.4.4 Differenz der Gehälter

Ermitteln Sie die Differenz zwischen dem größten und dem kleinsten Gehalt. Als Überschrift wählen Sie „Differenz“

Abfrage:

```
SELECT  Max(salary)-Min(salary) Differenz
FROM employee
```

Ergebnis:

DIFFERENZ
98977065

10.4.5 Statistik der Abteilungen

Erstellen Sie eine Abfrage, um für jede Abteilung die Abteilungsnummer, die Mitarbeiterzahl und das jeweilige Durchschnittsgehalt anzuzeigen. Als Überschrift wählen Sie „Abt_Nr“, „Anz_Ang“ und „Durchschnitts_Gehalt“.Sortiert werden soll absteigend nach dem Gehalt.

Abfrage:

```
SELECT dept_no Abt_Nr ,  count(emp_no) Anz_Ang,  AVG(salary) Durchschnitts_Gehalt
FROM employee
GROUP BY dept_no
ORDER BY AVG(salary) DESC;
```

Ergebnis:

ABT_NR	**ANZ_ANG**	**DURCHSCHNITTS_GEHALT**
125	1	99000000
115	2	6740000
123	1	390500
000	2	133321,5
121	1	110000
140	1	100914
130	2	94521,46875
900	2	92791,3125
100	2	77631,25
671	3	73155,0625
670	2	71268,75
621	4	69184,875
600	2	66450
110	2	65221,40625

623	5	57551,65
180	2	53688,75
622	3	53409,16667
672	2	45647,5
120	3	31926,5625

10.4.6 Welche Personen haben ein größeres Gehalt als das Durchschnittsgehalt?

```
SELECT last_name
FROM employee
WHERE salary > (
                    SELECT avg(salary)
                    FROM employee
                )
```

10.4.7 Mitarbeiterabfrage

Welche Mitarbeiter sind in der gleichen Abteilung wie der Mitarbeiter mit dem größten Gehalt?

```
SELECT last_name, job_code, salary
FROM employee
WHERE dept_no = (
                    SELECT dept_no
                    FROM employee
                    WHERE salary =
                                (
                                    SELECT MAX(salary)
                                    FROM employee
                                )
                    );
```

10.4.8 Welche Abteilungen liegen über den Durchschnittsgehalt?

Das Durchschnittsgehalt wird über alle Mitarbeiter bestimmt.

```
SELECT dept_no, avg(salary)
FROM employee
GROUP BY dept_no
HAVING AVG(salary) > (
                        SELECT AVG(salary)
                        FROM employee
                    )
```

10.4.9 Welche Abteilungen liegen über den Durchschnittsgehalt?

Das Durchschnittsgehalt wird über alle Mitarbeiter bestimmt.

```
SELECT dept_no, avg(salary)
FROM employee
GROUP BY dept_no
HAVING AVG(salary) > (
                        SELECT AVG(salary)
                        FROM employee
                    )
```

11 Indexverzeichnis

A

B

C

D

E

F

G

L

M

N

O

S

T

U