

Oracle und SQL

Gesamtinhaltsverzeichnis

1 Das relationale Datenbanksystem ORACLE	1-3
1.1 Einführung in das relationale Datenbanksystem.....	1-3
1.1.1 Begriffsdefinitionen	1-5
1.1.2 Drei-Ebenen-Architektur eines DBMS.....	1-11
1.2 Dr. Codds 12 Regeln	1-14
1.2.1 Basisregeln (Regel 0 und 12)	1-14
1.2.2 Strukturelle Regeln (Regel 1 und 6).....	1-15
1.2.3 Integritätsregeln (Regel 3 und 10)	1-15
1.2.4 Bearbeitungsregeln (Regel 2, 4, 5 und 7)	1-16
1.2.5 Datenunabhängigkeitsregeln (Regel 8, 9 und 11).....	1-17
1.3 Design, Entwicklung und Implementierung von Datenbanken.....	1-18
1.3.1 Datenbankdesign	1-19
1.3.2 Entity-Relationship Modell (ERM)	1-21
1.3.3 Physisches Modell	1-35
1.4 Relationale Algebra	1-36
1.5 Die Sprache SQL.....	1-38
1.5.1 DDL-Befehle	1-38
1.5.2 DML-Befehle.....	1-38
1.5.3 DCL-Befehle	1-38
1.5.4 Transaktionsbefehle.....	1-38
1.5.5 SQL-Befehlsauflistung	1-39
1.6 Data Dictionary	1-41
1.6.1 Aufteilung des Data Dictionary.....	1-43
2 Eingabeoberflächen.....	2-3
2.1 SQL*Plus Grundlagen	2-3
2.1.1 Leistungsmerkmale.....	2-3
2.1.2 Starten von SQL*Plus	2-4
2.1.3 Befehlspuffer und Editoraufruf	2-7
2.1.4 Umgebungsparameter	2-9
2.1.5 Befehlsdateien	2-13
2.1.6 Spool-Dateien	2-15
2.1.7 Weitere Befehle	2-17
2.2 SQL Developer	2-19
2.2.1 Connections.....	2-19

2.2.2	SQL Worksheet	2-21
2.2.3	Output-Möglichkeiten des SQL Developer.....	2-22
3	SELECT Anweisungen mit einer Tabelle	3-3
3.1	EBNF (Erweiterte Backus-Naur-Form)	3-4
3.2	Die einfache SELECT Anweisung	3-8
3.2.1	Kürzester SELECT.....	3-8
3.2.2	Aufbau der FROM-Klausel.....	3-9
3.2.3	Spaltenauswahl und Spaltenreihenfolge.....	3-10
3.2.4	Qualifizierte Angabe	3-12
3.2.5	Ausblendung doppelter Einträge.....	3-14
3.2.6	Verwendung von Aliasnamen	3-17
3.3	Operatoren im SELECT-Statement	3-20
3.3.1	NULL-Werte	3-22
3.3.2	Die NVL-Funktion	3-23
3.3.3	Die COALESCE-Funktion.....	3-25
3.3.4	Verwendung arithmetischer Operatoren	3-27
3.3.5	Der Verkettungsoperator.....	3-28
3.4	Die WHERE-Klausel	3-30
3.5	Vergleichsprädikate	3-32
3.5.1	Direkte Vergleichsprädikate	3-34
3.5.2	Intervallprädikat	3-35
3.5.3	Platzhaltersymbole (wildcard character)	3-37
3.5.4	Ähnlichkeitsprädikat.....	3-37
3.5.5	NULL-Test Prädikat.....	3-38
3.5.6	IN Prädikat	3-39
3.5.7	ALL-ANY-SOME Prädikate.....	3-41
3.5.8	EXISTS Prädikat.....	3-43
3.6	Die ORDER BY-Klausel.....	3-44
3.7	Zeilen limitieren	3-47
3.8	Datentypen	3-49
3.8.1	Datentypen	3-49
4	Funktionen.....	4-3
4.1	Arithmetische Funktionen	4-3
4.2	Zeichenkettenfunktionen	4-6
4.2.1	Groß- / Kleinschreibung:.....	4-6
4.2.2	Zeichen vom Rand “abschneiden“	4-7
4.2.3	Weitere Zeichenkettenfunktionen	4-8
4.2.4	Konvertierungsfunktionen	4-10

4.2.5	Datumsfunktionen	4-13
4.2.6	Die DECODE-Funktion.....	4-15
4.2.7	Reguläre Ausdrücke	4-17
4.2.8	Zusammenfassung Single Row Funktionen.....	4-27
4.3	Gruppenfunktionen	4-29
4.4	Die GROUP BY-Klausel	4-31
4.4.1	GROUP BY mit ROLLUP- und CUBE-Operatoren	4-35
4.5	Die HAVING-Klausel	4-39
5	SELECT Anweisungen mit mehreren Tabellen.....	5-3
5.1	JOIN-Operation	5-3
5.1.1	Kartesisches Produkt.....	5-4
5.1.2	JOIN-Operation Allgemein	5-8
5.1.3	Definition der JOIN-Operation in SQL.....	5-10
5.1.4	INNER JOIN-Operation.....	5-12
5.1.5	OUTER JOIN-Operation	5-14
5.1.6	LEFT OUTER JOIN	5-15
5.1.7	RIGHT OUTER JOIN	5-17
5.1.8	FULL OUTER JOIN.....	5-19
5.1.9	Der EQUI vs. NON EQUI JOIN	5-21
5.1.10	Der SELF JOIN	5-23
5.1.11	Der CROSS JOIN.....	5-25
5.1.12	Der NATURAL JOIN	5-25
5.1.13	Der EQUI JOIN mit USING-Klausel	5-27
5.2	Unterabfragen.....	5-29
5.2.1	Nicht korrelierte Unterabfragen.....	5-30
5.2.2	Korrelierte Unterabfragen	5-31
5.2.3	Verwendung von Single Row Vergleichsoperatoren	5-33
5.2.4	Verwendung mehrzelliger Vergleichsoperatoren	5-36
5.3	Pivot und Unpivot	5-45
5.3.1	Pivot.....	5-45
5.3.2	Unpivot.....	5-47
5.4	Hierarchische (rekursive) Abfragen	5-49
5.4.1	Formatieren hierarchischer Berichte	5-53
5.4.2	Einzelne Zweige ausblenden	5-55
5.4.3	Weitere Optionen zu hierarchischen Abfragen	5-57
5.5	SET-Operatoren	5-60
6	Data Manipulation Language (DML)	6-3
6.1	INSERT - Datensätze einfügen.....	6-5

6.1.1	Verwendung der VALUES-Klausel:.....	6-6
6.1.2	Verwendung einer Unterabfrage:.....	6-6
6.2	UPDATE - Datensätze aktualisieren	6-8
6.3	DELETE - Datensätze löschen	6-10
6.4	Das MERGE-Statement.....	6-12
6.5	Das Transaktionskonzept	6-14
6.5.1	SAVEPOINT	6-15
6.5.2	Die SET TRANSACTION-Anweisung	6-16
7	Data Definition Language (DDL)	7-3
7.1	Objekt TABLE	7-5
7.1.1	Der CREATE TABLE Befehl	7-5
7.1.2	CREATE TABLE und unsichtbare Spalten	7-6
7.1.3	Erstellung einer neuen Tabelle aus einer bestehenden Tabelle	7-8
7.1.4	CREATE TABLE – Neuerungen	7-10
7.1.5	CREATE TABLE – Neuerungen	7-11
7.1.6	Der ALTER TABLE Befehl.....	7-12
7.1.7	ALTER TABLE – Neuerungen	7-14
7.1.8	Der DROP TABLE Befehl	7-15
7.1.9	Der TRUNCATE TABLE Befehl	7-16
7.1.10	Der RENAME Befehl.....	7-18
7.1.11	Umbenennung von Spalten und Constraints	7-19
7.1.12	Spaltenlänge ändern.....	7-20
7.2	Objekt INDEX	7-24
7.2.1	Der CREATE INDEX Befehl	7-26
7.2.2	Der ALTER INDEX Befehl.....	7-27
7.2.3	Neuerungen ab 12c bezüglich Indizes	7-28
7.2.4	Der DROP INDEX Befehl	7-29
7.3	Objekt VIEW	7-30
7.3.1	Der CREATE VIEW Befehl	7-32
7.3.2	Views und DML-Befehle	7-34
7.3.3	Inline Views.....	7-35
7.3.4	Der ALTER VIEW Befehl	7-36
7.3.5	Der DROP VIEW Befehl.....	7-37
7.3.6	Der RENAME Befehl.....	7-37
7.3.7	Top-n / Bottom-n-Analysen	7-38
7.3.8	Zeilen limitieren.....	7-40
7.4	Objekt SEQUENCE	7-42

7.4.1	Der CREATE SEQUENCE Befehl.....	7-42
7.4.2	Neue Parameter	7-44
7.4.3	Pseudospalten der SEQUENCE.....	7-45
7.4.4	Der ALTER SEQUENCE Befehl.....	7-47
7.4.5	Der DROP SEQUENCE Befehl.....	7-48
7.4.6	Der RENAME Befehl.....	7-50
7.4.7	Die IDENTITY Klausel	7-51
7.5	Objekt SYNONYM	7-54
7.5.1	Der CREATE SYNONYM Befehl.....	7-54
7.5.2	Ändern eines SYNONYMS.....	7-55
7.5.3	Der DROP SYNONYM Befehl	7-55
7.5.4	Der RENAME Befehl.....	7-56
7.6	Constraints	7-57
7.6.1	Der NOT NULL-Constraint	7-64
7.6.2	Der UNIQUE-Constraint.....	7-64
7.6.3	Der PRIMARY KEY Constraint	7-64
7.6.4	Der FOREIGN KEY Constraint.....	7-65
7.6.5	Der CHECK Constraint.....	7-67
7.6.6	Constraints hinzufügen und löschen	7-67
7.6.7	Constraintstatus ändern	7-68
7.6.8	Kaskadierende Constraints	7-71
8	Datenschutz und DCL-Befehle.....	8-3
8.1	Konzept	8-3
8.2	Datenbankbenutzer und SCHEMA.....	8-5
8.3	Benutzerverwaltung	8-7
8.4	Befehle zur Benutzerverwaltung	8-9
8.4.1	ALTER USER Befehl	8-13
8.4.2	DROP USER Befehl	8-14
8.4.3	Data Dictionary Views zur Benutzerverwaltung	8-15
8.5	Privilegien	8-16
8.5.1	Systemprivilegien.....	8-16
8.5.2	Objektprivilegien	8-18
8.6	Befehle zur Rechteverwaltung (DCL)	8-22
8.6.1	Der GRANT Befehl für Systemprivilegien	8-22
8.6.2	Der GRANT Befehl für Objektprivilegien.....	8-24
8.6.3	Der REVOKE Befehl	8-25
8.6.4	Das Rollenkonzept.....	8-28
8.6.5	Der CREATE ROLE Befehl.....	8-31

8.6.6	Der ALTER ROLE Befehl.....	8-32
8.6.7	Der SET ROLE Befehl.....	8-32
8.6.8	Der DROP ROLE Befehl.....	8-34
8.6.9	Setzen von Default Rollen	8-34
8.6.10	Vordefinierte Rollen	8-36
8.7	Benutzerprofile	8-39
8.7.1	Der CREATE PROFILE Befehl	8-41
8.7.2	Der ALTER PROFILE Befehl	8-44
8.7.3	Der DROP PROFILE Befehl	8-44
8.7.4	Data Dictionary Views für Profiles.....	8-45
9	Analytische Funktionen.....	9-3
9.1	Aggregatfunktionen als analytische Funktion	9-3
9.2	LAG und LEAD	9-7
9.3	Rekursive Berechnungen	9-10
9.4	Ranking Funktionen (RANK, DENSE_RANK)	9-11
9.5	LISTAGG.....	9-13
10	Anhang A.....	10-3
10.1	Systemarchitektur.....	10-3
10.2	Die Komponenten der ORACLE-Datenbank	10-5
10.3	Die logische und physische ORACLE-Struktur.....	10-8
10.3.1	Logische Datenbankstruktur	10-8
10.3.2	Physische Datenbankstruktur	10-9
10.4	Informationen zu ORACLE – Dateien auf Betriebssystemebene	10-11
11	Anhang B SQL*Plus.....	11-3
11.1	Editierbefehle	11-3
11.2	Austauschvariablen	11-4
11.3	Neuerungen.....	11-4
12	Anhang C Syntaxdiagramme	12-3
12.1	Vorbemerkung	12-3
12.2	CREATE TABLE / ALTER TABLE.....	12-3
12.3	CREATE INDEX / ALTER INDEX.....	12-10
12.4	CREATE VIEW.....	12-13
12.5	ALTER SEQUENCE	12-15
12.6	SELECT-Anweisung	12-17
12.7	DML-Befehle	12-20
12.7.1	INSERT.....	12-20
12.7.2	UPDATE	12-21

12.7.3 DELETE	12-23
13 Literaturverzeichnis	13-3
Gesamtindex.....	IDX-1

1

Das relationale Datenbanksystem ORACLE

1.1	Einführung in das relationale Datenbanksystem.....	1-3
1.1.1	Begriffsdefinitionen	1-5
1.1.2	Drei-Ebenen-Architektur eines DBMS.....	1-11
1.2	Dr. Codds 12 Regeln	1-14
1.2.1	Basisregeln (Regel 0 und 12)	1-14
1.2.2	Strukturelle Regeln (Regel 1 und 6).....	1-15
1.2.3	Integritätsregeln (Regel 3 und 10)	1-15
1.2.4	Bearbeitungsregeln (Regel 2, 4, 5 und 7)	1-16
1.2.5	Datenunabhängigkeitsregeln (Regel 8, 9 und 11).....	1-17
1.3	Design, Entwicklung und Implementierung von Datenbanken.....	1-18
1.3.1	Datenbankdesign	1-19
1.3.2	Entity-Relationship Modell (ERM)	1-21
1.3.3	Physisches Modell	1-35
1.4	Relationale Algebra	1-36
1.5	Die Sprache SQL.....	1-38
1.5.1	DDL-Befehle	1-38
1.5.2	DML-Befehle.....	1-38
1.5.3	DCL-Befehle	1-38
1.5.4	Transaktionsbefehle.....	1-38
1.5.5	SQL-Befehlsauflistung	1-39
1.6	Data Dictionary	1-41

1.6.1 Aufteilung des Data Dictionary..... 1-43

1 Das relationale Datenbanksystem ORACLE

1.1 Einführung in das relationale Datenbanksystem

Die Theorie des relationalen Datenmodells wurde in den Jahren 1968-1973 von **Dr. E. F. Codd** entwickelt und veröffentlicht. Es zeichnet sich durch eine extrem einfache Strukturierung aus. Daten und Beziehungen werden durch **Relationen** oder, vereinfacht gesagt, Tabellen dargestellt. Jede **Zeile** in einer Tabelle spezifiziert ein bestimmtes Objekt dieser Tabelle, jede **Spalte** repräsentiert eine bestimmte Eigenschaft dieser Objekte.

In der Mitte der siebziger Jahre (1973 – 1979) entstand aus diesem Modell von Dr. Codd ein Forschungsprojekt von IBM. Entwickelt wurde dort der erste Prototyp, "System R". System R benutzte keine Relationale Algebra (**prozedural!**) sondern basierte auf der Datensprache SEQUEL (**mengenorientiert**), welche der Vorläufer von **SQL** 1974 war.

Als zweiter Prototyp wurde Anfang der 70er **Ingres** entwickelt.

Als eines der ersten kommerziellen relationalen Datenbanksysteme (RDBMS) neben Ingres wurde 1979 **ORACLE** entwickelt. Ein Jahr später zog IBM mit SQL/DS für DOS/VSE, VM/CMS nach. Im Jahr 1983 stellte IBM sein bis heute bekanntestes Datenbanksystem der Welt zur Verfügung: **DB2** (erst für MVS).

SQL als „Datenbanksprache“ entwickelte sich zum de facto **Standard** für relationale Datenbanksysteme und wurde **1986** als Standard des American National Standards Institute (ANSI) verabschiedet. Dieser von allen akzeptierte Standard verhalf den relationalen Datenbanksystemen zu ihrem fulminanten Durchbruch. In den Jahren **1989, 1992, 1999 und 2003** wurde dieser Standard erweitert.

Die Tabelle bildet den Grundbaustein eines relationalen Aufbaus. Eine **relationale Datenbank** wird vom Benutzer als eine **Menge von Tabellen** wahrgenommen. Sie kann aus beliebig vielen Tabellen bestehen. Es ist jedoch mindestens eine Tabelle zwingend notwendig.

Die einzelnen Tabellen setzen sich aus keiner, einer oder mehreren **Zeilen** und einer oder mehreren **Spalten** zusammen, wobei die Anzahl der Zeilen beliebig ist; die Anzahl der Spalten ist bei ORACLE auf 1000 limitiert. Eine Tabelle kann im Extremfall auch nur aus einer Zeile und einer Spalte bestehen.

Das relationale Modell



- Theorie entwickelt von Dr. E. F. Codd
- Aufbau einer DB aus Relationen/Tabellen
- SQL als Standard-DB-Sprache
- Historie:
 - 1968 – 73: Codd entwickelt die relationale Theorie
 - 1969 – 70: Codd: "A Relational Model of Data for Large Shared Data Banks", IBM San Jose Research Laboratory
 - 1973: Zwei Prototypen werden Entwickelt:
 - SYSTEM R von IBM
 - Ingres
 - 1974: SEQUEL in SYSTEM R (wird Ende 70er in SQL umbenannt)
 - 1979: erste kommerzielle Datenbanken:
 - Ingres
 - ORACLE
 - 1980: SQL/DS für DOS/VSE, VM/CMS
 - 1983: DB2 für MVS von IBM
 - 1986: ANSI verabschiedet ersten SQL Standard
 - 1989: ISO/ANSI Standard wird um Constraints und Embedded-SQL erweitert
 - 1992: Erweiterter SQL92 Standard von ISO/ANSI
 - 1999: SQL-3 Standard (OO, Trigger und Stored Procedures)
 - 2003: SQL03 Standard

1.1.1 Begriffsdefinitionen

Datenbank	Als Datenbank (DB) wird die eigentliche Datenbasis, vorliegend in Form von Dateien, bezeichnet. Die Datenbank ist der passive Teil des Datenbanksystems, eine Sammlung von Daten, die von sich aus unveränderlich bleiben und nur durch externe Aktivitäten verändert werden können.
Tabelle	Eine Tabelle ist eine in Spalten und Zeilen aufgeteilte organisierte Sammlung von Daten. Alle Daten einer Datenbank befinden sich ausschließlich in Tabellen!
View	(auch Abfrage, Sicht oder Query genannt) Eine "virtuelle" Tabelle. Das View greift auf eine oder mehrere Quellen (Tabellen oder Views) zu und filtert jedes Mal aufs Neue die Daten aus den Quellen. Das View selbst speichert KEINE Daten. Nach außen sieht ein View wie eine Tabelle aus!
Zeile	Eine Zeile wird auch als Datensatz bezeichnet. Sie enthält alle Informationen über ein durch die Tabelle beschriebenes Objekt. Zum Beispiel enthält eine Tabelle <code>Kunde</code> jeweils in einer Zeile die Daten zu einem bestimmten Kunden.
Spalte	Eine Spalte definiert eine bestimmte Eigenschaft/Attribut eines Objektes. Zum Beispiel ist <code>KundenName</code> eine Spalte in der Tabelle <code>Kunde</code> . Jede Spalte hat einen Namen und einen zugrundeliegenden Datentyp (Text, Zahl, Datum, etc.).
Feld	Ein Feld (eine Zelle) beinhaltet genau eine skalare Eigenschaft einer bestimmten Zeile. Zum Beispiel ist der <code>Kundenname „Müller“</code> ein Feld in der Zeile mit allen Daten zu dem Kunden Müller. Ein Feld kann auch mit NULL gefüllt werden.
NULL	bedeutet, dass der Wert in der Datenbank nicht definiert ist. Dies kann bedeuten: <ul style="list-style-type: none">– der Wert existiert nicht,– der Wert ist nicht bekannt,– der Wert ist nicht erfasst.

- Relation** ist der mathematische Begriff für Tabelle.
- Tupel** (auch Vektor) ist der mathematische Begriff für Zeile.
- Attribut** ist der mathematische Begriff für Spalte.

Begriffsdefinition

- Tabellen setzen zusammen sich aus:
 - Zeilen
 - keine
 - eine
 - mehrere
 - Spalten
 - eine
 - mehrere
- atomare (skalare)
Werte in den Feldern



Datenbank

Tabelle	
	Spalte

Tabelle

Tabelle	
Zeile	

Feld

2.5.0122 © Integrata Cegos GmbH ORACLE und SQL

Abb. 1-1: Grundstrukturen einer relationalen Datenbank

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17.12.80	800		20
7499	ALLEN	SALESMAN	7698	20.02.81	1600	300	30
7521	WARD	SALESMAN	7698	22.02.81	1250	500	30
7566	JONES	MANAGER	7839	02.04.81	2975		20
7654	MARTIN	SALESMAN	7698	28.09.81	1250	1400	30
7698	BLAKE	MANAGER	7839	01.05.81	2850		30
7782	CLARK	MANAGER	7839	09.06.81	2450		10
7788	SCOTT	ANALYST	7566	19.04.87	3000		20
7839	KING	PRESIDENT		17.11.81	5000		10
7844	TURNER	SALESMAN	7698	08.09.81	1500	0	30
7876	ADAMS	CLERK	7788	23.05.87	1100		20
7900	JAMES	CLERK	7698	03.12.81	950		30
7902	FORD	ANALYST	7566	03.12.81	3000		20
7934	MILLER	CLERK	7782	23.01.82	1300		10

Abb. 1-2: Eine Tabelle

Tabelle								
EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	
7369	SMITH	CLERK	7902	17.12.80	800		20	
7499	ALLEN	SALESMAN	7698	20.02.81	1600	300	30	
7521	WARD	SALESMAN	7698	22.02.81	1250	500	30	
7566	JONES	MANAGER	7839	02.04.81	2975		20	
7654	MARTIN	SALESMAN	7698	28.09.81	1250	1400	30	
7698	BLAKE	MANAGER	7839	01.05.81	2850		30	
7782	CLARK	MANAGER	7839	09.06.81	2450		10	
7788	SCOTT	ANALYST	7566	19.04.87	3000		20	
7839	KING	PRESIDENT		17.11.81	5000		10	
7844	TURNER	SALESMAN	7698	08.09.81	1500	0	30	
7876	ADAMS	CLERK	7788	23.05.87	1100		20	
7900	JAMES	CLERK	7698	03.12.81	950		30	
7902	FORD	ANALYST	7566	03.12.81	3000		20	
7934	MILLER	CLERK	7782	23.01.82	1300		10	

Das Konzept eines Datenbankmanagementsystems (DBMS)

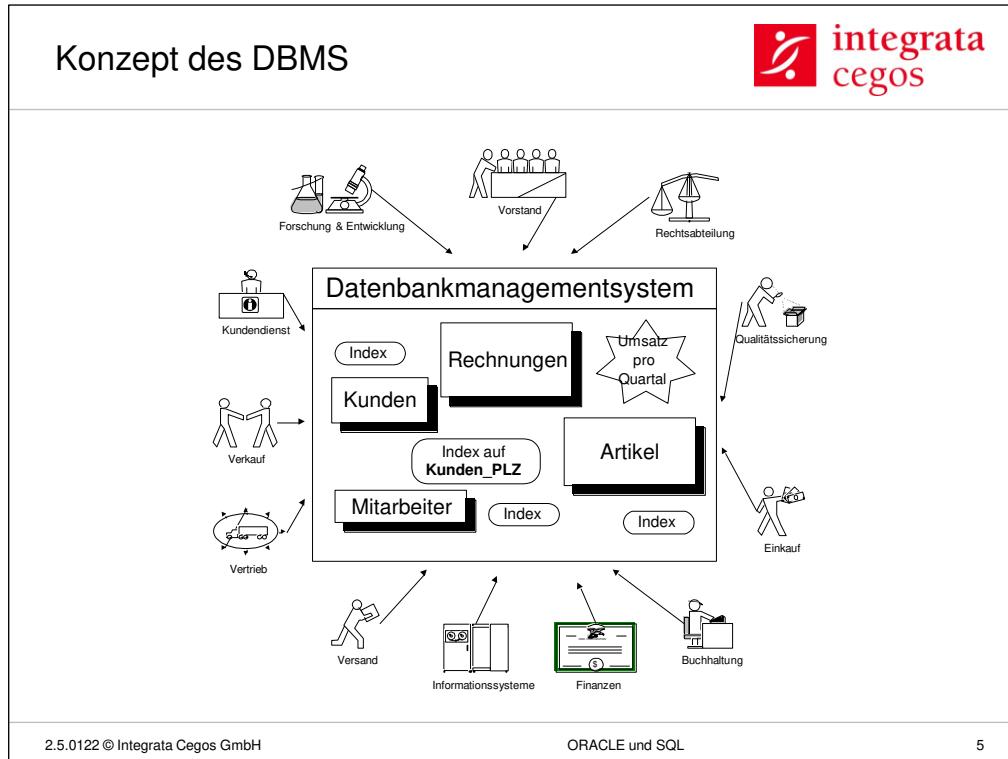


Abb. 1-3: Das Konzept des DBMS

Das Datenbankmanagementsystem (DBMS) kümmert sich um die Speicherung und Verwaltung der Daten in einer Datenbank. Eine Datenbank ist ein Container für die Daten. Alle Nutzer einer Datenbank wenden sich an das DBMS (genauer gesagt an die Werkzeuge des DBMS) während ihrer Arbeit.

Direkter Zugriff auf die physischen Daten ist nicht erlaubt. Der Benutzer kann dem DBMS nur mitteilen was für Daten angezeigt, bearbeitet oder gelöscht werden sollen. Das DBMS evaluiert **selbstständig** den besten Weg, um diese Daten auf dem physischen Speicher zu finden.

Diese Art des Zugriffs wird auch die "**beschreibende**" Art genannt. Deswegen bezeichnet man SQL als **beschreibende mengenorientierte** Sprache.

Vorteile der relationalen Datenbank (RDB) und der relationalen Datenbankmanagementsysteme (RDBMS)

- Datenbankmanagementsysteme (DBMS) ermöglichen es den Benutzern, alle Daten zentralisiert zu verwalten.
- Die Daten können von verschiedenen Anwendern gleichzeitig benutzt werden.
- Transaktionen können gewährleistet werden (ACID Regeln).
- Redundanzen können eingeschränkt und kontrolliert werden.
- Inkonsistenzen können vermieden werden.
- Datenintegrität (Widerspruchsfreiheit) kann aufrechterhalten werden.
- Datenschutz kann durchgesetzt werden.
- Standards können durchgesetzt werden.
- Der relationale Ansatz bietet eine einzige konzeptionelle Sicht auf alle Daten in der Datenbank.
- Der Anwender muss sich nicht um die physische Speicherung der Daten kümmern. Dies übernimmt das DBMS.
- Es existiert eine einzige Notation oder Datensprache, sowohl für den Anwender als auch den Entwickler und Administrator, mit der alle Ebenen und Funktionen der Datenbank angesprochen werden können.

Beim Zugriff auf die gewünschten Daten muss man also nicht mehr wissen, "WO" und "WIE" die Daten gespeichert werden, sondern nur noch "**WAS**" man haben will. Dadurch wird auch für den Endanwender die Datenbank transparent.

Die Regeln der realen Welt können mit Hilfe der Integritätsregeln durchgesetzt werden. Die Daten sind aus der Sicht des Systems konsistent, sofern die definierten Integritätsregeln eingehalten worden sind.

Vorteile der Relationalität

- Zentralisierte Datenverwaltung
- Zugriff durch mehrere Benutzer gleichzeitig
- Gewährleistung der Transaktionen
- Vermeidung der Redundanzen
- Vermeidung von Inkonsistenzen
- Gewährleistung der Datenintegrität
- Gewährleistung des Datenschutzes
- Durchsetzung von Standards
- Benutzersicht auf die Daten nur über die Relationen/Tabellen
- Keine Sorgen um physische Speicherung
- Eine einheitliche "Datenbanksprache" für alle Benutzer

1.1.2 Drei-Ebenen-Architektur eines DBMS

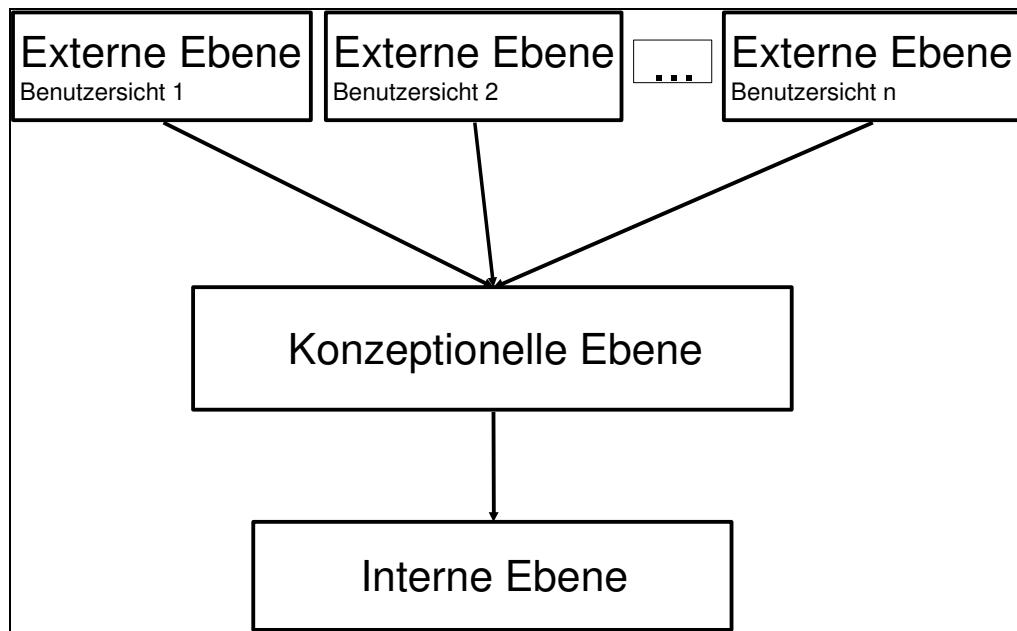


Abb. 1-4: Das Modell der Drei-Ebenen-Architektur

1.1.2.1 Interne Ebene

Die **interne Ebene** (auch **physische Ebene** genannt) kümmert sich um die Verteilung der Daten auf die der Datenbank zur Verfügung stehenden Speichermedien (Platte(n), CD, DVD, USB-Sticks, etc.) sowie um die für den jeweiligen Zweck am besten geeigneten physischen Datenstrukturen (Listen, Bäume, Plattenindexierung, etc.). Der DB-Benutzer kann diese Ebene nicht beeinflussen. Sie ist direkt in dem DBMS integriert.

1.1.2.2 Konzeptionelle Ebene

Die Konzeptionelle Ebene zeigt alle Daten der Datenbank in relationaler Form an. In dieser Ebene werden alle notwendigen Objekte der Datenbank (Tabellen, Views, etc.) angelegt und verwaltet. Insbesondere die Regeln, welche in der Datenbank durchgesetzt werden sollen, werden hier definiert.

1.1.2.3 Externe Ebene

Sie zeigt die Daten jedem Benutzer (Sekretärin, Vertrieb, Lagermitarbeiter) entsprechend seiner Bedürfnisse an und wird häufig in einer externen Programmiersprache (C, JAVA, C#, VB, ...) programmiert.

1.1.2.4 Datenunabhängigkeit

Es gibt zwei Arten der Unabhängigkeit:

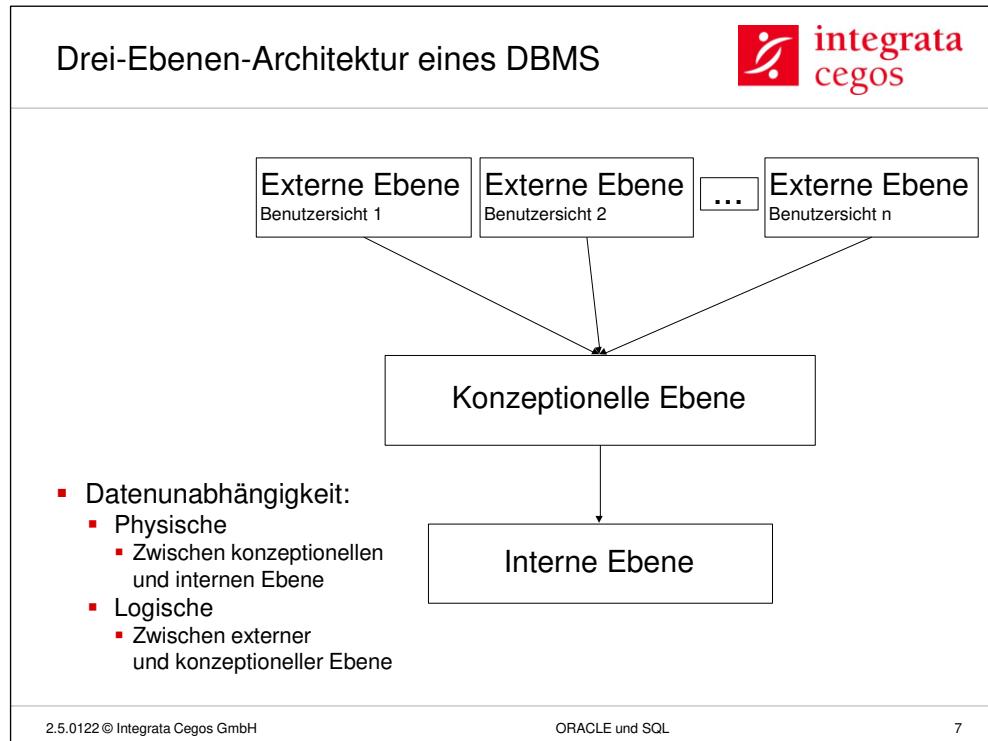
- Physische und
- Logische.

- **Physische Datenunabhängigkeit** garantiert den unbeeinflussten Zugriff auf die Daten, falls die physischen Datenstrukturen oder Zugriffsmechanismen auf selbige geändert wurden (z.B. Umstellung von FAT / FAT32 auf NTFS unter Windows). Die Anwender auf der konzeptionellen oder externen Ebene dürfen von diesen Änderungen nichts merken. Die physische Datenunabhängigkeit wird zwischen der konzeptionellen und der internen Ebene durchgesetzt. Um die Lösung dieses Problems kümmert sich das DBMS (oder der Hersteller des DBMS), und zwar im Allgemeinen relativ erfolgreich.
- **Logische Datenunabhängigkeit** garantiert einen fehlerfreien Zugriff der Anwender, falls Veränderungen an der logischen Struktur der Datenbank auf der konzeptionellen Ebene aufgetreten sind wie beispielsweise: die Tabelle Kunden wird in Personen umbenannt und bekommt eine neue Spalte Gewicht. Die logische Datenunabhängigkeit muss zwischen der externen und konzeptionellen Ebene durchgesetzt werden. Dafür ist der Pfleger der konzeptionellen Ebene zuständig.

Logische Datenunabhängigkeit ist ein weit schwierigeres Problem als die physische.

Falls an der physischen Ebene etwas geändert wird, kann der Datenbankhersteller einfach eine angepasste Version der Datenbank herausbringen, und es funktioniert alles (theoretisch) wieder reibungslos.

Falls jedoch an der logischen Ebene Veränderungen vorgenommen wurden, müssen meist mehrere interne Abteilungen eines Unternehmens bewegt werden und mehrere externe Programme mühsam angepasst werden.



1.2 Dr. Codds 12 Regeln

Ein DBMS ist nach Dr. Codd erst dann relational, wenn es jede einzelne seiner 12 Regeln erfüllt. Da diese Regeln schwer erfüllbar sind, nennen sich viele Datenbanksysteme stattdessen SQL-Datenbanksysteme.

Die 12 Regeln können Inhaltlich in 5 Gruppen unterteilt werden:

1. Basisregeln (Regel 0 und 12)
2. Strukturelle Regeln (Regel 1 und 6)
3. Integritätsregeln (Regel 3 und 10)
4. Bearbeitungsregeln (Regel 2, 4, 5 und 7)
5. Datenunabhängigkeitsregeln (Regel 8, 9 und 11)

1.2.1 Basisregeln (Regel 0 und 12)

1.2.1.1 Regel 0 – Grundregel

Jedes System, welches als relationales Datenbankmanagementsystem (RDBMS) bezeichnet wird (oder dies von sich behauptet), muss in der Lage sein, die gesamte Datenbank allein durch seine im relationalen Modell spezifizierten relationalen Fähigkeiten zu verwalten.

1.2.1.2 Regel 12 – Verletzen und Umgehen der relationalen Sprache

Falls ein relationales System über eine "Low-Level"-Sprache verfügt, welche einzelne Datensätze bearbeitet, so kann diese "Low-Level"-Sprache nicht dazu benutzt werden, die in der "High-Level" Sprache (z.B. SQL) – welche mehrere Datensätze gleichzeitig bearbeitet – vorgegebenen Integritätsregeln zu untergraben oder zu umgehen.

1.2.2 Strukturelle Regeln (Regel 1 und 6)**1.2.2.1 Regel 1 – Darstellung der Information**

Jede Information in einer relationalen Datenbank wird ausschließlich auf einer logischen Ebene (konzeptionelle Ebene) und nur durch Werte in Relationen (Tabellen) dargestellt.

1.2.2.2 Regel 6 – Aktualisieren von Views

Jede theoretisch aktualisierbare View kann im System auch aktualisiert werden.

1.2.3 Integritätsregeln (Regel 3 und 10)**1.2.3.1 Regel 3 – Systematische Behandlung von NULL**

In einem relationalen System werden unabhängig vom Datentyp Indikatoren (NULL) für fehlende oder aus der Sicht des Datentyps ungeeignete Informationen für die Darstellung auf der logischen Ebene verwendet. Diese Indikatoren (NULL) sind weder leere Zeichenketten noch Zeichenketten aus Leerzeichen. NULL ist nicht die Zahl 0 (Zero) oder irgendeine andere Zahl.

Neben den Indikatoren muss das relationale System die Manipulatoroperationen für diese Indikatoren unterstützen. Diese Operationen dürfen nicht vom Datentyp der fehlenden Information abhängig sein.

1.2.3.2 Regel 10 – Integritätsunabhängigkeit

Die für eine Datenbank spezifischen Integritätsbedingungen müssen mit Hilfe der relationalen Datenbeschreibungssprache definierbar und im Systemkatalog abgelegt sein. Sie dürfen nicht in den Anwendungsprogrammen abgelegt werden.

1.2.4 Bearbeitungsregeln (Regel 2, 4, 5 und 7)

1.2.4.1 Regel 2 – Garantierter Zugriff

Auf jeden skalaren Wert einer relationalen Datenbank kann immer mit einer Kombination aus Relationsnamen, Primärschlüsselwert und Spaltennamen zugegriffen werden.

1.2.4.2 Regel 4 – Dynamischer Online Katalog basierend auf dem relationalen Modell

Die Datenbankbeschreibung (Meta-Daten) wird auf die gleiche Weise wie gewöhnliche Daten auf der logischen Ebene dargestellt, so dass autorisierte Benutzer genau dieselbe Abfragesprache zum Durchsuchen dieser Daten benutzen können, welche auch für gewöhnliche Daten benutzt wird.

1.2.4.3 Regel 5 – Allumfassende Datenbanksprache

Ein relationales DBMS muss zumindest eine Sprache unterstützen, wobei gilt:

- Die Statements der Sprache müssen über eine wohldefinierte Syntax in Form von Zeichenketten ausdrückbar sein,
- Die Sprache ist allumfassend, indem sie alle folgenden Punkte erfüllt:
 - Datendefinition
 - Viewdefinition
 - Datenmanipulation (interaktiv und über Programme)
 - Integritätsregeln
 - Autorisierung
 - Transaktionskontrolle (Begin, End, Commit, Rollback)

1.2.4.4 Regel 7 – High-Level DML (INSERT, UPDATE, DELETE)

Die Fähigkeit, eine Basis- oder eine Ergebnisrelation (View) wie einen einfachen Operanden zu handhaben trifft zu auf:

- Selektieren,
 - Einfügen,
 - Ändern und
 - Löschen
- von Daten.

1.2.5 Datenunabhängigkeitsregeln (Regel 8, 9 und 11)

1.2.5.1 Regel 8 – Physische Datenunabhängigkeit

Anwendungsprogramme und Anwendungsoberflächen bleiben unbeeinträchtigt, wenn Veränderungen an der Speicherstruktur oder den Zugriffsmethoden auf die Speicherstrukturen vorgenommen werden.

1.2.5.2 Regel 9 – Logische Datenunabhängigkeit

Anwendungsprogramme und Anwendungsoberflächen bleiben unbeeinträchtigt, wenn Veränderungen an der logischen Speicherstruktur einer Datenbank vorgenommen werden.

1.2.5.3 Regel 11 – Verteilungsunabhängigkeit

Ein relationales DBMS ist Verteilungsunabhängig. Die Abfragen und Programme müssen identisch ablaufen unabhängig davon ob:

1. ein verteiltes DBMS eingeführt wird
2. ein bereits verteiltes DBMS wieder zusammengefügt wird.

Codds 12 Regeln	
<ul style="list-style-type: none">▪ Basisregeln:<ul style="list-style-type: none">▪ Regel 0 - Grundregel▪ Regel 12 - Verletzen und Umgehen der relationalen Sprache▪ Strukturelle Regeln:<ul style="list-style-type: none">▪ Regel 1 - Darstellung der Information▪ Regel 6 - Aktualisieren von Views▪ Integritätsregeln:<ul style="list-style-type: none">▪ Regel 3 - Systematische Behandlung von NULL▪ Regel 10 - Integritätsunabhängigkeit▪ Bearbeitungsregeln:<ul style="list-style-type: none">▪ Regel 2 - Garantiertes Zugriff▪ Regel 4 - Dynamischer Online Katalog basierend auf dem relationalen Modell▪ Regel 5 - Allumfassende Datenbanksprache▪ Regel 7 - High-Level DML (INSERT, UPDATE, DELETE)▪ Datenunabhängigkeitsregeln:<ul style="list-style-type: none">▪ Regel 8 - Physische Datenunabhängigkeit▪ Regel 9 - Logische Datenunabhängigkeit▪ Regel 11 - Verteilungsunabhängigkeit	
2.5.0122 © Integrata Cegos GmbH	ORACLE und SQL

1.3 Design, Entwicklung und Implementierung von Datenbanken

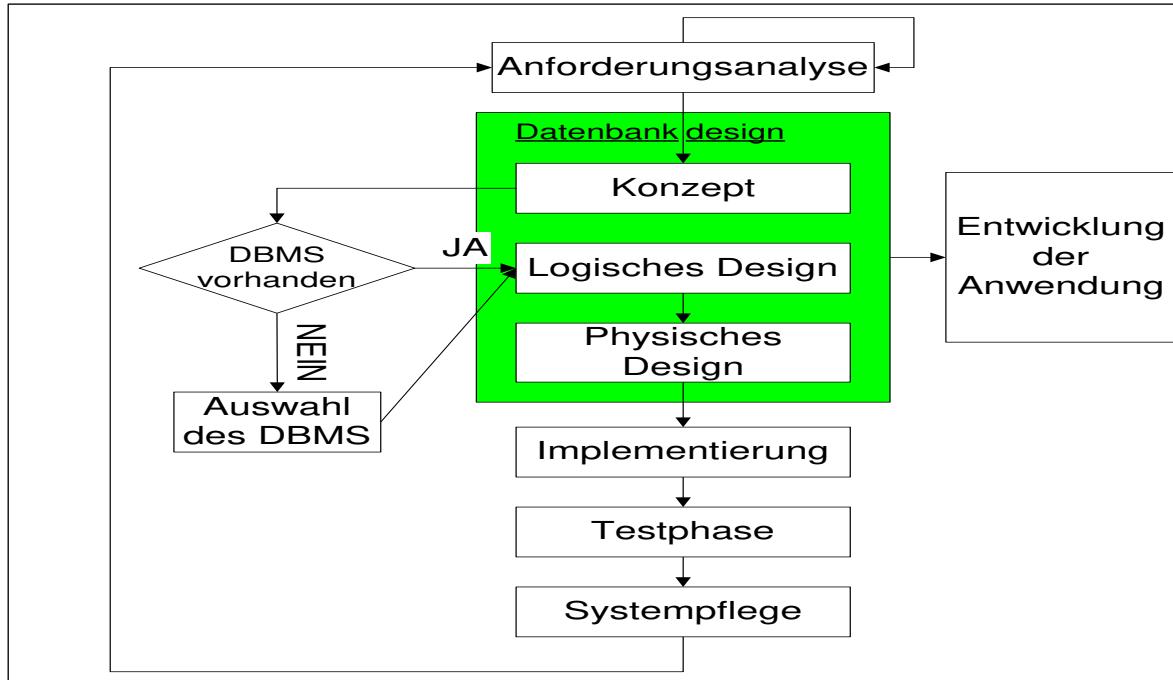


Abb. 1-5: Vereinfachtes Ablaufdiagramm der Datenbankentwicklung

Dieses umfangreiche Thema gehört nicht primär zu diesem Seminar und wird hier nur oberflächlich angesprochen. Die Bedeutung des richtigen Designs muss hervorgehoben werden, da es erfahrungsgemäß in Unternehmen sehr oft relativ stiefmütterlich behandelt wird. Mehr wegweisende Informationen zu diesem Thema kann Ihnen Ihr Referent mitteilen (s. auch das Seminar 4062).

Ein fehlerhaftes oder wie so oft gar kein vorhandenes Design kann die Datenbank unbrauchbar für praktischen Einsatz machen. Die oftmals gewünschten Tuning- und Optimierungsmaßnahmen können nur an einer halbwegen gut durchdachten Datenbank durchgeführt werden. Die Datenbank in diesem Sinne ist KEINE bloße Ansammlung von Tabellen, Views, Indizes, etc.!

Eine bereits im täglichen Einsatz befindliche Datenbank kann nur mit enormem Einsatz aller Beteiligten (DB-Admin, SYS-Admin, Anwendungsprogrammierer, mehrere betroffene Benutzer der Anwendung, etc.) geändert werden.

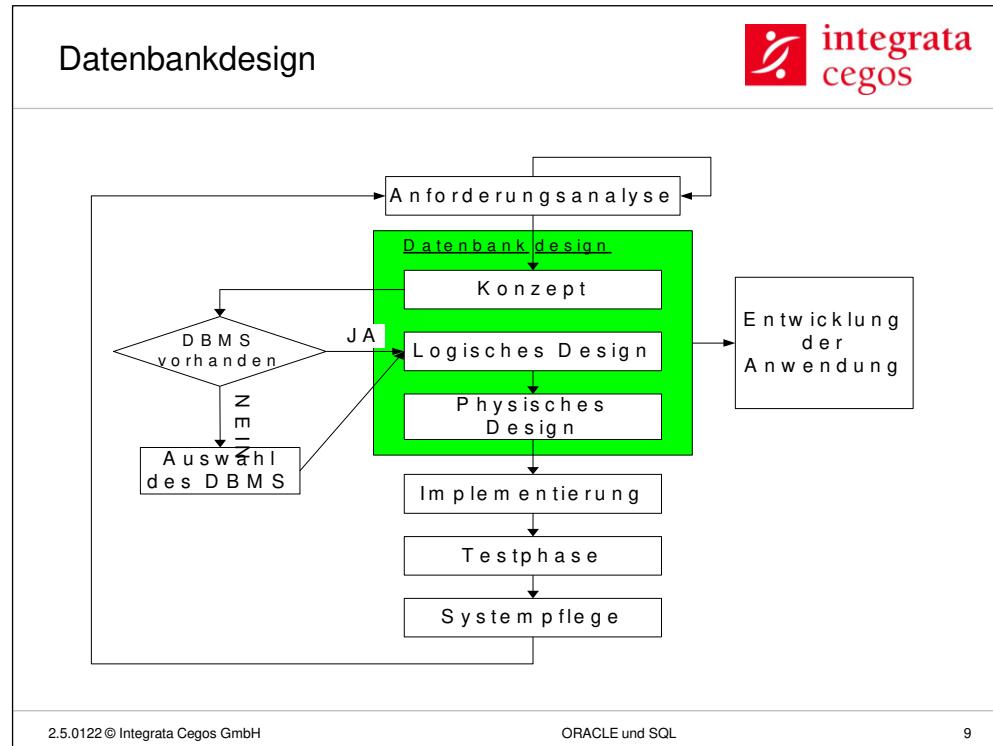
Deswegen sollte das ganze System gut durchdacht werden, bevor es endgültig in Betrieb genommen wird!

1.3.1 Datenbankdesign

Datenbankdesign beinhaltet das **konzeptuelle**, **logische** und **physische** Design der Datenbank.

Um eine Datenbank überhaupt designen zu können, müssen die **Anforderungen** an die zu entwerfende Datenbank gesammelt werden.

Ziel des Designs ist es, eine für die Zwecke der **Datenbankbenutzer** geeignete **Abstraktion** der realen Welt zu bekommen. Ein geeignetes Modell für den gewünschten Zweck zu finden ist eine **Kunst**.



Zu diesem Zwecke sind mehrere Methoden entwickelt worden. Dazu gehören u. a.:

- **ERM** (Entity-Relationship Modellierung): für die Abstraktion der realen Welt im traditionellen Bereich der Datenbankanwendung.
- **EERM** (Enhanced Entity-Relationship Modellierung): für die Abstraktion der realen Welt in:
 - CAD (Computer Aided Design),
 - CAM (Computer Aided Manufacturing),
 - CASE (Computer Aided Software Engineering) und
 - Multimedia,
 - Allgemein Objektorientierung,
 - Bereichen, für welche das "normale" ERM keine Abbildungsmöglichkeiten hat.
- **Normalisierung:** Erstellen des für die Zwecke der Datenbank besten relationalen Modells (Anmerkung: Für Data Warehouse im Prinzip De-Normalisierung).

<p>Datenbankdesign</p> <ul style="list-style-type: none"> ▪ Datenbankdesign: <ul style="list-style-type: none"> ▪ Konzeptuelles <ul style="list-style-type: none"> ▪ Abstraktion der Welt ▪ Abstraktes Modell ▪ Logisches <ul style="list-style-type: none"> ▪ Relationales Modell ▪ Physisches <ul style="list-style-type: none"> ▪ Tabellen ▪ Regeln ▪ Indizes ▪ etc. ▪ Methoden <ul style="list-style-type: none"> ▪ ERM ▪ EERM ▪ Normalisierung 	
<small>2.5.0122 © Integrata Cegos GmbH</small>	
<small>ORACLE und SQL</small>	
<small>10</small>	

1.3.2 Entity-Relationship Modell (ERM)

Eine der Methoden des Datenbankdesigns ist das ERM. ERM ist eine komplexe und mächtige Methode für Abstraktionen. Hier wird zu Präsentationszwecken ein kleiner Ausschnitt des ERM gezeigt. Richtige Handhabung kann in diesem Rahmen **nicht** vermittelt werden.

Die Elemente des ERM sind:

1. Entitätstypen:

stellen mehrere Objekte der realen Welt mit denselben Merkmalen dar. Entitätstypen können sowohl reale als auch abstrakte Objekte repräsentieren.

2. Beziehungstypen:

stellen sinnvolle Verbindungen zwischen Entitätstypen dar. Die Kardinalität der Beziehung soll hier aus allen ihren Eigenschaften hervorgehoben werden, da eine m:n-Kardinalität nur in einer Tabelle des logischen Modells abgebildet werden kann. Die Kardinalitäten sind:

- 1:1
- 1:n
- n:1
- m:n.

3. Attributtypen:

stellen eine Eigenschaft des Entitäts- oder Beziehungs- typen dar. Die Attributtypen können:

- einfach oder zusammengesetzt,
- ein- oder mehrwertig,
- abgeleitet

sein. Aus der Menge aller Attributtypen werden Schlüsselkandidaten (für Primärschlüssel) bestimmt.

Modellierung eines Unternehmens



- Entitätstypen:
 - Abteilung (schwacher Entitätstyp)
 - Mitarbeiter (starker Entitätstyp)
- Beziehungstypen
 - Arbeitet in
 - Kardinalität 1:1
 - Besteht aus
 - Kardinalität 1:n
- Attributen
 - Abteilung:
 - Name
 - Ort
 - Mitarbeiter:
 - Personalnummer
 - Name
 - Stellenbeschreibung
 - Gehalt
 - Gehaltsstufe

1.3.2.1 Modellierungsbeispiel

Wir modellieren in unserem Beispiel ein Unternehmen:

- a) Es besteht aus Abteilungen in welchen wiederum Mitarbeiter beschäftigt sind.
- b) Eine Abteilung existiert nur, falls auch Mitarbeiter in ihr beschäftigt sind.
- c) Ein Mitarbeiter kann nur für eine einzige Abteilung arbeiten.
- d) Es gibt keine "abteilungsfreien" Mitarbeiter.
- e) Eine Abteilung ist in einer bestimmten Stadt untergebracht.
- f) Ein Mitarbeiter hat eine Personalnummer, Namen und Stellenbeschreibung.
- g) Ein Mitarbeiter bezieht Gehalt und wird einer Gehaltsstufe zugeordnet.

Tiefer, korrekter und feiner wollen wir an dieser Stelle die Unternehmensstruktur absichtlich nicht analysieren.

Als **Entitätstypen** ergeben sich in diesem Beispiel "Mitarbeiter", als starker Entitätstyp und "Abteilung" als schwacher Entitätstyp (da sie ohne Mitarbeiter nicht existieren kann).

Die **Beziehungstypen** "arbeitet in" (1:1) und "besteht aus" (1:n) sind zwischen den beiden Entitätstypen vorhanden.

Attributstypen der Entitätstypen sind:

1. Abteilung:

- Name
- Ort

2. Mitarbeiter:

- Personalnummer
- Name
- Stellenbeschreibung
- Gehalt
- Gehaltsstufe

Fangfrage: Was ist "Vorgesetzter der Abteilung"? Ein Beziehungstyp oder ein Attributstyp?

Modellierung eines Unternehmens

- Ein Unternehmen besteht aus Abteilungen in welchen wiederum Mitarbeiter beschäftigt sind.
- Eine Abteilung existiert nur, falls auch Mitarbeiter in ihr beschäftigt sind.
- Ein Mitarbeiter kann nur für eine einzige Abteilung arbeiten.
- Es gibt keine "abteilungsfreien" Mitarbeiter.
- Eine Abteilung ist in einer bestimmten Stadt untergebracht.
- Ein Mitarbeiter hat eine Personalnummer, Namen und Stellenbeschreibung.
- Ein Mitarbeiter bezieht Gehalt und wird einer Gehaltsstufe zugeordnet.

1.3.2.2 Graphische Darstellung des ERM

Es gibt mehrere erarbeitete graphische Darstellungsmöglichkeiten eines ERM (mehr dazu im Seminar 4062).

Bei der graphischen Darstellung eines solchen Modells werden Beziehungstypen zwischen den Entitätstypen in Form von durchgezogenen ("MUSS"-Beziehung) oder unterbrochenen ("KANN"-Beziehung) Linien dargestellt. Zum Beispiel muss einer Abteilung auch mindestens ein Mitarbeiter zugeordnet werden, und umgekehrt wird jeder Mitarbeiter auch einer Abteilung zugeordnet ("MUSS"-Beziehung).

Es könnte aber auch vorkommen, dass ein Mitarbeiter keiner Abteilung fest zugeordnet werden muss. Sollte dies in der betreffenden Firma möglich sein, so würde dies als "KANN"-Beziehung in das Modell einfließen.

Jeder Abteilung kann mehr als ein Mitarbeiter zugeordnet sein, während der gleiche Mitarbeiter immer nur einer einzigen Abteilung zugeordnet sein darf. Dies bezeichnet man als Grad oder Kardinalität der Beziehung ("ein einziger" oder "einer oder mehrere"); graphisch wird "ein einziger" als einzelne Linie dargestellt, "einer oder mehrere" dagegen durch einen so genannten Krähenfuß.

1.3.2.3 ORACLES graphische ERM-Notation



Abb. 1-6: ERM des Unternehmens

Logisches Datenbankdesign

Aus dem ERM wird in diesem Schritt nun ein relationales Modell unseres ERM erstellt.

Zu **eigenständigen Relationen** des logischen Modells werden:

- Alle Entitätstypen,
- Alle Beziehungstypen mit höherer Kardinalität (m:n),
- Abgeleitete Attributstypen (Abhängig von der Normalisierungsstufe).

Zu **Attributen** der Relationen werden:

- Alle Attributstypen der Entitätstypen,
- Alle Beziehungstypen mit niedrigerer Kardinalität (1:1 und 1:n). In unserem Beispiel wird später die Spalte "Abteilungsnummer" in der Relation "Mitarbeiter" zu diesem Zweck gebraucht.

Verfeinerungen können über die Normalisierung weiter betrieben werden.

Modellierung eines Untemehmens	
<ul style="list-style-type: none">▪ Entitätstypen:<ul style="list-style-type: none">▪ Abteilung (schwacher Entitätstyp)▪ Mitarbeiter (starker Entitätstyp)▪ Beziehungstypen<ul style="list-style-type: none">▪ Arbeitet in<ul style="list-style-type: none">▪ Kardinalität 1:1▪ Besteht aus<ul style="list-style-type: none">▪ Kardinalität 1:n▪ Attributen<ul style="list-style-type: none">▪ Abteilung:<ul style="list-style-type: none">▪ Name▪ Ort▪ Mitarbeiter:<ul style="list-style-type: none">▪ Personalnummer▪ Name▪ Stellenbeschreibung▪ Gehalt▪ Gehaltsstufe	
<small>25.0122 © Integrata Cegos GmbH</small>	<small>ORACLE und SQL</small>
	<small>11</small>

1.3.2.4 Beziehungen zwischen Relationen

Datenintegrität bedeutet Korrektheit und Vollständigkeit des Datenbestandes. Ein wichtiger Punkt, um diese zu bewahren, ist die Festlegung und Berücksichtigung der Beziehungen zwischen den einzelnen Relationen.

Untersucht werden dabei immer zwei Relationen, die in irgendeiner Beziehung zueinander stehen (s. als erstes ERM), zum Beispiel die Tabellen "MITARBEITER" und "ABTEILUNG". Diese Beziehung wird dann über das Primär- und Fremdschlüsselprinzip definiert.

Relationale Beziehungen	
<ul style="list-style-type: none">▪ Beziehungen zwischen Relationen<ul style="list-style-type: none">▪ sind wichtig zum Erhalt der Datenintegrität▪ werden über das Primär- und Fremdschlüssel Prinzip festgelegt▪ Primärschlüssel<ul style="list-style-type: none">▪ identifiziert eindeutig eine Entität▪ darf keine NULL-Werte enthalten▪ jede Relation sollte einen Primärschlüssel besitzen▪ Fremdschlüssel<ul style="list-style-type: none">▪ sind Attribute, die in anderen Tabellen Primärschlüssel oder UNIQUE-Keys sind▪ bilden die Beziehung zwischen zwei Relationen▪ darf nur Werte annehmen, die schon in referenzierten Spalte existieren	

1.3.2.5 Primär- und Fremdschlüsselprinzip

Primärschlüssel (PRIMARY KEY → PK) Der Primärschlüssel identifiziert eindeutig einen Datensatz in einer Relation. In der Spalte, auf der er liegt, muss immer ein Wert eingetragen werden, sie darf also weder leer bleiben noch den Wert **NULL** enthalten. Der Wert muss eindeutig sein. In jeder Relation sollte es einen Primärschlüssel geben.

Beispiele: Abteilungsnummer in der Relation "Abteilung"
Personalnummer in der Relation "Mitarbeiter"

Fremdschlüssel (FOREIGN KEY → FK) Fremdschlüssel sind immer Attribute, die in einer anderen Relation als Primärschlüssel oder **UNIQUE**-Key definiert sind. Fremdschlüsselbeziehungen stellen die Beziehung zwischen den Relationen dar. Ein Fremdschlüssel kann nur Werte annehmen, die in der referenzierten Spalte bereits vorhanden sind. Sogenannte **NULL**-Werte (d.h., kein Eintrag in die entsprechende Spalte) sollte man bei Fremdschlüsseln nicht zulassen, da dies evtl. zu Anomalien führen kann.

Beispiele: Abteilungsnummer in der Tabelle "MITARBEITER"
DEPTNO in der Tabelle "EMP"

Sowohl Primär- als auch Fremdschlüssel können **zusammengesetzt** sein (wie in der Relation Abteilung $\{Name, Ort\}$), sich also über mehrere Spalten hinziehen.

Die Primärschlüssel können auch "**künstlich**" erzeugt werden, wie hier "Abteilungsnummer" der Relation Abteilung. Aus der Normalisierungssicht ist dies falsch, aus der Performance- oder Übersichtlichkeitssicht sind solche Vorgehen durchaus üblich.

Hinweis: Natürlicher PK-Kandidat in unserer Relation Abteilung ist $\{Name, Ort\}$. Man kann sich vorstellen, welche Algorithmen (Suchen, Einfügen, Ändern, Löschen, Indexieren in Listen, Bäumen, etc.) schneller sind:

- a) zu einem Textvektor: $\{Name, Ort\}$
- b) zu einer Zahl: Abteilungsnummer.

1.3.2.6 Relationales Modell

Aus einem ERM wird schließlich ein relationales Modell erstellt. Die Entitytypen und m:n Beziehungstypen werden dabei zu Relationen. Ein-fache Beziehungstypen können wahlweise als Relationen oder Spalten einer anderen Relation dargestellt werden.

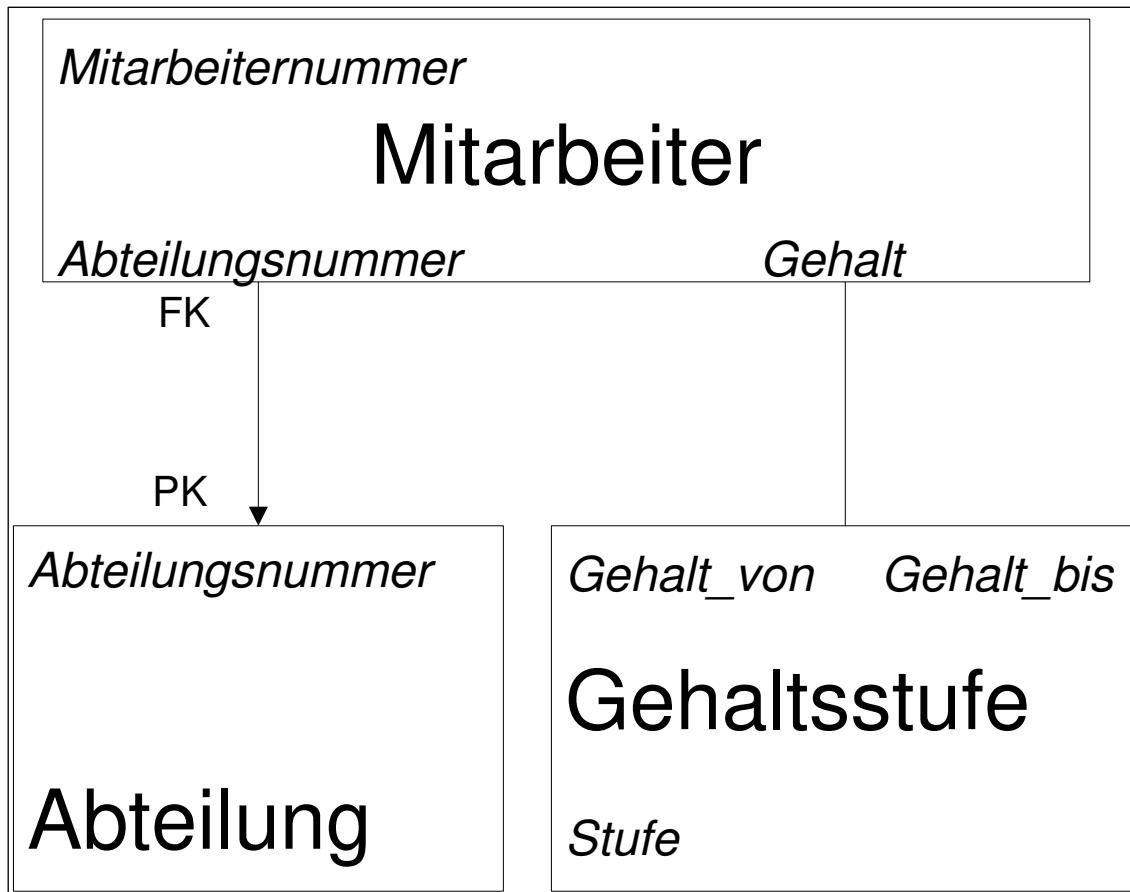
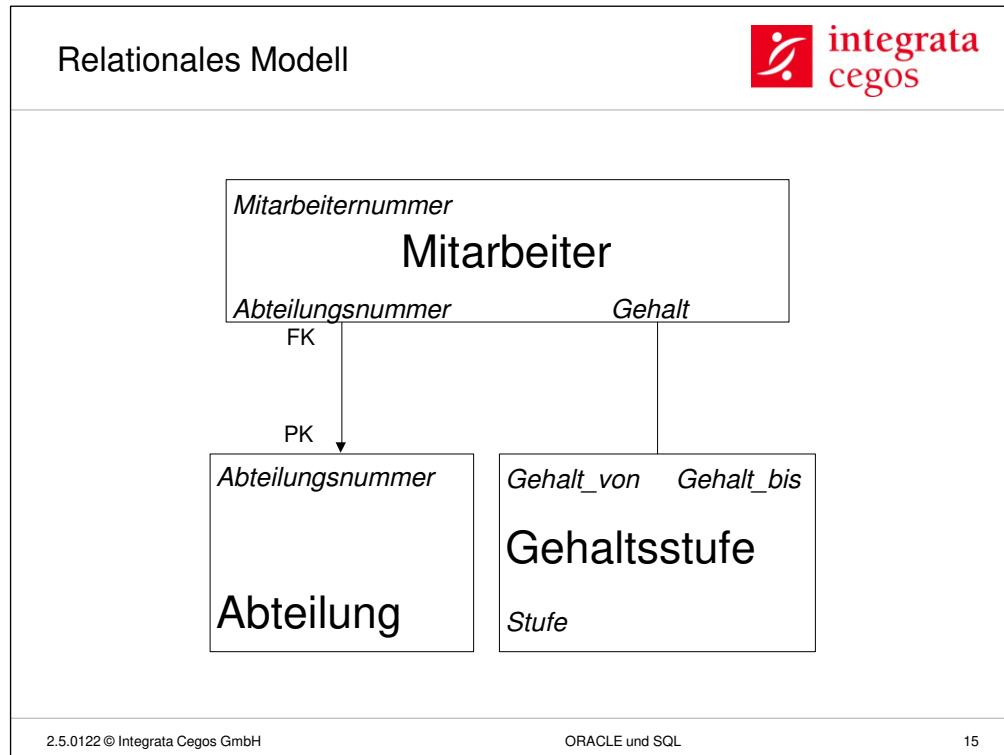


Abb. 1-7: Relationales Modell des Unternehmens mit Schlüsseln ohne "normale" Attribute



1.3.2.7 Normalisierung

Das relationale Modell kann noch nach den gewünschten Eigenschaften weiter verfeinert werden. Diese Aufgabe kann durch die **Normalisierung** der Relationen erledigt werden.

Es gibt folgende Normalformen:

1. unnormalisierte Form (Non first Normal form): NNF oder NF²
2. erste Normalform (1NF)
3. zweite Normalform (2NF)
4. dritte Normalform (3NF)
5. erweiterte dritte Normalform,
auch Boyce-Codd-Normalform genannt (BCNF)
6. vierte Normalform (4NF)
7. fünfte Normalform (5NF)
auch Project Join Normalform genannt (PJNF)

In der Praxis sind 1NF, 2NF und 3NF häufiger anzutreffen als die restlichen Normalformen und reichen in den meisten Fällen völlig aus. Höhere Normalformen sind für Lösungen von speziellen Problemen von Interesse. Bei großen und komplexen Datenbanken ist es schwierig, überhaupt so weit zu normalisieren.

Oft wird in der theoretischen Literatur empfohlen, die Datenbanken bis mindestens in die 5NF zu normalisieren. Die Normalisierung bis zur 5NF kann nicht immer als optimale Lösung empfohlen werden. Manchmal sind Performancegründe wichtiger als die durch Normalisierung abgefangenen Anomalien. Man denke nur an Data Warehouse Lösungen.

Mit jeder höheren Normalisierungsstufe steigt die Datensicherheit, die Flexibilität der Daten, die Erkennbarkeit der logischen Beziehungen und der **interne Verwaltungsaufwand**.

Das zu programmierende GUI, welches die Datenbank bedienen soll, wird auch mit jeder höheren Normalform komplexer. Es gibt keine Regel, welche Normalform wann die beste ist. Jeder muss es selber nach seinen subjektiven Kriterien situationsabhängig entscheiden.

1.3.2.8 Normalformen

Hier werden die Normalformen kurz vorgestellt. Wer an der genauen mathematischen Definition interessiert ist, findet diese in der weiterführenden Literatur.

1. unnormalisierte Form (Non first Normal form: NFNF)

In der Tabelle werden ein oder mehrere Attributen mehrfach wiederholt.

2. erste Normalform (1NF)

Alle Werte in den Feldern einer Relation sind atomar (skalar).

3. zweite Normalform (2NF)

Die Relation ist in 1NF und jedes Nicht-Primärschlüsselattribut ist voll funktional abhängig vom Primärschlüssel.

4. dritte Normalform (3NF)

Die Relation ist in 1NF und 2NF und kein Nicht-Primärschlüsselattribut ist transitiv vom Primärschlüssel abhängig.

5. erweiterte dritte Normalform, auch Boyce-Codd-Normalform (BCNF)

Die Relation ist in 3NF und jede Determinante stellt einen Schlüsselkandidaten dar.

6. vierte Normalform (4NF)

Die Relation ist in BCNF und es sind keine mehrwertigen trivialen Abhängigkeiten vorhanden.

7. fünfte Normalform (5NF) auch Project Join Normalform (PJNF)

Die Relation enthält keine Verknüpfungsabhängigkeiten.

Normalisierung



- unnormalisierte Form (Non first Normal form): NFNF, NF²
- erste Normalform (1NF)
- zweite Normalform (2NF)
- dritte Normalform (3NF)
- erweiterte dritte Normalform, auch Boyce-Codd Normalform (BCNF)
- vierte Normalform (4NF)
- fünfte Normalform (5NF)

1.3.3 Physisches Modell

Das konzeptionelle Modell wird mit konkreten SQL-Befehlen zum Erzeugen der Tabellen, Indizes, Bedingungen, Zugriffsrechten, etc. beendet:

z. B.:

```
SQL> CREATE TABLE Abteilung
      ( Abteilungsnummer      NUMBER(4) PRIMARY KEY,
        Name                  VARCHAR2(25) NOT NULL,
        Ort                  VARCHAR2(40) NOT NULL
      );
```

Mehr dazu im Kapitel zu DDL-Befehlen (s. Kapitel 7).

1.4 Relationale Algebra

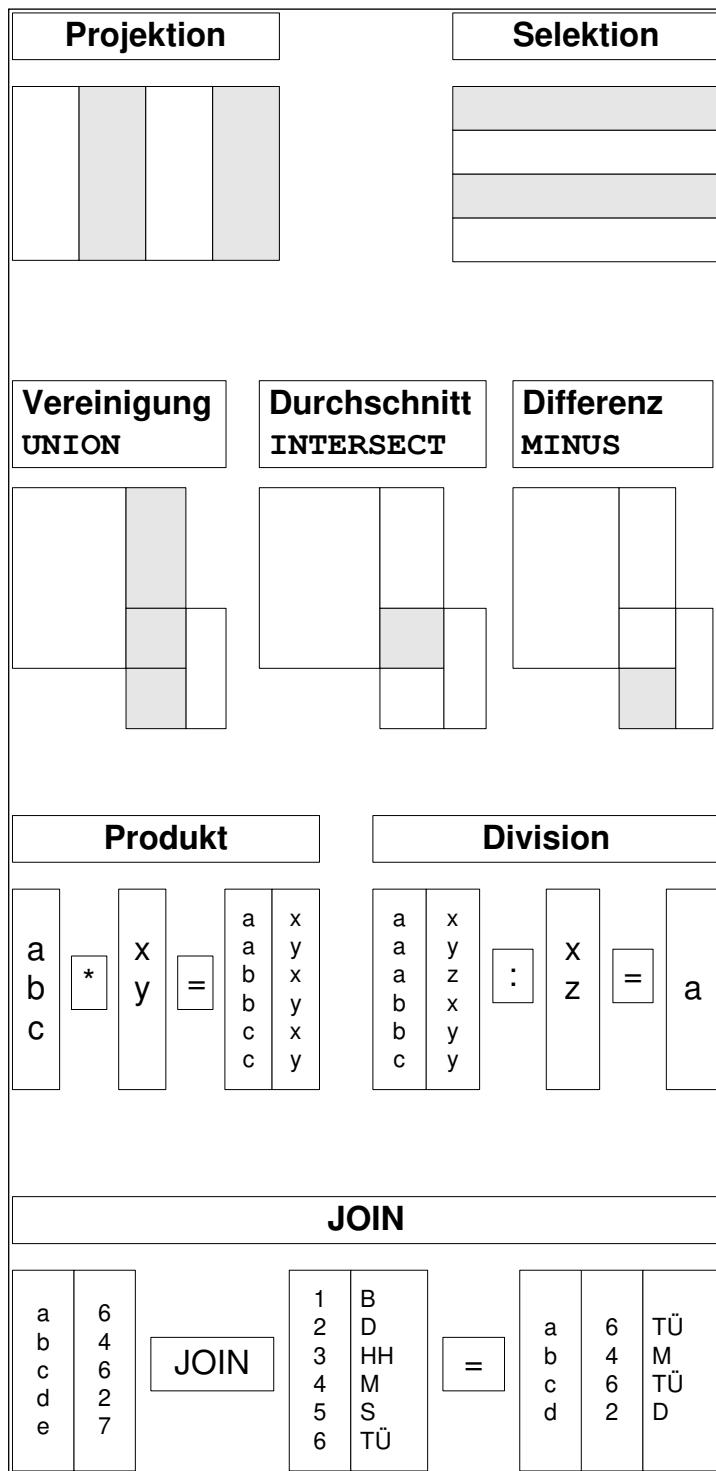
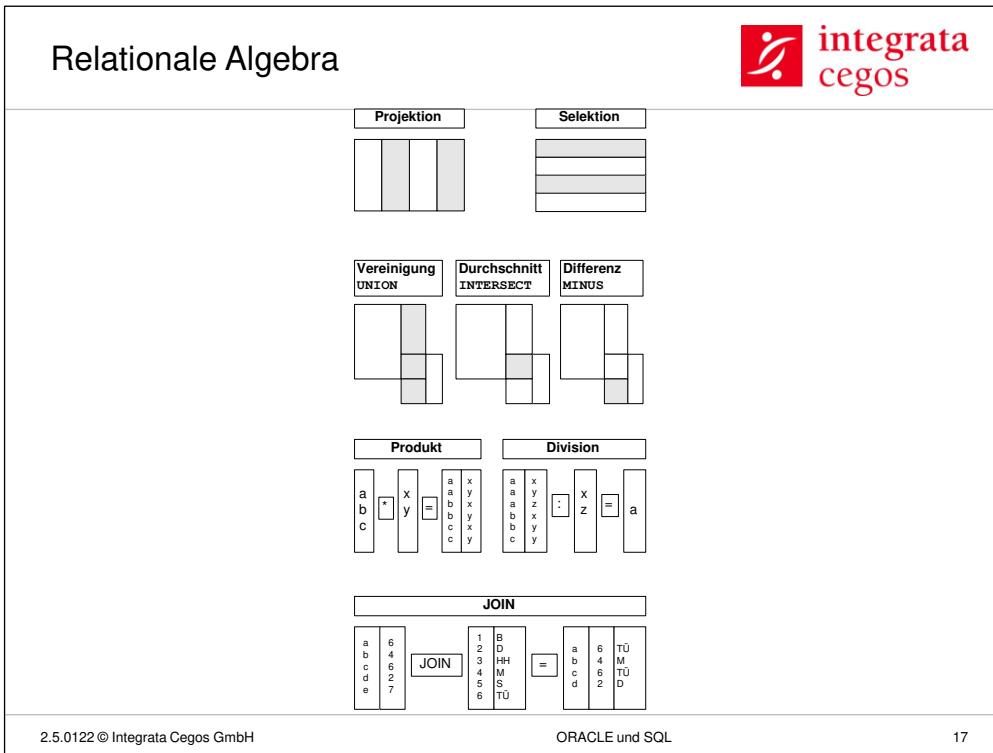


Abb. 1-8: Die Mathematischen Operationen auf die Relationen



1.5 Die Sprache SQL

SQL (Structured Query Language) ist eine beschreibende mengenorientierte Sprache.

Es existieren neben dem **mengenorientierten** Programmierparadigma noch das prozedurale, objektorientierte, funktionale und logische Programmierparadigma.

Die Sprache SQL teilt sich allgemein in 4 Gebiete:

1. DDL Datendefinition (Data Definition Language)
2. DML Datenmanipulation (Data Manipulation Language)
3. DCL Datenkontrolle (Data Control Language)
4. Transaktionsbefehle

1.5.1 DDL-Befehle

Mit Hilfe der **DDL**-Befehle werden die Objekte (Tabellen, Indizes, Views, etc.) einer Datenbank:

- erstellt (CREATE),
- angepasst (ALTER) und
- gelöscht (DROP).

1.5.2 DML-Befehle

Die **DML**-Befehle dienen dazu, die Daten einer Datenbank zu handhaben. Die Daten können mit Hilfe dieser Befehle:

- eingefügt (INSERT),
- bearbeitet (UPDATE) und
- gelöscht (DELETE)

werden.

Die Filterung der Daten (SELECT) gehört auch in die Klasse der DML-Befehle (darüber streiten sich die Geister), obwohl mit diesem Befehl die Daten in der Datenbank **keinesfalls** manipuliert werden.

1.5.3 DCL-Befehle

Die Klasse der **DCL**-Befehle (GRANT, REVOKE) ist für die Rechteverwaltung und Umsetzung der Sicherheitsrichtlinien einer Datenbank zuständig.

1.5.4 Transaktionsbefehle

Zu den SQL-Befehlen gehört auch der Satz der Befehle zur **Transaktionsteuerung**. Die Transaktionsbefehle passen aber in keine der ersten drei Rubriken, deswegen werden sie separat aufgeführt.

1.5.5 SQL-Befehlsauflistung

- DDL Datendefinition (Data Definition Language)
 - CREATE
 - ALTER
 - DROP
- DML Datenmanipulation (Data Manipulation Language)
 - SELECT
 - INSERT
 - UPDATE
 - DELETE
- DCL Datenkontrolle (Data Control Language)
 - GRANT
 - REVOKE
- Transaktionsbefehle
 - COMMIT
 - ROLLBACK
 - SAVEPOINT
 - SET TRANSACTION
 - LOCK TABLE

SQL

- SQL = Structured Query Language
- Datenabfragesprache
 - beschreibend
 - mengenorientiert

SQL-Befehle

- | | |
|-------------------|-----------------------|
| ▪ DDL | Datendefinition |
| ▪ CREATE | |
| ▪ ALTER | |
| ▪ DROP | |
| ▪ DML | Datenmanipulation |
| ▪ SELECT | |
| ▪ INSERT | |
| ▪ UPDATE | |
| ▪ DELETE | |
| ▪ MERGE | |
| ▪ DCL | Datenkontrolle |
| ▪ GRANT | |
| ▪ REVOKE | |
| ▪ TCL | Transaktionskontrolle |
| ▪ COMMIT | |
| ▪ ROLLBACK | |
| ▪ SAVEPOINT | |
| ▪ SET TRANSACTION | |
| ▪ LOCK TABLE | |

1.6 Data Dictionary

Betrachtet man den **inhaltlichen** Aufbau einer ORACLE-Datenbank, so besteht sie immer aus:

- einem **Data Dictionary**, in dem alle Daten gespeichert werden, die für das ordnungsgemäße Funktionieren der Datenbank verantwortlich sind,
- aus beliebig vielen **Benutzern** mit unterschiedlichen **Privilegien**,
- aus **Datenbankobjekten** wie Tabellen, Indizes, Views, usw., die von ORACLE-Benutzern angelegt wurden.

Im Data Dictionary werden also die Meta-Daten, die Definitionen aller Datenbankobjekte, die Zugangs- und Zugriffsberechtigungen und die Informationen über den physischen Datenbankaufbau verwaltet. Das Data Dictionary selbst ist realisiert durch Views, die den ORACLE-Benutzern `SYS` und `SYSTEM` gehören. Auf diese Views kann mit SQL-Befehlen zugegriffen werden.

Das Data Dictionary kann in zwei Ebenen unterteilt werden:

- **Interne Ebene**

Hier werden die eigentlichen Tabellen und ihre Dateninhalte betrachtet. Der normale ORACLE-Benutzer hat hierauf keinen Zugriff.

- **Externe Ebene**

Hier handelt es sich um eine Vielzahl von virtuellen Tabellen, das sind Sichten (`VIEWS`) auf die Basistabellen, die jedem ORACLE-Benutzer zugänglich sind.

Die Benutzerkennungen `SYS` und `SYSTEM` werden bei der Erstellung einer Datenbank automatisch eingerichtet.

Alle lesenden Zugriffe auf das Data Dictionary sollten über die zur Verfügung stehenden Views erfolgen, da die Struktur dieser externen Ebene auch bei Versionsänderungen halbwegs konstant bleibt.

Data Dictionary

- Inhalt des Data Dictionary:
 - Benutzer und deren Privilegien
 - Tabellen und deren Spaltenbezeichnungen und Datentypen
 - Statistiken über die Tabellen und Indizes
 - Indexinformationen
 - Zugriffsberechtigungen auf Tabellen
 - Profilinformation und die Zuordnung zu den Benutzern
 - Freiplatzverwaltung
- zwei Ebenen des Data Dictionary:
 - Interne Ebene:
Eigentliche Tabellen und ihre Dateninhalte
(für normale ORACLE Benutzer nicht zugänglich)
 - Externe Ebene:
Views auf die Basistabellen
(über SQL zugänglich)

1.6.1 Aufteilung des Data Dictionary

Wie bereits erwähnt, bietet die externe Data Dictionary-Ebene dem ORACLE-Benutzer die Möglichkeit, sich mittels SQL alle relevanten Informationen anzeigen zu lassen. Die Views des externen Data Dictionary sind in drei Gruppen eingeteilt, die eine unterschiedliche Sichtweise erlauben:

- **USER_** Views dieser Gruppe stellen alle Datenbankobjekte dar, deren Eigentümer der jeweilige angemeldete Benutzer ist.

Beispiel: Alle **Tabellen**, die dem angemeldeten Benutzer gehören, zeigt der Befehl:

```
SQL> SELECT * FROM USER_TABLES;
```

Alle **Views**, die dem angemeldeten Benutzer gehören, zeigt analog der Befehl:

```
SQL> SELECT * FROM USER_VIEWS;
```

- **ALL_** Views dieser Gruppe liefern Informationen über alle Datenbankobjekte, auf die der Benutzer Zugriff hat. Der vollständige Befehl lautet hier:

```
SQL> SELECT * FROM ALL_TABLES;
```

- **DBA_** Alle Views mit diesem Präfix sind nur für Benutzer mit DBA-Privileg zugänglich. Diese Views bieten eine Gesamtsicht auf die Datenbank. Beispiel: Alle Tabellen aller Benutzer im gesamten System zeigt der Befehl:

```
SQL> SELECT * FROM DBA_TABLES;
```

- **CDB_** Ab Version 12.1 kann die Datenbank als Pluggable Database installiert werden. Hier werden viele Pluggable Datenbanken (PDB's) in einer Container Datenbank (CDB) zusammengefasst. Die CDB_View hat die Sicht über alle PDB's hinweg.

Die Namen der einzelnen Views unterscheiden sich nur im Präfix und im Informationsgehalt, nicht jedoch in der grundsätzlichen Funktion. Die Auflistung aller Dictionary Views und eine kurze Beschreibung der Funktion bekommt man durch den Befehl:

```
SQL> SELECT * FROM DICTIONARY;
```

oder

```
SQL> SELECT * FROM DICT;
```

Dreiteilung des Data Dictionary



- Dreiteilung des Data Dictionary
 - USER_VIEWS
 - ALL_VIEWS
 - DBA_VIEWS
 - CDB_VIEWS (bei Pluggable Datenbanken ab 12.1)
- Beispiele:
 - SQL> SELECT * FROM USER_TABLES;
 - SQL> SELECT * FROM USER_TAB_COLUMNS;
 - SQL> SELECT * FROM ALL_INDEXES;
 - SQL> SELECT * FROM DBA_VIEWS;
 - SQL> SELECT * FROM DICTIONARY;

Beispiel:

Liste aller Data Dictionary Views mit Prefix ALL bei Oracle 12.2 sortiert nach dem Namen.

SQL>

```
SELECT          table_name
  FROM          dict
 WHERE         table_name LIKE 'ALL%'
 ORDER BY      table_name;
```

Auszug aus der Liste der DD-Views



Beispiel:

Alle DD-Views mit dem Prefix ALL

SQL>

```
SELECT          table_name
  FROM          dict
 WHERE         table_name LIKE 'ALL%'
 ORDER BY      table_name;
```

TABLE_NAME
ALL_ALL_TABLES
ALL_APPLY
...
ALL_COL_COMMENTS
ALL_COLL_TYPES
ALL_COL_PENDING_STATS
ALL_COL_PRIVS
ALL_COL_PRIVS_MADE
ALL_COL_PRIVS_REC'D
...
ALL_CONSTRAINTS
...
ALL_OBJECTS
...
ALL_TAB_COLS
ALL_TAB_COL_STATISTICS
ALL_TAB_COLUMNS
...
ALL_TABLES
...
334 Zeilen ausgewählt.

2

Eingabeoberflächen

2.1	SQL*Plus Grundlagen	2-3
2.1.1	Leistungsmerkmale.....	2-3
2.1.2	Starten von SQL*Plus	2-4
2.1.3	Befehlspuffer und Editoraufruf	2-7
2.1.4	Umgebungsparameter	2-9
2.1.5	Befehlsdateien	2-13
2.1.6	Spool-Dateien	2-15
2.1.7	Weitere Befehle	2-17
2.2	SQL Developer	2-19
2.2.1	Connections.....	2-19
2.2.2	SQL Worksheet	2-21
2.2.3	Output-Möglichkeiten des SQL Developer.....	2-22

2 Eingabeoberflächen

2.1 SQL*Plus Grundlagen

2.1.1 Leistungsmerkmale

SQL*Plus ist eine interaktive SQL-Umgebung, mit der SQL-Befehle eingegeben, ausgeführt und auf dem Bildschirm sichtbar gemacht werden können. SQL*Plus ist nicht zu verwechseln oder gleichzusetzen mit der Sprache SQL. SQL*Plus ist nur ein Hilfsmittel zur Benutzung dieser Sprache.

SQL*Plus besitzt folgende Funktionalitäten:

- Verwendung der Sprache SQL zur interaktiven Kommunikation mit Oracle,
- Zugriffskontrolle durch Überprüfung von Benutzernamen und Passwort,
- Speichern des letzten SQL Befehls im so genannten Befehlspuffer,
- Formatierungsmöglichkeiten der SQL Ergebnisse,
- Online Hilfsfunktionen,
- Komplette Befehlshistorie ab Start der Shell.

Vorteile:

- In jeder Oracle-Umgebung verfügbar,
- Es können ALLE Kommandos an die Datenbank abgeschickt werden.

Nachteil: für Ungeübte (oder nicht Konsolenliebhaber) wenig komfortabel.

2.1.2 Starten von SQL*Plus

Der Benutzer meldet sich wie gewöhnlich bei dem Computer an. Danach kann auf der Shell-Ebene folgender Befehl eingegeben werden:

```
sqlplus [benutzernname [[/passwort] [@Dienstname]]]
```

Man kann optional Benutzernamen und Passwort als Befehlsparameter angeben. Ansonsten wird interaktiv danach gefragt. Dienstname (z.B. ORCL) sollte angegeben werden, falls der Benutzer nicht direkt am Datenbankserver arbeitet.

```
sqlplus
```

Es erscheint dann folgende Meldung

```
SQL*Plus: Release 18.2.0.6.0 - Production on Mi Jan  
7 17:26:35 2018
```

```
Copyright (c) 1982, 2019, oracle. All rights re-  
served.
```

```
Benutzernamen eingeben:
```

Geben Sie nun einen gültigen Benutzernamen ein. Folgende Meldung erscheint:

```
Kennwort eingeben:
```

Geben Sie nun das gültige Passwort ein.

```
Verbunden mit:
```

```
Oracle Database 18c Enterprise Edition Release  
18.2.0.6.0 - Production  
with the Partitioning, OLAP, Data Mining and Real  
Application Testing options
```

```
SQL>
```

Zum **Verlassen von SQL*Plus** geben sie folgenden Befehl ein:

```
SQL> exit
```

Sie können während Sie in SQL*Plus sind **Betriebssystem-Befehle durchführen** ohne SQL*Plus zu verlassen, indem Sie den HOST-Befehl benutzen:

```
SQL> HOST dir
```

SQL*Plus

c:\>sqlplus scott/tiger

SQL*Plus: Release 18.0.0.0.0 - Production on Di Sep 24 13:24:51 2019
Version 18.3.0.0.0
Copyright (c) 1982, 2019, Oracle. All rights reserved.
Letzte erfolgreiche Anmeldezeit: Do Sep 12 2019 08:46:47 +02:00
Verbunden mit:
Oracle Database 18c Enterprise Edition Release 19.0.0.0.0 - Production
Version 18.3.0.0.0

SQL>

2.5.0122 © Integrata Cegos GmbH

ORACLE und SQL

2

SQL*Plus Leistungsmerkmale



- Verwendung der Sprache SQL zur interaktiven Kommunikation mit Oracle
- Zugriffskontrolle durch Überprüfung von Benutzernamen und Passwort
- Speichern des letzten SQL -Befehls im so genannten Befehlpuffer
- Formatierungsmöglichkeiten der SQL -Ergebnisse
- Online Hilfsfunktionen

2.1.3 Befehlspuffer und Editoraufruf

SQL Befehle unter SQL*Plus werden im sogenannten Befehlspuffer abgespeichert (nicht jedoch SQL*Plus Befehle!). Der Befehlspuffer kann normalerweise immer nur einen SQL Befehl aufnehmen. Wird eine neue Eingabe getätigt, so wird die alte überschrieben. Sql*Plus in der Console kann mehrere Befehle puffern. Durch diese Befehle kann mit den „Pfeiltasten“ „Up“ und „Down“ geblättert werden.

1. Wird nach dem SQL Befehl ein Semikolon ";" eingegeben, dann wird der Befehl sofort ausgeführt.
2. Mit dem Befehl "HELP INDEX" werden alle SQL*Plus Befehle aufgelistet.
3. Mit dem Befehl "HELP *kommando*" wird die Hilfe zu jedem SQL*Plus Befehl angezeigt
4. Mit dem Befehl "LIST" kann der Inhalt des Puffers angezeigt werden.
5. Der Befehl "RUN" oder "R" zeigt zunächst den Pufferinhalt an und führt dann den angezeigten SQL-Befehl aus.
6. Der Befehl "/" führt den SQL-Befehl aus dem Puffer aus, ohne ihn vorher anzuzeigen.

Der aktuelle SQL Befehl kann durch einen einfachen Zeileneditor in SQL*Plus editiert werden. Die einfachste Möglichkeit dazu besteht im Aufruf des Editors durch Eingabe des Befehls **ed**.

Daneben steht noch eine Reihe von Editierfunktionen zur Verfügung, die in Anhang B aufgeführt sind.

SQL*Plus Kommandos



- Nur SQL Befehle werden im Befehlspuffer gespeichert, keine SQL*Plus Befehle.
- Wird ein SQL Befehl mit einem Semikolon abgeschlossen, so wird er sofort ausgeführt.
- Der Befehl "HELP INDEX" zeigt sämtliche SQL*Plus Befehle an.
- Der Befehl "HELP *kommando*" zeigt die Hilfe zu jedem SQL*Plus Befehl an.
- Der Befehl "LIST" zeigt den aktuellen Befehl an, ohne ihn auszuführen.
- Der Befehl "RUN" (oder "R") zeigt den aktuellen Befehl an und führt ihn auch aus.
- Der Befehl "/" führt den aktuellen Befehl aus, ohne ihn anzuzeigen.
- Der Editor zur Bearbeitung des aktuellen Befehls wird mit " ed" aufgerufen.

2.1.4 Umgebungsparameter

Umgebungseinstellungen werden in globalen Variablen abgespeichert. Alle Einstellungen besitzen Standardwertvorgaben. Diese können verändert werden.

Soll die Umgebung automatisch bei jedem Einloggen eingestellt werden, so können die entsprechenden Parameter in einer Systemdatei mit dem Namen **GLOGIN.SQL** angegeben werden.

Diese befindet sich unter %ORACLE_HOME%\sqlplus\admin:

- bei ORACLE 11g z.B. im Verzeichnis
C:\app\Administrator\product\11.2.0\db_1\sqlplus\admin.
- bei ORACLE 12c z.B. im Verzeichnis
C:\oracle\product\12.2.0\Dbhome_1\sqlplus\admin
- Die glogin.sql ist seit Oracle 11g leer bzw. erhält nur Kommentare.

Mit dem Befehl **SET** können alle Einstellungen manipuliert werden.

Beispiel:

```
SQL> SET TIMING ON LINES 2000 PAGES 99 FEED 1;
```

Mit dem Befehl **SHOW** kann man sich die Belegung bzw. die Werte der Variablen anschauen.

Einige der wichtigsten Parameter sind hier aufgeführt:

ECHO on | off Bestimmt, ob die Kommandos aus der Befehlsdatei beim START-Befehl angezeigt werden oder nicht.
Default: **on**

FEED [BACK] on | off | n Gibt bei **on** die Anzahl der Datensätze zurück, die selektiert wurden. **off** unterdrückt diese Meldung. Wird **n** angegeben, dann erscheint die Meldung nur, wenn mindestens **n** Sätze selektiert wurden. ("x rows selected")
Default: **6**

HEA[DING] on off	Spaltenüberschriften werden ein- bzw. ausge-blendet. Default: on
HEADS [EP] on off zeichen	Definition des Zeichens zur Trennung von Spaltennamen, Titeln usw. Default: " "
LIN[ESIZE] zahl	Maximale Zeichenanzahl pro Zeile. Default: 80
LONG zahl	Standardbreite für die Darstellung langer Datentyp-Spalten. Default: 80 Wertebereich: 1 bis 32.767
NEWP [AGE] zahl	Anzahl der durchzuführenden Zeilenvorschübe bei Beginn einer neuen Seite. Beträgt die Anzahl 0, wird automatisch ein Seitenvorschub durchgeführt. Default: 1
NULL "zeichen"	Bestimmung der Zeichenkette, die für einen NULL-Wert ausgegeben wird. Default: Leerzeichen " "
NUM[WIDTH] zahl	Festlegen der Anzeigenbreite von numerischen Spalten. Diese kann von spezifischen Spalten-formaten überschrieben werden. Default: 10
PAGES [IZE] zahl	Angabe der Standard-Anzahl Zeilen einer Seite abzüglich Zeilenvorschübe am Beginn einer Seite. Default: 24
PAU[SE] on off text	PAUSE on veranlasst SQL*Plus, am Anfang ei-ner jeden Ergebnisseite anzuhalten. Erst wenn Return eingegeben wird, wird die Anzeige fort-gesetzt. Es kann ein beschreibender Text mit-gegeben werden. Default: off

SPACE [CE] <i>zahl</i>	Festlegen der Anzahl Leerschritte zwischen den einzelnen Spalten. Default: 1 Wertebereich: 1-10
SQLP [ROMPT] <i>text</i>	Festlegen des Promptzeichens für SQL*Plus. Default: "SQL"
SQLT [TERMINATOR] <i>zeichen</i> on off	Definition des Zeichens zum Beenden und Ausführen eines SQL-Befehls. Wird es auf off gesetzt, so wird ein Befehl erst ausgeführt, wenn eine Leerzeile eingegeben wird. Default: on und ";"
TERM [OUT] on off	Mit diesem Befehl kann gesteuert werden, ob eine Bildschirmausgabe erfolgen soll, wenn ein Kommando aus einer Befehlsdatei heraus gestartet wird. Diese Einstellung hat keine Auswirkungen auf Befehle, die interaktiv gestartet wurden. Default: on
TI [ME] on off	Bei on wird vor dem SQL-Prompt die aktuelle Zeit ausgegeben. Default: off
TIMI [NG] on off	Bei on werden nach jedem Ausführen eines SQL-Befehls Informationen über die verbrauchte Rechnerzeit angegeben. Default: off
UND [ERLINE] on off <i>zeichen</i>	Definition des Standardzeichens zum Unterstreichen von Spaltenüberschriften. Mit on/off kann diese Unterstreichung ein- bzw. ausgetauscht werden. Default: on und "-"
VER [IFY] on off	Bestimmt, ob SQL*Plus den Befehlstext vor und nach der Ersetzung von Variablen anzeigt. Default: on

Umgebungsvariablen			
Variable	Kurzform	Default-Wert	
ECHO on off	-	on	
FEEDBACK on off	FEED	6	
HEADING on off	HEA	on	
HEADSEP on off p"ü"che'n	HEADS	on und " "	
LINESIZEzahl	LIN	80	
LONGzahl	-	80 Werte: 1 bis 32.767	
NEWPAGEzahl	NEWP	1	
NULL Zeiche'n	NULL	Leerzeichen " "	
NUMWIDTHzahl	NUM	10	
PAGESIZEzahl	PAGES	24	
PAUSE on offext	PAU	off	
SPACEzahl	SPA	1	
SQLPROMPTtext	SQLP	SQL	
SQLTERMINATOR zeichenon off	SQLT	on und ";"	
TERMOUT on off	TERM	on	
TIME on off	TI	off	
TIMING on off	TI	off	
UNDERLINE on o'f'f'chen	UND	on und -"	
VERIFY on off	VER	on	

2.1.5 Befehlsdateien

Die Inhalte des Befehlspuffers können in sogenannten Befehlsdateien abgespeichert werden. Diese Dateien können dann bei Bedarf wieder abgerufen, bearbeitet und ausgeführt werden. Für die Erstellung dieser Dateien kann auch der Systemeditor genutzt werden. Default mäßig wird der Befehlspuffer in der Datei AFIEDT.BUF abgespeichert.

Befehlsdateien können auch verschachtelt werden, d.h., innerhalb einer Prozedur kann mit dem Befehl START eine weitere Prozedur aufgerufen werden.

Folgende Kommandos können genutzt werden:

SAV[E] *dateiname [.ext]* [CRE[ATE] | REP[LACE] | APP[END]]

sichert den Inhalt des Befehls-Puffers in die angegebene Datei. Wird keine Extension angegeben, so wird die Default-Extension ".sql" benutzt. Wird keine Option oder die Option CREATE angegeben, so wird eine neue Datei erstellt (Default). Die Option REPLACE ersetzt die angegebene Datei, die Option APPEND fügt an die angegebene Datei an.

GET *dateiname [.ext]* [LIST | NOLIST]

lädt die angegebene Datei in den Befehlspuffer. Wird die Option LIST angegeben, so wird automatisch der Inhalt der Datei beim GET angezeigt (Default). Bei der Option NOLIST wird die Anzeige unterdrückt.

START *dateiname [.ext]* oder @*dateiname [.ext]*

führt die Befehle der angegebenen Datei aus.

Beispiel:

```
SQL> SELECT * FROM emp;  
SQL> SAVE alleMA.sql  
SQL> START alleMA.sql
```

Hinweis: Der Standard-Systemeditor kann durch Setzen der Variablen **DEFINE _EDITOR = "Editorname"** verändert werden.

Befehlsdateien



- SAV[E] *Datei[.ext]* [CRE[ATE] | REP[LACE] | APP[END]]
- GET *Datei[.ext]* [LIST | NOLIST]
- START *Datei[.ext]* bzw. @*Datei[.ext]*

Beispiel:

- SQL> SELECT * FROM emp;
- SQL> SAVE alleMA.sql;
- SQL> START alleMA.sql;

2.1.6 Spool-Dateien

Ebenso wie mit SQL-Worksheet können auch die unter SQL*Plus erzeugten Ein- und Ausgaben mittels Spool-Dateien protokolliert und gespeichert werden. Folgende Befehle können genutzt werden:

SPOOL *dateiname* [APPEND]

Mit SPOOL wird die Protokollierung eingeschaltet. Die Protokollierung wird in der angegebenen Datei gespeichert. Existiert diese Datei bereits, kann mit der Angabe APPEND die Ausgabe angehängt werden, ansonsten wird die Datei überschrieben. Die Protokollierung bleibt so lange aktiv, bis folgender Befehl eingegeben wird:

SPOOL OFF

Damit wird die Protokollierung deaktiviert.

Hinweis: Wenn eine SQL*Plus Sitzung beendet wird, ohne dass der Anwender vorher die Protokollierung ausgeschaltet hat, werden alle offenen Spool-Dateien automatisch geschlossen.

- **Beispiel:**

```
SQL> SPOOL tmp_dat;
SQL> SELECT deptno FROM dept;
deptno
-----
 10
 20
 30
 40
```

```
SQL> SPOOL OFF;
```

- Anhängen an eine bereits existierende Datei:

```
SQL> SPOOL tmp_dat APPEND;
      ... SQL> ...
SQL> SPOOL OFF;
```

SPOOL



- SPOOL dient der Protokollierung der Ausgabe in eine Textdatei

- Beispiel:

```
SQL> SPOOL tmp_dat;
SQL> SELECT deptno FROM dept;
deptno
-----
10
20
30
40
SQL> SPOOL OFF;

▪ Anhängen an existierende Datei:
SQL> SPOOL tmp_dat APPEND;
...
SQL> SPOOL OFF;
```

2.1.7 Weitere Befehle

CONN [ECT] [benutzername / password] [@Dienstname]

Der Befehl CONNECT verbindet Sie mit einer anderen gültigen Benutzer-Identifikation.

Beispiele:

```
SQL> CONN scott/tiger@ORCL;
```

DESC [RIBE] [benutzername .]tabelle

Mit dem Befehl DESCRIBE kann man sich Informationen zu Tabellen und Views ansehen (Spaltennamen, Datentyp der Spalten und 'NULL?', d.h. kann die Spalte leer bleiben oder ist ein Eintrag verpflichtend).

Beispiele:

```
SQL> DESC emp;
```

```
SQL> DESC dept;
```

HELP [befehl]

Mit dem Befehl HELP lassen sich je nach ORACLE-Version Hilfsinformationen zu SQL*Plus und zu SQL-Befehlen anzeigen. Die dazu notwendigen Dateien liegen im SQLPlus/admin-Verzeichnis und müssen vom Benutzer manuell ausgeführt werden.

COL [UMN] spalte FOR [MAT] [A]format

Mit dem Befehl COLUMN können Sie die Ausgabe einer (alpha-) numerischen Spalte formatieren.

Beispiele:

```
SQL> COL sal FORMAT L09,999.99;
```

```
SQL> COL job FORMAT A15;
```

```
SQL> COL sal CLEAR;
```

Weitere SQL*Plus befehle



- **CONN[ECT] [benutzername[/passwort]][@Dienstname]**
- **DESC[RIBE] [benutzername.]tabelle**
- **COL[UMN] spalte FOR[MAT] [A]format**

- **Beispiel:**
 - `SQL> CONN scott/tiger@orcl;`
 - `SQL> DESC emp;`
 - `SQL> DESC dept;`
 - `SQL> COL sal FORMAT L09,999.99;`
 - `SQL> COL job FORMAT A15;`
 - `SQL> COL sal CLEAR;`

2.2 SQL Developer

Nicht jeder möchte jedoch mit einem kommandozeilenbasierten Tool arbeiten. Dafür wurde ein Alternative veröffentlicht, der kostenlos von ORACLEs Webseite (<http://technet.ORACLE.com>) heruntergeladen werden kann. Es wird unter dem Namen Oracle SQL Developer kostenlos zur Verfügung gestellt.



Abb. Startsequenz des SQL Developer

2.2.1 Connections

Connection Name: Hier können Sie einen beliebigen Namen für Ihre Verbindung definieren.

Username: Benutzername für die Instanz.

Password: Passwort des Benutzers. Mit der Checkbox "Save Password" wird das Passwort von SQL*Developer permanent gespeichert.

Role: Default|Sysoper|Sysdba

Wenn Sie sich als Benutzer Sys anmelden möchten, geben Sie hier bitte die Optionen Sysdba oder Sysoper an.

Es stehen drei Verbindungsarten zur Verfügung:

Basic: Hier geben Sie den Hostnamen oder die IP-Adresse an. Dazu den Port (meist 1521) und den Namen der SID (oder den Parameter service_names).

Network Alias: Hier werden die Einträge aus der TNSNAMES.ORA angezeigt.

Advanced: Hier können Sie einen JDBC-URL Connect String angeben.

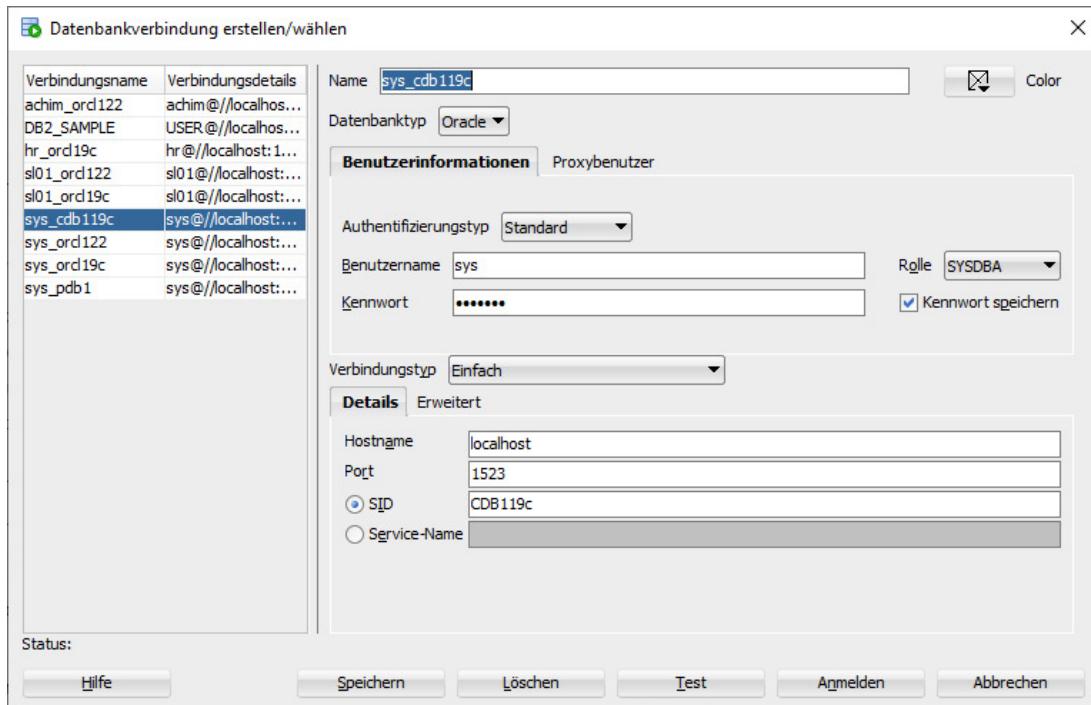
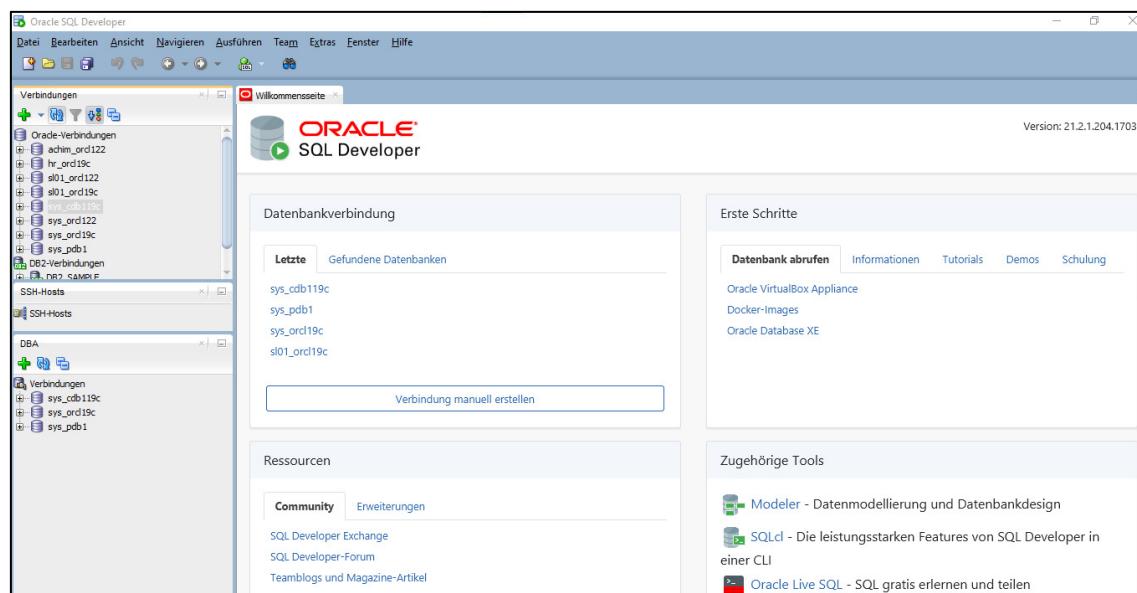


Abb. Connect-Informationen eingeben



2.2.2 SQL Worksheet

Im SQL Worksheet Fenster können die SQL-Befehle abgesetzt werden. Jedoch sind die SYSDBA-Optionen (startup, shutdown, recover database, ...) davon ausgenommen. Diese müssen über die Menue-Leiste Tools /SQL*Plus abgesetzt werden. Sollte diese Option noch grau sein, geben Sie bitte zuerst den Pfad zu einer SQL*Plus Version im Bereich Tools/Preferences/Database Connections/ SQL*Plus Executable an.

Folgende Kurzwahl-Tasten stehen zur Verfügung:

- F9 = Run = (Symbol: grünes Dreieck)
- F5 = Run Script = (Symbol: Dokument mit grünem Dreieck)
- F11 = Commit = (Symbol: Datenbank-Tonne mit grünem Hacken)
- F12 = Rollback = (Symbol: Datenbank-Tonne mit roten Pfeil)
- STRG-Q = Abbrechen
- F8 = SQL Historie = (Symbol: Tickende Uhr)
- F6 = Explain Plan = (Symbol: Datenbank mit Baum und grünem Pfeil)
- F6 = Autotrace = (Symbol: Datenbank mit Baum)
- STRG-D = Leeren = Bleistift mit Radiergummi

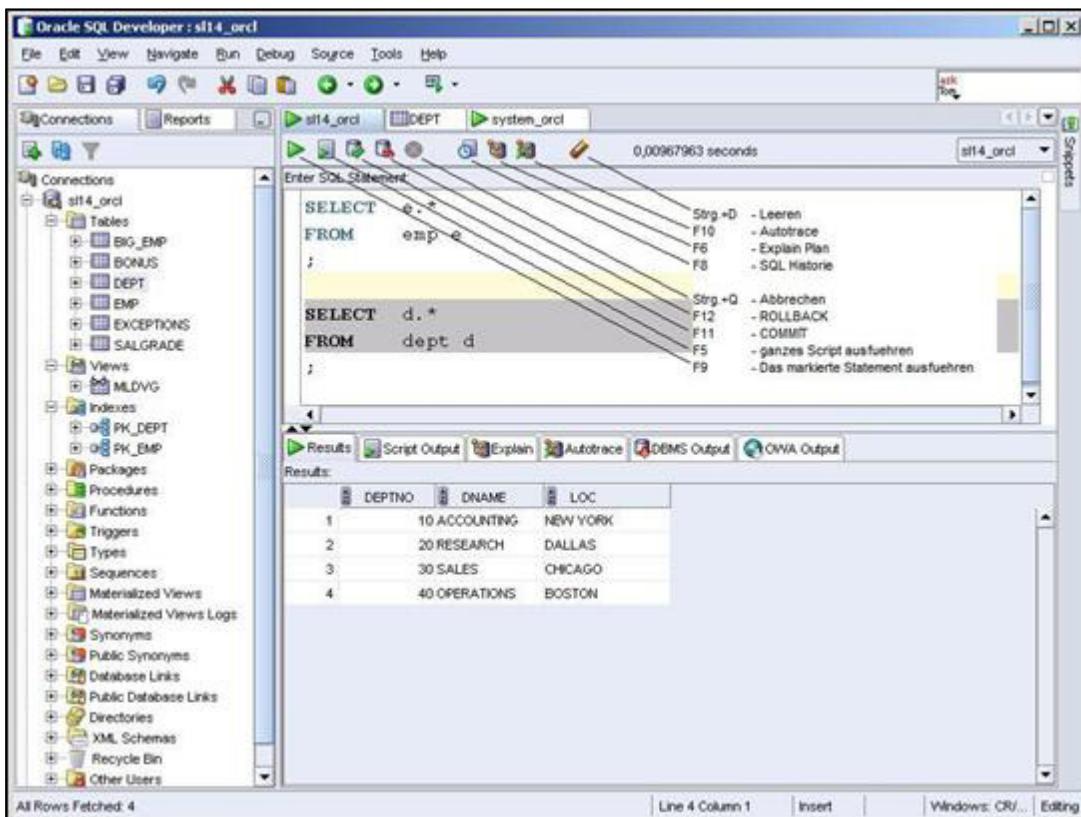


Abb.: Fenster SQL-Worksheet



2.2.3 Output-Möglichkeiten des SQL Developer

Sie können Texte mittels dbms_output oder htp.p ausgeben. Dafür existieren zwei Reiter (DBMS Output und OWA Output).

Klicken Sie zuerst jeweils links auf die Sprechblase um die Ausgabe freizuschalten.

Über das Diskettensymbol kann man die Ergebnisse in eine Textdatei exportieren.

Das Menü „File → Save As...“ steht einem natürlich auch zur Verfügung, um den aktuellen Teil des Fensters in eine Textdatei zu speichern.

The screenshot shows the Oracle SQL Developer interface. On the left is the 'Connections' sidebar with 'st14_orcl' selected. The main area has two SQL statements in the 'Enter SQL Statement' pane:

```

SELECT e.*  

FROM emp e  

;  
  

SELECT d.*  

FROM dept d  

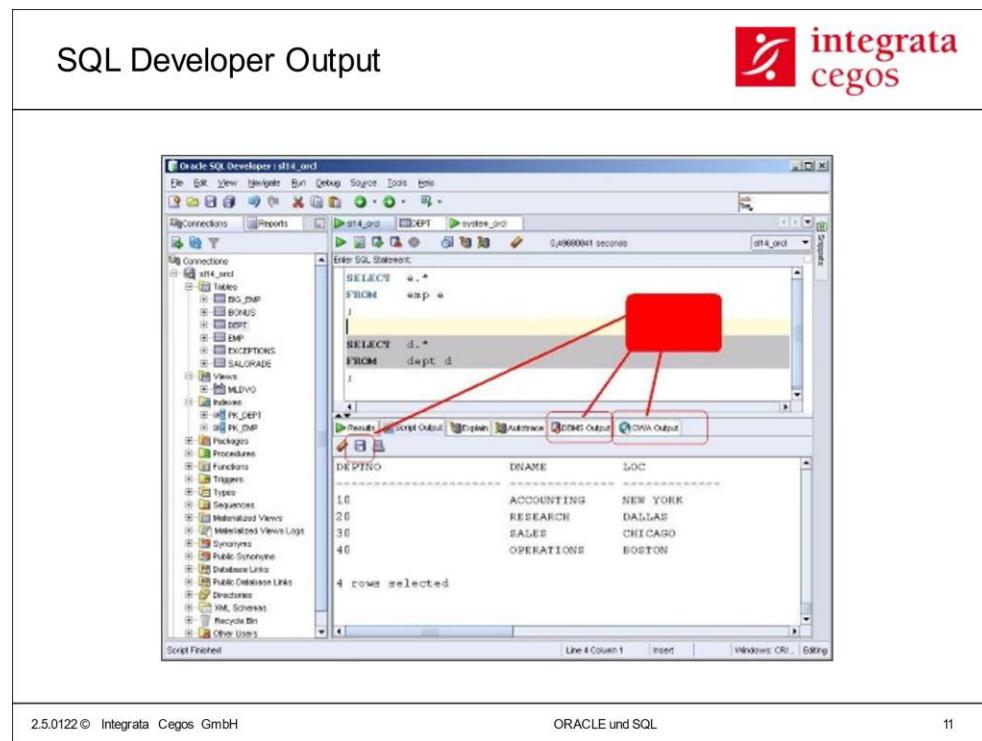
;
  
```

Below the statements is a toolbar with several buttons: 'Results' (highlighted with a red box), 'Script Output', 'Explain', 'Autotrace', 'DBMS Output' (highlighted with a red box), and 'OWA Output'. The results grid shows the following data:

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Text at the bottom of the results pane says '4 rows selected'.

Abb: SQL Developer Output



3

SELECT Anweisungen mit einer Tabelle

3.1	EBNF (Erweiterte Backus-Naur-Form)	3-4
3.2	Die einfache SELECT Anweisung	3-8
3.2.1	Kürzester SELECT.....	3-8
3.2.2	Aufbau der FROM-Klausel.....	3-9
3.2.3	Spaltenauswahl und Spaltenreihenfolge.....	3-10
3.2.4	Qualifizierte Angabe.....	3-12
3.2.5	Ausblendung doppelter Einträge.....	3-14
3.2.6	Verwendung von Aliasnamen	3-17
3.3	Operatoren im SELECT-Statement	3-20
3.3.1	NULL-Werte	3-22
3.3.2	Die NVL-Funktion	3-23
3.3.3	Die COALESCE-Funktion.....	3-25
3.3.4	Verwendung arithmetischer Operatoren	3-27
3.3.5	Der Verkettungsoperator.....	3-28
3.4	Die WHERE-Klausel	3-30
3.5	Vergleichsprädikate	3-32
3.5.1	Direkte Vergleichsprädikate	3-34
3.5.2	Intervallprädikat	3-35
3.5.3	Platzhaltersymbole (wildcard character)	3-37
3.5.4	Ähnlichkeitsprädikat.....	3-37

3.5.5	NULL-Test Prädikat.....	3-38
3.5.6	IN Prädikat	3-39
3.5.7	ALL-ANY-SOME Prädikate.....	3-41
3.5.8	EXISTS Prädikat.....	3-43
3.6	Die ORDER BY-Klausel	3-44
3.7	Zeilen limitieren	3-47
3.8	Datentypen	3-49
3.8.1	Datentypen	3-49

3 SELECT Anweisungen mit einer Tabelle

Der SELECT-Befehl wählt Daten zur Anzeige aus:

- Tabellen
- Views und
- Snapshots (Materialized Views oder MV).

Er gewährt also **lesenden** Zugriff auf die Datenbank. Im Weiteren ist der Einfachheit halber nur von Tabellen die Rede.

Achtung: Seit der Version 12.1.0.2 hat Oracle eine Unterscheidung bei den Rechten eingeführt. Es gibt nun ein neues Recht mit Namen "READ" bzw. "READ ANY TABLE"

	SELECT	READ
SELECT auf die Tabelle	X	X
SELECT FOR UPDATE auf Tabelle	X	
LOCK TABLE auf Tabelle	X	

Ist die SELECT-Anweisung erfolgreich, wird eine Ergebnistabelle zurückgeliefert. Vor dem Aufruf beachten: Besitzen Sie das CONNECT-Recht, so müssen Sie zusätzlich entweder Eigentümer der Tabelle sein oder das SELECT-Zugriffsrecht für jede verwendete Spalte besitzen.

Bei den Klauseln des SELECT-Statements muss die angegebene Reihenfolge eingehalten werden.

3.1 EBNF (Erweiterte Backus-Naur-Form)

Für die **exakten SQL-Syntaxspezifikationen** wird hier die **EBNF** (erweiterte Backus-Naur-Form) verwendet.

Die Syntaxspezifikationen werden nicht immer komplett auf einmal angegeben, sondern schrittweise erarbeitet.

Ebenso ist aus Einfachheits-, Verständlichkeits- und Übersichtlichkeitsgründen darauf **verzichtet** worden, die **exakte mathematische Korrektheit** der EBNF-Syntax permanent zu befolgen. Vielmehr sind die **SQL-Befehle klar und vor allem einfach dargestellt** worden. Syntaxfokus wird auf den aktuell betrachteten Sachverhalt gelegt.

Das Ziel dieses Seminars ist die Vermittlung von SQL für den praktischen Einsatz unter dem RDBMS ORACLE und keine genaue mathematische Spezifikation der SQL-Sprache.

Des Weiteren sind im Anhang dieser Broschüre entsprechende **Syntaxdiagramme** der Befehle enthalten.

Crashkurs in EBNF:

- Reservierte SQL-Worte werden großgeschrieben:

```
SELECT
```

- Optionsymbol
(kein- oder einmal): []

```
SELECT [ALL]
```

- Wiederholungssymbol
(kein-, ein- oder mehrmals): { }

```
FROM Tabelle {, Tabelle}
```

- Definitionssymbol: ::=
satz ::= wort {, wort}
- Alternativtrennung: |
operator ::= + | -

Alle hier nicht erwähnten Zeichen sind als Literale zu verstehen, Objektnamen erscheinen hier *kursiv*.

<p>EBNF - (Erweiterte Backus-Naur-Form)</p> <ul style="list-style-type: none"> ▪ Reservierte Worte in Großbuchstaben: SELECT ▪ Optionsymbol (kein- oder einmal): [] SELECT [ALL] ▪ Wiederholungssymbol (kein-, ein- oder mehrmals): { } FROM tabelle {, tabelle} ▪ Definitionssymbol: ::= satz ::= wort {, wort} ▪ Alternativtrennung: operator ::= + - 	
2.5.0122 © Integrata Cegos GmbH	ORACLE und SQL

SELECT Spezifikation

```

select-statement ::=

    select-spezifikation
    { [UNION [ALL] select-spezifikation]
      | [INTERSECT select-spezifikation]
      | [MINUS select-spezifikation] }
    [ORDER BY      orderbyausdruck];

select-spezifikation ::=

    SELECT [ALL | DISTINCT] spaltenausdruck
    FROM          tabellenausdruck
    [ WHERE        bedingungsausdruck-where ]
    [ GROUP BY    gruppenausdruck ]
    [ HAVING       bedingungsausdruck-having ];

```

- spaltenausdruck ist eine durch Kommata getrennte Liste der anzugegenden Spalten
- tabellenausdruck definiert die Quellen, aus welchen die Daten gefiltert werden
- bedingungsausdruck-where definiert einen Filter auf die Datensätze einer Tabelle
- gruppenausdruck ist eine durch Kommata getrennte Liste der Spalten, nach welchen die Daten gruppiert werden

- bedingungsausdruck-having definiert die Bedingungen an die gruppierten Daten
- orderbyausdruck ist eine durch Kommata getrennte Liste der Spalten, nach welchen die anzugebenden Ergebnisse sortiert werden sollen.

Die Einzelteile des select-statements werden in den folgenden Kapiteln erarbeitet.

SELECT Statement	 integrata cegos
<pre>select-statement ::= select-spezifikation { UNION [ALL] INTERSECT MINUS select-spezifikation } [ORDER BY orderbyausdruck] ;</pre>	
2.5.0122 © Integrata Cegos GmbH	ORACLE und SQL
	3

SELECT Spezifikation



```
select-spezifikation ::=
```

```
  SELECT [ ALL | DISTINCT ] spaltenausdruck
        FROM      tabellenausdruck
        [ WHERE     bedingungsausdruck-where ]
        [ GROUP BY  gruppenausdruck ]
        [ HAVING    bedingungsausdruck-having ]
;
```

3.2 Die einfache SELECT Anweisung

3.2.1 Kürzester SELECT

Eine minimale SELECT Anweisung (SELECT Statement) **muss** die Klauseln:

- SELECT und
- FROM

enthalten. **Diese zwei Klauseln sind obligatorisch.** Sie müssen in jeder SELECT Abfrage vorhanden sein.

In der SELECT-Klausel werden die anzuzeigenden Spalten eingegeben. Die Angabe der an der "Datenlieferung" beteiligten Tabellen erfolgt in der FROM-Klausel.

Sollen **alle Spalten einer Tabelle** angezeigt werden, so wird ein Stern "*" als Spaltenausdruck verwendet. Die Spalten werden in diesem Fall in der Reihenfolge ausgegeben, in welcher sie beim Erstellen der Tabelle angegeben worden sind.

Beispiel:

```
SQL> SELECT *  
      FROM emp;
```

Syntax:

```
SELECT      *  
FROM       Tabelle;
```

Kürzester SELECT



NUR lesender Zugriff auf die Daten

- **SELECT** und **FROM** Klauseln sind obligatorisch! Ohne diese kann keine Abfrage durchgeführt werden.

Syntax:

```
SELECT      *
FROM    Tabelle;
```

Beispiel:

```
SQL> SELECT      *
      FROM emp;
```

2.5.0122 © Integrata Cegos GmbH

ORACLE und SQL

5

3.2.2 Aufbau der **FROM**-Klausel

In der **FROM**-Klausel werden die Quellen angegeben, welche die Daten für die Filterung zur Verfügung stellen.

Der Einfachheit halber können wir an dieser Stelle sagen, dass in der **FROM**-Klausel Tabellen aufgezählt werden. Deswegen ist die Syntax der **FROM**-Klausel an dieser Stelle noch nicht vollständig. Später wird feiner differenziert (s. **JOIN** und Unterabfragen).

Beispiel:

```
SQL> SELECT      *
      FROM          emp;
SQL> SELECT      *
      FROM          dept;
SQL> SELECT      *
      FROM          salgrade;
```

Syntax:

```
SELECT      *
FROM        Tabelle { , Tabelle } ;
```

3.2.3 Spaltenauswahl und Spaltenreihenfolge

Werden gezielt bestimmte Spalten aus der Tabelle ausgewählt oder wird die Reihenfolge der Spaltenanzeige manipuliert, so werden die Spalten mit ihren Namen aufgezählt und die Aufzählung durch Komma-ta getrennt. Die Anzeige der Spalten erfolgt so, wie sie in dem SELECT-Statement angegeben wurden.

Beispiel:

```
SQL> SELECT empno, ename, sal, job  
      FROM emp;
```

EMPNO	ENAME	SAL	JOB
7369	SMITH	800	CLERK
7499	ALLEN	1600	SALESMAN
7521	WARD	1250	SALESMAN
7566	JONES	2975	MANAGER
7654	MARTIN	1250	SALESMAN
7698	BLAKE	2850	MANAGER
7782	CLARK	2450	MANAGER
7788	SCOTT	3000	ANALYST
7839	KING	5000	PRESIDENT
7844	TURNER	1500	SALESMAN
7876	ADAMS	1100	CLERK
7900	JAMES	950	CLERK
7902	FORD	3000	ANALYST
7934	MILLER	1300	CLERK

Syntax:

```
SELECT      * | Spalte {, Spalte}  
FROM       Tabelle {, Tabelle} ;
```

FROM Klausel und Spaltenauswahl



Syntax:

```
SELECT * | Spalte {, Spalte}
      FROM Tabelle {,Tabelle};
```

Beispiel:

```
SQL> SELECT empno, ename, sal, job
      FROM emp;
SQL> SELECT *
      FROM dept;
SQL> SELECT *
      FROM salgrade;
```

3.2.4 Qualifizierte Angabe

Vor den Spaltennamen kann der Tabellenname gesetzt werden, getrennt durch einen Punkt, um die eindeutige Zuordnung einer Spalte zu einer Tabelle **qualifiziert** anzugeben.

Tabellen können auch qualifiziert angesprochen werden, indem vor dem Tabellennamen der Besitzer der Tabelle (Schemaname) gefolgt von einem Punkt gesetzt wird.

Dadurch verbessert sich neben der Übersichtlichkeit auch die **Compilerzeit** der SQL-Befehle, da der Compiler nicht selbst suchen muss, in welcher Tabelle sich eine Spalte und in welchem Schema sich eine Tabelle befindet.

Beispiel:

```
SQL> SELECT
      emp.empno,
      emp.ename,
      s107.emp.sal, -- Vorsicht (s101 vs s107)
      emp.job
   FROM  s101.emp;
```

Syntax:

```
SELECT  [ [Schema.]Tabelle.]Spalte
        { , [ [Schema.]Tabelle.]Spalte }
   FROM    [Schema.]Tabelle;
```

Spaltenauswahl, qualifizierte Angaben



- Erhöhung der Übersichtlichkeit
- Beschleunigung des Compilervorgangs
- Syntax:

```
SELECT [ [Schema.] Tabelle.] Spalte  
FROM   [Schema.] Tabelle;
```

- Beispiel:

```
SQL> SELECT  
      empno,  
      ename,  
      job  
    FROM   emp;  
SQL> SELECT  
      emp.empno,  
      emp.ename,  
      s107.emp.sal, -- Vorsicht, Schemaname beachten  
      emp.job  
    FROM   s101.emp;
```

3.2.5 Ausblendung doppelter Einträge

In der bislang gezeigten einfachsten Form werden *alle* Zeilen in den gewählten Spalten einer Tabelle zurückgeliefert. Sollen doppelte Ausgaben vermieden werden, kann man dem SELECT das Schlüsselwort **DISTINCT** anhängen.

Beispiel:

```
SQL> SELECT DISTINCT  
          emp.job  
     FROM emp;
```

```
JOB  
-----  
CLERK  
SALESMAN  
PRESIDENT  
MANAGER  
ANALYST
```

Syntax:

```
SELECT [ALL | DISTINCT] Spalte { , Spalte }  
      FROM      Tabelle;
```

Hinweis: DISTINCT verursacht ggf. (bei großen Datenmengen) einen inneren Sort. Sortieren ist eine der teureren Operationen und sollte möglichst sparsam eingesetzt werden.

SELECT DISTINCT																											
<pre>SQL> SELECT DISTINCT emp.job FROM emp;</pre> <table border="1" style="margin-top: 10px;"> <thead> <tr><th>JOB</th></tr> </thead> <tbody> <tr><td>ANALYST</td></tr> <tr><td>CLERK</td></tr> <tr><td>MANAGER</td></tr> <tr><td>PRESIDENT</td></tr> <tr><td>SALESMAN</td></tr> </tbody> </table> <p><u>Syntax:</u></p> <pre>SELECT [ALL DISTINCT] Spalte {, Spalte} FROM Tabelle ;</pre>	JOB	ANALYST	CLERK	MANAGER	PRESIDENT	SALESMAN	<pre>SQL> SELECT DISTINCT emp.job, emp.deptno FROM emp;</pre> <table border="1" style="margin-top: 10px;"> <thead> <tr><th>JOB</th><th>DEPTNO</th></tr> </thead> <tbody> <tr><td>ANALYST</td><td>20</td></tr> <tr><td>CLERK</td><td>10</td></tr> <tr><td>CLERK</td><td>20</td></tr> <tr><td>CLERK</td><td>30</td></tr> <tr><td>MANAGER</td><td>10</td></tr> <tr><td>MANAGER</td><td>20</td></tr> <tr><td>MANAGER</td><td>30</td></tr> <tr><td>PRESIDENT</td><td>10</td></tr> <tr><td>SALESMAN</td><td>30</td></tr> </tbody> </table>	JOB	DEPTNO	ANALYST	20	CLERK	10	CLERK	20	CLERK	30	MANAGER	10	MANAGER	20	MANAGER	30	PRESIDENT	10	SALESMAN	30
JOB																											
ANALYST																											
CLERK																											
MANAGER																											
PRESIDENT																											
SALESMAN																											
JOB	DEPTNO																										
ANALYST	20																										
CLERK	10																										
CLERK	20																										
CLERK	30																										
MANAGER	10																										
MANAGER	20																										
MANAGER	30																										
PRESIDENT	10																										
SALESMAN	30																										
		2.5.0122 © Integrata Cegos GmbH ORACLE und SQL 8																									

Bei der Ausblendung doppelter Einträge werden immer die ganzen anzuzeigenden Zeilen (Vektoren, Tupel) miteinander verglichen. Eindeutig sind also die anzuzeigenden "Zeilenausprägungen".

Beispiel (Vergleich mit und ohne DISTINCT):

```
SQL> SELECT DISTINCT
      emp.job,
      emp.deptno
    FROM   emp;
```

JOB	DEPTNO
-----	-----
MANAGER	20
PRESIDENT	10
CLERK	10
SALESMAN	30
ANALYST	20
MANAGER	30
MANAGER	10
CLERK	30
CLERK	20

```
SQL> SELECT
      emp.job,
      emp.deptno
    FROM   emp;
```

JOB	DEPTNO
CLERK	20
SALESMAN	30
SALESMAN	30
MANAGER	20
SALESMAN	30
MANAGER	30
MANAGER	10
ANALYST	20
PRESIDENT	10
SALESMAN	30
CLERK	20
CLERK	30
ANALYST	20
CLERK	10

SELECT DISTINCT (Zeilenvergleich)



```
SQL> SELECT
      emp.job,
      emp.deptno
    FROM emp;
```

JOB	DEPTNO
CLERK	20
SALESMAN	30
SALESMAN	30
MANAGER	20
SALESMAN	30
MANAGER	30
MANAGER	10
ANALYST	20
PRESIDENT	10
SALESMAN	30
CLERK	20
CLERK	30
ANALYST	20
CLERK	10

```
SQL> SELECT DISTINCT
      emp.job,
      emp.deptno
    FROM emp;
```

JOB	DEPTNO
MANAGER	20
PRESIDENT	10
CLERK	10
SALESMAN	30
ANALYST	20
MANAGER	30
MANAGER	10
CLERK	30
CLERK	20

3.2.6 Verwendung von Aliasnamen

3.2.6.1 Alias für Spalten

Für Spaltennamen der Datenbank können in der SELECT-Anweisung auch Aliasnamen vergeben werden, die dann in der Ausgabe als Spaltenüberschrift erscheinen. Das Schlüsselwort AS ist dabei optional. Es wird aber empfohlen wegen der Übersichtlichkeit das AS zu benutzen.

Die Spaltenbezeichnung wird nur für dieses eine Statement angepasst und nicht in der Datenbank geändert.

Wird der Aliasname nicht in doppelte Anführungszeichen gesetzt (s.u. Arbeit), so wird er (wie alle Spaltennamen) in Großbuchstaben dargestellt.

Falls ein Alias Leer- oder Sonderzeichen enthalten soll (was nur bedingt ratsam ist), **muss** es unbedingt in doppelte Anführungszeichen gesetzt werden.

Beispiel:

```
SQL> SELECT DISTINCT
          emp.job           AS Arbeit,
          emp.deptno        AS "Abteilungsnummer"
      FROM    emp;
```

ARBEIT	Abteilungsnummer
MANAGER	20
PRESIDENT	10
CLERK	10
SALESMAN	30
ANALYST	20
MANAGER	30
MANAGER	10
CLERK	30
CLERK	20

Syntax:

```
SELECT  Spalte [ [AS] Spaltenalias]
        {, Spalte [ [AS] Spaltenalias ] }
      FROM   Tabelle;
```

Spaltenalias



Syntax:

```
SELECT      Spalte [AS] Spaltenalias
FROM        Tabelle;
```

Beispiel:

```
SQL>      SELECT DISTINCT
              emp.job      AS Arbeit,
              emp.deptno   AS "Abteilungsnummer"
            FROM  emp;
```

ARBEIT	Abteilungsnummer
ANALYST	20
CLERK	10
CLERK	20

2.5.0122 © Integrata Cegos GmbH ORACLE und SQL 10

3.2.6.2 Alias für Tabellen

Auch für Tabellennamen kann ein Aliasname vergeben werden. Wird ein Tabellenalias angegeben, so muss er im gesamten SELECT-Statement statt des originalen Tabellennamens verwendet werden.

Hinweis: Hier ist zu beachten, dass kein Schlüsselwort AS benutzt werden darf, ansonsten liefert ORACLE eine Fehlermeldung.

Beispiel:

```
SQL>      SELECT DISTINCT
              e.job      AS Arbeit,
              e.deptno   AS "Verdienst"
            FROM  s101.emp e;
```

ARBEIT	Verdienst
MANAGER	20
PRESIDENT	10
CLERK	10
SALESMAN	30
ANALYST	20
MANAGER	30
MANAGER	10
CLERK	30
CLERK	20

Syntax:

```
SELECT      Tabellenalias . Spalte [AS] Spaltenalias  
FROM        Tabelle Tabellenalias;
```

Tabellenalias



Syntax:

```
SELECT      Spalte [[AS] Spaltenalias]  
           [, Spalte [[AS] Spaltenalias]]  
FROM        Tabelle Tabellenalias;
```

Beispiel:

```
SQL> SELECT DISTINCT  
          e.job          AS Arbeit,  
          e.deptno       AS "Abteilungsnummer"  
     FROM s101.emp e;
```

ARBEIT	Abteilungsnummer
ANALYST	20
CLERK	10
CLERK	20

3.3 Operatoren im SELECT-Statement

Spaltenwerte können entweder so ausgegeben werden, wie sie in der Tabelle abgelegt sind oder in einer anderen für die Anzeige manipulierten Form (z.B. Bruttoreise aus Nettopreisen berechnen). Die Daten in den Tabellen werden dabei NICHT geändert. Die Änderung ist nur für diese eine Anzeige wirksam.

Zu den auf die Daten anwendbaren Operatoren zählen zum Beispiel arithmetische Operatoren (+, -, *, /) oder verschiedene Funktionen. Manche Funktionen sind extra für einen bestimmten Datentyp oder einen bestimmten Zweck entwickelt worden (`NVL`).

Beispiel:

```
SQL> SELECT
      e.empno          AS PersNr,
      e.sal            AS "Verdienst",
      (e.sal * 1.1)    AS "Hoheres Gehalt"
   FROM emp e;
```

PERSNR	Verdienst	Hoheres Gehalt
7369	800	880
7499	1600	1760
7521	1250	1375
7566	2975	3272,5
7654	1250	1375
7698	2850	3135
7782	2450	2695
7788	3000	3300
7839	5000	5500
7844	1500	1650
7876	1100	1210
7900	950	1045
7902	3000	3300
7934	1300	1430

Syntax:

```
SELECT Spalte operator Wert
      FROM      Tabelle;
```

operator ::=

arithmetischer-operator | verkettungsoperator

arithmetischer-operator ::= + | - | / | *

verkettungsoperator ::= ||

Spaltenoperatoren																						
<p>Syntax:</p> <pre>SELECT Spalte operator Wert FROM Tabelle;</pre> <p>operator ::= arithmetisch verkettung</p> <p>arithmetisch ::= + - / *</p> <p>Verkettung ::= </p> <p>▪ Einfache Operation:</p> <pre>SELECT e.empno AS PersNr, e.sal AS "Gehalt", (e.sal * 1.1)AS "Gehalt_neu" FROM emp e;</pre>																						
<table border="1" style="margin: auto; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center; padding: 2px;">PERSNR</th> <th style="text-align: center; padding: 2px;">Gehalt</th> <th style="text-align: center; padding: 2px;">Gehalt_neu</th> </tr> </thead> <tbody> <tr><td style="text-align: center; padding: 2px;">7369</td><td style="text-align: center; padding: 2px;">800</td><td style="text-align: center; padding: 2px;">880</td></tr> <tr><td style="text-align: center; padding: 2px;">7499</td><td style="text-align: center; padding: 2px;">1600</td><td style="text-align: center; padding: 2px;">1760</td></tr> <tr><td style="text-align: center; padding: 2px;">7521</td><td style="text-align: center; padding: 2px;">1250</td><td style="text-align: center; padding: 2px;">1375</td></tr> <tr><td style="text-align: center; padding: 2px;">7566</td><td style="text-align: center; padding: 2px;">2975</td><td style="text-align: center; padding: 2px;">3272,5</td></tr> <tr><td style="text-align: center; padding: 2px;">7654</td><td style="text-align: center; padding: 2px;">1250</td><td style="text-align: center; padding: 2px;">1375</td></tr> <tr><td style="text-align: center; padding: 2px;">7698</td><td style="text-align: center; padding: 2px;">2850</td><td style="text-align: center; padding: 2px;">3135</td></tr> </tbody> </table>		PERSNR	Gehalt	Gehalt_neu	7369	800	880	7499	1600	1760	7521	1250	1375	7566	2975	3272,5	7654	1250	1375	7698	2850	3135
PERSNR	Gehalt	Gehalt_neu																				
7369	800	880																				
7499	1600	1760																				
7521	1250	1375																				
7566	2975	3272,5																				
7654	1250	1375																				
7698	2850	3135																				
<small>2.5.0122 © Integrata Cegos GmbH</small> ORACLE und SQL 12																						

3.3.1 NULL-Werte

Ein **NULL-Wert** ist ein:

- nicht existenter,
- nicht erfasster oder
- nicht bekannter

Wert.

Ist in einem Datensatz in einer Spalte **nichts** eingetragen, so enthält diese Spalte implizit den Wert **NULL**, unabhängig davon, von welchem Datentyp die Werte in dieser Spalte sind.

Der Wert **NULL** ist **nicht zu verwechseln** mit:

- der Zahl "0" oder
- einem Leerzeichen oder
- einer leeren Zeichenkette!

Mit dem Wert **NULL** kann nicht operiert werden, deswegen muss die **NULL** für den temporären Einsatz situationsabhängig interpretiert werden.

NULL	 integrata cegos
<ul style="list-style-type: none"> ▪ Unabhängig vom Spaltendatentyp ▪ Ein NULL-Wert ist ein: <ul style="list-style-type: none"> ▪ nicht <u>existenter</u> Wert, ▪ nicht <u>erfasster</u> Wert oder ▪ nicht <u>bekannter</u> Wert ▪ Der Wert NULL ist nicht zu verwechseln mit: <ul style="list-style-type: none"> ▪ der Zahl "0" ▪ einem Leerzeichen ▪ einer leeren Zeichenkette 	

3.3.2 Die NVL-Funktion

Sie dient dazu, einem NULL-Wert zur Ausgabe(!) einen Ersatzwert zuzuweisen, der allerdings vom gleichen, der Spalte zugewiesenen Datentyp sein muss. Dies macht besonders dann Sinn, wenn auf Spaltenwerte Funktionen oder arithmetische Operationen angewandt werden sollen.

Beispiel:

```
SQL> SELECT
          e.empno,
          e.ename,
          NVL(e.comm, 15)           AS "Provision"
     FROM   emp e;
```

EMPNO	ENAME	Provision
7369	SMITH	15
7499	ALLEN	300
7521	WARD	500
7566	JONES	15
7654	MARTIN	1400
7698	BLAKE	15
7782	CLARK	15
7788	SCOTT	15
7839	KING	15
7844	TURNER	0
7876	ADAMS	15
7900	JAMES	15
7902	FORD	15
7934	MILLER	15

Definition der NVL-Funktion:

NVL (par1, par2) = **par1** falls par1 IS NOT NULL,
 par2 sonst

NVL



- Definition der NVL Funktion:
NVL (par1, par2) =
par1 Spalte, deren Werte angezeigt werden, falls
par1 IS NOT NULL,
par2 Alternativ-Wert für NULL-Werte
- Beispiel:

```
SELECT
    e.ename,
    NVL(e.comm, 15) AS "Provision"
FROM emp e;
```

ENAME	Provision
SMITH	15
ALLEN	300
WARD	500
JONES	15
MARTIN	1400
BLAKE	15
CLARK	15
SCOTT	15
KING	15
TURNER	0
ADAMS	15
JAMES	15
FORD	15
MILLER	15

3.3.3 Die COALESCE-Funktion

Sie dient genauso wie die NVL Funktion einen Ersatzwert zuzuweisen, der allerdings vom gleichen, der Spalte zugewiesenen Datentyp sein muss. Sie hat aber gegenüber der NVL Funktion folgende Vorteile

- Sie ist ANSI Konform (andere Datenbanken kennen diese Funktion auch (z.B. SQL Server und PostgreSQL))
- Sie kann mehr als zwei Parameter besitzen. Genommen wird der erste nicht NULL Wert von Links
- Sie berechnet den nächsten (rechten) Wert nur wenn der links davon NULL ist. Die NVL berechnet immer den rechten Ausdruck. Wenn das eine Funktion ist, kann das zu Laufzeitverzögerungen führen

Beispiel:

```
SELECT COALESCE(null,null,null,2,null) FROM dual;
```

⇒ 2

```
SQL> SELECT
      e.empno,
      e.ename,
      COALESCE(e.comm, e.mgr, 15)
           AS "Provision"
   FROM emp e;
```

EMPNO	ENAME	Provision
7369	SMITH	7902
7499	ALLEN	300
7521	WARD	500
7566	JONES	7839
7654	MARTIN	1400
7698	BLAKE	7839
7782	CLARK	7839
7788	SCOTT	7566
7839	KING	15
7844	TURNER	0
7876	ADAMS	7788
7900	JAMES	7698
7902	FORD	7566
7934	MILLER	7782

Definition der COALESCE Funktion:

COALESCE (par1, par2 { , parX }) =
erster Parameter aus der Liste, der NOT NULL ist.

COALESCE																																															
<ul style="list-style-type: none"> ▪ Definition der COALESCE Funktion: COALESCE (par1, par2 { , parX }) = erster Parameter der NOT NULL ist ▪ Beispiel: <pre>SELECT e.empno, e.ename, COALESCE(e.comm, e.mgr, 15) AS "Provision" FROM emp e;</pre> 																																															
<table border="1"> <thead> <tr> <th>EMPNO</th> <th>ENAME</th> <th>Provision</th> </tr> </thead> <tbody> <tr><td>7369</td><td>SMITH</td><td>7902</td></tr> <tr><td>7499</td><td>ALLEN</td><td>300</td></tr> <tr><td>7521</td><td>WARD</td><td>500</td></tr> <tr><td>7566</td><td>JONES</td><td>7839</td></tr> <tr><td>7654</td><td>MARTIN</td><td>1400</td></tr> <tr><td>7698</td><td>BLAKE</td><td>7839</td></tr> <tr><td>7782</td><td>CLARK</td><td>7839</td></tr> <tr><td>7788</td><td>SCOTT</td><td>7566</td></tr> <tr><td>7839</td><td>KING</td><td>15</td></tr> <tr><td>7844</td><td>TURNER</td><td>0</td></tr> <tr><td>7876</td><td>ADAMS</td><td>7788</td></tr> <tr><td>7900</td><td>JAMES</td><td>7698</td></tr> <tr><td>7902</td><td>FORD</td><td>7566</td></tr> <tr><td>7934</td><td>MILLER</td><td>7782</td></tr> </tbody> </table>	EMPNO	ENAME	Provision	7369	SMITH	7902	7499	ALLEN	300	7521	WARD	500	7566	JONES	7839	7654	MARTIN	1400	7698	BLAKE	7839	7782	CLARK	7839	7788	SCOTT	7566	7839	KING	15	7844	TURNER	0	7876	ADAMS	7788	7900	JAMES	7698	7902	FORD	7566	7934	MILLER	7782		
EMPNO	ENAME	Provision																																													
7369	SMITH	7902																																													
7499	ALLEN	300																																													
7521	WARD	500																																													
7566	JONES	7839																																													
7654	MARTIN	1400																																													
7698	BLAKE	7839																																													
7782	CLARK	7839																																													
7788	SCOTT	7566																																													
7839	KING	15																																													
7844	TURNER	0																																													
7876	ADAMS	7788																																													
7900	JAMES	7698																																													
7902	FORD	7566																																													
7934	MILLER	7782																																													
2.5.0122 © Integrata Cegos GmbH		ORACLE und SQL	15																																												

3.3.4 Verwendung arithmetischer Operatoren

Auf Spalteninhalte können Funktionen oder arithmetische Operatoren angewendet werden. Arithmetische Operatoren sind:

*	/	+	-
---	---	---	---

Multiplikation und Division haben Vorrang vor Addition und Subtraktion, wie gewohnt können Klammern zur Beeinflussung der Berechnungsreihenfolge benutzt werden.

Jede arithmetische Operation, die auf einen NULL-Wert angewandt wird (z.B. `4711 + NULL = NULL`), liefert als Ergebnis wieder NULL zurück. Aus diesem Grund sollte die NVL- oder COALESCE-Funktion verwendet werden.

Hinweis: Aus Performancegründen sollte man jedoch so wenige Funktionen wie möglich einsetzen, da sie für die Berechnung Rechenpower und Zeit brauchen.

Beispiel:

```
SQL> SELECT
      e.empno,
      e.sal AS "Verdienst",
      (e.sal * 1.2) AS "VerdienstHoch",
      (e.sal / 2) AS "VerdienstHalb",
      (e.comm + e.sal) AS "FALSCH",
      (NVL(e.comm, 0) + NVL(e.sal, 0)) AS "Gehalt"
   FROM emp e;
```

EMPNO	Verdienst	VerdienstHoch	VerdienstHalb	FALSCH	Gehalt
7369	800	960	400		800
7499	1600	1920	800	1900	1900
7521	1250	1500	625	1750	1750
7566	2975	3570	1487,5		2975
7654	1250	1500	625	2650	2650
7698	2850	3420	1425		2850
7782	2450	2940	1225		2450
7788	3000	3600	1500		3000
7839	5000	6000	2500		5000
7844	1500	1800	750	1500	1500
7876	1100	1320	550		1100
7900	950	1140	475		950
7902	3000	3600	1500		3000
7934	1300	1560	650		1300

Arithmetische Operatoren	
<ul style="list-style-type: none"> ▪ 4 Grundfunktionen der Mathematik: * / + - ▪ Angewandt auf NULL ergeben sie NULL!!! ▪ 4711 + NULL = NULL ▪ Beispiel: <pre>SELECT e.empno, e.sal AS "Verdienst", (e.sal * 1.2) AS "VerdienstHoch", (e.sal / 2) AS "VerdienstHalb", (e.comm + e.sal) AS "FALSCH", (NVL(e.comm, 0) + NVL(e.sal, 0)) AS "Gehalt" FROM emp e;</pre> 	

2.5.0122 © Integrata Cegos GmbH

ORACLE und SQL

16

3.3.5 Der Verkettungsoperator

Er wird dargestellt durch zwei vertikale Striche: "||", auch Pipezeichen genannt. Mit Hilfe des Verkettungsoperators können Spalten miteinander oder mit Zeichenketten ("Literale") verknüpft werden.

Literale in der SELECT-Anweisung müssen in **einfachen** Anführungszeichen stehen, sonst werden sie als Spaltennamen interpretiert.

Beispiel:

```
SQL> SELECT
      e.ename || ' verdient ' || e.sal
      AS "Einkommen pro Mitarbeiter"
   FROM emp e;
```

```
Einkommen pro Mitarbeiter
-----
SMITH verdient 800
ALLEN verdient 1600
WARD verdient 1250
JONES verdient 2975
MARTIN verdient 1250
BLAKE verdient 2850
CLARK verdient 2450
SCOTT verdient 3000
KING verdient 5000
TURNER verdient 1500
ADAMS verdient 1100
JAMES verdient 950
FORD verdient 3000
MILLER verdient 1300
```

Verkettungsoperator	 integrata cegos															
<ul style="list-style-type: none"> ▪ Verkettungsoperator ▪ <u>Beispiel:</u> <pre>SELECT e.ename ' verdient ' e.sal AS "Gehalt pro Mitarbeiter" FROM emp e;</pre> <p>Alternativ kann auch die Funktion concat verwendet werden:</p> <pre>SELECT concat('Hallo',' Kurs') FROM dual;</pre> 	<table border="1"> <thead> <tr> <th>Gehalt pro Mitarbeiter</th></tr> </thead> <tbody> <tr><td>SMITH verdient 800</td></tr> <tr><td>ALLEN verdient 1600</td></tr> <tr><td>WARD verdient 1250</td></tr> <tr><td>JONES verdient 2975</td></tr> <tr><td>MARTIN verdient 1250</td></tr> <tr><td>BLAKE verdient 2850</td></tr> <tr><td>CLARK verdient 2450</td></tr> <tr><td>SCOTT verdient 3000</td></tr> <tr><td>KING verdient 5000</td></tr> <tr><td>TURNER verdient 1500</td></tr> <tr><td>ADAMS verdient 1100</td></tr> <tr><td>JAMES verdient 950</td></tr> <tr><td>FORD verdient 3000</td></tr> <tr><td>MILLER verdient 1300</td></tr> </tbody> </table>	Gehalt pro Mitarbeiter	SMITH verdient 800	ALLEN verdient 1600	WARD verdient 1250	JONES verdient 2975	MARTIN verdient 1250	BLAKE verdient 2850	CLARK verdient 2450	SCOTT verdient 3000	KING verdient 5000	TURNER verdient 1500	ADAMS verdient 1100	JAMES verdient 950	FORD verdient 3000	MILLER verdient 1300
Gehalt pro Mitarbeiter																
SMITH verdient 800																
ALLEN verdient 1600																
WARD verdient 1250																
JONES verdient 2975																
MARTIN verdient 1250																
BLAKE verdient 2850																
CLARK verdient 2450																
SCOTT verdient 3000																
KING verdient 5000																
TURNER verdient 1500																
ADAMS verdient 1100																
JAMES verdient 950																
FORD verdient 3000																
MILLER verdient 1300																

2.5.0122 © Integrata Cegos GmbH

ORACLE und SQL

18

SELECT Spezifikation	 integrata cegos
<pre>select-spezifikation ::= SELECT [ALL DISTINCT] spaltenausdruck FROM tabellenausdruck [WHERE bedingungsausdruck-where] ... ;</pre>	

2.5.0122 © Integrata Cegos GmbH

ORACLE und SQL

19

3.4 Die WHERE-Klausel

Mit der **optionalen** WHERE-Klausel können die zurückgegebenen Zeilen nach gewünschten Kriterien eingegrenzt werden.

Sie folgt im SELECT-Statement unmittelbar auf die FROM-Klausel. Wenn ein SELECT-Statement keine WHERE-Klausel enthält, werden alle Datensätze einer Tabelle selektiert.

Die WHERE-Klausel definiert anhand von **Vergleichsoperatoren Suchkriterien** innerhalb einer Tabelle. In der WHERE-Klausel können auch manche Verknüpfungen (JOINS) zwischen den Tabellen abgebildet werden.

Es können **mehrere** logische **Suchkriterien** (Prädikate) angegeben werden, welche mit den logischen Operatoren AND, OR oder NOT verknüpft werden. Jedes der einzelnen Prädikate kann als Antwort ein WAHR, ein FALSCH oder UNBEKANNT (bei NULL) zurückgeben.

Nur solche Zeilen, in denen die WHERE-Klausel WAHR ist, werden in der Ergebnismenge **angezeigt**.

Beispiel:

```
SQL> SELECT
      e.empno,
      e.ename,
      e.sal
    FROM   emp e
  WHERE
    e.sal > 2000;
```

EMPNO	ENAME	SAL
7566	JONES	2975
7698	BLAKE	2850
7782	CLARK	2450
7788	SCOTT	3000
7839	KING	5000
7902	FORD	3000

Syntax:

```
SELECT      Spalte
FROM        Tabelle
WHERE       bedingungen;
```

WHERE Klausel



- Reihenfolge der Klauseln:

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

- Definition von Bedingungen
- Verknüpfungen zwischen den Quellen (JOIN)
- Verknüpfungen der Kriterien nach der booleschen Algebra:
 - AND
 - OR
 - NOT

Beispiele:

```
SQL> SELECT e.*  
      FROM emp e  
     WHERE e.deptno = 30;  
  
SQL> SELECT e.*  
      FROM emp e  
     WHERE e.sal > 2000  
          OR e.deptno = 10;  
  
SQL> SELECT e.*  
      FROM emp e  
     WHERE e.deptno <= 20;  
  
SQL> SELECT e.*  
      FROM emp e  
     WHERE e.job != 'CLERK'  
          AND e.deptno = 10;
```

Syntax der WHERE-Klausel



- Syntax:

```
SELECT Spalte
FROM Tabelle
WHERE bedingungen;
```

```
bedingungen ::=
  [NOT] praedikat
  { [ OR [NOT] praedikat] | [ AND [NOT] praedikat] }
```

3.5 Vergleichsprädikate

Die Vergleichsprädikate unterteilen sich in sieben Arten:

1. Direkte Vergleichsprädikate `deptno = 10`
2. Intervallprädikate `sal BETWEEN 10 AND 3000`
3. Ähnlichkeitsprädikate `ename LIKE 'SMITH'`
4. NULL-Test Prädikate `comm IS NULL`
5. IN Prädikate `deptno IN (10, 20)`
6. ALL-ANY-SOME Prädikate `e.deptno < ALL
(SELECT deptno
FROM dept)`
7. EXISTS Prädikate `EXISTS (SELECT deptno
FROM dept WHERE ...)`

Syntax:

bedingungen ::=

```
[NOT] praedikat
{ [OR [NOT] praedikat] | [AND [NOT] praedikat] }
```

praedikat ::=

```
direktvergleich-pr
| interval-pr
| aehnlichkeit-pr
| nulltest-pr
| in-pr
| all-any-some-pr
| exists-pr
```

Prädikate	 integrata cegos
<ul style="list-style-type: none">▪ sieben Arten<ul style="list-style-type: none">▪ Direkte Vergleichsprädikate▪ Intervallprädikate▪ Ähnlichkeitsprädikate▪ NULL-Test Prädikate▪ IN Prädikate▪ ALL-ANY-SOME Prädikate▪ EXISTS Prädikate	<ul style="list-style-type: none">▪ <u>Syntax:</u><pre>praedikat ::= direktvergleich-pr interval-pr aehnlichkeit-pr nulltest-pr in-pr all-any-some-pr exists-pr</pre>
2.5.0122 © Integrata Cegos GmbH	ORACLE und SQL

3.5.1 Direkte Vergleichsprädikate

Mit den direkten Vergleichsoperatoren werden zwei Werte direkt miteinander verglichen. Die Datentypen beider Werte sollten identisch sein.

Operator	Bedeutung
=	gleich
!=	ungleich
<>	ungleich
>	größer als
>=	größer gleich
<	kleiner als
<=	kleiner gleich

Beispiel:

```
SQL> SELECT e.*  
      FROM emp e  
     WHERE e.deptno = 10;  
  
SQL> SELECT e.*  
      FROM emp e  
     WHERE e.sal >= 1000;
```

Syntax:

```
SELECT      Spalte  
FROM        Tabelle  
WHERE       Ausdruck direktvergleichsoperator Ausdruck;
```

```
direktvergleichsoperator ::=  
= | <> | != | < | > | <= | >=
```

Direkte Vergleichsprädikate	 integrata cegos
<ul style="list-style-type: none"> ▪ Direkter Vergleich zwischen zwei Werten ▪ <u>Syntax:</u> SELECT <i>Spalte</i> FROM <i>Tabelle</i> WHERE <i>Ausdruck direktvergleichsoperator Ausdruck;</i> <i>direktvergleichsoperator ::=</i> <i>= <> != < > <= >=</i> ▪ <u>Beispiel:</u> <pre>SQL> SELECT e.* FROM emp e WHERE e.deptno = 10; SQL> SELECT e.* FROM emp e WHERE e.sal >= 1000;</pre> 	
2.5.0122 © Integrata Cegos GmbH ORACLE und SQL 23	

3.5.2 Intervallprädikat

Mit diesem Prädikat (Operator BETWEEN) wird ein Wert auf die Zugehörigkeit zu einem Intervall geprüft.

Beispiel:

```
SQL> SELECT e.*  
      FROM emp e  
     WHERE e.sal BETWEEN 1000 AND 3500;  
  
SQL> SELECT e.*  
      FROM emp e  
     WHERE e.ename BETWEEN 'A' AND 'L';  
  
SQL> SELECT e.*  
      FROM emp e  
     WHERE e.hiredate  
   BETWEEN to_date('01.01.1980', 'DD.MM.YYYY')  
   AND to_date('01.01.1982', 'DD.MM.YYYY');
```

Syntax:

```
SELECT Spalte  
FROM Tabelle  
WHERE Spalte [NOT] BETWEEN Wert AND Wert;
```

Intervallprädikate



- Zugehörigkeit des Wertes zu einem Intervall

- Syntax:

```
SELECT      Spalte
FROM        Tabelle
WHERE       Spalte [NOT] BETWEEN Wert AND Wert;
```

- Beispiel:

```
SQL> SELECT e.*
      FROM emp e
      WHERE e.sal BETWEEN 1000 AND 3500;
```

```
SQL> SELECT e.*
      FROM emp e
      WHERE e.ename BETWEEN 'A' AND 'L';
```

3.5.3 Platzhaltersymbole (wildcard character)

- % für jede beliebige Zeichenkette (0 bis beliebig viele Zeichen).
- _ für jedes beliebige Zeichen (genau ein Zeichen).

3.5.4 Ähnlichkeitsprädikat

Mit diesem Prädikat (Operator `LIKE`) wird ein Wert auf die Ähnlichkeit zu einem anderen Wert geprüft. Der `LIKE` Operator wird meist gemeinsam mit Platzaltern (Wildcards) eingesetzt.

Beispiel:

```
SQL> SELECT e.*  
      FROM emp e  
     WHERE e.ename LIKE '_A%';  
  
SQL> SELECT e.*  
      FROM emp e  
     WHERE e.job NOT LIKE 'M%';
```

Syntax:

```
SELECT Spalte  
FROM Tabelle  
WHERE Spalte [NOT] LIKE Wert [ESCAPE Literal];
```

Ähnlichkeitsprädikate



- Ähnlichkeit eines Wertes zu einem anderen
 - Platzhalter: % beliebige Zeichenkette (kein, ein oder beliebig viele Zeichen)
 - _ beliebiges Zeichen (genau ein Zeichen)
- Syntax:

```
SELECT Spalte  
FROM Tabelle  
WHERE Spalte [NOT] LIKE Wert [ESCAPE Literal];
```
- Beispiel:

```
SQL> SELECT e.*  
      FROM emp e  
     WHERE e.ename LIKE '_A%';  
  
SQL> SELECT e.*  
      FROM emp e  
     WHERE e.job NOT LIKE 'M%';
```

3.5.5 NULL-Test Prädikat

Mit diesem Prädikat wird ein Wert unabhängig vom Datentyp auf seinen Inhaltszustand abgefragt.

Es wird nur geprüft, ob der Wert `NULL` oder `NOT NULL` ist.

Operator	Bedeutung
<code>IS NULL</code>	Wahr, wenn der Wert <code>NULL</code>
<code>IS NOT NULL</code>	Wahr, wenn der Wert nicht <code>NULL</code> ist

Beispiel:

```
SQL> SELECT e.*  
      FROM emp e  
     WHERE e.mgr IS NULL;  
  
SQL> SELECT e.*  
      FROM emp e  
     WHERE e.comm IS NOT NULL;
```

Syntax:

```
SELECT Spalte  
FROM Tabelle  
WHERE Spalte IS [NOT] NULL;
```

IS NULL Prädikat



- Zustandsabfrage eines Wertes unabhängig vom Datentyp
 - Ist der Inhalt des Wertes `NULL` oder nicht?

- Syntax:

```
SELECT Spalte  
FROM Tabelle  
WHERE Spalte IS [NOT] NULL;
```

- Beispiel:

```
SQL> SELECT e.*  
      FROM emp e  
     WHERE e.mgr IS NULL;  
  
SQL> SELECT e.*  
      FROM emp e  
     WHERE e.comm IS NOT NULL;
```

3.5.6 IN Prädikat

Das ist einer der **Mengenvergleichsoperatoren** von SQL.

Es wird geprüft ob ein Wert (auch ein **Vektor**) in einer Menge (Vektormenge) vorkommt.

Die zu prüfende Menge kann statisch vorgegeben oder zu dem Zeitpunkt der Filterung dynamisch (s. Unterabfragen im Kapitel 5.2) ermittelt werden. Es sind maximal 1000 IN-Werte erlaubt.

Beispiel:

```
SQL> SELECT e.*  
      FROM emp e  
     WHERE e.deptno IN (10, 30);
```

Beispiel für einen Vektorvergleich:

```
SQL> SELECT e.*  
      FROM emp e  
     WHERE (e.comm, e.deptno)  
           IN ( SELECT e2.comm,  
                  e2.deptno  
                 FROM emp e2  
                WHERE e2.sal >= 1500  
              )  
;
```

Syntax:

```
SELECT Spalte  
FROM Tabelle  
WHERE Spalte [NOT] IN (Wert {, Wert});
```

IN Prädikat



- Prüfung ob ein Wert/Vektor in einer Menge vorkommt
 - Menge: statisch | dynamisch

- Syntax:

```
SELECT    Spalte
FROM      Tabelle
WHERE     Spalte [NOT] IN (Wert {,Wert});
```

- Beispiel (mehr zu Unterabfragen im Kapitel 5.2):

```
SQL> SELECT e.*
      FROM   emp e
      WHERE  e.deptno IN (10, 30);

SQL> SELECT e.*
      FROM   emp e
      WHERE  (e.comm, e.deptno)
             IN ( SELECT e2.comm,
                         e2.deptno
                  FROM   emp e2
                  WHERE  e2.sal >= 1500);
```

3.5.7 ALL-ANY-SOME Prädikate

gehören auch zu den **Mengenvergleichsoperatoren**. Mit diesen Operatoren wird geprüft, in welcher direkten Vergleichsbeziehung ein Wert zu ALLEN Werten einer Menge steht. Sie funktionieren nur mit skalaren Werten.

Diese Operatoren sollten gemieden werden, da es bessere Möglichkeiten gibt (mehr dazu später).

Beispiel:

```
-- alle: ALL
SQL> SELECT e.*  
      FROM emp e  
     WHERE  
       e.deptno > ALL (10, 20);  
  
-- irgendeiner: ANY bzw. SOME
SQL> SELECT e.*  
      FROM emp e  
     WHERE  
       e.sal > ANY (100, 200, 300, 850, 3000);
```

Syntax:

```
SELECT      Spalte  
FROM        Tabelle  
WHERE       Spalte direktvergleichsoperator  
           [ ALL | ANY | SOME ] (Wert {, Wert});
```

ALL-ANY-SOME Prädikat



- Direktvergleich eines Wertes mit jedem Element einer Menge

- Syntax:

```
SELECT      Spalte
           FROM      Tabelle
           WHERE      Spalte direktvergleichsoperator
                      [ ALL | ANY | SOME ] (Wert {, Wert});
```

- Beispiel:

```
--alle
SQL> SELECT e.*
      FROM emp e
      WHERE e.deptno > ALL (10, 20);
-- irgendeiner ANY bzw. SOME
SQL> SELECT e.*
      FROM emp e
      WHERE e.sal > ANY (100, 200, 300, 850, 3000);
```

3.5.8 EXISTS Prädikat

Gehören neben den beiden vorher erwähnten ebenso zu den **Men-genvergleichsoperatoren**.

Mit diesem Operator wird geprüft, ob eine Unterabfrage Ergebnisse hat.

Beispiel:

```
--Spitzenverdiener
SQL> SELECT e.*  
      FROM          emp e  
      WHERE  
            NOT EXISTS  
            (  
                  SELECT    1  
                  FROM      emp e2  
                  WHERE    e2.sal > e.sal  
            )  
;
```

Syntax:

```
SELECT      Spalte  
FROM        Tabelle  
WHERE       [NOT] EXISTS (select-spezifikation);
```

EXISTS Prädikat <ul style="list-style-type: none"> ▪ Vergleicht ob eine Menge Ergebnisse liefert ▪ <u>Syntax:</u> <pre>SELECT Spalte FROM Tabelle WHERE [NOT] EXISTS (select-spezifikation);</pre> ▪ <u>Beispiel:</u> <pre>-- Spitzenverdiener SQL> SELECT e.* FROM emp e WHERE NOT EXISTS (SELECT 1 FROM emp e2 WHERE e2.sal > <u>e.sal</u>) ;</pre> 	
<small>2.5.0122 © Integrata Cegos GmbH</small> <small>ORACLE und SQL</small> <small>29</small>	

3.6 Die ORDER BY-Klausel

Diese optionale Klausel steht am Ende eines SELECT-Statements und erlaubt die sortierte Ausgabe der gefilterten Ergebnismenge. Nur durch eine explizite Sortierangabe - also die ORDER BY-Klausel - ist sichergestellt, dass die Datensätze in der gewünschten Reihenfolge ausgegeben werden.

In der ORDER BY-Klausel können ein oder mehrere Spaltennamen angegeben werden, nach denen (je nach Datentyp der Spalte) entweder alphabetisch oder numerisch sortiert wird.

Des Weiteren kann angegeben werden, ob die Ergebnisse:

- aufsteigend (ASC - ascending, Standardeinstellung) oder
- absteigend (DESC - descending)

sortiert werden sollen.

NULL-Werte haben höchste Priorität, sie erscheinen bei ASC zuletzt, bei DESC zuerst.

Die ORDER BY-Klausel akzeptiert als Eingaben:

- Spaltennamen
- Ziffern (Reihenfolge der Spalte aus der SELECT-Klausel) und/oder
- **Aliasnamen** der Spalten aus der SELECT-Klausel.

Beispiel:

```
SQL> SELECT
      e.empno,
      e.ename AS "Nachname",
      e.sal,
      e.deptno
    FROM emp e
   ORDER BY e.deptno ASC,
            3 DESC,
            "Nachname";
```

EMPNO	Nachname	SAL	DEPTNO
7839	KING	5000	10
7782	CLARK	2450	10
7934	MILLER	1300	10
7902	FORD	3000	20
7788	SCOTT	3000	20
7566	JONES	2975	20
7876	ADAMS	1100	20
7369	SMITH	800	20
7698	BLAKE	2850	30
7499	ALLEN	1600	30
7844	TURNER	1500	30
7654	MARTIN	1250	30
7521	WARD	1250	30
7900	JAMES	950	30

Syntax:

```
SELECT      Spalte
FROM        Tabelle
ORDER BY {orderbyausdruck [ ASC | DESC ]
          [, orderbyausdruck [ ASC | DESC ] ] }
```

orderbyausdruck ::=
 Spalte | Ziffer | Spaltenalias ;

ORDER BY-Klausel



- Am Ende des SELECT-Statements
- Sortieren nur durch ORDER BY gesichert

- Sortierangabe:
 - ASC (ascending – aufsteigend) Standardwert
 - DESC (descending – absteigend)

- NULL-Werte haben höchste Priorität:
 - bei ASC zuletzt
 - bei DESC zuerst

- Angabe:
 - Spaltennamen
 - Spaltenalias
 - Ziffern (Spaltenreihenfolge in SELECT-Klausel)

ORDER BY-Klausel Bsp.

▪ Syntax:

```
SELECT Spalte
  FROM Tabelle
 ORDER BY {ausdruck [ASC | DESC][, ausdruck [ ASC | DESC ]]}
```

ausdruck ::= Spalte | Ziffer | Spaltenalias

```
SQL> SELECT e.empno, e.ename, e.sal,
      FROM emp e
      WHERE e.comm IS NOT NULL
      ORDER BY e.deptno DESC, e.ename;
```

```
SQL> SELECT e.empno, e.ename
      FROM emp e
      ORDER BY e.sal;
```

```
SQL> SELECT e.empno,
      e.ename AS "Name",
      e.sal,
      e.deptno
      FROM emp e
      ORDER BY e.deptno ASC,
              3           DESC,
              "Nachname";
```

EMPNO	Name	SAL	DEPTNO
7839	KING	5000	10
7782	CLARK	2450	10
7934	MILLER	1300	10
7902	FORD	3000	20
7788	SCOTT	3000	20

3.7 Zeilen limitieren

Seit 12c ist es möglich, die Ergebnismenge durch Limitieren einzuschränken.

Syntax für die Limitierung (nach der ORDER BY Klausel)

```
OFFSET <offset> { ROW | ROWS } ]
[ FETCH { FIRST | NEXT } [ { <rowcount> | <percent> PERCENT } ]
{ ROW | ROWS } { ONLY | WITH TIES}
```

WITH TIES ist eine Zusatzfunktion und wird verwendet, um zusätzliche Zeilen in einer ORDER BY Klausel anzuzeigen. Dabei betrachtet Oracle die zuletzt ausgegebene Zeile aus dem ORDER BY und gibt alle Zeilen mit demselben Ergebnis zusätzlich aus.

Zeilen limitieren														
<ul style="list-style-type: none"> ▪ 5 Top Verdienster ausgeben <pre>▪ SELECT ename, sal FROM scott.emp ORDER BY sal desc FETCH FIRST 5 ROWS ONLY;</pre> 														
<table border="1"> <thead> <tr> <th>ENAME</th> <th>SAL</th> </tr> </thead> <tbody> <tr><td>KING</td><td>5000</td></tr> <tr><td>SCOTT</td><td>3000</td></tr> <tr><td>FORD</td><td>3000</td></tr> <tr><td>JONES</td><td>2975</td></tr> <tr><td>BLAKE</td><td>2850</td></tr> </tbody> </table> 			ENAME	SAL	KING	5000	SCOTT	3000	FORD	3000	JONES	2975	BLAKE	2850
ENAME	SAL													
KING	5000													
SCOTT	3000													
FORD	3000													
JONES	2975													
BLAKE	2850													
<ul style="list-style-type: none"> ▪ Die nächsten 5 TOP Verdienster: <pre>▪ SELECT ename, sal FROM scott.emp ORDER BY sal desc OFFSET 5 ROWS FETCH NEXT 5 ROWS ONLY;</pre> 														
<table border="1"> <thead> <tr> <th>ENAME</th> <th>SAL</th> </tr> </thead> <tbody> <tr><td>CLARK</td><td>2450</td></tr> <tr><td>ALLEN</td><td>1600</td></tr> <tr><td>TURNER</td><td>1500</td></tr> <tr><td>MILLER</td><td>1300</td></tr> <tr><td>WARD</td><td>1250</td></tr> </tbody> </table> 			ENAME	SAL	CLARK	2450	ALLEN	1600	TURNER	1500	MILLER	1300	WARD	1250
ENAME	SAL													
CLARK	2450													
ALLEN	1600													
TURNER	1500													
MILLER	1300													
WARD	1250													
<ul style="list-style-type: none"> ▪ Es können auch prozentuale Angaben gemacht werden: <pre>▪ SELECT ename, sal FROM scott.emp ORDER BY sal desc FETCH FIRST 5 PERCENT ROWS ONLY;</pre> 														
		<table border="1"> <thead> <tr> <th>ENAME</th> <th>SAL</th> </tr> </thead> <tbody> <tr><td>KING</td><td>5000</td></tr> </tbody> </table>	ENAME	SAL	KING	5000								
ENAME	SAL													
KING	5000													
2.5.0122 © Integrata Cegos GmbH		ORACLE und SQL												
		32												

Zeilen limitieren (f)



- Syntax für die Limitierung (nach der ORDER BY Klausel)

```
OFFSET <offset> { ROW | ROWS } ]
[ FETCH { FIRST | NEXT } [ { <rowcount> | <percent> PERCENT } ]
{ ROW | ROWS } { ONLY | WITH TIES}
```

WITH TIES: Wenn in der letzten Ausgabezeile Werte stehen, die in den folgenden Zeilen den gleichen Wert haben, werden auch diese ausgegeben

- Die 2 Topverdiener ausgeben. Wenn noch ein Mitarbeiter das selbe Gehalt verdient wie der Zweitplatzierte, dann diesen auch mit ausgeben.

```
■ SELECT ename, sal
  FROM scott.emp
  ORDER BY sal desc
  FETCH FIRST 2 rows WITH TIES;
```

ENAME	SAL
1 KING	5000
2 SCOTT	3000
3 FORD	3000

3.8 Datentypen

Die wichtigsten Datentypen unter Oracle sind:

- NUMBER,
- DATE und
- VARCHAR2 .

Sie sind durch entsprechende Konvertierungsfunktionen ineinander überführbar.

3.8.1 Datentypen

NUMBER (n , m)	definiert eine numerische Spalte. Werte bis zu einer Genauigkeit von 38 Stellen können gespeichert werden. Die Zahl <i>n</i> gibt an, wie viele Stellen insgesamt (Vor- und Nachkommastellen) gespeichert werden. Wenn <i>m</i> (optional) eine positive Zahl ist, gibt sie die Anzahl der Nachkommastellen an. Ist <i>m</i> eine negative Zahl, so gibt sie den Rundungsfaktor für <i>n</i> an.
BINARY_FLOAT	32 Bit Gleitkommazahlen untere Grenze 1e-45 obere Grenze 3.4e+38
BINARY_DOUBLE	64 Bit Gleitkommazahlen untere Grenze 1e-130 obere Grenze 9.9e+125
DATE	nimmt Datums- und Zeitwerte auf. Für jedes Datum werden folgende Daten gespeichert: Jahrhundert, Jahr, Monat, Tag, Stunde, Minute, Sekunde. Standard Oracle Datumsformat ist: DD-MON-YY (13-NOV-01) .
CHAR (n)	wird für alphanumerische Spalten verwendet, deren feste Länge <i>n</i> Bytes betragen soll. Die maximale Größe von <i>n</i> beträgt 2000 . Der Standardwert und gleichzeitig die minimale Größe ist 1 Byte.
VARCHAR2 (n)	wird für Spalten verwendet, die alphanumerische Werte variabler Länge enthalten können. Die maximale Länge muss festgelegt werden und kann bis zu 4000 Bytes betragen.

	Jede Kombination von Buchstaben, Ziffern, Sonderzeichen oder Leerstellen kann gespeichert werden.
NCHAR und NVARCHAR2 (n)	zur Darstellung von Multibyte-Zeichensätzen wie z. B. Chinesisch und Japanisch.
LONG	wird für Spalten verwendet, die alphanumerische Werte variabler Länge enthalten können; die maximale Länge muss festgelegt werden. Die maximale Größe beträgt 2 GB . Es kann nur eine LONG -Spalte innerhalb einer Tabelle definiert werden. Anmerkung: Der LONG -Datentyp wird nur aus Abwärtskompatibilität gehalten. In neuen Applikationen sollten LOB -Datentypen benutzt werden.
LOB, BLOB, CLOB, NCLOB und BFILE	Ermöglichen die Speicherung großer Blöcke unstrukturierter Daten (Text, Bilder, Videos, Sound). Größe bis zu 4 GB .
RAW (n)	Binärdaten mit einer Größe von <i>n</i> Bytes. Die maximale Größe beträgt 2000 Bytes . Werte müssen als Character Strings und hexadezimal eingegeben werden.
LONG RAW	ein LONG RAW hat die gleiche Bedeutung wie RAW . Die maximale Länge beträgt allerdings 2 GB .
ROWID	wird für Hexadezimale Zeichenketten verwendet, die Zeilenadressen einer Tabelle aufnehmen sollen. Mit Oracle 10g wurde die ROWID so erweitert, dass extrem Große Big File Tablespaces (BFT) von bis zu 128 TB erlaubt werden.
INTERVAL YEAR (n) TO MONTH	Intervalldatentyp für " grobe " Zeitintervalle. Die Zahl <i>n</i> gibt an, wie viele Ziffern (max. 9) in das Jahresintervall eingeben werden können.
INTERVAL DAY (n) TO SECOND (m)	Intervalldatentyp für " genaue " Zeitintervalle. Die Zahl <i>n</i> gibt an, wie viele Ziffern (max. 6) für die Anzahl der Tage verwendet werden. Die Zahl <i>m</i> gibt an, wie viele Ziffern (max. 9) für Sekundenbruchteile eingeben werden können.
TIMESTAMP [(n)]	Datentyp für Zeitpunkte mit großer Genauigkeit, bis zu Sekundenbruchteilen. <i>n</i> liegt im Intervall [0 bis 9]; der Standardwert ist 6 (= Millionstel Sekunden)

TIMESTAMP [(n)] WITH TIME ZONE	beinhaltet auch die Abweichung von UTC (Coordinated Universal Time); zwei Werte werden als gleich erachtet, wenn sie unter Berücksichtigung ihrer Zeitzone den gleichen globalen Zeitpunkt beschreiben
TIMESTAMP [(n)] WITH LOCAL TIME ZONE	beinhaltet ebenfalls die Abweichung von UTC, doch wird diese nicht explizit in der DB gespeichert; stattdessen wird der Wert auf die Zeitzone der DB umgerechnet und als solche eingetragen; SELECT liefert lokale Session-Zeit

Datentypen	Beispiel	Inhalt
NUMBER	NUMBER(9, 2)	4332.22
DATE	DATE	"13.12.1999"
CHAR	CHAR(20)	"dies ist Text"
VARCHAR2	VARCHAR2(20)	"dies ist Text"
NCHAR	NCHAR(20)	Nationale Zeichen
NVARCHAR2	NVARCHAR2(20)	Nationale Zeichen
BLOB	BLOB	Binäre Daten
CLOB	CLOB	Character Daten
NCLOB	NCLOB	Nationale Character Daten
BFILE	BFILE	Pfadname einer Datei
LONG	LONG	"dies ist Text"
RAW	RAW(200)	binäre Daten
LONG RAW	LONG RAW	binäre Daten
ROWID	ROWID	0000001F.00000D.000019
INTERVAL YEAR (n) TO MONTH	INTERVAL YEAR (3) TO MONTH	989-3
INTERVAL DAY (n) TO SECOND (m)	INTERVAL DAY (3) TO SECOND (9)	0 0:0:5.789789789
TIMESTAMP[(n)]	TIMESTAMP(3)	25.11.18 14:27:58,000
TIMESTAMP[(n)] WITH TIME ZONE	TIMESTAMP(3) WITH TIME ZONE	25.11.18 14:27:58,000 +2:00
TIMESTAMP[(n)] WITH LOCAL TIME ZONE	TIMESTAMP(3) WITH LOCAL TIME ZONE	25.11.18 12:27:58,000
<u>BINARY FLOAT</u>	<u>BINARY FLOAT</u>	<u>2.4e-12</u>
<u>BINARY DOUBLE</u>	<u>BINARY DOUBLE</u>	<u>3.4e+21</u>
<u>SIMPLE INTEGER</u>	<u>SIMPLE INTEGER</u>	<u>-2,147,483,648</u>

Neuere Datentypen

Ab der Oracle 11g Version wurde SIMPLE_INTEGER als ein Subtyp von pls_integer eingeführt.

Er bringt mächtige Vorteile bei nativer Compilierung in PL/SQL und ist bei SQL von keinem größeren Interesse.

Hinweis: Vorsicht bei Überlauf und Unterlauf. Es wird einfach vom größten Wert auf den kleinsten gesprungen und umgekehrt!

SIMPLE_INTEGER Subtyp von pls_integer

untere Grenze -2,147,483,648

obere Grenze 2,147,483,647

4

Funktionen

4.1	Arithmetische Funktionen	4-3
4.2	Zeichenkettenfunktionen	4-6
4.2.1	Groß- / Kleinschreibung:	4-6
4.2.2	Zeichen vom Rand “abschneiden“	4-7
4.2.3	Weitere Zeichenkettenfunktionen	4-8
4.2.4	Konvertierungsfunktionen	4-10
4.2.5	Datumsfunktionen.....	4-13
4.2.6	Die DECODE-Funktion.....	4-15
4.2.7	Reguläre Ausdrücke	4-17
4.2.8	Zusammenfassung Single Row Funktionen.....	4-27
4.3	Gruppenfunktionen	4-29
4.4	Die GROUP BY-Klausel	4-31
4.4.1	GROUP BY mit ROLLUP- und CUBE-Operatoren	4-35
4.5	Die HAVING-Klausel	4-39

4 Funktionen

Zu den Spaltenoperatoren wurden bereits neben arithmetischen Funktionen auch die NVL- und COALESCE-Funktionen erwähnt. ORACLE bietet eine Fülle an nützlichen Funktionen, welche nicht nur in SELECT-Klauseln eingesetzt werden können.

Als Parametereingabe erwarten Funktionen einen oder mehrere Werte eines bestimmten Datentyps. Als Rückgabe liefern sie auch einen Wert in einem bestimmten Datentyp. Beim Einsatz von Funktionen muss beachtet werden, dass **Anzahl** und **Datentypen der Parameter** einer Funktion stimmen! Dies ist eine häufige Fehlerquelle.

Einige der für die tägliche Arbeit sehr nützlichen Funktionen werden hier vorgestellt.

Allg. Syntax:

```
funktion ::=  
    funktionsname [([parameter [, parameter}])]
```

Hinweis: Die Tabelle dual ist eine Data Dictionary Tabelle mit nur einer Spalte DUMMY CHAR(1) und einer Zeile mit dem Wert "x". Diese Tabelle dient u. a. dazu, Konstantenberechnungen durchzuführen.

4.1 Arithmetische Funktionen

ABS (n) gibt den Absolutbetrag von **n** zurück.

Beispiel: SELECT ABS(-15) AS "Betrag"
FROM dual;

Ergebnis: 15

CEIL (n) gibt die kleinste ganze Zahl zurück, die $\geq n$ ist.

Beispiel: SELECT CEIL(15.7) AS "Ceiling"
FROM dual;

Ergebnis: 16

FLOOR (n) gibt die größte ganze Zahl zurück, die $\leq n$ ist.

Beispiel: SELECT FLOOR(15.7) AS "Floor"
FROM dual;

Ergebnis: 15

MOD (m , n)	gibt den Rest der Division von m/n zurück.
Beispiel:	SELECT MOD(7, 5) AS "Modulo" FROM dual;
Ergebnis:	2
POWER (m , n)	gibt m^n zurück (Potenzierung).
Beispiel:	SELECT POWER(3, 2) AS "Potenz" FROM dual;
Ergebnis:	9
ROUND (n [, m])	Rundet die Zahl n auf m Stellen. Wird m nicht angegeben, ist dies identisch mit $m = 0$. m kann auch negativ sein, um bereits vor dem Komma zu runden.
Beispiel:	SELECT ROUND(15.193, 1) "Rundung" FROM dual;
Ergebnis:	15.2
Beispiel:	SELECT ROUND(163, -1) AS "Rundung" FROM dual;
Ergebnis:	160
SIGN (n)	wenn $n < 0$ Rückgabewert = -1; wenn $n = 0$ Rückgabewert = 0; wenn $n > 0$ Rückgabewert 1..
Beispiel:	SELECT SIGN(-15) AS "SIGN" FROM dual;
Ergebnis:	-1
SQRT (n)	Quadratwurzel von n .
Beispiel:	SELECT SQRT(25) AS "Wurzel" FROM dual;
Ergebnis:	5
TRUNC (n [, m])	Abschneiden der Dezimalstellen der Zahl n hinter der m -ten Nachkommastelle. Ist m nicht gesetzt, gilt der Wert 0. m kann auch negativ sein (analog zu ROUND).
Beispiel:	SELECT TRUNC(15.791, 1) "Trunc" FROM dual;
Ergebnis:	15.7

Arithmetische Funktionen		
Funktion	Beispiel	Wert
ABS (n)	ABS (-15)	15
CEIL (n)	CEIL (15.7)	16
FLOOR (n)	FLOOR (15.7)	15
MOD (m, n)	MOD (7, 5)	2
POWER (m, n)	POWER (3, 2)	9
ROUND (n[, m])	ROUND (15.193, 1)	15.2
SIGN (n)	SIGN (-15)	-1
SQRT (n)	SQRT (25)	5
TRUNC (n[, m])	TRUNC (15.791, 1)	15.7

2.5.0122 © Integrata Cegos GmbH ORACLE und SQL 2

4.2 Zeichenkettenfunktionen

4.2.1 Groß- / Kleinschreibung:

INITCAP (string)

gibt *string* in Kleinbuchstaben mit dem ersten Buchstaben groß aus.

Beispiel: SELECT INITCAP('sPeChT')
FROM dual;

Ergebnis: Specht

LOWER (string)

gibt *string* in Kleinbuchstaben aus.

Beispiel: SELECT LOWER('sPeChT')
FROM dual;

Ergebnis: specht

UPPER (string)

gibt *string* in Großbuchstaben aus.

Beispiel: SELECT UPPER('sPeChT')
FROM dual;

Ergebnis: SPECHT

4.2.2 Zeichen vom Rand “abschneiden”

LTRIM(*string* [, *c*])

löscht das Zeichen *c* im String *string* von links.

Ist kein Zeichen angegeben dann ist

c = ' ' (Leerzeichen).

Beispiel: SELECT LTRIM('VVVBejic', 'V')
FROM dual;

Ergebnis: Bejic

RTRIM(*string* [, *c*])

löscht das Zeichen *c* im String *string* von rechts.

Ist kein Zeichen angegeben dann ist

c = ' ' (Leerzeichen).

Beispiel: SELECT RTRIM(' BejicVV', 'V')
FROM dual;

Ergebnis: Bejic

TRIM([LEADING | TRAILING | BOTH] [*c* FROM] *string*)

löscht das Zeichen *c* im String *string* von

links (LEADING) oder

rechts (TRAILING) oder

von beiden Seiten (BOTH, Standardwert).

Ist kein Zeichen angegeben, dann ist

c = ' ' (Leerzeichen).

In diesem Fall wird auch “FROM“ weggelassen.

Beispiel: SELECT TRIM(LEADING 'V' FROM
'V BejicV') AS "Name"
FROM dual;

Ergebnis: BejicV

Beispiel: SELECT
TRIM('V' FROM 'VBejicV')
AS "Name"
FROM dual;

Ergebnis: Bejic

4.2.3 Weitere Zeichenkettenfunktionen

SUBSTR (string, m [, n])

gibt einen Teilstring aus einer Zeichenkette aus, beginnend ab Position **m**, insgesamt **n** Zeichen lang.

Ohne Angabe von **n** wird bis zum Ende des Strings ausgegeben.

Bei negativem **m** beginnt die Zählung am Ende des Strings.

Beispiel: SELECT SUBSTR(ename, 2, 4)
FROM emp
WHERE ename = 'TURNER';

Ergebnis: URNE

REPLACE (string1, string2 [, string3])

ersetzt in **string1** alle Vorkommen von **string2** durch **string3**.

Falls **string3** nicht spezifiziert wird, wird **string2** ersatzlos gelöscht.

Beispiel: SELECT REPLACE('Maier',
'ai', 'ei')
FROM dual;

Ergebnis: Meier

CONCAT (string1, string2)

verknüpft **string1** mit **string2**,
Entspricht dem Verknüpfungsoperator ||

Beispiel: SELECT CONCAT('Mato ',
'Bejic') AS "Name"
FROM dual;

Ergebnis: Mato Bejic

LENGTH (string)

gibt die Länge von **string** aus.

Beispiel: SELECT LENGTH ('SPECHT')
AS "Laenge"
FROM dual;

Ergebnis: 6

INSTR (string, n, m, i) gibt die Position des *i*-ten Vorkommens nach der Stelle **m** von Zeichenkette **n** in **string** aus.

Beispiel: SELECT INSTR('SPECHT', 'E')
FROM dual;

Ergebnis: 3

LPAD (string , m , n) erzeugt rechtsbündigen Blocksatz, indem die Spalte von links mit der Zeichenkette **n** aufgefüllt wird, bis sie **m** Zeichen breit ist.

Beispiel: SELECT LPAD(sal, 7, '*')
AS "Gehalt"
FROM emp
WHERE ename = 'SMITH';

Ergebnis: ****800

RPAD (string , m , n) analog, erzeugt linksbündigen Blocksatz.

Beispiel: SELECT RPAD(sal, 7, '*')
AS "Gehalt"
FROM emp
WHERE ename = 'SMITH';

Ergebnis: 800****

Zeichenkettenfunktionen



Funktion	Beispiel	Wert
INITCAP(string)	INITCAP ('sPeChT')	Specht
LOWER(string)	LOWER ('sPeChT')	specht
UPPER(string)	UPPER ('sPeChT')	SPECHT
LTRIM(string [,c])	LTRIM('VVVBejic', 'V')	Bejic
RTRIM(string [,c])	RTRIM('BejicVV', 'V')	Bejic
TRIM([LEADING TRAILING BOTH] [c FROM] string)	TRIM(' Bejic ')	Bejic
SUBSTR(string, m [, n])	SUBSTR('TURNER', 2, 4)	URNE
REPLACE(string1, string2[, string3])	REPLACE('Maier', 'ai', 'ei')	Meier
CONCAT(string1, string2)	CONCAT('Marco ', 'Patzwahl')	Mato Bejic
LENGTH(string)	LENGTH ('SPECHT')	6
INSTR(string, n, m, i)	INSTR('SPECHT', 'E')	3
LPAD(string, m, n)	LPAD(800, 7, '*')	****800
RPAD(string, m, n)	RPAD(800, 7, '*')	800****

4.2.4 Konvertierungsfunktionen

TO_CHAR konvertiert ein Datum oder eine Zahl zu einem String. Wird benötigt, wenn die Ausgabe nicht dem Standardformat entsprechen soll, sondern einem selbst definierten Format.

Beispiel:

```
SELECT  
  TO_CHAR (SYSDATE, 'DD.MM.YYYY HH24:MI')  
FROM dual;
```

Ergebnis: 20.12.2004 10:35

TO_DATE konvertiert eine Zeichenkette anhand der angegebenen Formatmaske zu einem Datum, falls dies möglich ist.

Beispiel:

```
UPDATE emp  
SET hiredate =  
  (SELECT  
    TO_DATE('26:06:2001', 'DD:MM:YYYY')  
   FROM dual)  
 WHERE ename LIKE 'KING';
```

TO_NUMBER konvertiert eine Zeichenkette zu einer Zahl.

Beispiel:

```
UPDATE emp  
SET sal = TO_NUMBER('100.00', '999.99')  
 WHERE ename = 'SMITH';
```

Um die Daten benutzen (insbesondere anzeigen) zu können, müssen diese konvertiert werden. Vereinfacht gesagt, kann ein menschliches Individuum nur "Texte" lesen. Alle anderen Datentypen seien es Zahlen oder Datumswerte müssen erst ins "menschliche" Format übersetzt werden. Dies geschieht mit der Funktion `TO_CHAR`.

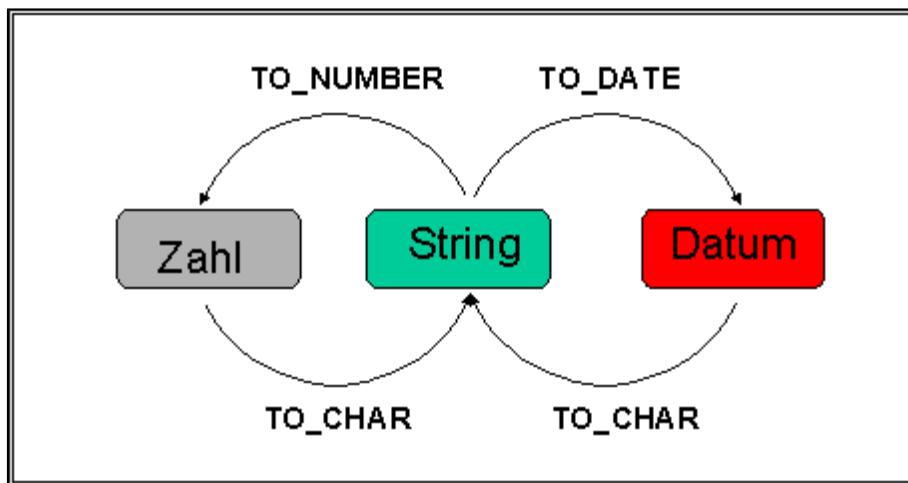


Abb.: Zusammenspiel der Konvertierungsfunktionen

Besonders gewöhnungsbedürftig ist die Benutzung der Datumstypen. Aus den Datumswerten können viele Informationen berechnet werden (Jahr, Monat, KW, Quartal, ...). Es müssen nur entsprechende Parameter der `TO_CHAR` Funktion gegeben werden.

Die Parameterliste ist sehr lang. Hier ein kleiner Auszug:

Parameter	Bedeutung
D	Tag der Woche (1-7)
DD	Tag des Monats (1-31)
DDD	Tag der Jahres (1-366)
HH	Uhrzeit (1-12)
HH24	Uhrzeit (0-23)
IW	KW(1-53)
MM	Monat(1-12)
MON	Monat abgekürzt ausgeschrieben (JAN - DEZ)
Month	Monat ausgeschrieben (Januar - Dezember)
Q	Quartal (1-4)
YYYY	Jahr (4-stellig)

Konvertierungsfunktionen

**integrata
cegos**

- `TO_CHAR TO_CHAR(SYSDATE, 'DD.MM.YYYY HH24:MI:SS')`
- `TO_NUMBER TO_NUMBER('124.73', '999.99')`
- `TO_DATE TO_DATE('26.!.06#1993', 'DD.!.MM#YYYY')`

▪ Beispiel:

- `TO_CHAR(TO_DATE('26.!.06#1993', 'DD.!.MM#YYYY'), 'DD.MM.YYYY HH:MI')`
- **Ergebnis:** 26.06.1993 12:00

2.5.0122 © Integrata Cegos GmbH ORACLE und SQL 4

Parameter der Funktion TO_CHAR

**integrata
cegos**

Parameter	Bedeutung
D	Tag der Woche (1-7)
DD	Tag des Monats (1-31)
DDD	Tag des Jahres (1-366)
HH	Uhrzeit (1-12)
HH24	Uhrzeit (0-23)
IW	KW(1-53)
MM	Monat(1-12)
MON	Monat abgekürzt ausgeschrieben (JAN - DEZ)
Month	Monat ausgeschrieben (Januar - Dezember)
Q	Quartal (1-4)
YYYY	Jahr (4-stellig)

2.5.0122 © Integrata Cegos GmbH ORACLE und SQL 5

4.2.5 Datumsfunktionen

SYSDATE	gibt das aktuelle Datum aus. Beispiel: SELECT SYSDATE FROM dual; Ergebnis: heutiger Tag
MONTHS_BETWEEN (datum , datum)	ermittelt die Anzahl an Monaten zwischen zwei Datumswerten. Die Zahl hinter dem Komma gibt den Teil des Monats an. Beispiel: SELECT MONTHS_BETWEEN (SYSDATE, TO_DATE ('01.01.2014', 'DD.MM.YYYY')) FROM DUAL; Ergebnis z. B.: 71,4385683
ROUND (datum [, fmt])	rundet das angegebene Datum nach einem bestimmten Formatmodell <i>fmt</i> Beispiel: SELECT ROUND(TO_DATE ('27.10.98'), 'YEAR') AS "First of Year" FROM dual; Ergebnis: 01.01.99
TRUNC	arbeitet ähnlich, schneidet das Datum immer ab.
ADD_MONTHS (datum , n)	addiert die im Parameter <i>n</i> angegebene Anzahl der Monate an das angegebene <i>datum</i> .
LAST_DAY (datum)	Ermittelt den letzten Tag eines Monats.
NEXT_DAY (datum , char)	Ermittelt an welches Datum nach <i>datum</i> nächster in <i>char</i> angegebener Tag fällt. Beispiel: Auf welches Datum fällt der nächste Montag nach Sylvester 2004. SELECT NEXT_DAY('31.12.2004', 'MONDAY') AS "ErsterArbeitstag" FROM dual; Ergebnis: 03.01.2005

EXTRACT (char FROM datum)

Ermittelt den gewünschten Datumsteil aus einem Datum

Beispiel:

```
SELECT
  EXTRACT (
    YEAR FROM DATE '2004-03-07') AS
  "Jahr"
FROM dual;
```

Ergebnis: 2004

CURRENT_DATE

gibt das aktuelle Datum der Sitzung als Daten-
typ DATE zurück (entspricht SYSDATE).

CURRENT_TIMESTAMP

gibt das aktuelle Datum und Uhrzeit als Daten-
typ TIMESTAMP WITH TIME ZONE zurück.

LOCALTIMESTAMP

gibt das aktuelle Datum und Uhrzeit als Daten-
typ TIMESTAMP zurück.

DBTIMEZONE

gibt die Zeitzone der Datenbank zurück.

SESSIONTIMEZONE

gibt Zeitzone der Session zurück.

Datumsfunktionen		
Funktion	Beispiel	Wert
SYSDATE	SYSDATE	heutiger Tag
MONTHS_BETWEEN (datum, datum)	MONTHS_BETWEEN (SYSDATE, hiredate) von 'KING'	277,179807
ROUND (d[,fmt])	ROUND (TO_DATE('27.10.98'), 'YEAR')	01.01.99
TRUNC	TRUNC(TO_DATE('27.10.98'), 'YEAR')	01.01.98
ADD_MONTHS (datum, n)	ADD_MONTHS (SYSDATE, 4)	Datum in 4 Monaten
LAST_DAY (datum)	LAST_DAY ('25.11.18')	30.11.18
NEXT_DAY (datum, char)	NEXT_DAY ('01.01.2018', 'MONDAY')	05.01.18
CURRENT_DATE	CURRENT_DATE	heutiger Tag
EXTRACT(datatype)	EXTRACT(YEAR FROM DATE '2018-03-07')	2018
CURRENT_TIMESTAMP	CURRENT_TIMESTAMP	22.12.18 13:44:18,171000 +01:00
LOCALTIMESTAMP	LOCALTIMESTAMP	22.12.18 13:44:18,171000
DBTIMEZONE	DBTIMEZONE	+00:00
SESSIONTIMEZONE	SESSIONTIMEZONE	+01:00

2.5.0122 © Integrata Cegos GmbH ORACLE und SQL 6

4.2.6 Die DECODE-Funktion

Sie entspricht der IF-Then-Else-Logik diverser Programmiersprachen. In einer einzigen Abfrage kann nach mehreren Werten gesucht und, abhängig davon, ein unterschiedliches Ergebnis geliefert werden.

Beispiel:

```
SQL> SELECT
      e.empno,
      e.job,
      e.sal,
      DECODE(job, 'ANALYST', e.sal * 1.1,
             'CLERK', e.sal * 1.15,
             'MANAGER', e.sal * 1.2,
             e.sal) AS "Neues Gehalt"
   FROM emp e
  ORDER BY "Neues Gehalt";
```

EMPNO	JOB	SAL	Neues Gehalt
7369	CLERK	800	920
7900	CLERK	950	1092,5
7521	SALESMAN	1250	1250
7654	SALESMAN	1250	1250
7876	CLERK	1100	1265
7934	CLERK	1300	1495
7844	SALESMAN	1500	1500
7499	SALESMAN	1600	1600
7782	MANAGER	2450	2940
7788	ANALYST	3000	3300
7902	ANALYST	3000	3300
7698	MANAGER	2850	3420
7566	MANAGER	2975	3570
7839	PRESIDENT	5000	5000

Der Rückgabewert ist in diesem Fall abhängig vom Eintrag in der Spalte job.

Für alle Angestellten mit der Berufsbezeichnung ANALYST soll das Gehalt um 10% erhöht angezeigt werden (* 1.1), für alle CLERKS um 15%, für alle MANAGER um 20%, während alle anderen Berufsbezeichnungen unverändert bleiben (SALESMAN, PRESIDENT).

Syntax:

```
DECODE(Spalte, Wert, Rueckgabewert
       [, Wert, Rueckgabewert]
       [, Default-Rueckgabewert])
```

DECODE Funktion

Syntax:

```
DECODE(Spalte, Wert, Rueckgabewert
      {, Wert, Rueckgabewert}
      [, Default-Rueckgabewert])
```

Beispiel:

```
SELECT
    e.empno,
    e.job,
    e.sal,
    DECODE(job,'ANALYST', e.sal * 1.1,
           'CLERK', e.sal * 1.15,
           'MANAGER', e.sal * 1.2,
           e.sal) AS "Neues Gehalt"
  FROM emp e
 ORDER BY "Neues Gehalt";
```

EMPNO	JOB	SAL	Neues Gehalt
7369	CLERK	800	920
7900	CLERK	950	1092,5
7521	SALESMAN	1250	1250
7654	SALESMAN	1250	1250
7876	CLERK	1100	1265
7934	CLERK	1300	1495
7844	SALESMAN	1500	1500
7499	SALESMAN	1600	1600
7782	MANAGER	2450	2940
7788	ANALYST	3000	3300
7902	ANALYST	3000	3300
7698	MANAGER	2850	3420
7566	MANAGER	2975	3570
7839	PRESIDENT	5000	5000

4.2.7 Reguläre Ausdrücke

Reguläre Ausdrücke sind eine der Neuerungen bereits aus der Version ORACLE 10g.

4.2.7.1 Allgemein

Reguläre Ausdrücke (engl.: regular expressions) sind ein leistungsstarkes Werkzeug zur Verarbeitung von Texten und Daten.

Mit regulären Ausdrücken können sehr komplexe und subtile Textprobleme elegant beschrieben und gelöst werden. Reguläre Ausdrücke sind in der UNIX Welt (egrep, vi, awk, emacs, ...) seit langem verbreitet.

Beispiel:

Suche alle Zeilen aus der Datei protokoll.log, welche mit dem Wort "Fehler" oder dem Wort "Error" anfangen:

```
% egrep '^(\bFehler\b|\bError\b)' protokoll.log
```

Die Programmiersprachen Java, Visual Basic, VBScript, JavaScript, C, C++, C#, (.NET), elist, perl, Python, Tcl, Ruby, PHP, etc. können mit regulären Ausdrücken umgehen.

Ein produktiver Einsatz ist für Ungeübte jedoch kein einfaches Unterfangen, da sie ein sehr mächtiges und komplexes Werkzeug darstellen. Mehr über reguläre Ausdrücke im Allgemeinen erfährt man im Buch von J.E.F. Friedl: "Reguläre Ausdrücke".

Reguläre Ausdrücke im Zusammenhang mit Oracle sind in einer zusammengefassten Form im Buch "Oracle Regular Expressions" zu finden.

4.2.7.2 Einsatz

Mögliche Einsatzgebiete in ORACLE sind:

- das Suchen eines bestimmten Musters in einer Zeichenkette
- das Ersetzen eines bestimmten Musters durch ein anderes in einer Zeichenkette
- das Extrahieren bestimmter Muster aus einer Zeichenkette
- das Prüfen des Auftretens eines bestimmten Musters in einer Zeichenkette

Reguläre Ausdrücke können bei ORACLE auf folgende Datentypen angewandt werden:

- CHAR,
- VARCHAR2,
- CLOB,
- NCHAR,
- NVARCHAR,
- NCLOB

Reguläre Ausdrücke



- Reguläre Ausdrücke (über REGEXP Funktionen)
- Anwendung auf:
 - CHAR
 - VARCHAR2
 - CLOB
 - NCHAR
 - NVARCHAR2
 - NCLOB

4.2.7.3 REGEXP_Funktionen

Für reguläre Ausdrücke wurden vier neue Funktionen entwickelt:

1. REGEXP_INSTR
2. REGEXP_SUBSTR
3. REGEXP_LIKE
4. REGEXP_REPLACE
5. REGEXP_COUNT

4.2.7.3.1 REGEXP_INSTR

- REGEXP_INSTR stellt ähnlich wie der INSTR-Operator fest, an welcher Stelle das gesuchte Muster auftritt.

Beispiel:

Angestellte mit mehr als einem 'A' im Namen suchen:

```
SQL> SELECT e.*  
      FROM emp e  
     WHERE REGEXP_INSTR (e.ename, '[A]+', 1, 2) > 0;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7876	ADAMS	CLERK	7788	23.05.87	1100		20

4.2.7.3.2 REGEXP_SUBSTR

- REGEXP_SUBSTR ist ähnlich wie der SUBSTR-Operator. Gibt den Teilstring aus, welcher auf das gesuchte Muster passt.

Beispiel:

Alle Buchstaben bis zum ersten A ausgeben, sofern A nicht der erste Buchstabe ist.

```
SQL> SELECT
      e.empno,
      e.ename,
      REGEXP_SUBSTR (e.ename, '[^A]+') AS Test
    FROM    emp e;
```

EMPNO	ENAME	TEST
7369	SMITH	SMITH
7499	ALLEN	LLEN
7521	WARD	W
7566	JONES	JONES
7654	MARTIN	M
7698	BLAKE	BL
7782	CLARK	CL
7788	SCOTT	SCOTT
7839	KING	KING
7844	TURNER	TURNER
7876	ADAMS	D
7900	JAMES	J
7902	FORD	FORD
7934	MILLER	MILLER

14 Zeilen ausgewählt.

4.2.7.3.3 REGEXP_LIKE

- REGEXP_LIKE prüft ähnlich wie der LIKE-Operator, ob die Muster gleich sind.

Beispiel:

Alle die mit 'J' anfangen und ein 'N' im Namen haben:

```
SQL> SELECT e.empno,
           e.ename
      FROM   emp e
     WHERE
        REGEXP_LIKE (e.ename, '^J.*N');
```

EMPNO	ENAME
7566	JONES

4.2.7.3.4 REGEXP_REPLACE

- REGEXP_REPLACE ist ähnlich wie der REPLACE-Operator. Sucht und ersetzt die beschriebenen Muster.

Beispiel:

Ersetzen von ,L's durch das benachbarte Nachfolzezeichen, sofern dieses kein weiteres ,L' ist.

```
SQL> SELECT
      e.empno,
      e.ename,
      REGEXP_REPLACE (e.ename, 'L([^\L])', '\1\1')
                           AS Test
  FROM    emp e;
```

EMPNO	ENAME	TEST
7369	SMITH	SMITH
7499	ALLEN	ALEEN
7521	WARD	WARD
7566	JONES	JONES
7654	MARTIN	MARTIN
7698	BLAKE	BAAKE
7782	CLARK	CAARK
7788	SCOTT	SCOTT
7839	KING	KING
7844	TURNER	TURNER
7876	ADAMS	ADAMS
7900	JAMES	JAMES
7902	FORD	FORD
7934	MILLER	MILEER

14 Zeilen ausgewählt.

4.2.7.3.5 REGEXP_COUNT

Seit ORACLE 11g gibt es auch noch die Funktion REGEXP_COUNT.

Beispiel:

Zählen wie viele A's in jedem Namen vorkommen.

```
SQL> SELECT
      e.empno,
      e.ename,
      REGEXP_COUNT(e.ename, 'A') AS AnzAs
    FROM emp e;
```

EMPNO	ENAME	ANZAS
7369	SMITH	0
7499	ALLEN	1
7521	WARD	1
7566	JONES	0
7654	MARTIN	1
7698	BLAKE	1
7782	CLARK	1
7788	SCOTT	0
7839	KING	0
7844	TURNER	0
7876	ADAMS	2
7900	JAMES	1
7902	FORD	0
7934	MILLER	0

REGEXP																							
<ul style="list-style-type: none">▪ REGEXP Funktionen:<ul style="list-style-type: none">▪ REGEXP_INSTR▪ REGEXP_SUBSTR▪ REGEXP_LIKE▪ REGEXP_REPLACE▪ REGEXP_COUNT▪ Beispiel:<pre>SELECT e.ename, REGEXP_REPLACE (e.ename, 'L([^\L])', '\1\1') AS Test FROM emp e;</pre>	<table border="1" style="width: 100%; border-collapse: collapse;"><thead><tr><th style="background-color: #d9534f; color: white;">ENAME</th><th style="background-color: #d9534f; color: white;">TEST</th></tr></thead><tbody><tr><td>SMITH</td><td>SMITH</td></tr><tr><td>ALLEN</td><td>ALEEN</td></tr><tr><td>WARD</td><td>WARD</td></tr><tr><td>MARTIN</td><td>MARTIN</td></tr><tr><td>BLAKE</td><td>BAKE</td></tr><tr><td>CLARK</td><td>CAARK</td></tr><tr><td>SCOTT</td><td>SCOTT</td></tr><tr><td>KING</td><td>KING</td></tr><tr><td>TURNER</td><td>TURNER</td></tr><tr><td>MILLER</td><td>MILEER</td></tr></tbody></table>	ENAME	TEST	SMITH	SMITH	ALLEN	ALEEN	WARD	WARD	MARTIN	MARTIN	BLAKE	BAKE	CLARK	CAARK	SCOTT	SCOTT	KING	KING	TURNER	TURNER	MILLER	MILEER
ENAME	TEST																						
SMITH	SMITH																						
ALLEN	ALEEN																						
WARD	WARD																						
MARTIN	MARTIN																						
BLAKE	BAKE																						
CLARK	CAARK																						
SCOTT	SCOTT																						
KING	KING																						
TURNER	TURNER																						
MILLER	MILEER																						
2.5.0122 © Integrata Cegos GmbH	ORACLE und SQL	9																					

4.2.7.4 Die Metazeichen der Regulären Ausdrücke

Die Implementierung in ORACLE basiert auf dem POSIX ERE (Extended Regular Expression) Standard.

In "Oracle SQL Reference" sind folgende Metazeichen aufgeführt:

Tabelle 1. Metazeichen in regulären Ausdrücken

\	Der Backslash kann abhängig vom Kontext vier verschiedene Bedeutungen haben. Er kann: <ul style="list-style-type: none">• für sich selbst stehen• das nächste Zeichen zitieren• einen Operator einbringen• nichts tun
*	Passt bei beliebig vielen Vorkommen (auch bei keinen)
+	Passt bei mindestens einem Vorkommen
?	Passt bei keinem oder genau einem Vorkommen
	Trennzeichen für alternative Muster
^	Passt auf den Zeilenanfang
\$	Passt auf das Zeilenende
.	Passt auf jedes unterstützte Zeichen ausgenommen von NULL
[]	Klammerausdruck enthält eine Liste von Ausdrücken, von denen ein beliebiger passen darf. Eine Ausschlussliste beginnt mit einem Circumflex (^).
()	Gruppierung, wird als einzelner Teilausdruck behandelt.
{ m }	Passt bei genau <i>m</i> Auftreten.
{ m, }	Passt bei mindestens <i>m</i> Auftreten.
{ m, n }	Passt bei mindestens <i>m</i> Auftreten, jedoch nicht mehr als <i>n</i> .
\n	Eine Backreference (<i>n</i> ist eine Ziffer zwischen 1 und 9) passt auf den <i>n</i> -ten Teilausdruck vor dem \v
[. .]	Steht für ein (evtl. zusammengesetztes) Textzeichen, das aus mehreren Zeichen bestehen kann, z.B. [.ch.] im Spanischen.
[: :]	Steht für eine Zeichenklasse (z.B. [:alpha:]). Passt auf jedes Zeichen dieser Klasse.
[==]	Steht für eine Äquivalenzklasse. So passt z.B. [=o=] auf o, ô, ö, ò, usw.

Metazeichen von Regulären Ausdrücken		
\	Der Backslash kann abhängig vom Kontext vier verschiedene Bedeutungen haben. Er kann: für sich selbst stehen das nächste Zeichen zitieren einen Operator einbringen nichts tun	
*	Passt bei beliebig vielen Vorkommen (auch bei keinen)	
+	Passt bei mindestens einem Vorkommen	
?	Passt bei keinem oder genau einem Vorkommen	
	Trennzeichen für alternative Muster	
^	Passt auf den Zeilenanfang	
\$	Passt auf das Zeilenende	
.	Passt auf jedes unterstützte Zeichen ausgenommen von NULL	
[]	Klammerausdruck enthält eine Liste von Ausdrücken, von denen ein beliebiger passen darf. Eine Ausschlussliste beginnt mit einem Circumflex (^).	
()	Gruppierung, wird als einzelner Teilausdruck behandelt.	
{m}	Passt bei genau m Auftreten.	
{m,}	Passt bei mindestens m Auftreten.	
{m,n}	Passt bei mindestens m Auftreten, jedoch nicht mehr als n.	
\n	Eine Backreference (n ist eine Ziffer zwischen 1 und 9) passt auf den n-ten Teilausdruck vor dem \n	
[..]	Steht für ein (evtl. zusammengesetztes) Textzeichen, das aus mehreren Zeichen bestehen kann, z. B. [.ch.] im Spanischen.	
[: :]	Steht für eine Zeichenklasse (z.B. [:alpha:]). Passt auf jedes Zeichen dieser Klasse.	
[==]	Steht für eine Äquivalenzklasse. So passt z.B. [=o=] auf o, ö, ö, ö, usw.	

2.5.0122 © Integrata Cegos GmbH ORACLE und SQL 10

4.2.8 Zusammenfassung Single Row Funktionen

Alle bisher erwähnten Funktionen werden auch als **Single Row Funktionen** bezeichnet, da sie für **jede Zeile** einer Tabelle einen Rückgabewert liefern und nicht den Inhalt mehrerer Zeilen zusammenfassen.

Single Row Funktionen können beliebig ineinander verschachtelt werden. In diesem Fall wird ein Ausdruck von innen nach außen abgearbeitet.

Die Rückgabedatentypen der inneren Funktion müssen mit den Parameterdatentypen der äußeren Funktion übereinstimmen.

Beispiel:

```
SQL> SELECT e.empno,
      NVL(TO_CHAR(e.comm), 'Keine Provision')
      AS Provision
     FROM emp e;

EMPNO PROVISION
----- -----
 7369 Keine Provision
 7499 300
 7521 500
 7566 Keine Provision
 7654 1400
 7698 Keine Provision
 7782 Keine Provision
 7788 Keine Provision
 7839 Keine Provision
 7844 0
 7876 Keine Provision
 7900 Keine Provision
 7902 Keine Provision
 7934 Keine Provision
```

Zuerst wird `comm` in den Datentyp Character umgewandelt (innere Funktion `TO_CHAR`), erst dann kann der Spalte als Ersatzwert ein String zugeordnet werden(äußere Funktion `NVL`).

Tuningtipp: Die Funktionen brauchen natürlich für ihre Abarbeitung eine gewisse Zeit und Rechenpower. Also sparsam einsetzen.

Zusammenfassung Single Row Funktionen



- Manipulation der Werte
- Arbeiten jede Zeile einzeln ab
- Sind ineinander verschachtelbar

- Datentypen
- Anzahl der Parameter

- Brauchen Zeit für Abarbeitung

4.3 Gruppenfunktionen

Gruppenfunktionen, die auch als Aggregatfunktionen bezeichnet werden, fassen im Gegensatz zu Single Row Funktionen **mehrere Zeilen** zusammen und liefern *ein* einziges Ergebnis zurück. Sie alle **ignorieren** NULL-Werte, mit Ausnahme von COUNT(*) .

AVG (n) gibt den Durchschnittswert von **n** aus. **n** muss numerischen Datentyps sein.

Beispiel: SELECT AVG(sal) FROM emp;

Ergebnis: 2073,2143

COUNT (n) gibt die Anzahl der Zeilen **n** zurück

Beispiel: SELECT COUNT(comm) FROM emp;

Ergebnis: 4

COUNT (*) gibt die Anzahl der Zeilen in der Tabelle zurück

Beispiel: SELECT COUNT(*) FROM emp;

Ergebnis: 14

MAX (n) gibt den größten Wert für **n** zurück

Beispiel: SELECT MAX(comm) FROM emp;

Ergebnis: 1400

MIN (n) gibt den kleinsten Wert für **n** zurück

Beispiel: SELECT MIN(comm) FROM emp;

Ergebnis: 0 (nicht NULL!)

SUM (n) gibt die Summe für **n** zurück. **n** muss numerisch sein.

Beispiel: SELECT SUM(sal) FROM emp;

Ergebnis: 29025

Weitere Funktionen:

STDDEV (Standardabweichung), VARIANCE (Varianz) sowie weitere statistische Funktionen.

Alle Gruppenfunktionen sind auch mit DISTINCT kombinierbar:

Beispiel: SELECT COUNT(DISTINCT sal)
FROM emp;

Ergebnis: 12

Gruppenfunktionen												
<code>SELECT AVG(e.sal)</code>	<code>FROM emp e;</code>											
<code>SELECT COUNT(e.comm)</code>	<code>FROM emp e;</code>											
<code>SELECT COUNT(*)</code>	<code>FROM emp e;</code>											
<code>SELECT MAX(e.comm)</code>	<code>FROM emp e;</code>											
<code>SELECT MIN(e.comm)</code>	<code>FROM emp e;</code>											
<code>SELECT SUM(e.sal)</code>	<code>FROM emp e;</code>											
Beispiel:												
<code>SELECT</code>												
<code> AVG(e.sal), SUM(e.sal), MAX(e.sal), MIN(e.sal), COUNT(*)</code>												
<code>FROM emp e;</code>												
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr style="background-color: #e67e22; color: white;"> <th>AVG(E.SAL)</th> <th>SUM(E.SAL)</th> <th>MAX(E.SAL)</th> <th>MIN(E.SAL)</th> <th>COUNT(*)</th> </tr> </thead> <tbody> <tr> <td>2073,21429</td> <td>29025</td> <td>5000</td> <td>800</td> <td>14</td> </tr> </tbody> </table>			AVG(E.SAL)	SUM(E.SAL)	MAX(E.SAL)	MIN(E.SAL)	COUNT(*)	2073,21429	29025	5000	800	14
AVG(E.SAL)	SUM(E.SAL)	MAX(E.SAL)	MIN(E.SAL)	COUNT(*)								
2073,21429	29025	5000	800	14								
2.5.0122 © Integrata Cegos GmbH		ORACLE und SQL										
		12										

Beispiel:

```
SQL> SELECT
      AVG(e.sal),
      SUM(e.sal),
      MAX(e.sal),
      MIN(e.sal),
      COUNT(*)
    FROM emp e;
```

AVG(E.SAL)	SUM(E.SAL)	MAX(E.SAL)	MIN(E.SAL)	COUNT(*)
2073,21429	29025	5000	800	14

4.4 Die GROUP BY-Klausel

Die GROUP BY-Klausel ist nur in Kombination mit Gruppenfunktionen üblich.

Sie erlaubt die Gruppierung der Ergebnismengen nach den angegebenen Spalten. Enthält ein SELECT-Statement eine (oder mehrere) Gruppenfunktion(en), so *muss* jede weitere Spalte der SELECT-Klausel auch in der GROUP BY-Klausel stehen.

Gruppenfunktionen dürfen *nicht* in der GROUP BY-Klausel oder der WHERE-Klausel stehen.

Nicht zulässig:

```
SQL> SELECT      e.deptno,
          AVG(e.sal)
        FROM        emp e;
SQL> SELECT      e.deptno,
          AVG(e.sal)
        FROM        emp e
        WHERE      AVG(e.sal) > 1000;
SQL> SELECT      e.deptno,
          AVG(e.sal)
        FROM        emp e
        GROUP BY AVG(e.sal);
```

Zulässig:

```
SQL> SELECT      e.deptno,
          AVG(e.sal)
        FROM        emp e
        GROUP BY e.deptno;
```

DEPTNO	AVG (E.SAL)
30	1566,66667
20	2175
10	2916,66667

```
SQL> SELECT e.deptno,
           AVG(e.sal)
      FROM emp e
 WHERE e.job = 'CLERK'
 GROUP BY e.deptno;
```

DEPTNO	AVG (E.SAL)
30	950
20	950
10	1300

```
SQL> SELECT e.deptno,
           AVG(e.sal)
      FROM emp e
 GROUP BY e.deptno, e.job;
```

DEPTNO	AVG (E.SAL)
20	950
30	1400
20	2975
30	950
10	5000
30	2850
10	1300
10	2450
20	3000

GROUP BY-Klausel



Beispiel:

```
SQL> SELECT e.deptno, AVG(e.sal)
      FROM emp e
      GROUP BY e.deptno;
```

DEPTNO	AVG (SAL)
10	2916,66667
20	2175
30	1566,66667

GROUP BY-Klausel



```
SQL> SELECT e.deptno, AVG(e.sal)
      FROM emp e
      WHERE e.job = 'CLERK'
      GROUP BY e.deptno;
```

DEPTNO	AVG (E. SAL)
10	1300
20	950
30	950

```
SQL> SELECT e.deptno, AVG(e.sal)
      FROM emp e
      GROUP BY e.deptno, e.job;
```

DEPTNO	AVG (E. SAL)
10	1300
10	2450
10	5000
20	950
20	3000
20	2975
30	950
30	2850
30	1400

Syntax:

```
SELECT      Spalte {, Spalte} {, Gruppenfunktion}
FROM        Tabelle
[WHERE      bedingungen]
[GROUP BY   groupby-ausdruck];
```

groupby-ausdruck ::=

```
[ROLLUP | CUBE] Spalte {, Spalte}
```

gruppenfunktion ::=

```
MIN (Spalte)
| MAX (Spalte)
| SUM (Spalte)
| AVG (Spalte)
| COUNT (Spalte)
| COUNT (DISTINCT Spalte)
| COUNT (*)
```

Syntax GROUP BY



▪ **Syntax:**

```
SELECT      Spalte {, Spalte} {, Gruppenfunktion}
FROM        Tabelle
[ WHERE      bedingungen]
[ GROUP BY   groupby-ausdruck];
```

```
groupby-ausdruck ::= 
[ROLLUP | CUBE] Spalte {, Spalte}
```

gruppenfunktion ::=

```
MIN (Spalte)
| MAX (Spalte)
| SUM (Spalte)
| AVG (Spalte)
| COUNT (Spalte)
| COUNT (DISTINCT Spalte)
| COUNT (*)
```

4.4.1 GROUP BY mit ROLLUP- und CUBE-Operatoren

Diese beiden Operatoren ermöglichen die Ausgabe von Zwischen- und Endergebnissen in Zeilen mit Gruppengesamtwerten (“subtotal rows”), zusätzlich zu den regulär gruppierten Zeilen (“regular rows”). ROLLUP und CUBE stehen innerhalb der GROUP BY-Klausel.

4.4.1.1 Der ROLLUP-Operator

Beispiel:

```
SQL> SELECT
      e.deptno,
      e.job,
      COUNT(*),
      SUM(e.sal)
    FROM emp e
   GROUP BY ROLLUP (e.deptno, e.job);
```

DEPTNO	JOB	COUNT (*)	SUM (E.SAL)
10	CLERK	1	1300
10	MANAGER	1	2450
10	PRESIDENT	1	5000
10		3	8750
20	CLERK	2	1900
20	ANALYST	2	6000
20	MANAGER	1	2975
20		5	10875
30	CLERK	1	950
30	MANAGER	1	2850
30	SALESMAN	4	5600
30		6	9400
		14	29025

Ohne den ROLLUP-Operator würde in erster Präferenz nach deptno und in zweiter Präferenz nach job sortiert und für jede Kombination aus deptno und job ein Wert ausgegeben (regular rows). Mit ROLLUP wird zusätzlich ein Wert für jede Abteilung (gesamt, nicht nach job aufgeschlüsselt, subtotal rows) sowie ein Gesamtwert über alle Abteilungen (grand total) ausgegeben.

GROUP BY mit ROLLUP

Beispiel:

```
SELECT
    e.deptno,
    e.job,
    COUNT(*) ,
    SUM(e.sal)
FROM emp e
GROUP BY ROLLUP
    (e.deptno, e.job);
```

DEPTNO	JOB	COUNT (*)	SUM (SAL)
10	CLERK	1	1300
10	MANAGER	1	2450
10	PRESIDENT	1	5000
10		3	8750
20	CLERK	2	1900
20	ANALYST	2	6000
20	MANAGER	1	2975
20		5	10875
30	CLERK	1	950
30	MANAGER	1	2850
30	SALESMAN	4	5600
30		6	9400
		14	29025

4.4.1.2 Der CUBE-Operator

CUBE ist eine Erweiterung von ROLLUP. Es werden die gleichen Zeilen ausgegeben wie bei ROLLUP. Zusätzlich kommen noch weitere Zeilen mit Zwischenergebnissen dazu, nämlich je eine Aufgliederung pro zu gruppierender Spalte.

Beispiel:

```
SQL> SELECT
      e.deptno,
      e.job,
      COUNT(*),
      SUM(e.sal)
    FROM   emp e
  GROUP BY CUBE (e.deptno, e.job)
  ORDER BY e.deptno, e.job;
```

DEPTNO	JOB	COUNT (*)	SUM (E.SAL)
10	CLERK	1	1300
10	MANAGER	1	2450
10	PRESIDENT	1	5000
10		3	8750
20	ANALYST	2	6000
20	CLERK	2	1900
20	MANAGER	1	2975
20		5	10875
30	CLERK	1	950
30	MANAGER	1	2850
30	SALESMAN	4	5600
30		6	9400
	ANALYST	2	6000
	CLERK	4	4150
	MANAGER	3	8275
	PRESIDENT	1	5000
	SALESMAN	4	5600
		14	29025

GROUP BY mit CUBE

Beispiel:

```
SELECT
    e.deptno,
    e.job,
    COUNT(*),
    SUM(e.sal)
FROM emp e
GROUP BY CUBE (e.deptno, e.job)
ORDER BY e.deptno, e.job;
```

DEPTNO	JOB	COUNT (*)	SUM (E . SAL)
10	CLERK	1	1300
10	MANAGER	1	2450
10	PRESIDENT	1	5000
10		3	8750
20	ANALYST	2	6000
20	CLERK	2	1900
20	MANAGER	1	2975
20		5	10875
30	CLERK	1	950
30	MANAGER	1	2850
30	SALESMAN	4	5600
30		6	9400
	ANALYST	2	6000
	CLERK	4	4150
	MANAGER	3	8275
	PRESIDENT	1	5000
	SALESMAN	4	5600
		14	29025

4.5 Die HAVING-Klausel

Durch die WHERE-Bedingung können zwar einzelne Zeilen von der Verarbeitung ausgeschlossen werden, jedoch kann damit keine **Gruppe** aus der GROUP BY-Klausel **ausgeschlossen** werden. Dafür ist die HAVING-Klausel zuständig.

In der HAVING-Klausel können **Bedingungen** angegeben werden, die eine **Gruppe erfüllen** muss.

Im Unterschied zur WHERE-Bedingung dürfen in der HAVING-Bedingung auch **Gruppenfunktionen** angegeben werden.

Die HAVING-Klausel kann nur im **Zusammenhang** mit der GROUP BY-Klausel angegeben werden.

Beispiel:

```
SQL> SELECT
      e.deptno,
      MIN(e.sal),
      MAX(e.sal),
      AVG(e.sal)
    FROM emp e
   GROUP BY e.deptno
 HAVING AVG(e.sal) > 2000;
```

DEPTNO	MIN(E.SAL)	MAX(E.SAL)	AVG(E.SAL)
20	800	3000	2175
10	1300	5000	2916,66667

HAVING-Klausel													
<u>Beispiel:</u> <pre>SELECT e.deptno, MIN(e.sal), MAX(e.sal), AVG(e.sal) FROM emp e GROUP BY e.deptno HAVING AVG(e.sal) > 2000;</pre>	<ul style="list-style-type: none"> ▪ Ausschluss der Gruppen ▪ Bedingungen an Gruppenfunktionen ▪ Nur mit GROUP BY einzusetzen 												
<table border="1" data-bbox="366 714 1298 842"> <thead> <tr> <th>DEPTNO</th><th>MIN (E . SAL)</th><th>MAX (E . SAL)</th><th>AVG (E . SAL)</th></tr> </thead> <tbody> <tr> <td>10</td><td>1300</td><td>5000</td><td>2916,66667</td></tr> <tr> <td>20</td><td>800</td><td>3000</td><td>2175</td></tr> </tbody> </table>		DEPTNO	MIN (E . SAL)	MAX (E . SAL)	AVG (E . SAL)	10	1300	5000	2916,66667	20	800	3000	2175
DEPTNO	MIN (E . SAL)	MAX (E . SAL)	AVG (E . SAL)										
10	1300	5000	2916,66667										
20	800	3000	2175										
2.5.0122 © Integrata Cegos GmbH	ORACLE und SQL	18											

Syntax:

```
SELECT      Spalte { , Spalte } { , Gruppenfunktion }
FROM        Tabelle
[WHERE      bedingungen]
[GROUP BY   groupby-ausdruck]
[HAVING     bedingungen-having] ;
```

bedingungen-having ::=

```
havingbedingung
{ [OR havingbedingung] | [AND havingbedingung] }
```

havingbedingung ::=

```
[NOT] gruppenfunktion operator Wert
```

Syntax HAVING

▪ Syntax:

```
SELECT      Spalte {, Spalte} {, Gruppenfunktion}
FROM        Tabelle
[WHERE      bedingungen]
[GROUP BY   groupby-ausdruck]
[HAVING     bedingungen-having];

bedingungen-having ::= 
    havingbedingung
    {[OR havingbedingung] | [AND havingbedingung]}

havingbedingung ::= 
    [NOT] Gruppenfunktion operator Wert
```


5

SELECT Anweisungen mit mehreren Tabellen

5.1	JOIN-Operation	5-3
5.1.1	Kartesisches Produkt	5-4
5.1.2	JOIN-Operation Allgemein	5-8
5.1.3	Definition der JOIN-Operation in SQL	5-10
5.1.4	INNER JOIN-Operation	5-12
5.1.5	OUTER JOIN-Operation	5-14
5.1.6	LEFT OUTER JOIN	5-15
5.1.7	RIGHT OUTER JOIN	5-17
5.1.8	FULL OUTER JOIN	5-19
5.1.9	Der EQUI vs. NON EQUI JOIN	5-21
5.1.10	Der SELF JOIN	5-23
5.1.11	Der CROSS JOIN	5-25
5.1.12	Der NATURAL JOIN	5-25
5.1.13	Der EQUI JOIN mit USING-Klausel	5-27
5.2	Unterabfragen	5-29
5.2.1	Nicht korrelierte Unterabfragen	5-30
5.2.2	Korrelierte Unterabfragen	5-31
5.2.3	Verwendung von Single Row Vergleichsoperatoren	5-33
5.2.4	Verwendung mehrzelliger Vergleichsoperatoren	5-36
5.3	Pivot und Unpivot	5-45
5.3.1	Pivot	5-45

5.3.2	Unpivot.....	5-47
5.4	Hierarchische (rekursive) Abfragen	5-49
5.4.1	Formatieren hierarchischer Berichte	5-53
5.4.2	Einzelne Zweige ausblenden	5-55
5.4.3	Weitere Optionen zu hierarchischen Abfragen	5-57
5.5	SET-Operatoren	5-60

5 SELECT Anweisungen mit mehreren Tabellen

5.1 JOIN-Operation

Zur Redundanzvermeidung werden Tabellen in der Datenbank normalisiert. Sachlich zusammengehörende Informationen werden dabei in mehrere Tabellen verteilt. Um die normalisierten Daten wieder in eine gemeinsame Form zu bringen, müssen Abfragen gestartet werden, welche aus mehreren Quellen (Tabellen, Views, Unterabfragen) gleichzeitig die Daten holen. Dieser Vorgang nennt sich **JOIN**.

Der JOIN oder Verbund bezeichnet eine definierte Verknüpfung zwischen zwei Quellen.

Um es am Anfang ein wenig einfacher zu haben, werden zuerst stellvertretend für Quellen nur die Tabellen erwähnt und benutzt.

Falls Daten aus zwei Tabellen gleichzeitig gefiltert werden sollen, müssen die Daten dieser Tabellen zueinander mit einer zu diesem Zwecke definierten Beziehung verknüpft werden.

Die Beziehung kann einerseits strukturell (z.B. Fremdschlüssel) andererseits benutzerspezifisch (z.B. „die Gehaltsstufe zwischen den Tabellen emp und salgrade“) vorgegeben worden sein.

Falls die Daten zweier Quellen in keine Beziehung zueinander gesetzt werden, so wird **jede** Zeile einer Tabelle mit **allen** Zeilen der anderen Tabelle kombiniert (und ggf. am Bildschirm oder Drucker(!) ausgegeben). Das nennt man **kartesisches Produkt**. Falls es sich dabei um große Datenmengen handelt, kann der DB-Server (temporäre Berechnungen) schnell in die Knie gehen.

5.1.1 Kartesisches Produkt

Falls bei einer Abfrage mit zwei Tabellen keine Join-Bedingung gestellt wird, wird die Kombination von jeder Zeile der einen Tabelle mit allen Zeilen der anderen Tabelle gebildet. Dies nennt man ein **kartesisches Produkt** oder **Kreuzprodukt**.

Beispiel:

```
SQL> SELECT e.* , d.*
      FROM emp e, dept d;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	DEPTNO	DNAME	LOC	
7369	SMITH	CLERK	7902	17.12.80	800		20	10	ACCOUNTING	NEW YORK	
7499	ALLEN	SALESMAN	7698	20.02.81	1600	300	30	10	ACCOUNTING	NEW YORK	
7521	WARD	SALESMAN	7698	22.02.81	1250	500	30	10	ACCOUNTING	NEW YORK	
7566	JONES	MANAGER	7839	02.04.81	2975		20	10	ACCOUNTING	NEW YORK	
7654	MARTIN	SALESMAN	7698	28.09.81	1250	1400	30	10	ACCOUNTING	NEW YORK	
7698	BLAKE	MANAGER	7839	01.05.81	2850		30	10	ACCOUNTING	NEW YORK	
7782	CLARK	MANAGER	7839	09.06.81	2450		10	10	ACCOUNTING	NEW YORK	
7788	SCOTT	ANALYST	7566	19.04.87	3000		20	10	ACCOUNTING	NEW YORK	
7839	KING	PRESIDENT		17.11.81	5000		10	10	ACCOUNTING	NEW YORK	
7844	TURNER	SALESMAN	7698	08.09.81	1500		0	30	10	ACCOUNTING	NEW YORK
7876	ADAMS	CLERK	7788	23.05.87	1100		20	10	ACCOUNTING	NEW YORK	
7900	JAMES	CLERK	7698	03.12.81	950		30	10	ACCOUNTING	NEW YORK	
7902	FORD	ANALYST	7566	03.12.81	3000		20	10	ACCOUNTING	NEW YORK	
7934	MILLER	CLERK	7782	23.01.82	1300		10	10	ACCOUNTING	NEW YORK	
7369	SMITH	CLERK	7902	17.12.80	800		20	20	RESEARCH	DALLAS	
7499	ALLEN	SALESMAN	7698	20.02.81	1600	300	30	20	RESEARCH	DALLAS	
7521	WARD	SALESMAN	7698	22.02.81	1250	500	30	20	RESEARCH	DALLAS	
7566	JONES	MANAGER	7839	02.04.81	2975		20	20	RESEARCH	DALLAS	
7654	MARTIN	SALESMAN	7698	28.09.81	1250	1400	30	20	RESEARCH	DALLAS	
7698	BLAKE	MANAGER	7839	01.05.81	2850		30	20	RESEARCH	DALLAS	
7782	CLARK	MANAGER	7839	09.06.81	2450		10	20	RESEARCH	DALLAS	
7788	SCOTT	ANALYST	7566	19.04.87	3000		20	20	RESEARCH	DALLAS	
7839	KING	PRESIDENT		17.11.81	5000		10	20	RESEARCH	DALLAS	
7844	TURNER	SALESMAN	7698	08.09.81	1500		0	30	20	RESEARCH	DALLAS
7876	ADAMS	CLERK	7788	23.05.87	1100		20	20	RESEARCH	DALLAS	
7900	JAMES	CLERK	7698	03.12.81	950		30	20	RESEARCH	DALLAS	
7902	FORD	ANALYST	7566	03.12.81	3000		20	20	RESEARCH	DALLAS	
7934	MILLER	CLERK	7782	23.01.82	1300		10	20	RESEARCH	DALLAS	
7369	SMITH	CLERK	7902	17.12.80	800		20	30	SALES	CHICAGO	
7499	ALLEN	SALESMAN	7698	20.02.81	1600	300	30	30	SALES	CHICAGO	
7521	WARD	SALESMAN	7698	22.02.81	1250	500	30	30	SALES	CHICAGO	
7566	JONES	MANAGER	7839	02.04.81	2975		20	30	SALES	CHICAGO	
7654	MARTIN	SALESMAN	7698	28.09.81	1250	1400	30	30	SALES	CHICAGO	
7698	BLAKE	MANAGER	7839	01.05.81	2850		30	30	SALES	CHICAGO	
7782	CLARK	MANAGER	7839	09.06.81	2450		10	30	SALES	CHICAGO	
7788	SCOTT	ANALYST	7566	19.04.87	3000		20	30	SALES	CHICAGO	
7839	KING	PRESIDENT		17.11.81	5000		10	30	SALES	CHICAGO	
7844	TURNER	SALESMAN	7698	08.09.81	1500		0	30	30	SALES	CHICAGO
7876	ADAMS	CLERK	7788	23.05.87	1100		20	30	SALES	CHICAGO	
7900	JAMES	CLERK	7698	03.12.81	950		30	30	SALES	CHICAGO	
7902	FORD	ANALYST	7566	03.12.81	3000		20	30	SALES	CHICAGO	
7934	MILLER	CLERK	7782	23.01.82	1300		10	30	SALES	CHICAGO	
7369	SMITH	CLERK	7902	17.12.80	800		20	40	OPERATIONS	BOSTON	
7499	ALLEN	SALESMAN	7698	20.02.81	1600	300	30	40	OPERATIONS	BOSTON	
7521	WARD	SALESMAN	7698	22.02.81	1250	500	30	40	OPERATIONS	BOSTON	
7566	JONES	MANAGER	7839	02.04.81	2975		20	40	OPERATIONS	BOSTON	
7654	MARTIN	SALESMAN	7698	28.09.81	1250	1400	30	40	OPERATIONS	BOSTON	
7698	BLAKE	MANAGER	7839	01.05.81	2850		30	40	OPERATIONS	BOSTON	
7782	CLARK	MANAGER	7839	09.06.81	2450		10	40	OPERATIONS	BOSTON	
7788	SCOTT	ANALYST	7566	19.04.87	3000		20	40	OPERATIONS	BOSTON	
7839	KING	PRESIDENT		17.11.81	5000		10	40	OPERATIONS	BOSTON	
7844	TURNER	SALESMAN	7698	08.09.81	1500		0	30	40	OPERATIONS	BOSTON
7876	ADAMS	CLERK	7788	23.05.87	1100		20	40	OPERATIONS	BOSTON	
7900	JAMES	CLERK	7698	03.12.81	950		30	40	OPERATIONS	BOSTON	
7902	FORD	ANALYST	7566	03.12.81	3000		20	40	OPERATIONS	BOSTON	
7934	MILLER	CLERK	7782	23.01.82	1300		10	40	OPERATIONS	BOSTON	

56 rows selected.

Beispiel:

Im System sind zwei **relativ kleine** Tabellen: Kunden mit 1.000 Datensätzen und Rechnungen mit 10.000 Datensätzen vorhanden. Jede Rechnung ist nur einem Kunden zugeordnet. Ein Kunde kann aber auch mehrere Rechnungen bekommen haben. Falls in einer Abfrage auf diese zwei Tabellen ein kartesisches Produkt gebildet wird, erhalten wir **1.000 x 10.000 = 10.000.000** Datensätze zurück.

Ergebnis:

- 10 Millionen Datensätze werden auf dem Bildschirm (oder Drucker oder Excel-Datei oder Homepage, etc.) ausgegeben
- 9.990.000 von diesen Datensätzen sind **FALSCH** (99,9%)!!!
- **Korrekt** sind nur 10.000 (**0,1%** in Worten: **null Komma ein Prozent**) der gelieferten Datensätze
- 99,9% der Zeit hat der DB-Server umsonst gerechnet
- 99,9% der Daten haben die (temporären) Tablespaces und somit die Platte umsonst **zugemüllt**
- Jeder gelieferte Datensatz sei nur 1 kB groß. Somit hätte der DB-Server 9.990.000 kB \approx **9,99 GB (ca. 9,53 GiB)** Müll produziert.

Man stelle sich vor, es existiere noch eine kleine Tabelle **Mitarbeiter** in dieser Datenbank mit nur 100 Einträgen. Jeder Kunde wird von einem Mitarbeiter betreut. Wenn zu Kunden und Rechnungen noch die Tabelle **Mitarbeiter** hinzugenommen wird, sieht es noch schlimmer aus:

$$\mathbf{10.000.000 \times 100 = 1.000.000.000 \text{ Datensätze}}$$

Von einer Milliarde Datensätze sind hier auch nur 10.000 korrekt (0,001%). Die restlichen Ergebnisse können Interessierte selbst nachrechnen.

Kartesisches Produkt



- Kartesisches Produkt oder Kreuzprodukt
- Verknüpft
 - jeden Datensatz der Tabelle emp mit allen Datensätzen der Tabelle dept
- Beispiel:

```
SQL> SELECT e.* , d.*
      FROM   emp e, dept d;
```

2.5.0122 © Integrata Cegos GmbH

ORACLE und SQL

2

Kartesisches Produkt 2



EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	DEPTNO	DNAME	LOC
7369	SMITH	CLERK	7902	17.12.80	800		20	10	ACCOUNTING	NEW YORK
7499	ALLEN	SALESMAN	7698	20.02.81	1600	300	30	10	ACCOUNTING	NEW YORK
7521	WARD	SALESMAN	7698	22.02.81	1250	500	30	10	ACCOUNTING	NEW YORK
7566	JONES	MANAGER	7839	02.04.81	2975		20	10	ACCOUNTING	NEW YORK
7654	MARTIN	SALESMAN	7698	28.09.81	1250	1400	30	10	ACCOUNTING	NEW YORK
7698	BLAKE	MANAGER	7839	01.05.81	2850		30	10	ACCOUNTING	NEW YORK
7782	CLARK	MANAGER	7839	09.06.81	2450		10	10	ACCOUNTING	NEW YORK
→ 7788	SCOTT	ANALYST	7566	19.04.87	3000		20	10	ACCOUNTING	NEW YORK
→ 7839	KING	PRESIDENT		17.11.81	5000		10	10	ACCOUNTING	NEW YORK
7844	TURNER	SALESMAN	7698	08.09.81	1500	0	30	10	ACCOUNTING	NEW YORK
7876	ADAMS	CLERK	7788	23.05.87	1100		20	10	ACCOUNTING	NEW YORK
7900	JAMES	CLERK	7698	03.12.81	950		30	10	ACCOUNTING	NEW YORK
7902	FORD	ANALYST	7566	03.12.81	3000		20	10	ACCOUNTING	NEW YORK
→ 7934	MILLER	CLERK	7782	23.01.82	1300		10	10	ACCOUNTING	NEW YORK
→ 7369	SMITH	CLERK	7902	17.12.80	800		20	20	RESEARCH	DALLAS
7499	ALLEN	SALESMAN	7698	20.02.81	1600	300	30	20	RESEARCH	DALLAS
7521	WARD	SALESMAN	7698	22.02.81	1250	500	30	20	RESEARCH	DALLAS
→ 7566	JONES	MANAGER	7839	02.04.81	2975		20	20	RESEARCH	DALLAS
7654	MARTIN	SALESMAN	7698	28.09.81	1250	1400	30	20	RESEARCH	DALLAS
7698	BLAKE	MANAGER	7839	01.05.81	2850		30	20	RESEARCH	DALLAS
7782	CLARK	MANAGER	7839	09.06.81	2450		10	20	RESEARCH	DALLAS
→ 7788	SCOTT	ANALYST	7566	19.04.87	3000		20	20	RESEARCH	DALLAS
...										

2.5.0122 © Integrata Cegos GmbH

ORACLE und SQL

3

Kartesisches Produkt 3



- Beispiel:
 - Zwei Tabellen:
 - Kunden mit 1.000 Datensätzen
 - Rechnungen mit 10.000 Datensätzen
 - Ergebnisse des Kreuzprodukts
 - **1.000 x 10.000 = 10.000.000 Datensätze (10 Millionen)**
 - 9.990.000 Datensätze FALSCH (99,9%)
 - 10.000 Datensätze korrekt (0,1%)
 - Ca. 99,9% der Rechenzeit des DB-Servers umsonst
 - Sei 1 Datensatz nur 1 kB groß
→ $9.990.000 \times 1 \text{ kB} \sim 9.990 \text{ MB}$ (ca. 9.755 MiB) $\sim 9,99 \text{ GB}$ falsche DS

2.5.0122 © Integrata Cegos GmbH

ORACLE und SQL

4

5.1.2 JOIN-Operation Allgemein

Um dem Kreuzprodukt vorzubeugen, werden die Tabellen miteinander gejoint.

Im SQL gibt es im Prinzip zwei Arten des JOIN:

1. INNER JOIN und
2. OUTER JOIN

Weitere JOIN-Unterarten lassen sich auf das Prinzip eines INNER oder OUTER JOIN zurückführen.

Tabelle Dept:

Deptno	5.1.2.1.1.1	Dname
3	Marketing	
4	Verkauf	
8	EDV	
9	Fertigung	

Tabelle Emp:

Empno	Ename	Deptno
14	Meier	3
18	Fischer	8
19	Müller	3
30	Fischer	NULL

Inner Join beider Tabellen über Spalte deptno:

Empno	Ename	Deptno	Dname
14	Meier	3	Marketing
18	Fischer	8	EDV
19	Müller	3	Marketing

Abb. 5-1: JOIN Prinzip

Beispiel für Inner Join																												
Tabelle Dept:		Tabelle Emp:																										
<table border="1"> <thead> <tr> <th>Deptno</th><th>Dname</th></tr> </thead> <tbody> <tr><td>3</td><td>Marketing</td></tr> <tr><td>4</td><td>Verkauf</td></tr> <tr><td>8</td><td>EDV</td></tr> <tr><td>9</td><td>Fertigung</td></tr> </tbody> </table>		Deptno	Dname	3	Marketing	4	Verkauf	8	EDV	9	Fertigung	<table border="1"> <thead> <tr> <th>Empno</th><th>Ename</th><th>Deptno</th></tr> </thead> <tbody> <tr><td>14</td><td>Meier</td><td>3</td></tr> <tr><td>18</td><td>Fischer</td><td>8</td></tr> <tr><td>19</td><td>Müller</td><td>3</td></tr> <tr><td>30</td><td>Fischer</td><td>NULL</td></tr> </tbody> </table>		Empno	Ename	Deptno	14	Meier	3	18	Fischer	8	19	Müller	3	30	Fischer	NULL
Deptno	Dname																											
3	Marketing																											
4	Verkauf																											
8	EDV																											
9	Fertigung																											
Empno	Ename	Deptno																										
14	Meier	3																										
18	Fischer	8																										
19	Müller	3																										
30	Fischer	NULL																										
2.5.0122 © Integrata Cegos GmbH		ORACLE und SQL																										
		5																										

Beispiel für Inner Join																			
<ul style="list-style-type: none"> ▪ Inner Join 		<p>zwischen den Tabellen Dept und Emp über die Spalten Dept.Deptno und Emp.Deptno</p>																	
<table border="1"> <thead> <tr> <th>Empno</th><th>Ename</th><th>Deptno</th><th>Dname</th></tr> </thead> <tbody> <tr><td>14</td><td>Meier</td><td>3</td><td>Marketing</td></tr> <tr><td>18</td><td>Fischer</td><td>8</td><td>EDV</td></tr> <tr><td>19</td><td>Müller</td><td>3</td><td>Marketing</td></tr> </tbody> </table>				Empno	Ename	Deptno	Dname	14	Meier	3	Marketing	18	Fischer	8	EDV	19	Müller	3	Marketing
Empno	Ename	Deptno	Dname																
14	Meier	3	Marketing																
18	Fischer	8	EDV																
19	Müller	3	Marketing																
2.5.0122 © Integrata Cegos GmbH		ORACLE und SQL																	
		6																	

5.1.3 Definition der JOIN-Operation in SQL

ORACLE bietet zwei Möglichkeiten eine JOIN-Operation zu definieren:

1. ANSI-Konform in der FROM-Klausel (erst seit der Version 9i)
2. In der WHERE-Klausel

ANSI-konforme Definition der JOIN-Operation in der FROM-Klausel

Beispiel:

```
SQL> SELECT e.empno, e.ename, d.deptno, d.dname
      FROM emp e INNER JOIN dept d
        ON e.deptno = d.deptno;
```

Syntax:

```
SELECT      Spalte {, Spalte}
FROM        Tabelle joinausdruck Tabelle
ON          bedingungen
{joinausdruck Tabelle
ON          bedingungen}
```

```
[WHERE ...;
```

```
joinausdruck ::=
```

```
[ [INNER]
| [LEFT OUTER] | [RIGHT OUTER] | [FULL OUTER]
] JOIN
```

JOIN Syntax



▪ Syntax:

```
SELECT Spalte {, Spalte}
FROM   Tabelle joinausdruck Tabelle
ON    bedingungen
{joinausdruck Tabelle
ON    bedingungen}

[WHERE ...;

joinausdruck ::=
[[INNER] | [LEFT OUTER] | [RIGHT OUTER] | [FULL OUTER]] JOIN
```

▪ Beispiel:

```
SQL> SELECT e.empno, e.ename,
      d.deptno, d.dname
      FROM emp e INNER JOIN dept d
        ON e.deptno = d.deptno;
```

Definition der JOIN-Operation in der WHERE-Klausel

Beispiel:

```
SQL> SELECT e.empno, e.ename, d.deptno, d.dname
      FROM emp e, dept d
     WHERE e.deptno = d.deptno;
```

Syntax:

```
SELECT      Spalte {, Spalte}
FROM        Tabelle, Tabelle {, Tabelle}
WHERE       bedingungenjoin [bedingungen];
```

bedingungenjoin ::=

Tabelle.Spalte [(+)] operator Tabelle.Spalte [(+)]

JOIN Syntax	 integrata cegos
<ul style="list-style-type: none"> ▪ <u>Syntax:</u> <pre style="margin: 0;">SELECT Spalte {, Spalte} FROM Tabelle, Tabelle{, Tabelle} WHERE bedingungenjoin [bedingungen];</pre> <p style="margin-top: 20px;">bedingungenjoin ::=</p> <pre style="margin-top: 0; margin-left: 40px;">Tabelle.Spalte [(+)] operator Tabelle.Spalte [(+)]</pre> <ul style="list-style-type: none"> ▪ <u>Beispiel (WHERE-Klausel):</u> <pre style="margin-top: 20px;">SQL> SELECT e.empno, e.ename, d.deptno, d.dname FROM emp e, dept d WHERE e.deptno = d.deptno;</pre>	

5.1.4 INNER JOIN-Operation

Die INNER JOIN-Operation filtert alle Datensätze aus zwei Tabellen, bei welchen die Werte in den zu joinenden Spalten identisch sind.

Beispiel:

Alle Personalnummern, Namen und Arbeitsorte der Angestellten ausgeben.

INNER JOIN-Operation in der WHERE-Klausel

```
SQL> SELECT e.empno, e.ename, d.deptno, d.dname
      FROM emp e, dept d
      WHERE
            e.deptno = d.deptno;
```

INNER JOIN-Operation mit INNER JOIN in der FROM-Klausel

Dies ist die ANSI Norm des JOINS.

```
SQL> SELECT e.empno, e.ename, d.deptno, d.dname
      FROM emp e INNER JOIN dept d
      ON e.deptno = d.deptno;
```

Die Ergebnisse beider Abfragen

EMPNO	ENAME	DEPTNO	DNAME
7782	CLARK	10	ACCOUNTING
7839	KING	10	ACCOUNTING
7934	MILLER	10	ACCOUNTING
7566	JONES	20	RESEARCH
7902	FORD	20	RESEARCH
7876	ADAMS	20	RESEARCH
7369	SMITH	20	RESEARCH
7788	SCOTT	20	RESEARCH
7521	WARD	30	SALES
7844	TURNER	30	SALES
7499	ALLEN	30	SALES
7900	JAMES	30	SALES
7698	BLAKE	30	SALES
7654	MARTIN	30	SALES

14 Zeilen ausgewählt.

INNER JOIN



INNER JOIN in der WHERE-Klausel

```
SQL> SELECT e.empno, e.ename, d.deptno,
      d.dname
        FROM emp e, dept d
       WHERE e.deptno = d.deptno;
```

INNER JOIN in der FROM-Klausel

```
SQL> SELECT e.empno, e.ename, d.deptno,
      d.dname
        FROM emp e INNER JOIN dept d
       ON e.deptno = d.deptno;
```

EMPNO	ENAME	DEPTNO	DNAME
7782	CLARK	10	ACCOUNTING
7839	KING	10	ACCOUNTING
7934	MILLER	10	ACCOUNTING
7566	JONES	20	RESEARCH

5.1.5 OUTER JOIN-Operation

Bei der OUTER JOIN-Operation werden alle Datensätze einer Tabelle ausgegeben. Aus der zweiten Tabelle werden nur solche Datensätze angezeigt, welche über die JOIN-Bedingung verknüpfbar sind. Der Rest wird mit NULL aufgefüllt!

Es gibt drei Arten des OUTER JOINS:

1. LEFT OUTER JOIN

Gibt alle Datensätze aus der linken Tabelle aus. Aus der rechten Tabelle werden nur solche Datensätze ausgegeben, welche sich mit der linken Tabelle verknüpfen lassen. Der Rest der rechten Tabelle wird mit NULL aufgefüllt.

2. RIGHT OUTER JOIN

Analog zu LEFT OUTER JOIN.

3. FULL OUTER JOIN

Alle Datensätze beider Tabellen anzeigen.

Daten, welche in beiden Tabellen vorhanden sind, verknüpfen.
Datensätze, welche nur in der linken Tabelle vorhanden sind, von rechts mit NULL auffüllen.

Datensätze, welche nur in der rechten Tabelle vorhanden sind, von links mit NULL auffüllen.

OUTER JOIN	 integrata cegos
<ul style="list-style-type: none">▪ LEFT OUTER JOIN▪ RIGHT OUTER JOIN▪ FULL OUTER JOIN	

5.1.6 LEFT OUTER JOIN

Gibt alle Datensätze aus der **linken** Tabelle aus.

Beispiel:

Eine Liste **aller** Abteilungen mit den Namen der dazugehörigen Mitarbeiter ausgeben (falls Mitarbeiter vorhanden).

5.1.6.1 LEFT OUTER JOIN in der WHERE-Klausel

In diesem Beispiel wird jeder JOIN-Bedingung der **rechten** (nicht linken) Tabelle in der WHERE-Klausel der String "(+)" angehängt.

```
SQL> SELECT d.deptno, d.dname,
           e.empno, e.ename
      FROM dept d, emp e
     WHERE d.deptno = e.deptno(+);
```

5.1.6.2 LEFT OUTER JOIN in der FROM-Klausel

In dem vom ANSI definierten LEFT OUTER JOIN wird die linke Tabelle in der FROM-Klausel links vom Ausdruck "LEFT OUTER JOIN" gestellt.

```
SQL> SELECT d.deptno, d.dname,
           e.empno, e.ename
      FROM dept d LEFT OUTER JOIN emp e
        ON d.deptno = e.deptno;
```

DEPTNO	DNAME	EMPNO	ENAME
10	ACCOUNTING	7782	CLARK
10	ACCOUNTING	7839	KING
10	ACCOUNTING	7934	MILLER
20	RESEARCH	7566	JONES
20	RESEARCH	7902	FORD
20	RESEARCH	7876	ADAMS
20	RESEARCH	7369	SMITH
20	RESEARCH	7788	SCOTT
30	SALES	7521	WARD
30	SALES	7844	TURNER
30	SALES	7499	ALLEN
30	SALES	7900	JAMES
30	SALES	7698	BLAKE
30	SALES	7654	MARTIN
40	OPERATIONS		

15 Zeilen ausgewählt.

LEFT OUTER JOIN



LEFT OUTER JOIN in der WHERE-Klausel

```
SQL> SELECT d.deptno, d.dname, e.empno, e.ename
   FROM dept d, emp e
 WHERE d.deptno = e.deptno(+);
```

LEFT OUTER JOIN in der FROM-Klausel

```
SQL> SELECT d.deptno, d.dname, e.empno, e.ename
   FROM dept d LEFT OUTER JOIN emp e
     ON d.deptno = e.deptno;
```

DEPTNO	DNAME	EMPNO	ENAME
10	ACCOUNTING	7782	CLARK
10	ACCOUNTING	7839	KING
10	ACCOUNTING	7934	MILLER
20	RESEARCH	7566	JONES
20	RESEARCH	7902	FORD
20	RESEARCH	7876	ADAMS
20	RESEARCH	7369	SMITH
20	RESEARCH	7788	SCOTT
30	SALES	7521	WARD
30	SALES	7844	TURNER
30	SALES	7654	MARTIN
40	OPERATIONS		

5.1.7 RIGHT OUTER JOIN

Analog zu LEFT OUTER JOIN. Gibt alle Datensätze aus der **rechten** Tabelle aus.

Beispiel:

Eine Liste **aller** Abteilungen mit den Namen der dazugehörigen Mitarbeiter ausgeben (falls Mitarbeiter vorhanden).

5.1.7.1 RIGHT OUTER JOIN in der WHERE-Klausel

In diesem Beispiel wird jeder JOIN-Bedingung der **linken** (nicht rechten) Tabelle in der WHERE-Klausel der String "(+)" angehängt.

```
SQL> SELECT e.empno, e.ename,
           d.deptno, d.dname
      FROM dept d, emp e
     WHERE e.deptno(+) = d.deptno;
```

5.1.7.2 RIGHT OUTER JOIN in der FROM-Klausel

In dem vom ANSI definierten RIGHT OUTER JOIN wird die rechte Tabelle in der FROM-Klausel rechts vom Ausdruck "RIGHT OUTER JOIN" gestellt.

```
SQL> SELECT e.empno, e.ename,
           d.deptno, d.dname
      FROM emp e RIGHT OUTER JOIN dept d
     ON d.deptno = e.deptno;
```

EMPNO	ENAME	DEPTNO	DNAME
7782	CLARK	10	ACCOUNTING
7839	KING	10	ACCOUNTING
7934	MILLER	10	ACCOUNTING
7566	JONES	20	RESEARCH
7902	FORD	20	RESEARCH
7876	ADAMS	20	RESEARCH
7369	SMITH	20	RESEARCH
7788	SCOTT	20	RESEARCH
7521	WARD	30	SALES
7844	TURNER	30	SALES
7499	ALLEN	30	SALES
7900	JAMES	30	SALES
7698	BLAKE	30	SALES
7654	MARTIN	30	SALES
		40	OPERATIONS

15 Zeilen ausgewählt.

Das Ergebnis ist identisch mit dem beim LEFT OUTER JOIN.

RIGHT OUTER JOIN



RIGHT OUTER JOIN in der WHERE-Klausel

```
SQL> SELECT e.empno, e.ename, d.deptno, d.dname
  FROM dept d, emp e
 WHERE e.deptno(+) = d.deptno;
```

RIGHT OUTER JOIN in der FROM-Klausel

```
SQL> SELECT e.empno, e.ename, d.deptno, d.dname
  FROM   emp e RIGHT OUTER JOIN dept d
  ON d.deptno = e.deptno;
```

EMPNO	ENAME	DEPTNO	DNAME
7782	CLARK	10	ACCOUNTING
7839	KING	10	ACCOUNTING
7934	MILLER	10	ACCOUNTING
7566	JONES	20	RESEARCH
7902	FORD	20	RESEARCH
7876	ADAMS	20	RESEARCH
7369	SMITH	20	RESEARCH
7788	SCOTT	20	RESEARCH
7521	WARD	30	SALES
7844	TURNER	30	SALES
7499	ALLEN	30	SALES
7900	JAMES	30	SALES
7698	BLAKE	30	SALES
7654	MARTIN	30	SALES
		40	OPERATIONS

5.1.8 FULL OUTER JOIN

Gibt alle Datensätze aus **beiden** Tabellen aus.

Nicht verknüpfbare Datensätze aus der linken Tabelle werden von rechts mit `NULL` aufgefüllt, und nicht verknüpfbare Datensätze aus der rechten Tabelle werden von links mit `NULL` aufgefüllt.

Beispiel:

Eine Liste **aller** Abteilungen mit den Namen der dazugehörigen Mitarbeiter ausgeben (falls Mitarbeiter vorhanden).

5.1.8.1 FULL OUTER JOIN in der WHERE-Klausel

Funktioniert nicht!!!

```
SQL> SELECT e.empno, e.ename,
          d.deptno, d.dname
    FROM dept d, emp e
   WHERE e.deptno(+) = d.deptno(+);
```

5.1.8.2 FULL OUTER JOIN in der FROM-Klausel

```
SQL> SELECT e.empno, e.ename,
          d.deptno, d.dname
    FROM emp e FULL OUTER JOIN dept d
      ON d.deptno = e.deptno;
```

EMPNO	ENAME	DEPTNO	DNAME
7934	MILLER	10	ACCOUNTING
7839	KING	10	ACCOUNTING
7782	CLARK	10	ACCOUNTING
7902	FORD	20	RESEARCH
7876	ADAMS	20	RESEARCH
7788	SCOTT	20	RESEARCH
7566	JONES	20	RESEARCH
7369	SMITH	20	RESEARCH
7900	JAMES	30	SALES
7844	TURNER	30	SALES
7698	BLAKE	30	SALES
7654	MARTIN	30	SALES
7521	WARD	30	SALES
7499	ALLEN	30	SALES
		40	OPERATIONS

15 Zeilen ausgewählt.

Das Ergebnis ist identisch mit dem LEFT oder RIGHT OUTER JOIN.

FULL OUTER JOIN				
In der WHERE-Klausel NICHT				
<pre>SELECT e.empno, e.ename, d.deptno, d.dname FROM dept d, emp e WHERE e.deptno(+) = d.deptno(+); </pre>				
In der FROM-Klausel				
<pre>SELECT e.empno, e.ename, d.deptno, d.dname FROM emp e FULL OUTER JOIN dept d ON d.deptno = e.deptno;</pre>				
2.5.0122 © Integrata Cegos GmbH				ORACLE und SQL
				13

EMPNO	ENAME	DEPTNO	DNAME
7782	CLARK	10	ACCOUNTING
7839	KING	10	ACCOUNTING
7934	MILLER	10	ACCOUNTING
7566	JONES	20	RESEARCH
7902	FORD	20	RESEARCH
7786	ADAMS	20	RESEARCH
7369	SMITH	20	RESEARCH
7788	SCOTT	20	RESEARCH
7521	WARD	30	SALES
7844	TURNER	30	SALES
7499	ALLEN	30	SALES
7900	JAMES	30	SALES
7698	BLAKE	30	SALES
7654	MARTIN	30	SALES
		40	OPERATIONS

5.1.9 Der EQUI vs. NON EQUI JOIN

Abhängig von der Vergleichsart der zu joinenden Spalten gibt es:

1. *EQUI JOIN* Exaktheitsvergleich Operator: "="
2. *NON EQUI JOIN* Operatoren: <, <=, >, >=, BETWEEN

In bisherigen Beispielen haben wir immer den *EQUI JOIN* benutzt.

Beispiel für NON EQUI JOIN:

Namen aller Angestellten mit ihren Gehaltsstufen ausgeben.

Die Lösung erfolgt mit einem INNER JOIN in der FROM-Klausel. Jeder bislang gezeigte Weg, egal ob in WHERE- oder FROM-Klausel, INNER oder OUTER JOIN, würde uns ans Ziel bringen.

```
SQL> SELECT
      e.empno,
      e.ename,
      s.grade AS "Gehaltsstufe"
    FROM    emp e INNER JOIN salgrade s
    ON e.sal BETWEEN s.loSal AND s.hiSal;
```

EMPNO	ENAME	Gehaltsstufe
7369	SMITH	1
7900	JAMES	1
7876	ADAMS	1
7521	WARD	2
7654	MARTIN	2
7934	MILLER	2
7844	TURNER	3
7499	ALLEN	3
7782	CLARK	4
7698	BLAKE	4
7566	JONES	4
7788	SCOTT	4
7902	FORD	4
7839	KING	5

14 Zeilen ausgewählt.

EQUI vs. NON EQUI JOIN



```

SELECT
    e.empno,
    e.ename,
    s.grade AS "Gehaltsstufe"
FROM emp e INNER JOIN salgrade s
    ON e.sal BETWEEN s.loSal AND s.hiSal;
  
```

Vergleichsoperatoren von NON EQUI JOIN:

<, <=, >, >=, BETWEEN

EMPNO	ENAME	Gehaltsstufe
7369	SMITH	1
7900	JAMES	1
7876	ADAMS	1
7521	WARD	2
7654	MARTIN	2
7934	MILLER	2
7844	TURNER	3
7499	ALLEN	3
7782	CLARK	4
7698	BLAKE	4
7566	JONES	4
7788	SCOTT	4
7902	FORD	4
7839	KING	5

5.1.10 Der *SELF JOIN*

Er stellt im Grunde genommen keinen eigenen Typ dar. Mit *SELF JOIN* wird nur die Verknüpfung einer Tabelle mit sich selbst bezeichnet.

In diesem Fall muss mit Aliasnamen für die Tabelle gearbeitet werden, um zwei getrennte Tabellen vorzutäuschen.

Welche JOIN Art dabei benutzt wird ist unerheblich.

Beispiel:

Eine Liste aller Angestellten mit dem Namen des Vorgesetzten ausgeben.

```
SQL> SELECT
      ma.empno      AS "MA PersNr",
      ma.ename      AS "MA Name",
      bosse.empno   AS "Vorg PersNr",
      bosse.ename   AS "Vorg Name"
  FROM   emp ma INNER JOIN emp bosse
         ON ma.mgr = bosse.empno;
```

MA PersNr	MA Name	Vorg PersNr	Vorg Name
7902	FORD	7566	JONES
7788	SCOTT	7566	JONES
7844	TURNER	7698	BLAKE
7499	ALLEN	7698	BLAKE
7521	WARD	7698	BLAKE
7900	JAMES	7698	BLAKE
7654	MARTIN	7698	BLAKE
7934	MILLER	7782	CLARK
7876	ADAMS	7788	SCOTT
7698	BLAKE	7839	KING
7566	JONES	7839	KING
7782	CLARK	7839	KING
7369	SMITH	7902	FORD

13 Zeilen ausgewählt.

SELF JOIN



```
SELECT ma.empno AS "MA PersNr",
       ma.ename AS "MA Name",
       boss.empno AS "Vorg PersNr",
       boss.ename AS "Vorg Name"
  FROM emp ma INNER JOIN emp boss
    ON ma.mgr = boss.empno;
```

MA PersNr	MA Name	Vorg PersNr	Vorg Name
7902	FORD	7566	JONES
7788	SCOTT	7566	JONES
7844	TURNER	7698	BLAKE
7499	ALLEN	7698	BLAKE
7521	WARD	7698	BLAKE
7900	JAMES	7698	BLAKE
7654	MARTIN	7698	BLAKE
7934	MILLER	7782	CLARK
7876	ADAMS	7788	SCOTT
7698	BLAKE	7839	KING
7566	JONES	7839	KING
7782	CLARK	7839	KING
7369	SMITH	7902	FORD

2.5.0122 © Integrata Cegos GmbH

ORACLE und SQL

15

5.1.11 Der CROSS JOIN

Entspricht dem kartesischen Produkt der beteiligten Tabellen.

```
SQL> SELECT e.empno, e.ename, d.deptno, d.dname  
      FROM emp e CROSS JOIN dept d;
```

5.1.12 Der NATURAL JOIN

Der NATURAL JOIN stellt eine Erweiterung des *EQUI JOIN* dar und verknüpft die Tabellen **automatisch über alle Spalten**, die den **gleichen Namen** und den **gleichen Datentyp** haben.

Die zur Verknüpfung verwendeten Spalten (hier z.B. *deptno*) dürfen in ganzem SELECT-Statement jedoch keinen Tabellen- oder Aliasnamen als Präfix besitzen.

```
SQL> SELECT e.empno, e.ename, deptno, d.dname  
      FROM emp e NATURAL JOIN dept d;
```

EMPNO	ENAME	DEPTNO	DNAME
7782	CLARK	10	ACCOUNTING
7839	KING	10	ACCOUNTING
7934	MILLER	10	ACCOUNTING
7566	JONES	20	RESEARCH
7902	FORD	20	RESEARCH
7876	ADAMS	20	RESEARCH
7369	SMITH	20	RESEARCH
7788	SCOTT	20	RESEARCH
7521	WARD	30	SALES
7844	TURNER	30	SALES
7499	ALLEN	30	SALES
7900	JAMES	30	SALES
7698	BLAKE	30	SALES
7654	MARTIN	30	SALES

14 Zeilen ausgewählt.

```
SQL> SELECT e.empno, e.ename, deptno, d.dname
  FROM emp e NATURAL RIGHT OUTER JOIN dept d;

  EMPNO ENAME   DEPTNO DNAME
  ----- -----
    7782 CLARK      10 ACCOUNTING
    7839 KING       10 ACCOUNTING
    7934 MILLER     10 ACCOUNTING
    7566 JONES      20 RESEARCH
    7902 FORD       20 RESEARCH
    7876 ADAMS      20 RESEARCH
    7369 SMITH      20 RESEARCH
    7788 SCOTT      20 RESEARCH
    7521 WARD       30 SALES
    7844 TURNER     30 SALES
    7499 ALLEN      30 SALES
    7900 JAMES      30 SALES
    7698 BLAKE      30 SALES
    7654 MARTIN     30 SALES
                           40 OPERATIONS
```

15 Zeilen ausgewählt.

Bei „SELECT * ...“ werden die JOIN-Spalten nur einmal angezeigt.

```
SQL> SELECT *
  FROM emp e NATURAL JOIN dept d;
```

DEPTNO	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DNAME	LOC
10	7782	CLARK	MANAGER	7839	09.06.81	2450		ACCOUNTING	NEW YORK
10	7839	KING	PRESIDENT		17.11.81	5000		ACCOUNTING	NEW YORK
10	7934	MILLER	CLERK	7782	23.01.82	1300		ACCOUNTING	NEW YORK
20	7566	JONES	MANAGER	7839	02.04.81	2975		RESEARCH	DALLAS
20	7902	FORD	ANALYST	7566	03.12.81	3000		RESEARCH	DALLAS
20	7876	ADAMS	CLERK	7788	23.05.87	1100		RESEARCH	DALLAS
20	7369	SMITH	CLERK	7902	17.12.80	800		RESEARCH	DALLAS
20	7788	SCOTT	ANALYST	7566	19.04.87	3000		RESEARCH	DALLAS
30	7521	WARD	SALESMAN	7698	22.02.81	1250	500	SALES	CHICAGO
30	7844	TURNER	SALESMAN	7698	08.09.81	1500	0	SALES	CHICAGO
30	7499	ALLEN	SALESMAN	7698	20.02.81	1600	300	SALES	CHICAGO
30	7900	JAMES	CLERK	7698	03.12.81	950		SALES	CHICAGO
30	7698	BLAKE	MANAGER	7839	01.05.81	2850		SALES	CHICAGO
30	7654	MARTIN	SALESMAN	7698	28.09.81	1250	1400	SALES	CHICAGO

14 Zeilen ausgewählt.

Hinweis: Bei dem NATURAL JOIN besteht die Gefahr, dass es in den Tabellen durchaus Spalten mit gleicher Bezeichnung und gleichen Datentyp geben kann (Bemerkung, Memo, Feld1, Feld2, ...), über welche aber keinesfalls der JOIN aufgebaut werden sollte.

5.1.13 Der *EQUI JOIN* mit **USING**-Klausel

Im Gegensatz zum NATURAL JOIN wird beim *EQUI JOIN* über die USING-Klausel explizit eine bestimmte Spalte angegeben, über die verknüpft werden soll.

Auch hier muss die entsprechende Spalte im gesamten SELECT-Statement ohne Präfix verwendet werden.

```
SQL> SELECT *
      FROM emp e INNER JOIN dept d USING (deptno);
```

DEPTNO	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DNAME	LOC
10	7782	CLARK	MANAGER	7839	09.06.81	2450		ACCOUNTING	NEW YORK
10	7839	KING	PRESIDENT		17.11.81	5000		ACCOUNTING	NEW YORK
10	7934	MILLER	CLERK	7782	23.01.82	1300		ACCOUNTING	NEW YORK
20	7566	JONES	MANAGER	7839	02.04.81	2975		RESEARCH	DALLAS
20	7902	FORD	ANALYST	7566	03.12.81	3000		RESEARCH	DALLAS
20	7876	ADAMS	CLERK	7788	23.05.87	1100		RESEARCH	DALLAS
20	7369	SMITH	CLERK	7902	17.12.80	800		RESEARCH	DALLAS
20	7788	SCOTT	ANALYST	7566	19.04.87	3000		RESEARCH	DALLAS
30	7521	WARD	SALESMAN	7698	22.02.81	1250	500	SALES	CHICAGO
30	7844	TURNER	SALESMAN	7698	08.09.81	1500	0	SALES	CHICAGO
30	7499	ALLEN	SALESMAN	7698	20.02.81	1600	300	SALES	CHICAGO
30	7900	JAMES	CLERK	7698	03.12.81	950		SALES	CHICAGO
30	7698	BLAKE	MANAGER	7839	01.05.81	2850		SALES	CHICAGO
30	7654	MARTIN	SALESMAN	7698	28.09.81	1250	1400	SALES	CHICAGO

14 Zeilen ausgewählt.

```
SQL> SELECT e.empno, e.ename, deptno, d.dname
      FROM emp e INNER JOIN dept d USING (deptno);
```

EMPNO	ENAME	DEPTNO	DNAME
7782	CLARK	10	ACCOUNTING
7839	KING	10	ACCOUNTING
7934	MILLER	10	ACCOUNTING
7566	JONES	20	RESEARCH
7902	FORD	20	RESEARCH
7876	ADAMS	20	RESEARCH
7369	SMITH	20	RESEARCH
7788	SCOTT	20	RESEARCH
7521	WARD	30	SALES
7844	TURNER	30	SALES
7499	ALLEN	30	SALES
7900	JAMES	30	SALES
7698	BLAKE	30	SALES
7654	MARTIN	30	SALES

14 Zeilen ausgewählt.

```
SQL> SELECT e.empno, e.ename, deptno, d.dname
      FROM emp e      RIGHT OUTER JOIN dept d
                    USING (deptno);
```

EMPNO	ENAME	DEPTNO	DNAME
7782	CLARK	10	ACCOUNTING
7839	KING	10	ACCOUNTING
7934	MILLER	10	ACCOUNTING
7566	JONES	20	RESEARCH
7902	FORD	20	RESEARCH
7876	ADAMS	20	RESEARCH
7369	SMITH	20	RESEARCH
7788	SCOTT	20	RESEARCH
7521	WARD	30	SALES
7844	TURNER	30	SALES
7499	ALLEN	30	SALES
7900	JAMES	30	SALES
7698	BLAKE	30	SALES
7654	MARTIN	30	SALES
		40	OPERATIONS

15 Zeilen ausgewählt.

Weitere JOIN Unterarten



- **CROSS JOIN (Kartesisches Produkt):**

```
SQL> SELECT e.empno, e.ename, d.deptno, d.dname
      FROM emp e CROSS JOIN dept d;
```

- **NATURAL JOIN:**

```
SQL> SELECT *
      FROM emp e NATURAL JOIN dept d;
SQL> SELECT e.empno, e.ename, deptno, d.dname
      FROM emp e NATURAL JOIN dept d;
SQL> SELECT e.empno, e.ename, deptno, d.dname
      FROM emp e NATURAL RIGHT OUTER JOIN dept d;
```

- **EQUI JOIN mit USING-Klausel:**

```
SQL> SELECT *
      FROM emp e INNER JOIN dept d USING (deptno);
SQL> SELECT e.empno, e.ename, deptno, d.dname
      FROM emp e INNER JOIN dept d USING (deptno);
SQL> SELECT e.empno, e.ename, deptno, d.dname
      FROM emp e RIGHT OUTER JOIN dept d USING (deptno);
```

5.2 Unterabfragen

Unterabfragen, auch Subqueries genannt, sind SELECT-Anweisungen, die Teil eines anderen SQL-Befehls sind.

Am gebräuchlichsten sind sie in der WHERE- und FROM-Klausel.

Generell gilt bei der Verwendung von Unterabfragen:

- Die Unterabfrage muss in Klammern gesetzt werden.
- Die ORDER BY-Klausel ist in der Unterabfrage nicht erlaubt (in der Hauptabfrage schon) **Ausnahme:** Views und Inline-Views (Unterabfrage in der FROM-Klausel).
- Bei Verwendung in Single Row Operatoren (=, !=, <, >, <=, >=) darf die Unterabfrage nur genau **einen** Wert zurückliefern.
- Vektorvergleiche sind erlaubt.
- Bei Verwendung in mehrzeiligen Operatoren (IN, ANY, ALL) darf die Unterabfrage mehrere Werte (Liste) zurückliefern.
- Unterabfragen in der SELECT-Klausel dürfen nur einen skalaren Wert liefern.

<h3>Unterabfragen</h3> <ul style="list-style-type: none">▪ Sind vollwertige SELECT-Statements▪ Dürfen keine ORDER BY-Klausel enthalten, ausgenommen in:<ul style="list-style-type: none">▪ Views▪ Inline Views (Unterabfrage in der FROM-Klausel)▪ Dürfen in jeder Klausel mit Ausnahme der ORDER BY vorkommen▪ Müssen geklammert werden▪ In der SELECT-Klausel müssen sie einen einzigen skalaren Wert liefern▪ Bei Single-Row Vergleichen dürfen Sie nur einen einzigen skalaren Wert liefern▪ Vektorvergleiche sind erlaubt	
2.5.0122 © Integrata Cegos GmbH ORACLE und SQL 17	

5.2.1 Nicht korrelierte Unterabfragen

Eine nicht korrelierte Unterabfrage ist ein vollständiges SELECT-Statement, das außerhalb der Hauptabfrage lauffähig ist. Sie kann zum Beispiel zum Testen aus der Hauptabfrage herausgenommen und eigenständig optimiert werden.

Beispiel:

```
SQL> SELECT
      e.empno,
      e.ename,
      e.sal
    FROM   emp e
   WHERE  e.sal = ( SELECT  MAX(e2.sal)
                     FROM   emp e2
                 )
;
```

EMPNO	ENAME	SAL
7839	KING	5000

Die Unterabfrage (`SELECT MAX(e2.sal) FROM emp e2`) ist eine vollständige und vollwertige Abfrage und liefert derzeit (!) als Ergebnis: **5000**.

Obige Beispielabfrage ist somit analog zu folgender Abfrage:

```
SQL> SELECT
      e.empno,
      e.ename,
      e.sal
    FROM   emp e
   WHERE  e.sal = 5000;
```

EMPNO	ENAME	SAL
7839	KING	5000

5.2.2 Korrelierte Unterabfragen

Sie werden verwendet, wenn in der WHERE-Klausel der Unterabfrage Spalten aus der Tabelle der Hauptabfrage referenziert werden.

Im Gegensatz zur nicht korrelierten Unterabfrage wird hier konzeptionell die Unterabfrage für jede Zeile der Hauptabfrage ausgeführt. Jede korrelierte Abfrage lässt sich auch mit JOIN ausdrücken!

Abgesehen davon gelten die gleichen Regeln wie für einfache Unterabfragen. Es sind auch die gleichen Operatoren zulässig.

Syntax:

```
SELECT    A.spaltenreferenz
FROM      quelle A
WHERE     A.spalte1 vergleichsoperator
          (
            SELECT    B.spaltenreferenz
            FROM      quelle_B B
            WHERE     B.spalte2 = A.spalte
          );
```

Beispiel:

```
SQL>  SELECT e.empno, e.ename, e.sal, e.deptno
      FROM emp e
      WHERE e.sal >
          (
            SELECT AVG(e2.sal)
            FROM emp e2
            WHERE e2.deptno = e.deptno
          );
```

EMPNO	ENAME	SAL	DEPTNO
7499	ALLEN	1600	30
7566	JONES	2975	20
7698	BLAKE	2850	30
7788	SCOTT	3000	20
7839	KING	5000	10
7902	FORD	3000	20

Nicht korrelierte vs. korrelierten Abfragen



```
SQL> SELECT
      e.empno,
      e.ename,
      e.sal
    FROM emp e
   WHERE e.sal = ( SELECT MAX(e2.sal)
                     FROM      emp e2);
```

EMPNO	ENAME	SAL
7839	KING	5000

```
SQL> SELECT e.empno, e.ename, e.sal,
      e.deptno
    FROM emp e
   WHERE e.sal >
        (
          SELECT AVG(e2.sal)
            FROM emp e2
           WHERE e2.deptno = e.deptno
        );
```

EMPNO	ENAME	SAL	DEPTNO
7499	ALLEN	1600	30
7566	JONES	2975	20
7698	BLAKE	2850	30
7788	SCOTT	3000	20
7839	KING	5000	10
7902	FORD	3000	20

In diesem Beispiel werden alle Mitarbeiter aufgelistet, die **mehr** als das **Durchschnittsgehalt in ihrer Abteilung** verdienen. Für jeden Datensatz wird konzeptionell zunächst die Unterabfrage ausgeführt, um das Durchschnittsgehalt der Abteilung zu errechnen. Dieser Wert wird dann mit dem jeweiligen Wert aus der Hauptabfrage verglichen, und es wird geprüft ob die Bedingung erfüllt ist. Die Berechnung der Unterabfrage wird konzeptionell so oft durchgeführt, wie es in der Hauptabfrage Datensätze gibt.

Hinweis: Die konzeptionelle Betrachtung der korrelierten Unterabfrage entspricht nicht zwingend notwendig der physischen Abarbeitung derselben. Der Optimizer kann eine Unterabfrage intern in einen äquivalenten JOIN umwandeln. Welches Statement in welchem Fall optimal ist, muss jedes Mal aufs neue empirisch in der **konkreten Umgebung** (Betriebssystem, Hardware, Oracle Version, Datenbestand, ...) ermittelt werden.

5.2.3 Verwendung von Single Row Vergleichsoperatoren

Solche Unterabfragen sind neben der `WHERE`-Klausel auch in der `HAVING`-Klausel üblich.

Liefern sie mehr als einen Wert zurück, so wird eine Fehlermeldung ausgegeben.

Kritischer ist der Fall, dass gar kein Wert zurückgegeben wird (`NULL`). In diesem Fall gibt es keine Fehlermeldung sondern nur die Nachricht "no rows selected".

Beispiel:

```
-- korreliert: Spitzenverdiener je Abteilung
SQL> SELECT
      e.empno,
      e.ename,
      e.sal
     FROM emp e
    WHERE e.sal = (    SELECT MAX(e2.sal)
                      FROM emp e2
                     WHERE e2.deptno = e.deptno);
```

EMPNO	ENAME	SAL
7698	BLAKE	2850
7788	SCOTT	3000
7839	KING	5000
7902	FORD	3000

```
-- nicht korreliert: Kollegen von empno=7839
SQL> SELECT
      e.empno,
      e.ename,
      e.sal
    FROM   emp e
   WHERE  e.deptno =
          (
              SELECT  e2.deptno
              FROM   emp e2
              WHERE  e2.empno=7839
          );
EMPNO ENAME      SAL
----- -----
 7782 CLARK      2450
 7839 KING       5000
 7934 MILLER    1300
```

```
-- Unterabfrage welche NULL zurückliefert
SQL> SELECT
      e.empno,
      e.ename,
      e.sal
    FROM   emp e
   WHERE  e.comm =
          (
              SELECT  e2.comm
              FROM   emp e2
              WHERE  e2.empno = 7839
          );
```

Ergebnis:

Es wurden keine Zeilen ausgewählt

Einfache Unterabfrage mit Single Row Vergleich																														
<pre>SQL> SELECT e.empno, e.ename, e.sal FROM emp e WHERE e.sal = (SELECT MAX(e2.sal) FROM emp e2 WHERE e2.deptno = <u>e.deptno</u>) ; SQL> SELECT e.empno, e.ename, e.sal FROM emp e WHERE e.deptno = (SELECT e2.deptno FROM emp e2 WHERE e2.empno=7839) ; SQL> SELECT e.empno, e.ename, e.sal FROM emp e WHERE e.comm = (SELECT e2.comm FROM emp e2 WHERE e2.empno = 7839) ;</pre>			<table border="1"> <thead> <tr> <th>EMPNO</th><th>ENAME</th><th>SAL</th></tr> </thead> <tbody> <tr> <td>7698</td><td>BLAKE</td><td>2850</td></tr> <tr> <td>7788</td><td>SCOTT</td><td>3000</td></tr> <tr> <td>7839</td><td>KING</td><td>5000</td></tr> <tr> <td>7902</td><td>FORD</td><td>3000</td></tr> </tbody> </table> <table border="1"> <thead> <tr> <th>EMPNO</th><th>ENAME</th><th>SAL</th></tr> </thead> <tbody> <tr> <td>7782</td><td>CLARK</td><td>2450</td></tr> <tr> <td>7839</td><td>KING</td><td>5000</td></tr> <tr> <td>7934</td><td>MILLER</td><td>1300</td></tr> </tbody> </table>	EMPNO	ENAME	SAL	7698	BLAKE	2850	7788	SCOTT	3000	7839	KING	5000	7902	FORD	3000	EMPNO	ENAME	SAL	7782	CLARK	2450	7839	KING	5000	7934	MILLER	1300
EMPNO	ENAME	SAL																												
7698	BLAKE	2850																												
7788	SCOTT	3000																												
7839	KING	5000																												
7902	FORD	3000																												
EMPNO	ENAME	SAL																												
7782	CLARK	2450																												
7839	KING	5000																												
7934	MILLER	1300																												
			Es wurden keine Zeilen ausgewählt																											
2.5.0122 © Integrata Cegos GmbH	ORACLE und SQL		19																											

5.2.4 Verwendung mehrzelliger Vergleichsoperatoren

Die Unterabfrage darf in diesem Fall mehrere Ergebnisse an die Hauptabfrage zurückliefern. Alle Operatoren können auch in Kombination mit NOT verwendet werden.

Auch hier ist kein Vergleich mit NULL-Werten möglich. Sollen sie mit einbezogen werden, so empfiehlt sich die Verwendung der NVL-Funktion.

Es gibt vier mehrzeilige Vergleichsoperatoren:

1. IN
 2. EXIST
 3. ALL
 4. ANY bzw. SOME

5.2.4.1 IN

IN vergleicht mit **jedem einzelnen** Wert einer Liste.

Beispiel:

```
SQL> SELECT
      d.deptno,
      d.dname
    FROM dept d
   WHERE d.deptno IN
         (      SELECT      e.deptno
              FROM      emp e);
```

DEPTNO	DNAME
10	ACCOUNTING
20	RESEARCH
30	SALES

Hinweis: Quantifizierte Vergleiche mit “= ANY”, “<> ANY”, “= ALL”, “<> ALL” etc. sind sehr verwirrend und fehleranfällig.

- IN ist gleichbedeutend mit = ANY
 - NOT IN ist nicht gleichbedeutend mit <> ANY
 - NOT IN ist gleichbedeutend mit <> ALL (sofern keine NULL's im Spiel sind)

Mehrzeilige Unterabfragen (IN)



- Können sowohl:
 - korreliert als auch
 - nicht korreliert formuliert werden

Operator: IN

```
SQL> SELECT
      d.deptno,
      d.dname
    FROM dept d
   WHERE d.deptno IN
        (SELECT e.deptno
         FROM emp e);
```

DEPTNO	DNAME
10	ACCOUNTING
20	RESEARCH
30	SALES

5.2.4.2 EXISTS

EXISTS ist wahr, wenn der SELECT mindestens eine Zeile zurückgibt.
EXISTS wird in der Regel bei synchronisierten (korrelierten) Unterabfragen verwendet.

Vorteil von EXISTS ist, dass es immer **wahr** oder **falsch** ist. Es liefert nie den Wahrheitswert **unbekannt** zurück.

Beispiel:

```
SQL> SELECT
      d.deptno,
      d.dname
    FROM dept d
   WHERE EXISTS
        (SELECT 1
         FROM emp e
        WHERE e.deptno = d.deptno
        ) ;

DEPTNO  DNAME
----- -----
      10  ACCOUNTING
      20  RESEARCH
      30  SALES
```

Mehrzeilige Unterabfragen (EXISTS)



- Können sowohl:
- korreliert als auch
- nicht korreliert formuliert werden

▪ Operator: EXISTS

```
SQL> SELECT
      d.deptno,
      d.dname
    FROM dept d
   WHERE EXISTS
        (SELECT 1
         FROM emp e
        WHERE e.deptno = d.deptno
        ) ;
```

DEPTNO	DNAME
10	ACCOUNTING
20	RESEARCH
30	SALES

ANY bzw. SOME

ANY bzw. SOME vergleicht einen Wert mit **jedem** Wert einer Liste oder Abfrage. Davor muss ein einfacher Vergleichsoperator stehen. Es reicht, wenn die Bedingung für **irgendeinen** Wert aus der Liste erfüllt ist.

Beispiel:**Variante 1 (mit ANY):**

```
SQL> SELECT
      d.dname,
      d.deptno
    FROM dept d
   WHERE d.deptno > ANY (
      SELECT e.deptno
        FROM emp e
       WHERE e.job = 'CLERK'
    ) ;
```

Variante 2 (mit MIN):

```
SQL> SELECT
      d.dname,
      d.deptno
    FROM dept d
   WHERE d.deptno > (
      SELECT MIN(e.deptno)
        FROM emp e
       WHERE e.job = 'CLERK'
    ) ;
```

Variante 3 (mit JOIN):

```
SQL> SELECT
      d.dname,
      d.deptno
    FROM
      dept d INNER JOIN
      (   SELECT MIN(e.deptno) AS dnumber
        FROM emp e
       WHERE e.job = 'CLERK' ) a
     ON d.deptno > a.dnumber
    ;
```

DNAME	DEPTNO
RESEARCH	20
SALES	30
OPERATIONS	40

Mehrzeilige Unterabfragen (ANY bzw. SOME) 

Variante1 (mit ANY):

```
SQL> SELECT
      d.dname, d.deptno
    FROM dept d
   WHERE d.deptno > ANY (
      SELECT e.deptno
        FROM emp e
       WHERE e.job = 'CLERK');
```

DNAME	DEPTNO
RESEARCH	20
SALES	30
OPERATIONS	40

Variante 2 (mit MIN):

```
SQL> SELECT
      d.dname, d.deptno
    FROM dept d
   WHERE d.deptno > (
      SELECT MIN(e.deptno)
        FROM emp e
       WHERE e.job = 'CLERK');
```

Variante 3 (mit JOIN):

```
SQL> SELECT
      d.dname, d.deptno
    FROM
      dept d INNER JOIN
      ( SELECT MIN(e.deptno) AS dnumber
        FROM emp e
       WHERE e.job = 'CLERK') a
   ON d.deptno > a.dnumber;
```

ALL

ALL vergleicht einen Wert mit **jedem** Wert einer Liste oder Abfrage. Davor muss ein einfacher Vergleichsoperator stehen. Die angegebene Bedingung muss für **jeden Wert** aus der Liste erfüllt sein.

Beispiel (ohne NULL-Werte in der Unterabfrage):**Variante 1 (mit ALL):**

```
SQL> SELECT
      d.dname,
      d.deptno
    FROM dept d
   WHERE d.deptno > ALL (
      SELECT      e.deptno
      FROM        emp e
      WHERE       job = 'CLERK'
    );
```

Variante 2 (mit MAX):

```
SQL> SELECT
      d.dname,
      d.deptno
    FROM dept d
   WHERE d.deptno > (
      SELECT      MAX(e.deptno)
      FROM        emp e
      WHERE       e.job = 'CLERK'
    );
```

Variante 3 (mit JOIN):

```
SQL> SELECT
      d.dname,
      d.deptno
    FROM
      dept d INNER JOIN
      (      SELECT      MAX(e.deptno) AS dnumber
            FROM        emp e
            WHERE       e.job = 'CLERK' ) a
    ON d.deptno > a.dnumber
;
```

DNAME	DEPTNO
OPERATIONS	40

Mehrzeilige Unterabfragen (ALL)					
Variante 1 (mit ALL):					
<pre>SQL> SELECT d.dname, d.deptno FROM dept d WHERE d.deptno > ALL (SELECT e.deptno FROM emp e WHERE job = 'CLERK');</pre>	<table border="1"> <thead> <tr> <th>DNAME</th><th>DEPTNO</th></tr> </thead> <tbody> <tr> <td>OPERATIONS</td><td>40</td></tr> </tbody> </table>	DNAME	DEPTNO	OPERATIONS	40
DNAME	DEPTNO				
OPERATIONS	40				
Variante 2 (mit MAX)					
<pre>SQL> SELECT d.dname, d.deptno FROM dept d WHERE d.deptno > (SELECT MAX(e.deptno) FROM emp e WHERE e.job = 'CLERK'); </pre>					
Variante 3 (mit JOIN)					
<pre>SQL> SELECT d.dname, d.deptno FROM dept d INNER JOIN (SELECT MAX(e.deptno) AS dnumber FROM emp e WHERE e.job = 'CLERK') a ON d.deptno > a.dnumber ;</pre>					
2.5.0122 © Integrata Cegos GmbH					
ORACLE und SQL					
23					

Hinweis: Vorsicht mit **ALL**-Vergleichen und Variationen wenn NULL-Werte im Spiel sind.

Beispiel mit einer leeren Unterabfrage:

Variante 1 (mit ALL):

```
SQL>      SELECT
              d.dname,
              d.deptno
          FROM dept d
         WHERE d.deptno > ALL  (
                  SELECT e.deptno
                  FROM emp e
                 WHERE job = 'TRAINER'
                );

```

DNAME	DEPTNO
ACCOUNTING	10
RESEARCH	20
SALES	30
OPERATIONS	40

Variante 2 (mit MAX):

```
SQL> SELECT
      d.dname,
      d.deptno
    FROM dept d
   WHERE d.deptno > (
      SELECT      MAX(e.deptno)
      FROM        emp e
      WHERE       e.job = 'TRAINER'
    );
```

Variante 3 (mit JOIN):

```
SQL> SELECT
      d.dname,
      d.deptno
    FROM
      dept d INNER JOIN
      (      SELECT      MAX(e.deptno) AS dnumber
            FROM        emp e
            WHERE       e.job = 'TRAINER') a
     ON d.deptno > a.dnumber
    ;
```

Ergebnisse der Variante 2 und 3:

Es wurden keine Zeilen ausgewählt

Beispiel mit Mischung aus NULL- und anderen Werten in der Unterabfrage:**Variante 1 (mit ALL):**

```
SQL> SELECT
      e.empno,
      e.ename,
      e.deptno
    FROM emp e
   WHERE e.comm > ALL (
      SELECT e.comm
        FROM emp e
       WHERE e.sal > 1500
     ) ;
```

Ergebnis:

Es wurden keine Zeilen ausgewählt

Variante 2 (mit MAX):

```
SQL> SELECT
      e.empno,
      e.ename,
      e.deptno
    FROM emp e
   WHERE e.comm > (
      SELECT MAX(e.comm)
        FROM emp e
       WHERE e.sal > 1500
     ) ;
```

Variante 3 (mit JOIN):

```
SQL> SELECT
      e1.empno,
      e1.ename,
      e1.deptno
    FROM emp e1 INNER JOIN
      ( SELECT MAX(e.comm) mcomm
        FROM emp e
       WHERE e.sal > 1500) a
   ON e1.comm > a.mcomm;
```

Ergebnisse der Variante 2 und 3:

EMPNO	ENAME	DEPTNO
7521	WARD	30
7654	MARTIN	30

5.3 Pivot und Unpivot

Ab Oracle 11g sind zwei neue Funktionalitäten eingeführt worden, welche vor Allem im BI und DWH-Bereich auf großes Interesse stoßen sollten. Das sind die PIVOT und UNPIVOT-Operationen.

5.3.1 Pivot

Pivot Operation wandelt die **Zeilen in Spalten** um, in dem sie den Dateninhalt aggregiert und rotiert.

Beispiel:

Beispiel:

Die Summe der Gehälter für jede Jobausprägung und je Abteilung ausgeben. Die Abteilungen sollen als Spalten dargestellt werden:

```
SQL>   SELECT      e1.*  
        FROM  
          (      (      SELECT  
                  e2.job,  
                  e2.deptno,  
                  SUM(e2.sal)  summegeh  
            FROM emp e2  
           GROUP BY  
                  e2.job, e2.deptno  
        ) e3  
      PIVOT      (      SUM(e3.summegeh)  
                    FOR DEPTNO IN (10, 20, 30)  
      )  
    ) e1;
```

JOB	10	20	30
CLERK	1300	1900	950
SALESMAN			5600
PRESIDENT	5000		
MANAGER	2450	2975	2850
ANALYST		6000	

5 Zeilen ausgewählt.

Pivot																									
<pre>SELECT e1.* FROM ((SELECT e2.job, e2.deptno, SUM(e2.sal) summegeh FROM emp e2 GROUP BY e2.job, e2.deptno) e3 PIVOT (SUM(e3.summegeh) FOR DEPTNO IN (10, 20, 30))) e1;</pre>																									
<table><thead><tr><th>JOB</th><th>10</th><th>20</th><th>30</th></tr></thead><tbody><tr><td>CLERK</td><td>1300</td><td>1900</td><td>950</td></tr><tr><td>SALESMAN</td><td></td><td></td><td>5600</td></tr><tr><td>PRESIDENT</td><td>5000</td><td></td><td></td></tr><tr><td>MANAGER</td><td>2450</td><td>2975</td><td>2850</td></tr><tr><td>ANALYST</td><td></td><td>6000</td><td></td></tr></tbody></table>		JOB	10	20	30	CLERK	1300	1900	950	SALESMAN			5600	PRESIDENT	5000			MANAGER	2450	2975	2850	ANALYST		6000	
JOB	10	20	30																						
CLERK	1300	1900	950																						
SALESMAN			5600																						
PRESIDENT	5000																								
MANAGER	2450	2975	2850																						
ANALYST		6000																							

5.3.2 Unpivot

Unpivot Operation wandelt die **Spalten in Zeilen** um, in dem sie den Dateninhalt aggregiert und rotiert.

Beispiel:

Gehaltsteilarten jedes Mitarbeiters darstellen:

```
SQL> SELECT e1.empno, e1.ename, e1.job,
      e1.zahlungsart, e1.jahreszahlen
    FROM   (      emp e2
              UNPIVOT ( Jahreszahlen FOR
                         Zahlungsart IN
                         (SAL AS 'Gehalt',
                          COMM AS 'Provision')
                     )
      ) e1
  ORDER BY e1.empno;
```

EMPNO	ENAME	JOB	ZAHLUNGSART	JAHRESZAHLEN
7369	SMITH	CLERK	Gehalt	800
7499	ALLEN	SALESMAN	Gehalt	1600
7499	ALLEN	SALESMAN	Provision	300
7521	WARD	SALESMAN	Gehalt	1250
7521	WARD	SALESMAN	Provision	500
7566	JONES	MANAGER	Gehalt	2975
7654	MARTIN	SALESMAN	Gehalt	1250
7654	MARTIN	SALESMAN	Provision	1400
7698	BLAKE	MANAGER	Gehalt	2850
7782	CLARK	MANAGER	Gehalt	2450
7788	SCOTT	ANALYST	Gehalt	3000
7839	KING	PRESIDENT	Gehalt	5000
7844	TURNER	SALESMAN	Gehalt	1500
7844	TURNER	SALESMAN	Provision	0
7876	ADAMS	CLERK	Gehalt	1100
7900	JAMES	CLERK	Gehalt	950
7902	FORD	ANALYST	Gehalt	3000
7934	MILLER	CLERK	Gehalt	1300

18 Zeilen ausgewählt.

Unpivot																																																																																																
<pre> SELECT e1.empno, e1.ename, e1.job, e1.zahlungsart, e1.jahreszahlen FROM (emp e2 UNPIVOT (Jahreszahlen FOR Zahlungsart IN (SAL AS 'Gehalt', COMM AS 'Provision'))) e1 ORDER BY e1.empno; </pre>																																																																																																
	<table border="1"> <thead> <tr> <th>EMPNO</th><th>ENAME</th><th>JOB</th><th>ZAHLUNGSART</th><th>JAHRESZAHLEN</th></tr> </thead> <tbody> <tr><td>7369</td><td>SMITH</td><td>CLERK</td><td>Gehalt</td><td>800</td></tr> <tr><td>7499</td><td>ALLEN</td><td>SALESMAN</td><td>Gehalt</td><td>1600</td></tr> <tr><td>7499</td><td>ALLEN</td><td>SALESMAN</td><td>Provision</td><td>300</td></tr> <tr><td>7521</td><td>WARD</td><td>SALESMAN</td><td>Gehalt</td><td>1250</td></tr> <tr><td>7521</td><td>WARD</td><td>SALESMAN</td><td>Provision</td><td>500</td></tr> <tr><td>7566</td><td>JONES</td><td>MANAGER</td><td>Gehalt</td><td>2975</td></tr> <tr><td>7654</td><td>MARTIN</td><td>SALESMAN</td><td>Gehalt</td><td>1250</td></tr> <tr><td>7654</td><td>MARTIN</td><td>SALESMAN</td><td>Provision</td><td>1400</td></tr> <tr><td>7698</td><td>BLAKE</td><td>MANAGER</td><td>Gehalt</td><td>2850</td></tr> <tr><td>7782</td><td>CLARK</td><td>MANAGER</td><td>Gehalt</td><td>2450</td></tr> <tr><td>7788</td><td>SCOTT</td><td>ANALYST</td><td>Gehalt</td><td>3000</td></tr> <tr><td>7839</td><td>KING</td><td>PRESIDENT</td><td>Gehalt</td><td>5000</td></tr> <tr><td>7844</td><td>TURNER</td><td>SALESMAN</td><td>Gehalt</td><td>1500</td></tr> <tr><td>7844</td><td>TURNER</td><td>SALESMAN</td><td>Provision</td><td>0</td></tr> <tr><td>7876</td><td>ADAMS</td><td>CLERK</td><td>Gehalt</td><td>1100</td></tr> <tr><td>7900</td><td>JAMES</td><td>CLERK</td><td>Gehalt</td><td>950</td></tr> <tr><td>7902</td><td>FORD</td><td>ANALYST</td><td>Gehalt</td><td>3000</td></tr> <tr><td>7934</td><td>MILLER</td><td>CLERK</td><td>Gehalt</td><td>1300</td></tr> </tbody> </table>	EMPNO	ENAME	JOB	ZAHLUNGSART	JAHRESZAHLEN	7369	SMITH	CLERK	Gehalt	800	7499	ALLEN	SALESMAN	Gehalt	1600	7499	ALLEN	SALESMAN	Provision	300	7521	WARD	SALESMAN	Gehalt	1250	7521	WARD	SALESMAN	Provision	500	7566	JONES	MANAGER	Gehalt	2975	7654	MARTIN	SALESMAN	Gehalt	1250	7654	MARTIN	SALESMAN	Provision	1400	7698	BLAKE	MANAGER	Gehalt	2850	7782	CLARK	MANAGER	Gehalt	2450	7788	SCOTT	ANALYST	Gehalt	3000	7839	KING	PRESIDENT	Gehalt	5000	7844	TURNER	SALESMAN	Gehalt	1500	7844	TURNER	SALESMAN	Provision	0	7876	ADAMS	CLERK	Gehalt	1100	7900	JAMES	CLERK	Gehalt	950	7902	FORD	ANALYST	Gehalt	3000	7934	MILLER	CLERK	Gehalt	1300
EMPNO	ENAME	JOB	ZAHLUNGSART	JAHRESZAHLEN																																																																																												
7369	SMITH	CLERK	Gehalt	800																																																																																												
7499	ALLEN	SALESMAN	Gehalt	1600																																																																																												
7499	ALLEN	SALESMAN	Provision	300																																																																																												
7521	WARD	SALESMAN	Gehalt	1250																																																																																												
7521	WARD	SALESMAN	Provision	500																																																																																												
7566	JONES	MANAGER	Gehalt	2975																																																																																												
7654	MARTIN	SALESMAN	Gehalt	1250																																																																																												
7654	MARTIN	SALESMAN	Provision	1400																																																																																												
7698	BLAKE	MANAGER	Gehalt	2850																																																																																												
7782	CLARK	MANAGER	Gehalt	2450																																																																																												
7788	SCOTT	ANALYST	Gehalt	3000																																																																																												
7839	KING	PRESIDENT	Gehalt	5000																																																																																												
7844	TURNER	SALESMAN	Gehalt	1500																																																																																												
7844	TURNER	SALESMAN	Provision	0																																																																																												
7876	ADAMS	CLERK	Gehalt	1100																																																																																												
7900	JAMES	CLERK	Gehalt	950																																																																																												
7902	FORD	ANALYST	Gehalt	3000																																																																																												
7934	MILLER	CLERK	Gehalt	1300																																																																																												

5.4 Hierarchische (rekursive) Abfragen

Hierarchische (rekursive) Abfragen sind dann möglich, wenn eine hierarchische Beziehung innerhalb einer Tabelle besteht. Ein Join ist in einer hierarchischen Abfrage nicht zulässig.

Beispiel:

In der Tabelle `emp` steht 'KING' ganz oben in der Hierarchie, die über die Spalte `mgr` aufgebaut werden kann. 'KING' ist unmittelbarer Vorgesetzter von 'JONES', 'BLAKE' und 'CLARK'. Alle drei sind wiederum Vorgesetzte für andere Mitarbeiter. 'JONES' beispielsweise hat 'SCOTT' und 'FORD' als Untergebene, und 'SCOTT' seinerseits ist der Vorgesetzte von 'ADAMS'.

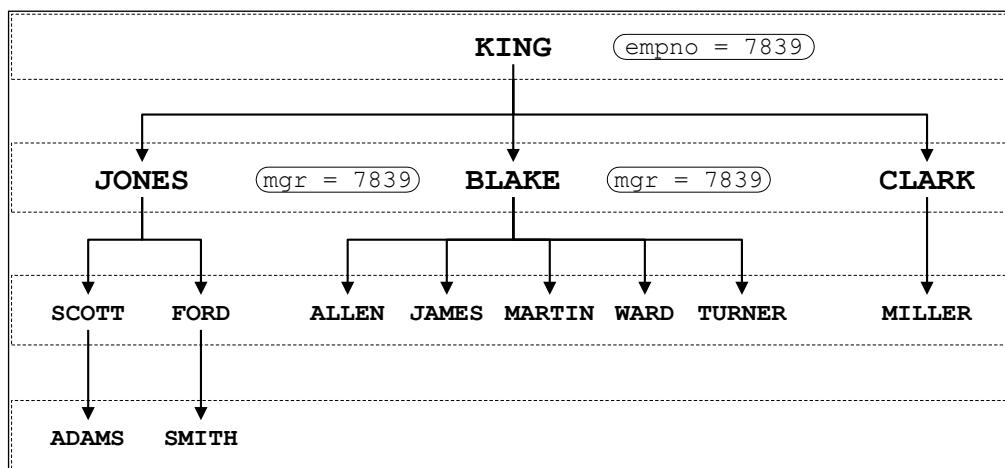
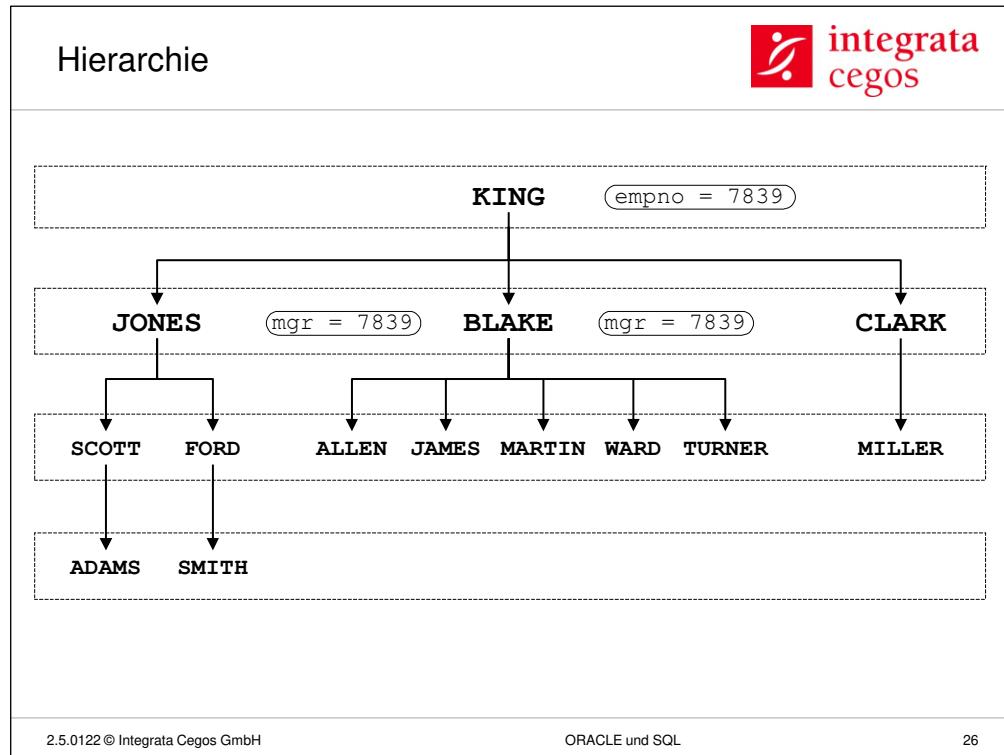


Abb.: Prinzip der Hierarchie

Für eine hierarchische Abfrage sind zwei weitere Klauseln nötig:

1. **START WITH**: legt den Anfangspunkt fest, ab dem die Hierarchie durchwandert werden soll
2. **CONNECT BY PRIOR**: spezifiziert die Beziehung zwischen Vater- und Kind-Datensätzen, wobei **PRIOR** den Vaterdatensatz kennzeichnet. Diese Klausel kann auch weitere Einschränkungen beinhalten.

LEVEL ist eine Pseudospalte, die bei hierarchischen Abfragen die Hierarchieebene angibt. Für die oberste Ebene ist **LEVEL = 1**, für die nächste Ebene ist **LEVEL = 2** usw..



Beispiel:

```
SQL> SELECT
      LEVEL,
      e.empno,
      e.ename,
      e.job,
      e.mgr
    FROM emp e
   CONNECT BY PRIOR e.empno = e.mgr
  START WITH e.mgr IS NULL;
```

LEVEL	EMPNO	ENAME	JOB	MGR
1	7839	KING	PRESIDENT	
2	7566	JONES	MANAGER	7839
3	7788	SCOTT	ANALYST	7566
4	7876	ADAMS	CLERK	7788
3	7902	FORD	ANALYST	7566
4	7369	SMITH	CLERK	7902
2	7698	BLAKE	MANAGER	7839
3	7499	ALLEN	SALESMAN	7698
3	7521	WARD	SALESMAN	7698
3	7654	MARTIN	SALESMAN	7698
3	7844	TURNER	SALESMAN	7698
3	7900	JAMES	CLERK	7698
2	7782	CLARK	MANAGER	7839
3	7934	MILLER	CLERK	7782

Das Verbindungsglied der Hierarchie ist `mgr`, die `empno` des "Vaters" ist `mgr` des "Kindes".

Die Auflistung startet mit dem Mitarbeiter ohne vorgesetzten Manager:
`START WITH e.mgr IS NULL.`

Hierarchische (rekursive) Abfragen



Nur innerhalb einer Tabelle

```
SQL> SELECT
      LEVEL,
      e.empno,
      e.ename,
      e.job,
      e.mgr
    FROM emp e
   CONNECT BY PRIOR e.empno = e.mgr
  START WITH e.mgr IS NULL;
```

LEVEL	EMPNO	ENAME	JOB	MGR
1	7839	KING	PRESIDENT	
2	7566	JONES	MANAGER	7839
3	7788	SCOTT	ANALYST	7566
4	7876	ADAMS	CLERK	7788
3	7902	FORD	ANALYST	7566
4	7369	SMITH	CLERK	7902
2	7698	BLAKE	MANAGER	7839
3	7499	ALLEN	SALESMAN	7698
3	7521	WARD	SALESMAN	7698
3	7654	MARTIN	SALESMAN	7698
3	7844	TURNER	SALESMAN	7698
3	7900	JAMES	CLERK	7698
2	7782	CLARK	MANAGER	7839
3	7934	MILLER	CLERK	7782

5.4.1 Formatieren hierarchischer Berichte

Die Ausgabe hierarchischer Abfragen kann mit Hilfe von LPAD und LEVEL so formatiert werden, dass jeder Eintrag entsprechend seiner Hierarchie-Ebene eingerückt wird.

Beispiel:

```
SQL>
SELECT
    LPAD('    ', 3 * LEVEL - 1) || e.ename AS Name,
    LEVEL,
    e.empno,
    e.mgr
FROM emp e
CONNECT BY PRIOR e.empno = e.mgr
START WITH e.mgr IS NULL;
```

NAME	LEVEL	EMPNO	MGR
KING	1	7839	
JONES	2	7566	7839
SCOTT	3	7788	7566
ADAMS	4	7876	7788
FORD	3	7902	7566
SMITH	4	7369	7902
BLAKE	2	7698	7839
ALLEN	3	7499	7698
WARD	3	7521	7698
MARTIN	3	7654	7698
TURNER	3	7844	7698
JAMES	3	7900	7698
CLARK	2	7782	7839
MILLER	3	7934	7782

14 Zeilen ausgewählt.

Hierarchische Abfragen			
			
<pre>SQL> SELECT LPAD(' ', 3 * LEVEL - 1) e.ename AS Name, LEVEL, e.empno, e.mgr FROM emp e CONNECT BY PRIOR e.empno = e.mgr AND e.ename != 'FORD' START WITH e.mgr IS NULL;</pre>			
NAME	LEVEL	EMPNO	MGR
KING	1	7839	
JONES	2	7566	7839
SCOTT	3	7788	7566
ADAMS	4	7876	7788
BLAKE	2	7698	7839
ALLEN	3	7499	7698
WARD	3	7521	7698
MARTIN	3	7654	7698
TURNER	3	7844	7698
JAMES	3	7900	7698
CLARK	2	7782	7839
MILLER	3	7934	7782

5.4.2 Einzelne Zweige ausblenden

Auch bei einer hierarchischen Abfrage kann man einzelne Datensätze durch eine WHERE-Bedingung ausblenden. Hierarchisch untergeordnete Datensätze bleiben aber erhalten.

Will man auch untergeordnete Datensätze ausblenden, so muss man die entsprechende Bedingung mit in die CONNECT BY-Klausel schreiben.

Beispiel:

```
-- blendet nur FORD selber aus (SMITH wird angezeigt)
```

```
SQL> SELECT
      LEVEL,
      e.empno,
      LPAD(' ', 3 * LEVEL - 1) || e.ename
                           AS Name
    FROM emp e
   WHERE e.ename != 'FORD'
   CONNECT BY PRIOR e.empno = e.mgr
 START WITH e.mgr IS NULL;
```

LEVEL	EMPNO	NAME
1	7839	KING
2	7566	JONES
3	7788	SCOTT
4	7876	ADAMS
4	7369	SMITH
2	7698	BLAKE
3	7499	ALLEN
3	7521	WARD
3	7654	MARTIN
3	7844	TURNER
3	7900	JAMES
2	7782	CLARK
3	7934	MILLER

13 Zeilen ausgewählt.

-- blendet FORD und seinen Untergebenen SMITH aus.

```
SQL> SELECT
      LEVEL,
      e.empno,
      LPAD(' ', 3 * LEVEL - 1) || e.ename
      AS Name
  FROM emp e
 CONNECT BY PRIOR e.empno = e.mgr
      AND e.ename != 'FORD'
 START WITH e.mgr IS NULL;
```

LEVEL	EMPNO	NAME
1	7839	KING
2	7566	JONES
3	7788	SCOTT
4	7876	ADAMS
2	7698	BLAKE
3	7499	ALLEN
3	7521	WARD
3	7654	MARTIN
3	7844	TURNER
3	7900	JAMES
2	7782	CLARK
3	7934	MILLER

12 Zeilen ausgewählt.

5.4.3 Weitere Optionen zu hierarchischen Abfragen

Hierarchische Abfragen können um:

- NOCYCLE
- CONNECT_BY_ISCYCLE
- CONNECT_BY_ISLEAF
- CONNECT_BY_ROOT

erweitert werden.

Erweiterung der Hierarchischen Abfragen



- Erweiterung von hierarchischen Abfragen:
 - NOCYCLE
- Pseudospalten:
 - CONNECT_BY_ISCYCLE (gibt es unendliche Schleife)
 - CONNECT_BY_ISLEAF (ist Blatt)
 - CONNECT_BY_ROOT (die Wurzel des Baumes)

Unendliche Schleife in der Rekursion

Falls in den Daten eine unendliche Schleife vorhanden ist, kann man diese mit **NOCYCLE** abfangen.

Schleife einbauen:

```
UPDATE    emp
SET        mgr = 7788
WHERE      empno = 7839;
```

Obige SELECT Abfrage liefert keine Ergebnisse mehr:

ERROR:

```
ORA-01436: CONNECT BY loop in user data
no rows selected
```

oder ab Oracle 10.2:

Es wurden keine Zeilen ausgewählt

Diesen Fehler kann man mit **NOCYCLE** abfangen:

```
COL „ganzer Pfad“ FOR a40;

SELECT      e.ename "MA Name",
            e.empno,
            LEVEL,
            SYS_CONNECT_BY_PATH(e.ename, '-')
                        AS "ganzer Pfad"
FROM        emp e
WHERE       LEVEL < 4
START WITH ename LIKE 'KING'
CONNECT BY NOCYCLE PRIOR e.empno = e.mgr;
```

Pseudospalten

Drei weitere Pseudospalten stehen bei hierarchischen Abfragen zur Verfügung:

- CONNECT_BY_ISCYCLE:

Zeigt ob die Zeile eine Schleife enthält.

- CONNECT_BY_ISLEAF

Zeigt ob die aktuelle Zeile ein Blatt des Rekursionsbaumes ist.

- CONNECT_BY_ROOT

Zeigt die Wurzel des Rekursionsbaumes an.

```
SELECT      e.empno,
            e.ename                      AS "MA Name",
            LEVEL,
            CONNECT_BY_ISCYCLE          AS "IsCycle",
            CONNECT_BY_ISLEAF           AS "IsLeaf",
            CONNECT_BY_ROOT e.empno     AS "Root"
FROM        emp e
START WITH  ename LIKE 'KING'
CONNECT BY NOCYCLE PRIOR e.empno = e.mgr;
```

EMPNO	MA Name	LEVEL	IsCycle	IsLeaf	Root
7839	KING	1	0	0	7839
7566	JONES	2	0	0	7839
7788	SCOTT	3	0	0	7839
7876	ADAMS	4	0	1	7839
7902	FORD	3	0	0	7839
7369	SMITH	4	0	1	7839
7698	BLAKE	2	0	0	7839
7499	ALLEN	3	0	1	7839
7521	WARD	3	0	1	7839
7654	MARTIN	3	0	1	7839
7844	TURNER	3	0	1	7839
7900	JAMES	3	0	1	7839
7782	CLARK	2	0	0	7839
7934	MILLER	3	0	1	7839

14 Zeilen ausgewählt.

5.5 SET-Operatoren

Die folgenden Operatoren verbinden die Ergebnisse von mehreren SELECT-Statements zu einer Ergebnismenge.

Abfragen, die einen dieser Operatoren verwenden, werden Compound_Abfragen genannt.

Voraussetzung ist, dass die **Anzahl** und der **Datentyp** der Spalten immer gleich sind. Es können nicht Character-Typen mit numerischen Typen gemischt werden.

Bei der Ausgabe werden die Spaltennamen der ersten Abfrage verwendet.

UNION	fügt mehrere SELECTs zu einer Ergebnismenge zusammen und eliminiert doppelte Zeilen aus beiden Mengen (entspricht der Vereinigungsmenge in der Mengenlehre). Führt einen inneren Sort durch.
UNION ALL	fügt mehrere SELECTs zu einer Ergebnismenge zusammen, ohne doppelte Zeilen zu eliminieren.
INTERSECT	gibt nur die Ergebnisse zurück, die in beiden SELECTs vorkommen (entspricht der Schnittmenge in der Mengenlehre).
MINUS	gibt die Ergebnisse aus dem ersten SELECT zurück, welche im zweiten SELECT nicht vorkommen (entspricht der Differenz zweier Mengen).

Syntax:

```
select-statement ::=

    select-spezifikation
    { [UNION [ALL] select-spezifikation]
    | [INTERSECT select-spezifikation ]
    | [MINUS      select-spezifikation] }
    [ORDER BY      orderbyausdruck];
```

SET Operatoren Syntax



Syntax:

```
select-statement ::=  
    select-spezifikation  
    {[UNION [ALL]      select-spezifikation]  
     | [INTERSECT      select-spezifikation ]  
     | [MINUS          select-spezifikation ]}  
    [ORDER BY          orderbyausdruck];
```

Beispiel:

```
-- UNION  
SQL>  SELECT e.empno,  
          e.ename  
      FROM emp e  
     WHERE e.sal < 1000  
UNION  
SELECT e2.empno,  
      e2.ename  
  FROM emp e2  
 WHERE e2.deptno = 20  
ORDER BY ename;
```

EMPNO ENAME

```
-----  
7876 ADAMS  
7902 FORD  
7900 JAMES  
7566 JONES  
7788 SCOTT  
7369 SMITH
```

```
--UNION ALL
```

```
SQL>  SELECT e.empno,  
          e.ename  
      FROM emp e  
     WHERE e.sal < 1000  
UNION ALL  
SELECT e2.empno,  
      e2.ename  
  FROM emp e2  
 WHERE e2.deptno = 20  
ORDER BY ename;
```

EMPNO ENAME

```
-----  
7876 ADAMS  
7902 FORD  
7900 JAMES  
7566 JONES  
7788 SCOTT  
7369 SMITH  
7369 SMITH
```

SET Operatoren																		
UNION																		
SELECT e.empno, e.ename FROM emp e WHERE e.sal < 1000 UNION SELECT e2.empno, e2.ename FROM emp e2 WHERE e2.deptno = 20 ORDER BY ename;		<table border="1"> <thead> <tr> <th>EMPNO</th><th>ENAME</th></tr> </thead> <tbody> <tr><td>7876</td><td>ADAMS</td></tr> <tr><td>7902</td><td>FORD</td></tr> <tr><td>7900</td><td>JAMES</td></tr> <tr><td>7566</td><td>JONES</td></tr> <tr><td>7788</td><td>SCOTT</td></tr> <tr><td>7369</td><td>SMITH</td></tr> </tbody> </table>	EMPNO	ENAME	7876	ADAMS	7902	FORD	7900	JAMES	7566	JONES	7788	SCOTT	7369	SMITH		
EMPNO	ENAME																	
7876	ADAMS																	
7902	FORD																	
7900	JAMES																	
7566	JONES																	
7788	SCOTT																	
7369	SMITH																	
UNION ALL																		
SELECT e.empno, e.ename FROM emp e WHERE e.sal < 1000 UNION ALL SELECT e2.empno, e2.ename FROM emp e2 WHERE e2.deptno = 20 ORDER BY ename;		<table border="1"> <thead> <tr> <th>EMPNO</th><th>ENAME</th></tr> </thead> <tbody> <tr><td>7876</td><td>ADAMS</td></tr> <tr><td>7902</td><td>FORD</td></tr> <tr><td>7900</td><td>JAMES</td></tr> <tr><td>7566</td><td>JONES</td></tr> <tr><td>7788</td><td>SCOTT</td></tr> <tr><td>7369</td><td>SMITH</td></tr> <tr><td>7369</td><td>SMITH</td></tr> </tbody> </table>	EMPNO	ENAME	7876	ADAMS	7902	FORD	7900	JAMES	7566	JONES	7788	SCOTT	7369	SMITH	7369	SMITH
EMPNO	ENAME																	
7876	ADAMS																	
7902	FORD																	
7900	JAMES																	
7566	JONES																	
7788	SCOTT																	
7369	SMITH																	
7369	SMITH																	
2.5.0122 © Integrata Cegos GmbH		ORACLE und SQL																
		31																

```
--INTERSECT
SQL> SELECT e.empno,
          e.ename
     FROM emp e
    WHERE e.sal < 1000
INTERSECT
SELECT e2.empno,
        e2.ename
   FROM emp e2
  WHERE e2.deptno = 20
 ORDER BY ename;
```

EMPNO	ENAME
-----	-----
7369	SMITH

```
--MINUS
SQL> SELECT e.empno,
          e.ename
     FROM emp e
    WHERE e.sal < 1000
MINUS
SELECT e2.empno,
       e2.ename
  FROM emp e2
 WHERE e2.deptno = 20
 ORDER BY ename;
```

EMPNO	ENAME
7900	JAMES

SET Operatoren						
<pre>INTERSECT SELECT e.empno, e.ename FROM emp e WHERE e.sal < 1000 INTERSECT SELECT e2.empno, e2.ename FROM emp e2 WHERE e2.deptno = 20 ORDER BY ename;</pre>						
<table border="1"> <thead> <tr> <th>EMPNO</th> <th>ENAME</th> </tr> </thead> <tbody> <tr> <td>7369</td> <td>SMITH</td> </tr> </tbody> </table>			EMPNO	ENAME	7369	SMITH
EMPNO	ENAME					
7369	SMITH					
<pre>MINUS SELECT e.empno, e.ename FROM emp e WHERE e.sal < 1000 MINUS SELECT e2.empno, e2.ename FROM emp e2 WHERE e2.deptno = 20 ORDER BY ename;</pre>						
<table border="1"> <thead> <tr> <th>EMPNO</th> <th>ENAME</th> </tr> </thead> <tbody> <tr> <td>7900</td> <td>JAMES</td> </tr> </tbody> </table>			EMPNO	ENAME	7900	JAMES
EMPNO	ENAME					
7900	JAMES					
2.5.0122 © Integrata Cegos GmbH		ORACLE und SQL				
		32				

6

Data Manipulation Language (DML)

6.1	INSERT - Datensätze einfügen.....	6-5
6.1.1	Verwendung der VALUES-Klausel:.....	6-6
6.1.2	Verwendung einer Unterabfrage:.....	6-6
6.2	UPDATE - Datensätze aktualisieren	6-8
6.3	DELETE - Datensätze löschen	6-10
6.4	Das MERGE-Statement.....	6-12
6.5	Das Transaktionskonzept	6-14
6.5.1	SAVEPOINT	6-15
6.5.2	Die SET TRANSACTION-Anweisung.....	6-16

6 Data Manipulation Language (DML)

DML-Befehle greifen schreibend auf die Datenbank zu und verändern den Inhalt bestehender Tabellen. Es gibt drei DML-Befehle:

- `INSERT` fügt neue Datensätze ein
- `UPDATE` manipuliert bestehende Felder
- `DELETE` löscht Datensätze

Während die Befehle `INSERT` und `DELETE` immer auf der Datensatzebene operieren, ist der `UPDATE` Befehl für einzelne Felder Bearbeitung zuständig.

Mit dem `INSERT` Befehl werden **ein** oder **mehrere** neue Datensätze in die Tabelle eingefügt.

Mit dem `DELETE` Befehl werden **ein** oder **mehrere** Datensätze aus der Tabelle entfernt.

Mit dem `UPDATE` Befehl wird der Inhalt der Felder eines oder mehrerer Datensätze:

- Geändert
- Gelöscht (auf `NULL` gesetzt)
- Eingefügt (von `NULL` auf anderen Wert gesetzt).

Während **DDL-** und **DCL-**Befehle im Allgemeinen **nicht mehr rückgängig** zu machen sind, ist es bei **DML-**Befehlen **noch möglich**, den alten Datenzustand **wiederherzustellen**, solange er nicht durch ein explizites oder implizites `COMMIT` festgeschrieben wurde.

Eng mit DML verknüpft ist daher der Begriff der **Transaktion**.

DML-Befehle



- #### ■ Syntax:

```
insert-statement ::=  
    insert-mitwerten | insert-mitunterabfrage
```

- #### ▪ Datensatzebene

- **INSERT** Datensätze einfügen
 - **DELETE** Datensätze löschen

- #### ■ Feldebene

- **UPDATE** Felder ändern | einfügen | löschen

- mit ROLLBACK rückgängig machen

- mit COMMIT festschreiben

2.5.0122 © Integrata Cegos GmbH

ORACLE und SQL

2

6.1 INSERT - Datensätze einfügen

INSERT fügt in eine bestehende Tabelle einen oder mehrere Sätze ein.

Hinweis: Vor dem Aufruf beachten: Besitzen Sie das CONNECT-Recht, so müssen Sie zusätzlich entweder das INSERT-Zugriffsrecht haben oder aber Tabelleneigentümer sein.

Es gibt zwei prinzipielle Möglichkeiten, Datensätze einzufügen:

1. Verwendung der VALUES-Klausel
2. Verwendung einer Unterabfrage

Syntax:

insert-statement ::=

insert-mitvalues | insert-mitunterabfrage

INSERT



- Mit VALUES Klausel:

Syntax:

```
INSERT INTO [Schema.]Tabelle [(Spalte {, Spalte})]
VALUES (Wert {, Wert });

INSERT INTO dept VALUES (60, 'Verkauf', 'Berlin');
INSERT INTO dept (dname, deptno)VALUES ('Aussendienst', 70);
```

- Mit Unterabfrage:

Syntax:

```
INSERT INTO [Schema.]Tabelle (Spalte {, Spalte})
( select-klausel );

INSERT INTO dept2(SELECT deptno, dname, loc
                  FROM dept
                  WHERE deptno > 20);
```

6.1.1 Verwendung der VALUES-Klausel:

Auf diese Art kann immer **nur ein Datensatz** eingefügt werden.

Beispiel:

```
SQL> INSERT INTO dept
      VALUES (60, 'Verkauf', 'Berlin');

SQL> INSERT INTO dept (dname, deptno)
      VALUES ('Aussendienst', 70);
```

Syntax:

```
insert-mitvalues ::=

    INSERT INTO [Schema.]Tabelle
        [ (Spalte {, Spalte} ) ]
    VALUES (Wert {, Wert} );
```

6.1.2 Verwendung einer Unterabfrage:

Auf diese Art können **mehrere Datensätze** gleichzeitig eingefügt werden (**mengenorientiert**).

Beispiel: (vorausgesetzt, es gibt eine dept2 Tabelle)

```
SQL> INSERT INTO dept2
      ( SELECT      deptno, dname, loc
        FROM        dept
        WHERE       deptno > 20);
```

Syntax:

```
insert-mitunterabfrage ::=

    INSERT INTO [Schema.]Tabelle
        [ (Spalte {, Spalte} ) ]
    (select-klausel);
```

Die Angabe von Spaltennamen ist optional. Wenn die angesprochenen Spalten nicht angegeben werden, wird die physikalische Reihenfolge der Spalten in der Tabelle benutzt, und die Werteliste muss vollständig sein. Soll in eine Spalte nichts eingetragen werden, dann muss explizit NULL angegeben werden.

Die einzufügenden Werte in der Werteliste müssen der **Reihenfolge** und dem **Datentyp** der Spaltenliste entsprechen. Durch Verwendung von Spaltennamen ist es möglich, für einzelne Spalten in der Werteliste keine Werte anzugeben. In dem Fall werden für die nicht aufgeführten Spalten Annahmewerte (Default-Werte) eingesetzt, soweit sie vorhanden sind, ansonsten wird NULL eingetragen.

Für Spalten mit Datentyp character (CHAR, VARCHAR2, ...) und DATE müssen die Werte in einfache Anführungszeichen ('text') eingeschlossen werden.

6.2 UPDATE - Datensätze aktualisieren

UPDATE ändert Spaltenwerte von Sätzen in einer Tabelle.

Hinweis: Vor dem Aufruf beachten: Besitzen Sie das CONNECT-Recht, so müssen Sie zusätzlich entweder das UPDATE-Recht für die entsprechenden Spalten haben oder Tabelleneigentümer sein.

Fehlt die WHERE-Bedingung, so werden **alle** Datensätze der Tabelle aktualisiert, ansonsten nur diejenige(n) Zeile(n), die der Bedingung entspricht/entsprechen.

In der SET-Klausel ist auch eine Unterabfrage möglich. Anzahl und Datentyp der Spalten in der Unterabfrage müssen dabei denjenigen in der SET-Klausel entsprechen.

Es ist auch möglich, den angegebenen Standardwert einer Spalte explizit zu referenzieren. Dazu muss das Schlüsselwort DEFAULT im SQL-Statement verwendet werden. Dies ist beispielsweise der Fall, wenn Sie einen Spaltenwert auf den Standardwert aktualisieren wollen.

Beispiel:

```
SQL> UPDATE emp
      SET comm = 0
      WHERE
            comm IS NULL;
```

Syntax:

```
update-statement ::=

    UPDATE Tabelle
        SET Spalte = update-ausdruck
            { , Spalte = update-ausdruck }
        [WHERE bedingungen];
```

```
update-ausdruck ::=

    DEFAULT | Wert | (select-spezifikation)
```

Beispiel (mehrere Werte gleichzeitig ändern):

```
SQL> UPDATE emp
      SET comm = comm + (sal * 0.1),
          sal = sal + 100
     WHERE comm IS NOT NULL;
```

Beispiel (mehrere Werte als TUPEL gleichzeitig ändern):

```
SQL> UPDATE emp
      SET (sal, comm) =
        (SELECT avg(e2.sal),
               avg(e2.comm)
        FROM emp e2
       )
     WHERE comm IS NULL;
```

UPDATE	 integrata cegos
<ul style="list-style-type: none"> ▪ <u>Syntax:</u> <pre>update-statement ::= UPDATE Tabelle SET Spalte = update-ausdruck [, Spalte = update-ausdruck] [WHERE bedingungen]; update-ausdruck ::= DEFAULT Wert (select-spezifikation)</pre>	
<ul style="list-style-type: none"> ▪ <u>Beispiele:</u> <pre>UPDATE emp SET comm = 0 WHERE comm IS NULL; UPDATE emp SET comm = comm + (sal * 0.1), sal = sal + 100 WHERE comm IS NOT NULL; UPDATE emp SET (sal, comm) = (SELECT avg(e2.sal), avg(e2.comm) FROM emp e2) WHERE comm IS NULL;</pre>	
<small>2.5.0122 © Integrata Cegos GmbH ORACLE und SQL 4</small>	

6.3 DELETE - Datensätze löschen

Mit der `DELETE`-Anweisung können Sätze aus einer Tabelle gelöscht werden.

Hinweis: Vor dem Aufruf zu beachten: Besitzen Sie das `CONNECT`-Zugriffsrecht, so müssen Sie zusätzlich entweder das `DELETE`-Zugriffsrecht haben oder aber Tabelleneigentümer sein.

Beispiel:

```
SQL> DELETE FROM dept  
      WHERE  
            deptno = 40;
```

Syntax:

```
delete-statement ::=  
    DELETE [FROM] [Schema.] Tabelle  
    [WHERE bedingungen];
```

Fehlt die `WHERE`-Bedingung, so werden **alle Datensätze der Tabelle gelöscht!** Die Tabelle als solche bleibt (leer) bestehen, und der Speicherplatz wird nicht freigegeben.

Wird eine `WHERE`-Bedingung angegeben, so werden alle Datensätze gelöscht, die dieser Bedingung entsprechen. In der `WHERE`-Bedingung sind auch Unterabfragen möglich.

DELETE



- Syntax:

```
delete-statement ::=  
    DELETE [FROM] [Schema.] Tabelle  
    [WHERE bedingungen];
```

- Beispiele:

```
DELETE FROM dept;  
  
DELETE FROM dept  
WHERE deptno = 40;  
  
DELETE FROM emp  
WHERE deptno = 10;
```

6.4 Das MERGE-Statement

Oracle bietet auch die Möglichkeit, INSERT- und UPDATE-Anweisungen zu kombinieren. Dies kann hilfreich sein, wenn in eine Hilfstabelle ständig neue Daten eingespielt werden, anhand derer Einträge in Tabellen zum Teil ergänzt, zum Teil aber auch geändert werden müssen.

Der MERGE-Befehl ermöglicht es, beide Vorgänge in einem Schritt durchzuführen.

Seit Version 10g ist der MERGE-Befehl noch um die Einsatzmöglichkeit des DELETE-Statements erweitert worden.

Beispiel:

```
SQL> MERGE INTO emp2 e2
      USING emp e
      ON (e2.empno = e.empno)
      WHEN MATCHED THEN
          UPDATE SET e2.sal = e2.sal * 1.1
          DELETE WHERE (e2.sal < 2500)
      WHEN NOT MATCHED THEN
          INSERT (empno, ename, sal)
          VALUES (e.empno, e.ename, e.sal * 0.9);
```

Syntax:

```
merge-statement ::=

    MERGE INTO [Schema.]Tabelle [Alias]
        USING [Schema.]Tabelle [Alias]
        ON (bedingung-on)
        WHEN MATCHED THEN
            update-statement
            [ delete-statement ]
        WHEN NOT MATCHED THEN
            insert-statement;
```

MERGE



- Syntax:

```
merge-statement ::=  
    MERGE INTO [Schema.] Tabelle [Alias]  
        USING [Schema.] Tabelle [Alias]  
        ON (bedingung-on)  
    WHEN MATCHED THEN  
        update-statement  
        [ delete-statement ]  
    WHEN NOT MATCHED THEN  
        insert-statement;
```

- Beispiel:

```
MERGE INTO emp2 e2  
    USING emp e  
    ON (e2.empno = e.empno)  
    WHEN MATCHED THEN  
        UPDATE SET e2.sal = e2.sal * 1.1  
        DELETE WHERE (e2.sal < 2500)  
    WHEN NOT MATCHED THEN  
        INSERT (empno, ename, sal)  
        VALUES (e.empno, e.ename, e.sal * 0.9);
```

6.5 Das Transaktionskonzept

Eine Transaktion beginnt normalerweise mit der Ausführung einer DML-Anweisung. Sie kann auf zweierlei Weise beendet werden:

1. Entweder alle Anweisungen seit Beginn der Transaktion werden permanent gemacht, oder
2. alle Anweisungen seit Beginn der Transaktion werden zurückgerollt.

Der erste Fall tritt auf, wenn explizit der Befehl

COMMIT;

eingegeben wird oder ein implizites COMMIT stattfindet.

Ein **implizites COMMIT** wird ausgelöst durch:

- jeden DDL-Befehl (auch wenn er fehlschlägt)
- jeden DCL-Befehl (auch wenn er fehlschlägt)
- ein ordnungsgemäßes Verlassen von SQL*Plus mit EXIT.

Der zweite Fall tritt auf, wenn explizit der Befehl

ROLLBACK;

eingegeben wird oder ein implizites ROLLBACK stattfindet.

Ein **implizites ROLLBACK** wird ausgelöst, wenn sonst die Konsistenz der Datenbank nicht mehr gewährleistet wäre. Dazu gehören Systemabstürze, sonstige schwerere Fehler oder ein nicht ordnungsgemäßes Verlassen von SQL*Plus.

```
SQL> UPDATE emp SET sal = 10;  
SQL> ROLLBACK;  
SQL> UPDATE emp SET comm = 100;  
SQL> COMMIT;
```

6.5.1 SAVEPOINT

Ein einzelner DML-Befehl innerhalb einer Transaktion kann nicht zurückgerollt werden. Man kann aber zwischendurch Haltepunkte setzen durch den Befehl

SAVEPOINT *Savepointname*;

In diesem Fall besteht die Möglichkeit, nur bis zu diesem SAVEPOINT zurückzurollen durch den Befehl

SQL> ROLLBACK TO SAVEPOINT *Savepointname*;

Nach Beendigung einer Transaktion werden alle von ihr genutzten Resourcen wieder freigegeben und alle Sperren aufgehoben.

```
SQL> UPDATE emp SET sal = 10;
SQL> ROLLBACK;
SQL> UPDATE emp SET comm = 100;
SQL> SAVEPOINT punkt1
SQL> UPDATE emp SET sal = sal * 1.1;
SQL> ROLLBACK TO SAVEPOINT punkt1;
SQL> COMMIT; -- comm = 100
SQL> ROLLBACK; -- bringt nichts mehr
```

Transaktionskonzept



- **Explizit vs. Implizit**
- **ROLLBACK**
- **COMMIT**
- **SAVEPOINT**

```
SQL> UPDATE emp SET sal = 10;
SQL> ROLLBACK;

SQL> UPDATE emp SET comm = 100;
SQL> SAVEPOINT punkt1;

SQL> UPDATE emp SET sal = sal * 1.1;
SQL> ROLLBACK TO SAVEPOINT punkt1;

SQL> COMMIT; -- comm = 100
SQL> ROLLBACK; -- bringt nichts mehr
```

6.5.2 Die SET TRANSACTION-Anweisung

SET TRANSACTION leitet explizit eine Transaktion ein. Diese Anweisung ist nur als erster Befehl zulässig, nicht innerhalb einer "pending", also laufenden Transaktion.

Mit SET TRANSACTION kann festgelegt werden:

- es findet nur Lesezugriff oder Lese- und Schreibzugriff statt
- Isolierungsstufe der Transaktion
- Verwendung eines bestimmten Rollback-Segments

SET TRANSACTION READ ONLY:

Nur SELECT ist erlaubt, sowie die Befehle LOCK TABLE, SET ROLE, ALTER SESSION und ALTER SYSTEM. Der Unterschied zum normalen Lesezugriff besteht darin, dass alle Änderungen, die während der laufenden Transaktion von anderen Usern gemacht wurden, nicht gesehen werden, auch dann nicht, wenn sie mit COMMIT abgeschlossen wurden. Dies kann Vorteile haben bei mehreren, länger dauernden Abfragen, die in sich konsistent sein sollen.

SET TRANSACTION READ WRITE:

Setzt explizit den Default-Status einer Transaktion: Lesen und Schreiben ist erlaubt, alle Änderungen anderer User werden gesehen, sobald sie durch COMMIT permanent gemacht wurden. (vgl. READ COMMITTED).

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE:

DML-Befehle innerhalb der Transaktion sind erlaubt, aber auf Datensätze beschränkt, die seit Beginn der Transaktion nicht von anderen Usern geändert wurden (es reicht, wenn nur das COMMIT erst nach Beginn der Transaktion erfolgt war).

SET TRANSACTION ISOLATION LEVEL READ COMMITTED:

Entspricht dem Default der ORACLE-Transaktionsverarbeitung. Jede Änderung wird akzeptiert und durchgeführt, sobald eine eventuelle Sperre des Datensatzes durch andere Benutzer aufgehoben ist.

Diese Einstellungen gelten für genau eine Transaktion, also bis zum nächsten expliziten oder impliziten COMMIT.

SET TRANSACTION



- Einstellungen gelten nur für eine einzige Transaktion
- Nur am Anfang der Transaktion möglich
- SET TRANSACTION READ ONLY:
 - Nur Lesezugriff, innerhalb der Transaktion konsistenter Zustand der Datenbank
 - Commitete Änderungen anderer User sind vor Transaktionsende nicht sichtbar
- SET TRANSACTION READ WRITE:
 - Commitete Änderungen anderer User sind innerhalb der Transaktion sichtbar
 - Default
- SET TRANSACTION ISOLATION LEVEL SERIALIZABLE:
 - Datensätze, die durch erst nach Beginn der Transaktion abgeschlossene fremde Transaktionen verändert wurden, sind nicht für DML-Befehle zugänglich
- SET TRANSACTION ISOLATION LEVEL READ COMMITTED:
 - Default

7

Data Definition Language (DDL)

7.1	Objekt TABLE	7-5
7.1.1	Der CREATE TABLE Befehl	7-5
7.1.2	CREATE TABLE und unsichtbare Spalten	7-6
7.1.3	Erstellung einer neuen Tabelle aus einer bestehenden Tabelle	7-8
7.1.4	CREATE TABLE – Neuerungen	7-10
7.1.5	CREATE TABLE – Neuerungen	7-11
7.1.6	Der ALTER TABLE Befehl	7-12
7.1.7	ALTER TABLE – Neuerungen	7-14
7.1.8	Der DROP TABLE Befehl	7-15
7.1.9	Der TRUNCATE TABLE Befehl	7-16
7.1.10	Der RENAME Befehl	7-18
7.1.11	Umbenennung von Spalten und Constraints	7-19
7.1.12	Spaltenlänge ändern	7-20
7.2	Objekt INDEX	7-24
7.2.1	Der CREATE INDEX Befehl	7-26
7.2.2	Der ALTER INDEX Befehl	7-27
7.2.3	Neuerungen ab 12c bezüglich Indizes	7-28
7.2.4	Der DROP INDEX Befehl	7-29
7.3	Objekt VIEW	7-30
7.3.1	Der CREATE VIEW Befehl	7-32

7.3.2	Views und DML-Befehle	7-34
7.3.3	Inline Views.....	7-35
7.3.4	Der ALTER VIEW Befehl	7-36
7.3.5	Der DROP VIEW Befehl.....	7-37
7.3.6	Der RENAME Befehl.....	7-37
7.3.7	Top-n / Bottom-n-Analysen	7-38
7.3.8	Zeilen limitieren.....	7-40
7.4	Objekt SEQUENCE	7-42
7.4.1	Der CREATE SEQUENCE Befehl.....	7-42
7.4.2	Neue Parameter	7-44
7.4.3	Pseudospalten der SEQUENCE.....	7-45
7.4.4	Der ALTER SEQUENCE Befehl.....	7-47
7.4.5	Der DROP SEQUENCE Befehl.....	7-48
7.4.6	Der RENAME Befehl.....	7-50
7.4.7	Die IDENTITY Klausel	7-51
7.5	Objekt SYNONYM	7-54
7.5.1	Der CREATE SYNONYM Befehl.....	7-54
7.5.2	Ändern eines SYONYMs.....	7-55
7.5.3	Der DROP SYNONYM Befehl	7-55
7.5.4	Der RENAME Befehl.....	7-56
7.6	Constraints	7-57
7.6.1	Der NOT NULL-Constraint	7-64
7.6.2	Der UNIQUE-Constraint.....	7-64
7.6.3	Der PRIMARY KEY Constraint	7-64
7.6.4	Der FOREIGN KEY Constraint.....	7-65
7.6.5	Der CHECK Constraint.....	7-67
7.6.6	Constraints hinzufügen und löschen	7-67
7.6.7	Constraintstatus ändern.....	7-68
7.6.8	Kaskadierende Constraints.....	7-71

7 Data Definition Language (DDL)

Die Datendefinitionssprache unter ORACLE umfasst die Befehle, die zum Anlegen einer logischen Datenbankstruktur verwendet werden (Datenbank Wartung). Dazu gehört die Verwaltung von Tabellen, Views, Synonymen, Clustern und Indizes, die unter dem Begriff **Datenbankobjekte** zusammengefasst werden.

Zu den DDL-Befehlen gehören:

- **CREATE** (erstellt ein Datenbankobjekt)
- **ALTER** (ändert ein Datenbankobjekt)
- **DROP** (löscht ein Datenbankobjekt)
- **TRUNCATE** (leert Tabellen oder Cluster)
- **RENAME** (benennt Objekte um)

Oracle führt vor und nach jedem DDL-Befehl ein implizites COMMIT aus.

Bei der Namensvergabe für ein Objekt sind folgende Regeln zu beachten:

- Maximal 30 Zeichen für Objektname. Ab Version 12.2 können 128 Zeichen verwendet werden.
- Keine reservierten SQL oder PL/SQL Wörter (SELECT, ALTER, ...)
- Keine Zahl oder Sonderzeichen am Anfang (007_Bond ist nicht erlaubt, Bond_007 ist zulässig)
- Nur folgende Sonderzeichen können verwendet werden: "_", "\$", "#"
- Objektname muss innerhalb des Schemas eindeutig sein

Data Definition Language – DDL



- CREATE Objekte erstellen
- ALTER Objekte ändern
- RENAME Objekte umbenennen
- DROP Objekte löschen
- TRUNCATE Tabelleninhalt schnell löschen

2.5.0122 © Integrata Cegos GmbH

ORACLE und SQL

2

DDL



- Regeln für Namensvergabe:
 - Maximal 30 Zeichen für Objektname (ab 12.2: 128 Zeichen)
 - Keine reservierten SQL oder PL/SQL Wörter
(SELECT, ALTER, ...)
 - Keine Zahl oder Sonderzeichen am Anfang
(007_Bond ist nicht erlaubt, Bond_007 ist zulässig)
 - Nur folgende Sonderzeichen:
 - "_",
 - "\$",
 - "#"
 - Objektname innerhalb des Schema eindeutig

2.5.0122 © Integrata Cegos GmbH

ORACLE und SQL

3

7.1 Objekt TABLE

7.1.1 Der CREATE TABLE Befehl

Um eine Tabelle erstellen zu können, braucht man das Privileg CREATE TABLE.

Beim Erstellen einer neuen Tabelle müssen angegeben werden:

- Tabelle
- Spaltennamen und
- Datentyp der Spalten.

Alle weiteren Angaben wie Constraints, Default-Werte und Speicherverwaltungsangaben sind optional. (Eine vollständige Syntax des Befehls findet sich in der Online Dokumentation zu ORACLE und ist bei Oracle 12c ca. 24 Seiten lang.)

Beispiel:

```
SQL> CREATE TABLE test
      (nr      NUMBER (5) NOT NULL,
       name    VARCHAR2 (10) DEFAULT 'Schmidt',
       geburtstag DATE);
```

Syntax:

```
create-table ::=*
  create-table-aus-SELECT | create-table-direkt

create-table-direkt ::=*
  CREATE [GLOBAL TEMPORARY] TABLE
    [Schema.]Tabelle (
      Spalte spalteneigenschaft
      {, Spalte spalteneigenschaft }
      {, constraint-zeile });

spalteneigenschaft ::=*
  datentyp [DEFAULT Wert] {constraint-spalte}

datentyp ::=*
  NUMBER | BINARY_FLOAT | BINARY_DOUBLE | DATE |
  CHAR | VARCHAR2 | NCHAR | NVARCHAR | NVARCHAR2 |
  LONG | LOB | BLOB | CLOB | NCLOB | BFILE | RAW |
  LONG RAW | INTERVAL YEAR TO MONTH | INTERVAL DAY
  TO SECOND | TIMESTAMP | TIMESTAMP WITH TIME ZONE
  | TIMESTAMP WITH LOCAL TIME ZONE
```

CREATE TABLE	
<ul style="list-style-type: none"> ▪ <u>Syntax:</u> <pre>create-table ::= create-table-aus-SELECT create-table-direkt create-table-direkt ::= CREATE [GLOBAL TEMPORARY] TABLE [Schema.]Tabelle (Spalte spalteneigenschaft { , Spalte spalteneigenschaft } { , constraint-zeile }); spalteneigenschaft ::= datentyp [DEFAULT Wert] {constraint-spalte} datentyp ::= NUMBER BINARY_FLOAT BINARY_DOUBLE DATE CHAR VARCHAR2 NCHAR NVARCHAR NVARCHAR2 LONG LOB BLOB CLOB NCLOB BFILE RAW LONG RAW INTERVAL YEAR TO MONTH INTERVAL DAY TO SECOND TIMESTAMP TIMESTAMP WITH TIME ZONE TIMESTAMP WITH LOCAL TIME ZONE</pre> 	
<ul style="list-style-type: none"> ▪ <u>Beispiel:</u> <pre>SQL> CREATE TABLE test (nr NUMBER (5) NOT NULL, name VARCHAR2 (40) DEFAULT 'Schmidt', geburtstag DATE);</pre>	
<small>2.5.0122 © Integrata Cegos GmbH</small> ORACLE und SQL	
<small>4</small>	

7.1.2 CREATE TABLE und unsichtbare Spalten

Neuerung der Version 12c

Spalten können nun auf "Invisible" gesetzt werden:

```
CREATE TABLE t (c1 INT, c2 INT INVISIBLE, c3 INT);
```

Nur c1 und c3 werden ausgegeben

Spalte wieder auf sichtbar setzen

```
ALTER TABLE t MODIFY (c2 VISIBLE);
```

Hinweis: Dadurch wird die Spalte als letzte angezeigt, aber intern an der "Original-Position" weiter vorgehalten

Spalte im Nachhinein auf unsichtbar setzen

```
ALTER TABLE t MODIFY (c2 INVISIBLE);
```

CREATE TABLE



- Mit der Einführung der Version 12c können Spalten nun auf "Invisible" gesetzt werden:
 - `CREATE TABLE t (c1 INT, c2 INT INVISIBLE, c3 INT);`
- Nur c1 und c3 werden ausgegeben
- Spalte wieder auf sichtbar setzen
 - `ALTER TABLE t MODIFY (c2 VISIBLE);`
- Hinweis: Dadurch wird die Spalte als letzte angezeigt
 - Aber intern an der "Original-Position" weiter vorgehalten
- Spalte im Nachhinein auf unsichtbar setzen
 - `ALTER TABLE t MODIFY (c2 INVISIBLE);`

2.5.0122 © Integrata Cegos GmbH

ORACLE und SQL

5

CREATE TABLE



- Wenn Sie eine Spalte auf invisible setzen wird deren Column_ID auf Null gesetzt
- `ALTER TABLE emp MODIFY (mgr INVISIBLE);`
- `ALTER TABLE emp MODIFY (mgr VISIBLE);`

OWNER	TABLE_NAME	COLUMN_NAME	COLUMN_ID	INTERNAL_COLUMN_ID
SCOTT	EMP	DEPTNO	7	8
SCOTT	EMP	COMM	6	7
SCOTT	EMP	SAL	5	6
SCOTT	EMP	HIREDATE	4	5
SCOTT	EMP	MGR	(null)	4
SCOTT	EMP	JOB	3	3
SCOTT	EMP	ENAME	2	2
SCOTT	EMP	EMPNO	1	1

```
SELECT owner,table_name,column_name,
column_id,internal_column_id
FROM dba_tab_cols
WHERE owner='SCOTT'
AND table_name='EMP';
```

OWNER	TABLE_NAME	COLUMN_NAME	COLUMN_ID	INTERNAL_COLUMN_ID
SCOTT	EMP	DEPTNO	7	8
SCOTT	EMP	COMM	6	7
SCOTT	EMP	SAL	5	6
SCOTT	EMP	HIREDATE	4	5
SCOTT	EMP	MGR	8	4
SCOTT	EMP	JOB	3	3
SCOTT	EMP	ENAME	2	2
SCOTT	EMP	EMPNO	1	1

2.5.0122 © Integrata Cegos GmbH

ORACLE und SQL

6

7.1.3 Erstellung einer neuen Tabelle aus einer bestehenden Tabelle

Es besteht auch die Möglichkeit, eine Tabelle anhand einer bereits bestehenden Tabelle zu erstellen und auch mit ersten Werten zu füllen.

Die Spaltenliste der neuen Tabelle ist optional. Falls sie nicht angegeben wird, übernimmt die neue Tabelle die Spaltennamen der alten Tabelle. Bei der expliziten Angabe der Spaltenliste der neuen Tabelle muss die Anzahl der in der Tabelle zu erstellenden Spalten mit der Anzahl der Spalten der Unterabfrage übereinstimmen. Ein Datentyp *darf nicht* angegeben werden, weil er aus den selektierten Spalten übernommen wird.

Die WHERE-Bedingung in der Unterabfrage entscheidet, welche Datensätze eingefügt werden.

Beispiel:

```
SQL> CREATE TABLE emp_verdienst_manager AS
      (SELECT e.empno, e.ename, e.sal
       FROM   emp e
       WHERE  e.job LIKE 'MANAGER');
```

```
SQL> CREATE TABLE emp_struktur_kopie AS
      (SELECT e.empno, e.ename, e.sal
       FROM   emp e
       WHERE  1 = 2);
```

Syntax:

```
create-table-aus-SELECT ::=

    CREATE [GLOBAL TEMPORARY] TABLE
        [Schema.] Tabelle
        [ (Spalte {, Spalte} ) ]
    AS ( select-spezifikation ) ;
```

<p>CREATE TABLE aus bereits vorhandenen Tabelle</p> <ul style="list-style-type: none"> ▪ Syntax: <pre>create-table-aus-SELECT ::= CREATE [GLOBAL TEMPORARY] TABLE [Schema.] Tabelle [(Spalte {, Spalte})] AS (select-spezifikation) ;</pre> <ul style="list-style-type: none"> ▪ Beispiel: <pre>SQL> CREATE TABLE emp_verdienst_manager AS (SELECT e.empno, e.ename, e.sal FROM emp e WHERE e.job LIKE 'MANAGER'); SQL> CREATE TABLE emp_struktur_kopie AS (SELECT e.empno, e.ename, e.sal FROM emp e WHERE 1 = 2);</pre>	
2.5.0122 © Integrata Cegos GmbH	ORACLE und SQL

7.1.3.1 Globale temporäre Tabellen

Es besteht die Möglichkeit, durch den Befehl:

```
SQL> CREATE GLOBAL TEMPORARY TABLE t12 ...
```

globale temporäre Tabellen anzulegen.

Diese Tabellen sind von überall aus sichtbar. Die Datensätze sind nur für die Sitzung sichtbar, welche sie eingefügt hat. Sonstige spezielle Tabellenfeatures wie Indexierung, Partitionierung, Clustering sind hier nicht verfügbar.

7.1.3.2 NULL-Wert in Tabellenspalten

NULL-Werte können für eine Spalte einer Tabelle verboten werden, indem bei der Spaltendefinition NOT NULL angeben wird. Werden NULL-Werte bei der Definition nicht ausgeschlossen, kann die Spalte den NULL-Wert annehmen.

Wird beim Einfügen eines Datensatzes (INSERT) für eine Spalte, welche NULL-Werte enthalten darf, kein Wert angegeben, trägt die Datenbank automatisch NULL ein.

NULL-Werte sollten in Spalten, die UNIQUE sind, nicht zugelassen werden, da Oracle NULL-Werte bei UNIQUE Indizes nicht als Duplicate Values betrachtet und daher beliebig viele NULL-Einträge zulässt.

NOT NULL ist der einzige von fünf Constraints, der nur auf Spaltenebene definiert werden kann.

7.1.4 CREATE TABLE – Neuerungen

Die Default-Klausel kann nun die Sequenzattribute NEXTVAL und CURRVAL benutzen, um damit standardmäßig Sequenzwerte in die Tabelle zu laden.

Voraussetzungen für die Nutzung dieser Pseudospalten als Spalten-Default sind, dass die gewünschte Sequenz bereits existiert und der Benutzer das SELECT-Recht auf die Sequenz besitzt.

Die Pseudospalten NEXTVAL und CURRVAL können somit beispielsweise auch für das automatische Laden von Master-Detail-Beziehungen in eine Tabelle genutzt werden.

Beispiel:

```
CREATE SEQUENCE master_seq;
CREATE SEQUENCE detail_seq;

CREATE TABLE master (
    id          NUMBER DEFAULT master_seq.NEXTVAL,
    description VARCHAR2(30)
);

CREATE TABLE detail (
    id          NUMBER DEFAULT detail_seq.NEXTVAL,
    master_id   NUMBER DEFAULT master_seq.CURRVAL,
    description VARCHAR2(30)
);

INSERT INTO master (description) VALUES ('Master 1');
INSERT INTO detail (description) VALUES ('Detail 1');
INSERT INTO detail (description) VALUES ('Detail 2');

INSERT INTO master (description) VALUES ('Master 2');
INSERT INTO detail (description) VALUES ('Detail 3');
INSERT INTO detail (description) VALUES ('Detail 4');

SELECT * FROM master;

      ID DESCRIPTION
----- 
      1 Master 1
      2 Master 2
```

CREATE TABLE – Neuerungen	
<ul style="list-style-type: none"> ▪ DEFAULT Klauseln können Sequenzattribute NEXTVAL und CURRVAL benutzen ▪ Voraussetzung: <ul style="list-style-type: none"> ▪ Sequence muss bereits existieren ▪ SELECT Recht auf Sequenz, um diese als Default-Spalte nutzen zu können ▪ <u>Beispiel:</u> <ul style="list-style-type: none"> ▪ CREATE SEQUENCE s; ▪ CREATE TABLE t (<ul style="list-style-type: none"> id NUMBER DEFAULT s.nextval, name VARCHAR2(99));) ▪ INSERT INTO t (name) <ul style="list-style-type: none"> VALUES ('Marco');) ▪ INSERT INTO t (name) <ul style="list-style-type: none"> VALUES ('Hans');) 	

2.5.0122 © Integrata Cegos GmbH

ORACLE und SQL

8

7.1.5 CREATE TABLE – Neuerungen

Die ADD COL Klausel wurde in der Version 12c für Spalten mit NULL-Werten, die eine Default-Klausel bekommen sollen, beschleunigt.

Die DEFAULTS ON NULL Klausel ersetzt Werte, die bei explizitem Übergeben von NULL-Werten sonst NULL wären.

CREATE TABLE – Neuerungen	
<ul style="list-style-type: none"> ▪ ADD COL Klausel wurde beschleunigt <ul style="list-style-type: none"> ▪ für Spalten mit NULL-Werten, die eine Default-Klausel bekommen sollen ▪ DEFAULTS ON NULL: ersetzt Werte die bei explizitem Übergeben von NULL-Werten sonst NULL wären <ul style="list-style-type: none"> ▪ <code>CREATE TABLE t (id NUMBER DEFAULT ON NULL 1, name VARCHAR2(99));</code> ▪ <code>INSERT INTO t (id,name) VALUES (null,'Marco');</code> 	

2.4.1019 © Integrata Cegos GmbH

ORACLE und SQL

9

7.1.6 Der ALTER TABLE Befehl

Mit dem ALTER TABLE Befehl kann die Definition einer schon bestehenden Tabelle verändert werden. Zum Verändern benötigt man die Besitzrechte, das ALTER-Privileg oder DBA-Recht.

Die ALTER TABLE-Anweisung enthält als zweite Klausel:

- **ADD**, fügt eine Spalte (oder einen Constraint) hinzu
- **MODIFY**, ändert eine vorhandene Spalte
- **DROP**, löscht eine vorhandene Spalte oder einen Constraint

Beispiel:

```
SQL> ALTER TABLE DEPT
      ADD abteilungsleiter NUMBER;
SQL> ALTER TABLE dept
      MODIFY abteilungsleiter VARCHAR2(10);
SQL> ALTER TABLE dept
      DROP COLUMN abteilungsleiter;
```

Syntax:

```
alter-table ::=

    altertab-add | altertab-modify | altertab-drop |
    altertab-rename

altertab-add ::=

    ALTER TABLE [Schema.]Tabelle
        ADD (Spalte spalteneigenschaft
              {, Spalte spalteneigenschaft}) ;

altertab-modify ::=

    ALTER TABLE [Schema.]Tabelle
        MODIFY (Spalte spalteneigenschaft) ;

altertab-drop ::=

    ALTER TABLE [Schema.]Tabelle
        DROP COLUMN Spalte [CASCADE CONSTRAINTS] ;
```

ALTER TABLE	
<p>▪ Syntax:</p> <pre>alter-table ::= altertab-add altertab-modify altertab-drop altertab-rename altered-add ::= ALTER TABLE [Schema.] Tabelle ADD (Spalte spalteneigenschaft { , Spalte spalteneigenschaft }); altered-modify ::= ALTER TABLE [Schema.] Tabelle MODIFY (Spalte spalteneigenschaft); altered-drop ::= ALTER TABLE [Schema.] Tabelle DROP COLUMN Spalte [CASCADE CONSTRAINTS];</pre>	
<p>▪ Beispiel:</p> <pre>SQL> ALTER TABLE DEPT ADD abteilungsleiter NUMBER; SQL> ALTER TABLE dept MODIFY abteilungsleiter VARCHAR2(10); SQL> ALTER TABLE dept DROP COLUMN abteilungsleiter;</pre>	

Mit `MODIFY` geändert werden können *prinzipiell*:

- NOT NULL-Constraints
 - Datentyp
 - Größe und
 - Default-Werte
- einer Spalte.

Es ist immer möglich, den Default-Wert zu verändern. Dies wirkt sich aber erst auf alle folgenden Einträge aus.

Auch eine Vergrößerung der Spalte ist immer möglich.

Eine Verkleinerung einer Spalte ist möglich, solange keine Datenverluste auftreten. Die Änderung des Datentyps ist nur möglich, solange die Spalte noch keine Werte enthält.

Eine NOT NULL-Spalte zu einer NULL-Spalte zu machen, ist auch problemlos möglich. Umgekehrt geht es nur, wenn die Spalte für jeden bereits vorhandenen Datensatz einen von `NULL` verschiedenen Wert enthält.

Es ist möglich, Spalten wieder zu löschen oder sie alternativ auf `UNUSED` zu setzen. Das geht schneller, gibt aber den Festplattenspeicher nicht frei. Auf `UNUSED`-Spalten kann nicht mehr zugegriffen werden, aber sie können pauschal gelöscht werden.

Eine vollständigere Syntax des `ALTER TABLE` Befehls findet sich im Anhang.

7.1.7 ALTER TABLE – Neuerungen

Mit der Syntax:

```
CREATE TABLE .. [NO] ROW ARCHIVAL
```

kann die Spaltenarchivierung aktiviert werden.

Dabei wird in der entsprechenden Tabelle die unsichtbar Spalte ORA_ARCHIVE_STATE erstellt.

Für alle bereits in der Tabelle enthaltenen Datensätze wird diese Spalte auf 0 gesetzt. Diese Zeilen können nachträglich anhand des Updates der Spalte ORA_ARCHIVE_STATE=1 auf archiviert gesetzt werden.

Das nachträglich An- bzw. Abschalten des Row Archival kann wie folgt vorgenommen werden:

```
ALTER TABLE .. [NO] ROW ARCHIVAL;
```

Einschränkungen: Existiert die Spalte ORA_ARCHIVE_STATE bereits in der Tabelle, kann für diese keine ROW ARCHIVAL Klausel verwendet werden. Für den Benutzer SYS kann keine NO ROW ARCHIVAL Klausel genutzt werden.

Ab 12.2 können bei einer Tabellen-Reorg auch die dazugehörigen Indizes aktualisiert werden:

```
ALTER TABLE scott.emp MOVE ONLINE
  STORAGE ( INITIAL 128K NEXT 64K)
  TABLESPACE users UPDATE INDEXES;
```

ALTER TABLE	 integrata cegos
<ul style="list-style-type: none"> ▪ <u>Syntax:</u> <ul style="list-style-type: none"> ▪ <code>CREATE TABLE .. [NO] ROW ARCHIVAL</code> ▪ <u>Beispiel:</u> <ul style="list-style-type: none"> ▪ <code>CREATE TABLE t (id NUMBER, text VARCHAR2(10)) ROW ARCHIVAL;</code> ▪ <code>INSERT INTO t VALUES (1,'X');</code> ▪ <u>Zeile auf archiviert setzen:</u> <ul style="list-style-type: none"> ▪ <code>UPDATE t SET ora_archive_state=1 WHERE id=1</code> ▪ <u>nachträgliches Abschalten des ROW ARCHIVAL</u> <ul style="list-style-type: none"> ▪ <code>ALTER TABLE t NO ROW ARCHIVAL;</code> ▪ <u>nachträgliches Anschalten des ROW ARCHIVAL</u> <ul style="list-style-type: none"> ▪ <code>ALTER TABLE t ROW ARCHIVAL;</code> 	

7.1.8 Der `DROP TABLE` Befehl

Mit Hilfe des `DROP TABLE`-Befehls lassen sich die Definitionen, die mit dem `CREATE TABLE`-Befehl festgelegt wurden, löschen.

Um eine Tabelle in einem fremden Schema löschen zu können, muss der Benutzer das `DROP ANY TABLE`-Recht besitzen.

Beispiel:

```
SQL> DROP TABLE emp;  
SQL> DROP TABLE dept CASCADE CONSTRAINTS;  
SQL> DROP TABLE emp PURGE;
```

Syntax:

`drop-table ::=`

```
    DROP TABLE [Schema.]Tabelle  
    [ CASCADE CONSTRAINTS ] [ PURGE ];
```

`DROP TABLE ... CASCADE CONSTRAINTS` löscht alle **referenzierenden Constraints** anderer Tabellen, die auf Primärschlüssel der gelöschten Tabelle verweisen. Wird diese Option weggelassen, obwohl Verweise von anderen Tabellen existieren, so reagiert ORACLE mit einer Fehlermeldung.

`DROP TABLE ... PURGE` löscht die Tabelle sofort aus dem Recycle Bin. Somit kann diese Tabelle nicht mehr mit evtl. möglichen `FLASHBACK`-Befehlen (DB-Admin) zurückgeholt werden.

Beim Löschen einer Tabelle werden intern folgende Operationen durchgeführt:

- alle Zeilen werden gelöscht.
- alle Indizes werden gelöscht (unabhängig davon, wer sie angelegt hat).
- alle Datenblöcke, die von der Tabelle benutzt wurden, werden wieder freigegeben.
- wird die Tabelle in Views, Prozeduren etc. verwendet, so werden diese Objekte als ungültig markiert, jedoch **nicht** gelöscht.
- die Tabellendefinition wird aus dem Data Dictionary gelöscht.

Hinweis: Ein `DROP TABLE`-Befehl und alle anderen `DROP`-Befehle werden ohne Rückfrage ausgeführt! Nach einem `DROP`-Befehl kann das jeweilige Objekt nicht wieder zurückgeholt sondern nur neu erstellt werden!!!

DROP TABLE	
<ul style="list-style-type: none"> ▪ <u>Syntax:</u> <pre>drop-table ::= DROP TABLE [Schema.] Tabelle [CASCADE CONSTRAINTS] [PURGE];</pre> 	
<ul style="list-style-type: none"> ▪ <u>Beispiel:</u> <pre>SQL> DROP TABLE emp; SQL> DROP TABLE dept CASCADE CONSTRAINTS; SQL> DROP TABLE emp PURGE;</pre> ▪ Nach einem DROP ist kein ROLLBACK möglich ▪ Jedoch mit FLASHBACK möglich, falls nicht PURGE angegeben 	
<small>2.5.0122 © Integrata Cegos GmbH</small>	<small>ORACLE und SQL</small>

12

7.1.9 Der TRUNCATE TABLE Befehl

Dieser DDL-Befehl steht nur für Tabellen zur Verfügung.

Er löscht **sämtliche Datensätze** der Tabelle, **nicht** jedoch die **Tabellenstruktur**, ebenso wie der Befehl

`DELETE FROM Tabelle;`

Im Gegensatz zu `DELETE` ist aber bei `TRUNCATE` keine WHERE-Bedingung möglich, und die Anweisung kann **nicht zurückgerollt** werden. Ein weiterer Unterschied zu `DELETE` besteht darin, dass `TRUNCATE` den Speicherplatz freigibt, der von der Tabelle benutzt wurde.

Vorteil des `TRUNCATE TABLE` Befehls ist seine Geschwindigkeit. Er leert jede Tabelle rasend schnell.

Beispiel:

`SQL> TRUNCATE TABLE dept;`

Syntax:

`truncate-table ::=`

`TRUNCATE TABLE [Schema.] Tabelle;`

TRUNCATE TABLE



- Nur auf Tabellen anwendbar
- Löscht ALLE Datensätze, aber nicht die Struktur
- Kein ROLLBACK
- Keine WHERE-Klausel
- Gibt Speicherplatz frei
- Rasend schnell

- Beispiel:
`SQL> TRUNCATE TABLE dept;`

- Syntax:
`truncate-table ::=
 TRUNCATE TABLE [Schema.] Tabelle;`

2.5.0122 © Integrata Cegos GmbH

ORACLE und SQL

13

Mit der Bereitstellung der Oracle Version 12c können nun auch Child-Tabellen rekursiv gelöscht. Dazu wurde die Option CASCADE im Zusammenspiel mit TRUNCATE eingeführt.

TRUNCATE TABLE <tabellen_name> CASCADE;

TRUNCATE TABLE



Die neue Option CASCADE erlaubt ab Version 12c recursive auch die Child-Tabellen zu löschen.

`TRUNCATE TABLE dept CASCADE;`

→ löscht Zeilen der DEPT (Parent) und EMP (Child) Tabelle

→ TRUNCATE war bis 11.2 nur möglich, wenn Child-Tabellen bereits leer waren

2.5.0122 © Integrata Cegos GmbH

ORACLE und SQL

14

7.1.10 Der RENAME Befehl

Mit diesem DDL-Befehl können Tabellen umbenannt werden.

Nur der Eigentümer einer Tabelle kann die Tabelle umbenennen.

Beispiel:

```
SQL> RENAME emp TO emp_neu;
```

Syntax:

```
rename ::=  
        RENAME Alter-name TO Neuer-name;
```

RENAME



- Umbenennung von Tabellen

- Syntax:

```
rename ::=  
        RENAME Alter-name TO Neuer-name;
```

- Beispiel:

```
SQL> RENAME emp TO emp_neu;
```

7.1.11 Umbenennung von Spalten und Constraints

Spaltennamen und Constraints können seit der Version 9.2 über den ALTER TABLE Befehl umbenannt werden:

Beispiel:

```
SQL> ALTER TABLE emp  
      RENAME COLUMN comm TO provision;
```

Syntax:

altertab-rename ::=

```
ALTER TABLE Tabelle RENAME  
          COLUMN | CONSTRAINT  
          Alter-name TO Neuer-name;
```

Umbenennung von Spalten und Constraints



- Syntax:

```
altertab-rename ::=  
ALTER TABLE Tabelle RENAME  
          COLUMN | CONSTRAINT  
          Alter-name TO Neuer-name;
```

- Beispiel:

```
SQL> ALTER TABLE emp  
      RENAME COLUMN comm TO provision;
```

7.1.12 Spaltenlänge ändern

Mit Einführung von Oracle 12c ist es nun möglich, die Spaltenlänge auf 32K zu erweitern.

Spaltenlänge ändern		
<ul style="list-style-type: none"> ▪ Ab Version 12.1 kann für Varchar2, NVARCHAR2 und Raw die Länge 32767 Bytes (nicht Zeichen!) verwendet werden ▪ Intern verwendet die Out-of-Line DB LOBs (Lob Segmente + Lob Index) ▪ Dazu muss die DB einmalig umgestellt werden 		
<p>Hinweis:</p> <p>Function Based Indizes werden ungültig, wenn Sie auf einer Spalte basieren, die länger sein kann als 87% von einen Index Blocks</p>		
<small>2.5.0122 © Integrata Cegos GmbH</small>	<small>ORACLE und SQL</small>	<small>17</small>

7.1.12.1 Prozedur zur Umstellung der Spaltenlänge

Umstellung einer Datenbank auf neue Spaltenlänge		
<ol style="list-style-type: none"> 1. Schließen Sie die Datenbank. <ul style="list-style-type: none"> ▪ <code>shutdown immediate</code> 2. DB Starten im Upgrade Modus <ul style="list-style-type: none"> ▪ <code>startup upgrade</code> 3. Einstellung von MAX_STRING_SIZE ändern <ul style="list-style-type: none"> ▪ <code>ALTER SYSTEM SET MAX_STRING_SIZE =extended;</code> 4. Starten Sie als Skript <ul style="list-style-type: none"> ▪ <code>@?/rdbms/admin/utl32k.sql</code> 5. DB Neu starten <ul style="list-style-type: none"> ▪ <code>shutdown immediate</code> ▪ <code>startup</code> 		
<small>2.5.0122 © Integrata Cegos GmbH</small>	<small>ORACLE und SQL</small>	<small>18</small>

7.1.12.2 Prozedur zur Umstellung der Spaltenlänge bei einer Container Datenbank

Umstellung einer CDB (Container Datenbank) auf neue Spaltenlänge	
<ol style="list-style-type: none">1. Datenbank schließen <code>shutdown immediate</code>2. DB Starten im Upgrade Modus <code>startup upgrade</code>3. Einstellung von MAX_SQL_STRING_SIZE ändern (default:standard) <code>ALTER SYSTEM SET MAX_STRING_SIZE =extended;</code>4. Skript im Root Container und allen PDB (Pluggable Datenbank) in der der CDB (Container Datenbank) starten → (Alternativ können Sie auch das Perl-Skript catcon.pl ausführen) <code>@?/rdbms/admin/utl32k.sql</code>5. DB Neu starten <code>shutdown immediate</code> <code>startup</code>	

7.1.12.3 Benutzen der neuen Spaltenlänge

Die neue Spaltenlänge wird über den CREATE TABLE Befehl mit angegeben.

Beispiel:

```
CREATE TABLE t (
  id NUMBER,
  text VARCHAR2(32767));
```

Nutzung der neuen Spaltenlänge



- Beispiel:
 - `CREATE TABLE t (`
 `id NUMBER,`
 `text VARCHAR2(32767));`
 - `INSERT INTO t (text) VALUES(lpad('#',32000,'x'));`
 - `SELECT length(text) FROM t;`
 `=> 32000`

7.1.12.4 Konsequenzen aus der Umstellung auf die neue Spaltenlänge

Konsequenzen aus der Umstellung der Spaltenlänge	
<ul style="list-style-type: none">▪ Intern verwendet Oracle die CLOB Technik um die VARCHAR2 Länge von 4k-32K darzustellen.▪ Folgende Einschränkungen sind dabei auffällig:<ul style="list-style-type: none">▪ <code>SELECT * FROM t_with_clob_col UNION SELECT * FROM t_with_clob_col;</code>▪ Ein Index auf einer 32K Spalte:<ul style="list-style-type: none">▪ <code>CREATE INDEX i_text_ix ON t(text);</code>▪ Endet mit einem<ul style="list-style-type: none">▪ ORA-01450: Maximale Schlüssellänge (6398) überschritten	

7.2 Objekt INDEX

Ein Index ist ein eigenes Datenbankobjekt, das bei der **Suche** nach bestimmten Einträgen in einer Tabelle den Festplattenzugriff reduziert. Bei anderen Operationen ist ein Index eher hinderlich!

Ein Index wird vom Oracle-Server automatisch und komplett unabhängig von der indizierten Tabelle verwendet und verwaltet.

Hinweis: Die interne Verwaltung eines Indexes ist **sehr teuer!!!**

Einen Index sollte man nur erstellen, wenn es sich um eine große Tabelle handelt und die meisten Abfragen nur ca. 2-5 % der Zeilen zurückliefern, sonst kann ein Index die Performance unter Umständen sogar verschlechtern.

Ein Index sollte auf Spalten gelegt werden, die

- häufig in WHERE-Klauseln referenziert werden
- häufig in der JOIN-Bedingungen (ON-Klausel) verwendet werden
- viele unterschiedliche Werte beinhalten
(Ausnahme: Bitmap Index)
- eine hohe Anzahl von NULL-Werten beinhalten

Ein Index kann maximal 32 Spalten enthalten. Auf einer Tabelle können beliebig viele Indizes angelegt werden, die sich in mindestens einer Spalte unterscheiden müssen.

Ein Index wird **automatisch** für jede PRIMARY KEY- und UNIQUE-Spalte erstellt. Er kann aber auch manuell erstellt werden.

INDEX	
<ul style="list-style-type: none">▪ Beschleunigung der <u>SUCHE</u> vs. Verlangsamung der Schreiboperationen▪ Automatische Verwaltung durch ORACLE▪ Verwaltung eines INDEX ist <u>sehr teuer</u>▪ Vorteilhaft bei:<ul style="list-style-type: none">▪ Großen Tabellen▪ Lieferung eines kleinen Ausschnitts des Datenbestandes (bis zu 2-5%)▪ JOIN▪ WHERE▪ Vielen unterschiedlichen Werten in einer Spalte▪ Hohem Anteil von NULL-Werten▪ Maximal 32 Spalten pro Index▪ Beliebig viele Indizes pro Tabelle▪ Automatische INDEX-Erstellung für PRIMARY KEY und UNIQUE Spalten	
2.5.0122 © Integrata Cegos GmbH	ORACLE und SQL

7.2.1 Der CREATE INDEX Befehl

Angegeben werden müssen: Indexname, Tabelle sowie Spaltenname(n). Weitere Angaben sind optional. (Eine vollständigere Syntax des Befehls findet sich im Anhang.)

Wenn die WHERE-Klausel statt einer Spalte eine Funktion auf eine Spalte enthält, wird ein normaler Index nicht verwendet. Es gibt aber die Möglichkeit, **funktionsbasierte Indizes** zu erstellen, die dann für die angegebene Funktion wirksam werden.

Beispiel:

```
SQL> CREATE INDEX idx_emp_sal
      ON emp (sal);
SQL> CREATE INDEX idx_emp_sal_comm
      ON emp (sal, comm);
SQL> CREATE INDEX idx_emp_UP_ename
      ON emp (UPPER(ename));
```

Syntax:

```
create-index ::=

    CREATE [UNIQUE | BITMAP] INDEX
    [ Schema.] Index
        ON Tabelle ([Spalte ASC | DESC
                     {, Spalte ASC | DESC} ] );
```

CREATE INDEX	 integrata cegos
<ul style="list-style-type: none"> ▪ <u>Syntax:</u> <pre>create-index ::= CREATE [UNIQUE BITMAP] INDEX [Schema.] Index ON Tabelle ([Spalte ASC DESC { , Spalte ASC DESC }]);</pre>	
<ul style="list-style-type: none"> ▪ <u>Beispiel:</u> <pre>SQL> CREATE INDEX idx_emp_sal ON emp (sal); SQL> CREATE INDEX idx_emp_sal_comm ON emp (sal, comm); SQL> CREATE INDEX idx_emp_UP_ename ON emp (UPPER(ename));</pre>	

7.2.2 Der ALTER INDEX Befehl

Über ALTER INDEX können Speicherparameter und Transaktionseinträge zur Performanceverbesserung verändert werden. Einzelheiten finden sich in dem Syntaxdiagramm im Anhang.

Mit den Parametern `ENABLE` und `DISABLE` können Indizes ein- und ausgeschaltet werden. Dies ist z.B. bei langen Schreiboperationen von Vorteil.

Mit dem Parameter `UNUSABLE` kann ein Index sofort ausgeschaltet werden. Ein ausgeschalteter Index kann nicht mehr eingeschaltet werden, sondern nur gelöscht und bei Bedarf neu erzeugt werden. Dies ist notwendig, wenn keine Rechenzeit auf dem DB-Server zur Verfügung steht um einen Index sofort zu entfernen.

Syntax:

```
alter-index ::=
```

```
    ALTER INDEX [Schema.]Index
    alterind-rename [ ENABLE | DISABLE | UNUSABLE ];
```

```
alterind-rename ::=
```

```
    RENAME name_alt TO name_neu;
```

Änderungen des INDEXes



- Syntax:

```
alter-index ::=

    ALTER INDEX [Schema.]Index
    alterind-rename [ ENABLE | DISABLE | UNUSABLE ];
```

- Ändern von:

- Speicherparameter und
- Transaktionseinträge zur Performanceverbesserung

- Umbenennen eines INDEXes

```
alterind-rename ::=

    RENAME name_alt TO name_neu;
```

7.2.3 Neuerungen ab 12c bezüglich Indizes

Mithilfe des INDEX COALESCE CLEANUP Befehls können verwaiste Index-Einträge ohne Bezug zu Tabelleneinträgen gelöscht werden.

Ob ein verwaister Index-Eintrag vorhanden ist, kann der Spalte ORPHANED_ENTRIES aus der Data Dictionary View USER_ | DBA_ | ALL_INDEXES entnommen werden.

Zudem sind ab der Version 12c mehrere Indizes auf der gleichen Spalte erlaubt, solange die Indizes folgende Voraussetzungen erfüllen:

- sie sind nicht vom gleichen Typ
- sie verwenden verschiedene Partitionstypen
- sie verwenden verschiedene Uniqueness-Eigenschaften
- nur einer der Indizes ist auf "VISIBLE" gesetzt

Diese Neuerung vereinfacht die Vorgehensweise beim Wechsel von einem Index zu einem anderen, z.B. wenn von einem B-Tree zu einem Bitmap-Index gewechselt werden soll, dahingehend, dass kein neuer Index erstellt werden, sondern der gewünschte Index lediglich auf VISIBLE gesetzt werden muss.

<p style="margin: 0;">Index – Neuerungen in 12c</p> <ul style="list-style-type: none"> ▪ INDEX COALESCE CLEANUP: löscht verwaiste Index-Einträge ohne Bezug zu Tabelleneintrag <ul style="list-style-type: none"> ▪ <code>ALTER INDEX scott.pk_emp COALESCE CLEANUP;</code> ▪ mehrere Indizes auf gleicher Spalte sind erlaubt <ul style="list-style-type: none"> ▪ Voraussetzungen: <ul style="list-style-type: none"> ▪ Indizes sind nicht vom gleichen Typ ▪ Indizes verwenden verschiedene Partitionstypen ▪ Indizes verwenden verschiedene Uniqueness Eigenschaften ▪ Nur ein Index ist auf "VISIBLE" gesetzt 	
<p>2.5.0122 © Integrata Cegos GmbH ORACLE und SQL 25</p>	

7.2.4 Der `DROP INDEX` Befehl

Mit Hilfe des `DROP INDEX` Befehls lassen sich Indizes, die mit dem `CREATE INDEX` Befehl angelegt wurden, löschen.

Um einen Index in einem fremden Schema löschen zu können, muss der Benutzer das `DROP ANY INDEX`-Recht besitzen.

Wurde der Index gelöscht, so werden alle zugeordneten Datenblöcke des entsprechenden Tablespaces wieder freigegeben.

Beispiel:

```
SQL> DROP INDEX idx_emp_sal;
```

Syntax:

`drop-index ::=`

```
    DROP INDEX [Schema.]Index;
```

DROP INDEX



- Syntax:

```
drop-index ::=  
    DROP INDEX [Schema.]Index;
```

- Beispiel:

```
SQL> DROP INDEX idx_emp_sal;
```

7.3 Objekt VIEW

Eine View (Abfrage, Sicht, Query) ist eine **virtuelle Tabelle**, die ihre Inhalte aus bestehenden Tabellen oder anderen Views holt. Die Besonderheit der View besteht darin, dass lediglich die SELECT-Abfrage gespeichert wird, welche die View definiert. **Die Daten sind nur in den zugrundeliegenden Tabellen vorhanden.** Eine View kann daher ohne Datenverlust gelöscht werden.

Views ermöglichen eine differenziertere Rechtevergabe und erleichtern komplexe Abfragen. Unterschiedlichen Mitarbeitergruppen kann eine unterschiedliche Sicht der Daten zugänglich gemacht werden.

Der Benutzer muss die zugrundeliegende(n) Tabelle(n) besitzen oder mindestens SELECT-Rechte besitzen, um eine View anzulegen. Auf die View selbst kann in analoger Weise über SELECT zugegriffen werden wie auf eine normale Tabelle.

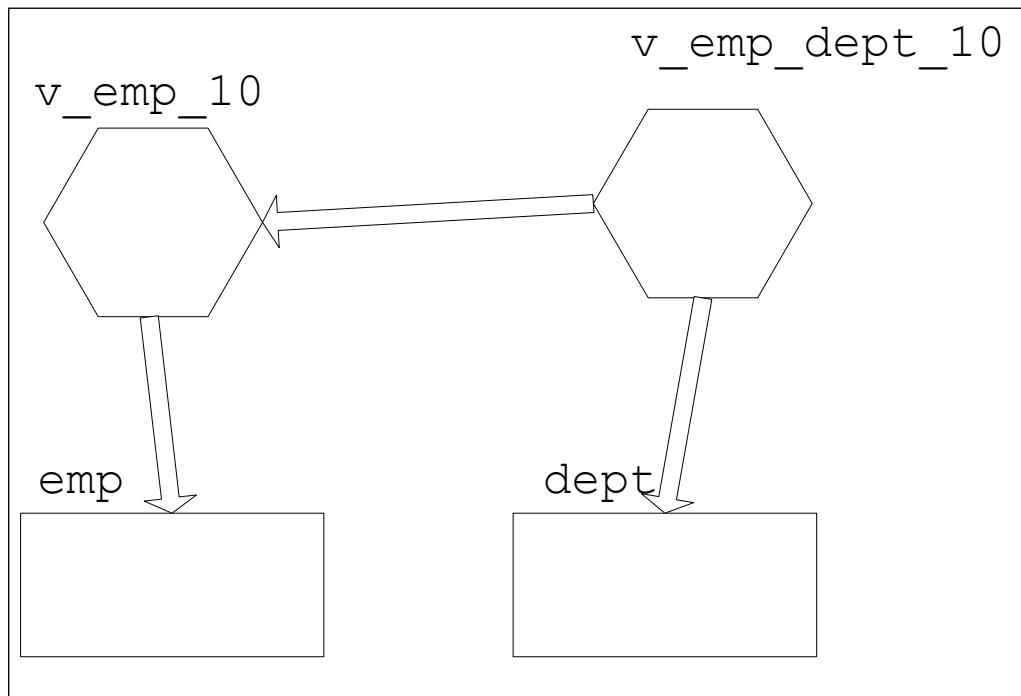
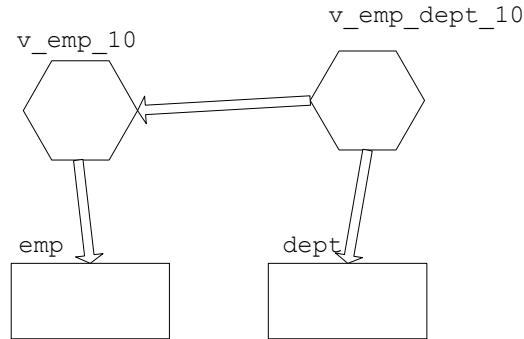


Abb.: View Konzept

VIEW (Abfrage, Sicht, Query)



- Virtuelle Tabelle
- Speichert keine Daten
- Filtert die Daten im Moment des Aufrufs



7.3.1 Der CREATE VIEW Befehl

Eine **einfache View** enthält nur Spalten aus *einer* Tabelle und keine Funktionen jeglicher Art.

Eine **komplexe View** enthält Spalten aus einer oder mehreren Tabellen, Views, arithmetische Operationen, Single Row Funktionen und / oder Gruppenfunktionen.

Syntax:

create-view ::=

```
CREATE [OR REPLACE] [[NO] FORCE] VIEW
[Schema.] View
    [ (spaltenalias {, spaltenalias} ) ]
AS select-klausel
[view-optionen]
;
```

view-optionen ::=

```
WITH
    READ ONLY |
    CHECK OPTION [ CONSTRAINT constraint ]
```

OR REPLACE Wenn es bereits eine **VIEW** dieses Namens gibt, wird sie ersetzt durch die neu erschaffene View. Der Vorteil gegenüber einem Löschen und neuem Anlegen liegt darin, dass die Rechte auf die View erhalten bleiben.

FORCE Die View wird auch dann erstellt, wenn die zugrunde liegende Tabelle noch nicht existiert. Default ist **NO FORCE**.

WITH READ ONLY

Auf die View ist nur lesender Zugriff gestattet.

WITH CHECK OPTION

Die Einhaltung der in der **SELECT**-Anweisung definierten Bedingung (**WHERE**-Klausel) wird bei einem schreibenden Zugriff überprüft. Sätze, welche die Bedingung nicht erfüllen, werden abgewiesen. Ist die Option nicht angegeben, können auch Sätze eingegeben werden, welche die Bedingung nicht erfüllen. Auf Diese kann jedoch anschließend nicht mehr über die View zugegriffen werden.

CREATE VIEW



- **Syntax:**

```
create-view ::=  
    CREATE [OR REPLACE] [ [ NO ] FORCE] VIEW  
    [Schema.]View  
        [[ spaltenalias { , spaltenalias } ] ]  
    AS select-klausel  
    [ view-optionen ]  
;  
  
view-optionen ::=  
    WITH  
        READ ONLY |  
        CHECK OPTION [ CONSTRAINT constraint ]
```
- **Beispiel:**

```
SQL> CREATE VIEW v_emp10 AS  
      (   SELECT e.*  
          FROM emp e  
         WHERE e.deptno = 10);
```

2.5.0122 © Integrata Cegos GmbH ORACLE und SQL 28

Beispiel:

```
SQL> CREATE VIEW v_emp10 AS  
      (   SELECT e.*  
          FROM emp e  
         WHERE e.deptno = 10);
```

Viewbenutzung:

```
SQL> SELECT *  
      FROM v_emp10;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7782	CLARK	MANAGER	7839	09.06.81	2450		10
7839	KING	PRESIDENT		17.11.81	5000		10
7934	MILLER	CLERK	7782	23.01.82	1300		10

3 Zeilen ausgewählt.

7.3.2 Views und DML-Befehle

DML-Operationen über eine einfache View sind in der Regel möglich, über eine komplexe View normalerweise nicht.

DML-Befehle sind nicht möglich, wenn die View folgendes enthält:

- Gruppenfunktionen
- GROUP BY-Klausel
- das Schlüsselwort DISTINCT
- die Pseudospalte ROWNUM
- SET-Operatoren

Ein UPDATE oder INSERT ist außerdem nicht möglich, wenn Spalten in der View durch Single Row Funktionen oder algebraische Ausdrücke definiert sind.

Ein INSERT ist auch nicht möglich, wenn die Basistabelle NOT NULL-Spalten enthält, die nicht in der View enthalten sind.

In allen anderen Fällen können die entsprechenden Änderungen in den Basistabellen über eine View vorgenommen werden, soweit dies nicht durch WITH READ ONLY explizit untersagt wurde.

View und DML Befehle	
<ul style="list-style-type: none"> ▪ DML Befehle möglich nur bei einfachen Views ▪ DML Befehle nicht möglich bei Verwendung von: <ul style="list-style-type: none"> ▪ Gruppenfunktionen ▪ GROUP BY-Klausel ▪ des Schlüsselworts DISTINCT ▪ der Pseudospalte ROWNUM ▪ SET-Operatoren ▪ Single Row Funktionen ▪ Algebraischen Ausdrücken ▪ WITH READ ONLY View ▪ INSERT zusätzlich nicht möglich bei Views mit fehlenden NOT NULL Spalten der Basistabelle. 	

7.3.3 Inline Views

Eine in der `FROM`-Klausel statt einer Tabelle verwendete Unterabfrage wird eine "**Inline View**" genannt. Die Inline View hat meist einen Aliasnamen. Sie ist kein eigenständiges Schemaobjekt.

Beispiel:

```
SQL> SELECT e2.Name
      FROM (SELECT e.ename Name, e.sal
            FROM emp e) e2;
```

Syntax:

```
SELECT * | ...
FROM quelle {[ , quelle ] | [JOIN quelle]}
```

`quelle` ::=

```
quellenname [ quellenalias ] |
select-statement quellenalias
```

Inline View <ul style="list-style-type: none"> ▪ Eine in der <code>FROM</code> Klausel verwendete Unterabfrage ▪ <u>Syntax:</u> <pre>quelle ::= quellenname [quellenalias] select-statement quellenalias</pre> ▪ <u>Beispiel:</u> <pre>SQL> SELECT e2.Name FROM (SELECT e.ename Name, e.sal FROM emp e) e2; SQL> SELECT ROWNUM, e2.ename, e2.sal FROM (SELECT e.ename, e.sal FROM emp e ORDER BY e.sal DESC) e2 WHERE ROWNUM <= 3;</pre> 	
<small>2.5.0122 © Integrata Cegos GmbH</small> <small>ORACLE und SQL</small> <small>30</small>	

7.3.4 Der ALTER VIEW Befehl

Eine View kann **nicht** wirklich geändert werden. Die View muss gelöscht und neu erzeugt oder mit REPLACE neu erzeugt werden.

Um einen ALTER VIEW-Befehl in einem fremden Schema ausführen zu können, muss der Benutzer das ALTER ANY TABLE-Recht besitzen.

Folgender Parameter kann verändert werden:

COMPILE mit diesem Befehl wird die View neu compiliert.

Wurde eine der Basistabellen der View verändert, so ermöglicht die neue Kompilierung eine Korrektur der View, um Laufzeitfehler zu vermeiden. Zusätzlich wird sichergestellt, dass die Veränderung der Basistabelle (n) keine View-Veränderungen mit sich gebracht hat.

Wenn bei der Kompilierung keine Fehler aufgetreten sind, hat die Änderung der Basistabellen keine Folgefehler bei der View hervorgerufen. Traten Kompilierungsfehler als Folge des Befehls auf, so muss die View entsprechend geändert werden.

Hinweis: Mit der ALTER VIEW-Anweisung können keine Views verändert werden. Um eine View zu ändern, muss die OR REPLACE-Option im CREATE VIEW-Befehl verwendet werden.

Beispiel:

```
SQL> ALTER VIEW v_emp10 COMPILE;
```

Syntax:

```
alter-view ::=  
ALTER VIEW [Schema.]View COMPILE;
```

7.3.5 Der `DROP VIEW` Befehl

Mit Hilfe des `DROP VIEW` Befehls lassen sich Views löschen.

Um eine View in einem fremden Schema löschen zu können, muss der Benutzer die `DROP ANY VIEW`-Rechte besitzen.

Wird eine View gelöscht, so werden weitere Views oder Prozeduren, die sich auf die gelöschte View beziehen, nicht gelöscht, jedoch als ungültig markiert.

Beispiel:

```
SQL> DROP VIEW v_emp10;
```

Syntax:

```
drop-view ::=
```

```
    DROP VIEW [Schema.]View  
        [ CASCADE CONSTRAINTS ] ;
```

7.3.6 Der `RENAME` Befehl

Mit diesem DDL-Befehl können Views umbenannt werden.

Nur der Eigentümer einer View kann die View umbenennen.

Beispiel:

```
SQL> RENAME v_emp10 TO v_emp10_neu;
```

Syntax:

```
rename ::=
```

```
    RENAME Alter-name TO Neuer-name;
```

<p>Änderungen von Views</p> <ul style="list-style-type: none"> ▪ ALTER VIEW: <ul style="list-style-type: none"> ▪ Beispiel: SQL>ALTER VIEW v_emp10 COMPILE; ▪ Syntax: <code>alter-view ::= ALTER VIEW [Schema.]View COMPILE;</code> ▪ DROP VIEW: <ul style="list-style-type: none"> ▪ Beispiel: SQL>DROP VIEW v_emp10; ▪ Syntax: <code>drop-view ::= DROP VIEW [Schema.]View [CASCADE CONSTRAINTS] ;</code> ▪ RENAME: <ul style="list-style-type: none"> ▪ Beispiel: SQL>RENAME v_emp10 TO v_emp10_neu; ▪ Syntax: <code>rename ::= RENAME Alter-name TO Neuer-name;</code> 	
2.5.0122 © Integrata Cegos GmbH ORACLE und SQL 31	

7.3.7 Top-n / Bottom-n-Analysen

Mit Hilfe einer Inline View mit ORDER BY-Klausel ist es möglich, die obersten oder untersten n Werte einer Tabelle zu ermitteln, wenn in der Hauptabfrage die Pseudospalte ROWNUM mit aufgeführt wird. Das bezeichnet man als **Top-n- bzw. Bottom-n-Analyse**.

Beispiel:

```
-- die drei Top-Verdiener der Firma ermitteln
SQL> SELECT ROWNUM, e2.ename, e2.sal
   FROM
     (SELECT e.ename, e.sal
      FROM emp e
     ORDER BY e.sal DESC
    ) e2
   WHERE ROWNUM <= 3;
```

ROWNUM	ENAME	SAL
-----	-----	-----
1	KING	5000
2	SCOTT	3000
3	FORD	3000

Top-n / Bottom-n-Analysen



- ermittelt die größten oder kleinsten Werte einer Spalte

- gehört zu den Inline-Views

- Syntax:

```
▪ SELECT [ROWNUM,] spaltenliste
      FROM (SELECT [spaltenliste]
              FROM tabelle
              ORDER BY top-N_spalte)
        WHERE ROWNUM <= n;
```

- Beispiel:

- Welche 4 Mitarbeiter wurden als letztes eingestellt?

Top-n / Bottom-n-Analysen



- Beispiel:

- Gesucht sind Namen und Gehalt der fünf Topverdiener aus der Tabelle emp:

```
▪ SELECT ROWNUM as rang, ename, gehalt
      FROM (SELECT ename, sal AS gehalt
              FROM emp
              ORDER BY gehalt desc)
        WHERE ROWNUM <= 5;
```

RANG	ENAME	GEHALT
1	KING	5000
2	FORD	3000
3	SCOTT	3000
4	JONES	2975
5	BLAKE	2850

7.3.8 Zeilen limitieren

Die `OFFSET` Klausel ermöglicht es, n Zeilen eines Ergebnissets zu überspringen und erst die darauffolgenden Zeilen auszugeben.

Die in der Klausel angegebene Zahl muss 0 oder positiv sein. Ist sie größer als die Anzahl der Zeilen des zugrundeliegenden Datensets, werden keine Zeilen ausgegeben.

Die `FETCH FIRST` Klausel kann mit der `OFFSET` Klausel kombiniert werden und limitiert die Anzahl der im Ergebnis zurückgegebenen Zeilen ein.

Die `FETCH FIRST` Klausel kann dafür genutzt werden, nur wenige Datensätze einer großen Datenmenge anzuzeigen und wird normalerweise mit der `ORDER BY` Klausel verwendet.

Die in der `FETCH FIRST` Klausel angegebene Zahl muss 1 oder höher sein. Der Defaultwert ist 1.

Zeilen limitieren	 integrata cegos
<ul style="list-style-type: none"> ▪ Syntax (nach ORDER BY): <ul style="list-style-type: none"> ▪ <code>OFFSET <offset> { ROW ROWS } []</code> ▪ <code>[FETCH { FIRST NEXT } [{ <rowcount> <percent> PERCENT }]</code> ▪ <code>{ ROW ROWS } { ONLY WITH TIES}</code> ▪ prozentuale Angaben: <ul style="list-style-type: none"> ▪ <code>SELECT ename, sal</code> ▪ <code>FROM scott.emp</code> ▪ <code>ORDER BY sal desc</code> ▪ <code>FETCH FIRST 5 PERCENT ROWS ONLY;</code> 	
2.5.0122 © Integrata Cegos GmbH	ORACLE und SQL
34	

Zeilen limitieren



- **5 Top Verdienster ausgeben**

- ```
SELECT ename, sal
FROM scott.emp
ORDER BY sal desc
FETCH FIRST 5 ROWS ONLY;
```

- **die nächsten 5 TOP Verdienster:**

- ```
SELECT ename, sal
FROM scott.emp
ORDER BY sal desc
OFFSET 5 ROWS FETCH NEXT 5 ROWS ONLY;
```

7.4 Objekt SEQUENCE

Eine Sequenz ist ein Datenbankobjekt, mit dessen Hilfe sich **fortlaufende Integerreihen** generieren lassen. Somit wird es auch für verschiedene Anwender möglich, fortlaufende Ganzzahlen zu benutzen, die in einer Tabellenspalte nur einmal vorkommen (unique integers). Dies eignet sich besonders zur Erzeugung des Primärschlüssels einer Tabelle, aber die Sequenz ist unabhängig von der Tabelle(!).

7.4.1 Der CREATE SEQUENCE Befehl

Um eine Sequenz im eigenen Schema anlegen zu können, ist das CREATE SEQUENCE-Privileg notwendig. Eine Sequenz in einem fremden Schema kann nur angelegt werden, wenn der Benutzer das CREATE ANY SEQUENCE-Privileg besitzt.

Beispiel:

```
SQL> CREATE SEQUENCE reihel
      INCREMENT BY 5
      START WITH 10
      MAXVALUE 10000
      CACHE 50;
```

Syntax:

```
create-sequence ::=

    CREATE SEQUENCE [Schema.]Sequence
    [ seq-parameter ];

seq-parameter ::=

    [ INCREMENT BY int ] | [ START WITH int ]
    | [ MAXVALUE int ] | [ NOMAXVALUE ]
    | [ MINVALUE int ] | [ NOMINVALUE ]
    | [ CYCLE ] | [ NOCYCLE ]
    | [ CACHE int ] | [ NOCACHE ]
    | [ ORDER ] | [ NOORDER ]
```

Parameter:

INCREMENT BY <i>n</i>	spezifiziert das Intervall zwischen den einzelnen Sequenznummern. Das Intervall kann positive wie auch negative Werte annehmen (nicht 0, Default 1).
MINVALUE <i>n</i>	gibt die kleinste Sequenznummer an.
NOMINVALUE	Default, Minimumvorgabe ist 1
MAXVALUE <i>n</i>	gibt die größte Sequenznummer an.
NOMAXVALUE	Default, Maximalvorgabe ist 10^{27}
START WITH <i>n</i>	gibt die erste zu generierende Sequenznummer an (Default 1).
CYCLE	mit dieser Option wird beim Erreichen der größten bzw. der kleinsten Sequenznummer wieder mit dem Startwert begonnen.
NOCYCLE	mit diesem Parameter ist ein erneutes Erstellen von Sequenznummern nach Erreichen des Maximalwertes nicht möglich (Default).
CACHE <i>n</i>	gibt an, wie viele Sequenznummern im Speicher gehalten werden. Minimum: 2 (Default: 20).
NOCACHE	Es werden keine Sequenznummern im Speicher gehalten.
ORDER	Die Nummern werden in der Reihenfolge erzeugt wie sie angefordert werden. Dies spielt hauptsächlich in einem RAC (Real Application Cluster) eine Rolle, wenn man die erzeugten Zahlen wie Timestamps verwenden will. In einer selbstständigen Datenbank ist dies automatisch der Fall.
NOORDER	Kein Order (Default).

SEQUENCE	 integrata cegos
<ul style="list-style-type: none"> ▪ Erzeugung fortlaufender Integerreihen ▪ Syntax: <pre>create-sequence ::= CREATE SEQUENCE [Schema.] Sequence [seq-parameter]; seq-parameter ::= [INCREMENT BY int] [START WITH int] [MAXVALUE int] [NOMAXVALUE] [MINVALUE int] [NOMINVALUE] [CYCLE] [NOCYCLE] [CACHE int] [NOCACHE] [ORDER] [NOORDER]</pre> ▪ Beispiel: <pre>SQL> CREATE SEQUENCE reihe1 INCREMENT BY 5 START WITH 10 MAXVALUE 10000 CACHE 50;</pre> 	

2.5.0122 © Integrata Cegos GmbH

ORACLE und SQL

36

7.4.2 Neue Parameter

Die `KEEP` Klausel beinhaltet, dass die Sequence den `NEXTVAL`-Wert auch dann behält, wenn ein reparierbarer Fehler aufgetreten ist. Voraussetzung hierfür ist, dass die Sequence im gleichen Schema angelegt ist wie die Tabelle.

Für diese Klausel muss ein zusätzliches Recht vergeben werden:

```
GRANT KEEP SEQUENCE ON emp_seq TO SCOTT;
```

Der Default beim Erstellen von Sequenzen ist die `NOKEEP` Klausel.

Die `SESSION` Klausel wird als spezieller Sequenz-Typ für Temporäre Tabellen genutzt. Die Sequenz ist dabei nur in der aktuellen Session gültig und liefert dort aufeinanderfolgende Werte.

Eine `SESSION`-Sequenz muss von einer read-write-Datenbank erstellt werden, steht aber auch read-only-Datenbanken zur Verfügung.

Die `CACHE`, `NOCACHE`, `ORDER` oder `NOORDER` Klauseln werden ignoriert, wenn die `SESSION` Klausel genutzt wird.

Die `GLOBAL` Klausel ist der Default beim Erstellen von Sequenzen.

SEQUENCE – Neue Parameter	
<ul style="list-style-type: none"> ▪ KEEP und NOKEEP Klausel <ul style="list-style-type: none"> ▪ KEEP: behält NEXTVAL-Wert auch, wenn reparierbarer Fehler aufgetreten ist ▪ Voraussetzung: Sequenz ist im gleichen Schema wie die Tabelle ▪ zusätzliches Recht: <ul style="list-style-type: none"> ▪ GRANT KEEP SEQUENCE ON emp_seq TO SCOTT; ▪ GLOBAL oder SESSION <ul style="list-style-type: none"> ▪ SESSION: Sequenz für Temporäre Tabellen ▪ Sequenz nur in aktueller Session gültig (liefert dort aufeinanderfolgende Werte) 	
2.5.0122 © Integrata Cegos GmbH	ORACLE und SQL

7.4.3 Pseudospalten der SEQUENCE

Die Sequenz besitzt zwei Pseudospalten:

NEXTVAL ermittelt den nächsten Wert einer Sequenz.

CURRVAL ermittelt den aktuellen Wert einer Sequenz.

Beispiel:

Mit

```
SQL> SELECT reihel.NEXTVAL FROM dual;
```

wird der aktuelle Sequenzwert (z.B. 10) ausgegeben. Bei jedem weiteren Aufruf des Formates Sequenz.NEXTVAL wird ein entsprechend den Angaben ermittelter nächster Wert zurückgegeben. In diesem Fall z.B. 15, 20, 25....

Mit

```
SQL> SELECT reihel.CURRVAL FROM dual;
```

wird der aktuelle Wert einer Sequenz ermittelt ohne diesen zu verbrauchen.

Der Vorteil der Sequenznummern ist die Möglichkeit, in Applikationen einen eindeutigen numerischen Schlüssel generieren zu können, der nicht explizit vom Anwendungsprogramm gepflegt werden muss, sondern von ORACLE geführt wird.

Weiterhin ist eine Sequenz nicht nur für eine Tabelle verwendbar sondern auch für mehrere Tabellen.

Ein konfliktfreier Zugriff ist möglich, da die Sequenznummer immer hochgezählt wird, egal ob eine Transaktion abgeschlossen wird oder nicht. Das heißt, wenn 2 Benutzer auf die Sequenz zugreifen, wird sie immer hochgezählt, so dass keine Zugriffskonflikte entstehen können. Dies führt jedoch auch zu Lücken.

Hinweis: Die Option `CACHE` ermöglicht einen schnelleren Zugriff auf die Nummern, doch es gehen alle im Cache gehaltenen Nummern beim Herunterfahren der Instanz verloren.

Pseudospalten der SEQUENCE		
▪ <code>NEXTVAL</code>	ermittelt den nächsten Wert einer Sequenz.	
▪ <code>CURRVAL</code>	ermittelt den aktuellen Wert einer Sequenz.	
▪ Beispiel:		
SQL>	<code>SELECT reihe1.NEXTVAL FROM dual;</code>	
SQL>	<code>SELECT reihe1.CURRVAL FROM dual;</code>	
2.5.0122 © Integrata Cegos GmbH		ORACLE und SQL
		38

7.4.4 Der ALTER SEQUENCE Befehl

Mit Hilfe des ALTER SEQUENCE Befehls lassen sich Einstellungen, die mit dem CREATE SEQUENCE Befehl festgelegt worden sind, ändern.

Um eine Sequenz in einem fremden Schema verändern zu können, muss der Benutzer das ALTER ANY SEQUENCE-Recht besitzen.

Die Argumente bzw. Parameter haben dabei die gleiche Bedeutung wie die des CREATE SEQUENCE Befehls.

Hinweis: Um den *Startwert* einer Sequenz zu ändern, muss die Sequenz explizit gelöscht und neu erstellt werden. Nur Werte, die später eingefügt werden, können über den ALTER SEQUENCE Befehl nachträglich geändert werden.

Beispielsweise kann auch kein neuer Maximalwert angegeben werden, der den momentanen Sequenzwert unterschreitet.

Beispiele:

```
SQL> ALTER SEQUENCE reihel  
      MAXVALUE 1500;
```

```
SQL> ALTER SEQUENCE reihel  
      CYCLE  
      CACHE 5;
```

Syntax:

```
alter-sequence ::=  
  
    ALTER SEQUENCE [Schema.]Sequence  
    [ seq-parameter ];
```

<p>ALTER SEQUENCE</p> <ul style="list-style-type: none"> ▪ <u>Syntax:</u> <code>alter-sequence ::= ALTER SEQUENCE [Schema.]Sequence [seq-parameter];</code> ▪ <u>Beispiele:</u> <code>SQL> ALTER SEQUENCE reihel MAXVALUE 1500; SQL> ALTER SEQUENCE reihel CYCLE CACHE 5;</code> 		
<p>2.5.0122 © Integrata Cegos GmbH</p>	<p>ORACLE und SQL</p>	<p>39</p>

7.4.5 Der **DROP SEQUENCE** Befehl

Mit Hilfe des **DROP SEQUENCE**-Befehls lassen sich Sequenzen löschen.

Um eine Sequenz in einem fremden Schema löschen zu können, muss der Benutzer das **DROP ANY SEQUENCE**-Recht besitzen.

Um eine Sequenz neu zu starten, muss sie erst gelöscht und dann wieder neu angelegt werden.

Beispiel:

Der neue Startwert soll 100 sein, der aktuelle ist 150;

```
SQL> DROP SEQUENCE seq1;
SQL> CREATE SEQUENCE seq1 START WITH 100;
```

Syntax:

```
drop-sequence ::=  
    DROP SEQUENCE [Schema.]Sequence;
```

DROP SEQUENCE



- Syntax:

```
drop-sequence ::=  
    DROP SEQUENCE [Schema.] Sequencename;
```

- Beispiel:

Der neue Startwert soll 100 sein, der aktuelle ist 150;
SQL> DROP SEQUENCE seq1;
SQL> CREATE SEQUENCE seq1 START WITH 100;

7.4.6 Der **RENAME** Befehl

Mit diesem DDL-Befehl können Sequenzen umbenannt werden.

Nur der Eigentümer einer Sequenz kann die Sequenz umbenennen.

Beispiel:

```
SQL> RENAME reihe1 TO reihe2;
```

Syntax:

```
rename ::=
```

```
    RENAME Alter-name TO Neuer-name;
```

RENAME



- Umbenennung einer SEQUENCE
- Syntax:

```
rename ::=  
    RENAME Alter-name TO Neuer-name;
```
- Beispiel:

```
SQL>RENAME reihe1 TO reihe2;
```

7.4.7 Die IDENTITY Klausel

Anhand der IDENTITY-Klausel kann ab der Version 12c auch ein automatischer Sequenz-Generator zum Füllen von PK/UK Spalten verwendet werden.

Beispiele:

```
SQL> CREATE TABLE t (
      id NUMBER GENERATED AS IDENTITY,
      text VARCHAR2(10));
      INSERT INTO t (text) VALUES ('TTT');
```

ID	TEXT
1	TTT

```
SQL> CREATE TABLE scott.t2 (id NUMBER GENERATED AS
      IDENTITY, text varchar2(99));
      SQL> INSERT INTO scott.t2 (text)
      SELECT owner FROM all_objects WHERE rownum<10;
```

IDENTITY KLAUSEL	 integrata cegos
<ul style="list-style-type: none">▪ Es kann auch ein automatischer Sequenz-Generator zum Füllen von PK/UK Spalten verwendet werden.▪ Beispiele:<ul style="list-style-type: none">▪ <code>CREATE TABLE t (id NUMBER GENERATED AS IDENTITY, text VARCHAR2(10)); INSERT INTO t (text) VALUES ('TTT');</code>▪ <code>CREATE TABLE t (id NUMBER GENERATED BY DEFAULT AS IDENTITY (START WITH 100 INCREMENT BY 10));</code>	

2.5.0122 © Integrata Cegos GmbH ORACLE und SQL 42

IDENTITY KLAUSEL	
<ul style="list-style-type: none"> ▪ Im Prinzip können die gleichen Optionen, wie bei einer Sequenz verwendet werden: <ul style="list-style-type: none"> ▪ ... (START WITH { <int> LIMIT VALUE } INCREMENT BY <int> MAXVALUE integer NOMAXVALUE MINVALUE integer NOMINVALUE CYCLE NOCYCLE CACHE integer NOCACHE ORDER NOORDER) ▪ Zusätzlich gibt es noch: <ul style="list-style-type: none"> ▪ ALWAYS: Spalte darf nur durch IDENTITY gefüllt werden. Manuell bei Insert oder Update ist nicht erlaubt. ▪ BY DEFAULT [ON NULL]: Spalte wird per Default durch IDENTITY gefüllt, manuelles Füllen ist jedoch möglich. ▪ ON NULL: Wenn ein Insert für die Spalte NULL ergibt, wird sie durch IDENTITY gefüllt. 	

Folgende Einschränkungen sind bei der `IDENTITY`-Klausel zu beachten:

- Pro Tabelle kann nur eine `Identity` Spalte festlegt werden. Wird die `IDENTITY`-Klausel verwendet, dann muss der Datentyp in der Spaltendefinition nummerisch sein. Ein benutzerdefinierter Datentyp kann nicht spezifiziert werden.
- Das Setzen eines Defaultwertes in der Spaltendefinition ist bei der Verwendung der `Identity` Klausel nicht erlaubt.
- Wird die `Identity` Klausel verwendet, ist ein NOT NULL Constraint automatisch gesetzt.
- Wird ein Inline Constraint benutzt, das im Widerspruch zu einem NOT NULL Constraint bzw. NOT DEFERRABLE Constraint steht, dann wird eine Fehlermeldung ausgegeben.
- Oracle empfiehlt, dass bei verschlüsselten `Identity` Spalten ein starker Verschlüsselungsalgorithmus verwendet wird, da ansonsten eventuelle Rückschlüsse auf den Algorithmus gezogen werden könnten.
- `CREATE TABLE AS SELECT` vererbt nicht die `IDENTITY` Eigenschaft.

IDENTITY KLAUSEL



- Funktionsweise
 - Oracle legt eine Sequenz im Schema der Tabelle an (mit Namen ISEQ\$\$<objnr> und als Default mit Cache=20)
 - Welche Tabelle verwendet IDENTITY Spalten :
 - `SELECT owner, table_name, has_identity
FROM all_tables WHERE owner='MARCO';`

OWNER	TABLE_NAME	HAS_IDENTITY
MARCO	T	YES

 - `SELECT owner, table_name, column_name,
identity_column
FROM all_tab_columns WHERE owner='MARCO';`

OWNER	TABLE_NAME	COLUMN_NAME	IDENTITY_COLUMN
MARCO	T	ID	YES
MARCO	T	TEXT	NO

Die IDENTITY-Klausel kann mit folgenden Befehlen gewartet werden:

- Bestehende IDENTITY löschen (Spalte bleibt erhalten)

```
ALTER TABLE t MODIFY (id DROP IDENTITY);
```

- IDENTITY nachträglich (inkl. Spalte) hinzufügen:

```
ALTER TABLE t ADD
(id_neu NUMBER GENERATED AS IDENTITY
(START WITH 100));
```

7.5 Objekt SYNONYM

Ein Synonym ist ein **Bezeichner** für ein Datenbankobjekt. Es dient oft dazu, lange Objektnamen abzukürzen. Beispielsweise vereinfacht es den Zugriff auf Tabellen eines anderen Eigentümers. Das Synonym selbst ist auch ein Datenbankobjekt.

7.5.1 Der CREATE SYNONYM Befehl

Beispiel:

```
SQL> CREATE PUBLIC SYNONYM e FOR scott.emp;
```

```
SQL> SELECT *  
      FROM          e;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17.12.80	800		20
7499	ALLEN	SALESMAN	7698	20.02.81	1600	300	30
7521	WARD	SALESMAN	7698	22.02.81	1250	500	30
7566	JONES	MANAGER	7839	02.04.81	2975		20
7654	MARTIN	SALESMAN	7698	28.09.81	1250	1400	30
7698	BLAKE	MANAGER	7839	01.05.81	2850		30
7782	CLARK	MANAGER	7839	09.06.81	2450		10
7788	SCOTT	ANALYST	7566	19.04.87	3000		20
7839	KING	PRESIDENT		17.11.81	5000		10
7844	TURNER	SALESMAN	7698	08.09.81	1500	0	30
7876	ADAMS	CLERK	7788	23.05.87	1100		20
7900	JAMES	CLERK	7698	03.12.81	950		30
7902	FORD	ANALYST	7566	03.12.81	3000		20
7934	MILLER	CLERK	7782	23.01.82	1300		10

Syntax:

```
create-synonym ::=  
    CREATE [OR REPLACE] [PUBLIC] SYNONYM  
    [Schema.] Synonym FOR [Schema.] Objekt;
```

Die Option PUBLIC macht ein Synonym allgemein zugänglich, ohne diese Option wird es als privates Synonym bezeichnet und ist nur seinem Eigentümer zugänglich. Allerdings kann nur ein Benutzer mit DBA-Rechten oder mit Privileg CREATE PUBLIC SYNONYM ein PUBLIC-Synonym anlegen.

CREATE SYNONYM	
<ul style="list-style-type: none"> ▪ SYNONYM ist ein "Bezeichner" (Alias) für ein Datenbankobjekt ▪ <u>Syntax:</u> <pre>create-synonym ::= CREATE [OR REPLACE] [PUBLIC] SYNONYM [Schema.] <i>Synonym</i> FOR [Schema.] <i>Objekt</i>;</pre> ▪ <u>Beispiel:</u> <pre>SQL>CREATE PUBLIC SYNONYM e FOR scott.emp;</pre> 	
2.5.0122 © Integrata Cegos GmbH	ORACLE und SQL

7.5.2 Ändern eines SYNONYMS

Synonyme können nicht geändert werden. Sie müssen gelöscht und dann neu erstellt werden.

7.5.3 Der DROP SYNONYM Befehl

Mit Hilfe des DROP SYNONYM Befehls lassen sich Synonyme löschen. PUBLIC Synonyme lassen sich nur mit DROP PUBLIC SYNONYM ... löschen.

Beispiel:

SQL> DROP SYNONYM e;

Syntax:

```
drop-synonym ::=  
      DROP [ PUBLIC ] SYNONYM  
      [Schema.] Synonym [ FORCE ] ;
```

7.5.4 Der **RENAME** Befehl

Mit diesem DDL-Befehl können Synonyme umbenannt werden.

Nur der Eigentümer eines Synonyms kann ein Synonym umbenennen.

Beispiel:

```
SQL> RENAME e TO e22;
```

Syntax:

```
rename ::=
```

```
    RENAME Alter-name TO Neuer-name;
```

Operationen auf SYNONYMen



- Änderung ist nicht möglich.

- **Löschen:**

- **Beispiel:**
SQL>DROP SYNONYM e;

- **Syntax:**

```
drop-synonym ::=  
    DROP [PUBLIC] SYNONYM  
    [Schema.] Synonym [FORCE];
```

- **Umbenennen:**

- **Beispiel:**
SQL>RENAME e TO e22;

- **Syntax:**

```
rename ::=  
    RENAME Alter-name TO Neuer-name;
```

7.6 Constraints

Von besonderer Bedeutung sind die Forderungen nach Kontrolle und Gewährleistung der Datenintegrität eines Datenbestandes. Unter 'Integrität' ist dabei die formale Richtigkeit der Daten zu verstehen. Es gibt dabei drei Problembereiche:

- Anwendungsspezifische Einschränkungen von Spaltenwerten.
- Entitätsintegrität. Das heißt, dass jede Tabelle einen Primärschlüssel haben muss.
- Referentielle Integrität. Zwischen Fremdschlüsseln und Primärschlüsseln herrschen logische Beziehungen, die für alle Datenbankoperationen sichergestellt sein müssen.

Bei den in ORACLE zur Verfügung stehenden Methoden zur Implementierung von Constraints (Regeln) handelt es sich um eine ANSI/ISO/DIN-konforme Möglichkeit der Definition von Integritätsbedingungen. Sie werden als Teil der Tabellendefinition angegeben und sind, wenn sie nicht explizit ausgeschaltet (disabled) sind, für alle Änderungsoperationen aktiv.

Folgende Constraint-Arten sind in ORACLE verfügbar:

- **PRIMARY KEY**
- **FOREIGN KEY**
- **UNIQUE**
- **CHECK**
- **NOT NULL**

NOT NULL kann als einziger Constraint nur auf Spaltenebene definiert werden, alle anderen Constraints können sowohl auf Spalten- als auch auf Tabellenebene definiert werden.

Constraints	
<ul style="list-style-type: none"> ▪ Constraints dienen zur: <ul style="list-style-type: none"> ▪ Kontrolle und ▪ Gewährleistung der Datenintegrität ▪ Problembereiche: <ul style="list-style-type: none"> ▪ Anwendungsspezifische Einschränkungen ▪ Entitätsintegrität ▪ Referentielle Integrität ▪ CONSTRAINTS: <ul style="list-style-type: none"> ▪ PRIMARY KEY ▪ FOREIGN KEY ▪ UNIQUE ▪ CHECK ▪ <u>NOT NULL</u> (nur auf Spaltenebene definierbar) 	

2.5.0122 © Integrata Cegos GmbH

ORACLE und SQL

47

create-table-direkt ::=

```
CREATE [GLOBAL TEMPORARY] TABLE
  [Schema.]Tabelle (
    Spalte spalteneigenschaft
    {, Spalte spalteneigenschaft}
    {, constraint-zeile});
```

spalteneigenschaft ::=

datentyp [DEFAULT wert] {constraint-spalte}

Erstellen von CONSTRAINTs		
SQL> CREATE TABLE Tbuch (BuchNr NUMBER(8) NOT NULL,	DEFAULT(2100),
ErschJ NUMBER(4)	NOT NULL,	
Preis NUMBER(7,2),	PRIMARY KEY (Buchnr),	
Titel VARCHAR2(127)	CHECK (Preis > 0),	
CONSTRAINT pk_tb	CHECK (ErschJ BETWEEN 1950 AND 2100)	
CONSTRAINT c_tb_Preis		
CONSTRAINT c_tb_EJ		
) ;		
SQL> CREATE TABLE Tisbn (
BuchNr NUMBER(8) NOT NULL,		
Isbn CHAR(10) NOT NULL,		
LfdNr NUMBER(1) NOT NULL,		
CONSTRAINT pk_ti PRIMARY KEY (Isbn),		
CONSTRAINT fk_ti_tb FOREIGN KEY (Buchnr)		
) ;	REFERENCES Tbuch (BuchNr)	ON DELETE CASCADE

2.5.0122 © Integrata Cegos GmbH

ORACLE und SQL

48

Syntax zur Erstellung eines Constraints auf Spaltenebene:

constraint-spalte ::=

```
[ CONSTRAINT Constraint ]
[ [ NOT ] NULL ] | [ UNIQUE ] | [ PRIMARY KEY ]
| [ CHECK ( bedingung-check ) ]
| [ FOREIGN KEY (spalte) ref-klausel ]
[ constraint-status ]
```

Syntax zur Erstellung eines Constraints auf Tabellenebene:

constraint-zeile ::=

```
[ CONSTRAINT Constraint ]
[ UNIQUE ] | [ PRIMARY KEY ]
| [ CHECK ( bedingung-check ) ]
| [ FOREIGN KEY (spalte) ref-klausel ]
[ constraint-status ]
```

ref-klausel ::=

```
REFERENCES [Schema.] Tabelle (Spalte)
[ ON DELETE ] [ CASCADE ] | [ SET NULL ]
```

```
constraint-status ::=

{ [ [ NOT ] DEFERABLE ]
| [ INITIALLY IMMEDIATE | DEFERRED ]
| [ ENABLE ] | [ DISABLE ]
| [ VALIDATE ] | [ NOVALIDATE ]
| [ using-index-klausel ] | [ exceptions-klausel ]
} }
```

Syntax



- **Syntax zur Erstellung eines Constraints auf Spaltenebene:**

```
constraint-spalte ::=

[ CONSTRAINT Constraint ]
[ [ NOT ] NULL ] | [ UNIQUE ] | [ PRIMARY KEY ]
| [ CHECK ( bedingung-check ) ]
| [ FOREIGN KEY (spalte) ref-klausel ]
[ constraint-status ]
```

- **Syntax zur Erstellung eines Constraints auf Tabellenebene:**

```
constraint-zeile ::=

[ CONSTRAINT Constraint ]
[ UNIQUE ] | [ PRIMARY KEY ]
| [ CHECK ( bedingung-check ) ]
| [ FOREIGN KEY (Spalte) ref-klausel ]
[ constraint-status ]
```

Syntax	 integrata cegos
<pre>▪ ref-klausel ::= REFERENCES [Schema.]Tabelle (Spalte) [ON DELETE] [CASCADE] [SET NULL] ▪ constraint-status ::= { [[NOT] DEFERABLE] [INITIALLY IMMEDIATE DEFERED] [ENABLE] [DISABLE] [VALIDATE] [NOVALIDATE] [using-index-klausel] [exceptions-klausel] }</pre>	

Obwohl es nicht zwingend notwendig ist, sollte jeder Constraint einen Namen erhalten, da:

- die Fehlersuche bei einem Verstoß gegen eine vordefinierte Regel erleichtert wird.
- die Möglichkeit, einen Constraint wieder zu löschen bzw. zu aktivieren oder deaktivieren, erleichtert wird.

Wenn kein Name vergeben wird, erzeugt ORACLE einen eigenen Namen in der Form '**SYS_Cn**', wobei *n* eine ganze Zahl beinhaltet.

Beispiel:

```
SQL> CREATE TABLE Tbuch (
    BuchNr  NUMBER(8)          NOT NULL,
    ErschJ  NUMBER(4)          DEFAULT(2100),
    Preis    NUMBER(7,2),
    Titel   VARCHAR2(127)      NOT NULL,
    CONSTRAINT pk_tb PRIMARY KEY (Buchnr),
    CONSTRAINT c_tb_Preis CHECK
        (Preis > 0),
    CONSTRAINT c_tb_EJ CHECK
        (ErschJ BETWEEN 1950 AND 2100)
) ;

SQL> CREATE TABLE Tisbn (
    BuchNr  NUMBER(8)          NOT NULL,
    Isbn    CHAR(10)            NOT NULL,
    LfdNr   NUMBER(1)           NOT NULL,
    CONSTRAINT pk_ti PRIMARY KEY (Isbn),
    CONSTRAINT fk_ti_tb FOREIGN KEY (Buchnr)
        REFERENCES Tbuch (BuchNr)
        ON DELETE CASCADE
) ;
```

Constraintstatus

DEFERRABLE	Der Constraint kann auch erst am Ende einer Transaktion überprüft werden.
NOT DEFERRABLE	Default. Der Constraint muss bei jedem DML-Befehl überprüft werden.
INITIALLY IMMEDIATE	Default. Der Constraint wird als Default-Einstellung bei jedem DML-Befehl überprüft.
INITIALLY DEFERRED	Der Constraint wird als Default-Einstellung erst am Ende einer Transaktion überprüft (nur bei DEFERRABLE möglich).
ENABLE	Default. Der Constraint ist aktiviert.
DISABLE	Der Constraint ist deaktiviert.
VALIDATE	Default. Auch bereits vorhandene Datensätze werden auf Constraint-Verletzungen überprüft.
NOVALIDATE	Nur neu hinzukommende Datensätze werden auf Constraint-Verletzungen überprüft.

Nicht alle Kombinationen sind möglich:

Nur ein DEFERRABLE-Constraint kann überhaupt auf INITIALLY DEFERRED gesetzt werden.

NOT DEFERRABLE bzw. keine Angabe impliziert automatisch auch INITIALLY IMMEDIATE.

ENABLE VALIDATE ist gleichbedeutend mit ENABLE und entspricht dem Default.

ENABLE NOVALIDATE muss explizit angegeben werden.

DISABLE NOVALIDATE entspricht DISABLE.

DISABLE VALIDATE kann zwar eingestellt werden, aber dann sind keine DML-Befehle auf die betroffenen Spalten zulässig.

7.6.1 Der NOT NULL-Constraint

Dieser Constraint verbietet es, in eine Spalte NULL-Werte einzutragen. Die Spalte *muss* also bei einem `INSERT`-Befehl einen definierten Wert erhalten.

Es sind beliebig viele `NOT NULL`-Constraints pro Tabelle erlaubt.

Beispiel:

Auf der Spalte `empno` der Tabelle `EMP` liegt ein `NOT NULL`-Constraint.

```
SQL> INSERT INTO emp (ename) VALUES ('Test');
      → Fehler!
SQL> INSERT INTO emp (empno) values (8000);
      → OK
```

7.6.2 Der UNIQUE-Constraint

Dieser Constraint verbietet es, in eine Spalte zweimal den gleichen Wert einzutragen. Jeder Eintrag muss `UNIQUE`, also *eindeutig* sein. Allerdings sind `NULL`-Werte erlaubt, und zwar beliebig viele.

Es sind beliebig viele `UNIQUE`-Constraints pro Tabelle erlaubt. Ein `UNIQUE` Key kann auch über mehrere Spalten gehen. In diesem Fall spricht man von einem zusammengesetzten eindeutigen Schlüssel.

Für jede Spalte mit `UNIQUE`-Constraint erzeugt ORACLE automatisch einen Index.

7.6.3 Der PRIMARY KEY Constraint

Die `PRIMARY KEY`-Klausel definiert einen Primärschlüssel für eine oder mehrere Spalten einer Tabelle. Abweichend vom Relationen-Modell ist die `PRIMARY KEY`-Klausel im `CREATE TABLE` Befehl fakultativ, doch sollte für jede Tabelle ein Primärschlüssel definiert werden.

Für jede `PRIMARY KEY` Spalte/Spaltenkombination werden **automatisch** auch ein `UNIQUE` Key und ein `NOT NULL`-Constraint definiert. Bei einem zusammengesetzten Primärschlüssel reicht es, wenn in *einer* der Spalten ein definierter Wert steht; es liegt nicht auf den beteiligten einzelnen Spalten ein `NOT NULL`-Constraint sondern nur auf ihrer Kombination.

Es wird auch automatisch ein `UNIQUE`-Index angelegt.

Es gibt immer nur *einen* Primärschlüssel pro Tabelle.

7.6.4 Der FOREIGN KEY Constraint

Er stellt sicher, dass für einen Fremdschlüssel nur Werte verwendet werden, die auch in der Menge der korrespondierenden Primärschlüsselwerte vorkommen. NULL-Werte sind allerdings zulässig.

Nehmen wir an, es soll in einer Angestellten-Tabelle ein Bezug auf eine Abteilungstabelle festgehalten werden. Die entsprechende Relation enthält unter anderem die Attribute: Abteilungsnummer (deptno) und Abteilungsname.

Das Kommando für die Erzeugung der Datenbank-Tabelle hätte folgendes Aussehen:

```
SQL> CREATE TABLE dept
      (deptno    NUMBER,
       dname     VARCHAR2(30),
       constraint PK_deptno
          PRIMARY KEY (deptno));
```

Die Angestelltentabelle erhält referentielle Integritätsbedingungen für den neu einzutragenden Fremdschlüssel deptno:

```
SQL> CREATE TABLE emp
      (empno    NUMBER,
       ename     VARCHAR2(40) NOT NULL,
       deptno   NUMBER,
       constraint PK_empno
          PRIMARY KEY (empno),
       FOREIGN KEY (deptno)
          REFERENCES dept (deptno);
```

Die Tabelle DEPT stellt die **Master-Tabelle** oder **referenzierte Tabelle** dar. EMP wird als **Detail-Tabelle** oder **referenzierende Tabelle** bezeichnet. Wird ein Datensatz in die Tabelle EMP eingetragen oder werden die Fremdschlüsselspalten von EMP verändert, prüft der DB-Server, ob dem neuen Fremdschlüsselwert ein entsprechender Wert in der Tabelle DEPT gegenübersteht. Wenn nicht, wird die Operation zurückgewiesen.

Bei einer Definition auf Spaltenebene wird nur das Schlüsselwort **REFERENCES** verwendet. Ihm folgen der Name der referenzierten Tabelle und der Name der referenzierten Spalte in der Form:

```
... REFERENCES Tabelle (Spalte)
```

Der FOREIGN KEY-Constraint wird auch als referentieller Integritäts-Constraint bezeichnet.

Eine Löschoperation in einer referenzierten Tabelle (Master-Tabelle) ist nur dann möglich, wenn kein abhängiger Datensatz in einer FOREIGN KEY Tabelle existiert.

Beispiel:

Aus der Tabelle DEPT kann der Datensatz mit der Kennung deptno=40 problemlos gelöscht werden, jeder andere Löschversuch dagegen führt zu einer Fehlermeldung.

Dieses Problem kann umgangen werden mit den Optionen FOREIGN KEY der Tabelle emp

- **ON DELETE CASCADE**: löscht abhängige Datensätze
- **ON DELETE SET NULL**: setzt den entsprechenden Wert abhängiger Datensätze auf NULL

Beispiele:

Wäre die FOREIGN KEY-Beziehung der Tabelle EMP auf die Tabelle DEPT in folgender Form erstellt worden:

```
CREATE TABLE emp ( ...
    FOREIGN KEY (deptno) REFERENCES dept (deptno)
    ON DELETE CASCADE);
```

dann könnte man jeden Datensatz aus der Tabelle DEPT löschen. Zur Wahrung der referentiellen Integrität würden vorher alle zugehörigen Datensätze aus der Tabelle EMP gelöscht.

Der Befehl

```
SQL> DELETE FROM dept WHERE deptno = 10;
```

würde zuerst aus der Tabelle EMP die drei abhängigen Datensätze löschen und dann aus der Tabelle DEPT eine Zeile entfernen.

Bei Verwendung von ON DELETE SET NULL würden in der Tabelle EMP die Datensätze mit deptno = 10 erhalten bleiben, aber ihre Werte für deptno auf NULL gesetzt werden.

7.6.5 Der CHECK Constraint

Der CHECK Constraint definiert eine Bedingung, der jeder Datensatz genügen muss. Innerhalb der Bedingung kann auf Werte des aktuellen Datensatzes zugegriffen werden. Zugriffe auf andere Tabellen (z.B. Unterabfragen) sind nicht möglich.

Durch den CHECK Constraint sind anwendungsspezifische Regeln durchsetzbar. Die CHECK Bedingung muss in Klammern stehen.

Mehrere CHECK Optionen pro Tabelle und auch pro Spalte sind möglich.

7.6.6 Constraints hinzufügen und löschen

Constraints können auch nachträglich einer bestehenden Tabelle hinzugefügt werden, wenn alle vorhandenen Datensätze den Bedingungen dieses Constraints genügen, da ein neu hinzugefügter Constraint auch nachträglich wirksam wird.

7.6.6.1 Einen NOT NULL Constraint hinzufügen

```
ALTER TABLE [Schema.]Tabelle  
MODIFY (Spalte [CONSTRAINT Constraintname]  
NOT NULL);
```

Beispiel: NOT NULL Constraint hinzufügen

```
SQL> ALTER TABLE emp MODIFY sal NOT NULL;
```

Beispiel: NULL Constraint hinzufügen

```
SQL> ALTER TABLE emp MODIFY sal NULL;
```

7.6.6.2 Alle übrigen Constraints hinzufügen

```
ALTER TABLE [Schema.]Tabelle  
ADD (constraint-zeile);  
  
SQL> ALTER TABLE emp ADD CONSTRAINT  
ch_emp_sal CHECK (sal BETWEEN 500 AND 10000);
```

7.6.6.3 Einen Constraint löschen

```
ALTER TABLE [Schema.]Tabelle  
DROP CONSTRAINT Constraintname;
```

```
SQL> ALTER TABLE emp DROP CONSTRAINT ch_emp_sal;
```

Wird ein Constraint gelöscht, so werden etwaige Indizes automatisch mitgelöscht.

Da ein PRIMARY KEY nur einmal vergeben werden darf, reicht in diesem Fall zum Löschen auch

```
ALTER TABLE [Schema.]Tabelle
    DROP PRIMARY KEY [ CASCADE ];
```

Um einen PRIMARY KEY oder UNIQUE-Constraint löschen zu können, müssen entweder erst alle abhängigen, diesen Constraint referenzierenden Constraints gelöscht werden, oder die Option CASCADE wird verwendet, die abhängige Constraints automatisch mitlöscht.

CONSTRAINTs hinzufügen und löschen
 integrata
cegos

- **NOT NULL Constraint hinzufügen:**
`ALTER TABLE [Schema.]Tabelle
 MODIFY (Spalte [CONSTRAINT Constraint]
 NOT NULL);`

- **Alle übrigen Constraints hinzufügen**
`ALTER TABLE [Schema.]Tabelle
 ADD (constraint-zeile);`

- **Einen Constraint löschen**
`ALTER TABLE [Schema.]Tabelle
 DROP CONSTRAINT Constraint;`

2.5.0122 © Integrata Cegos GmbH ORACLE und SQL 51

7.6.7 Constraintstatus ändern

Der Status eines Constraints kann nachträglich geändert werden, mit Ausnahme von DEFERRABLE und NOT DEFERRABLE. Hier ist eine Änderung nur durch Löschen und Neuanlegen möglich. Für alle anderen Zustände lautet die allgemeingültige Syntax zum Ändern:

```
ALTER TABLE [Schema.]Tabelle
    MODIFY CONSTRAINT Constraintname Constraintstatus
        [EXCEPTIONS INTO exceptionstabelle];
```

Wird ein Constraint deaktiviert (DISABLE), bleibt die Definition erhalten, aber etwaige Indizes werden dabei gelöscht und bei Reaktivierung neu angelegt.

Mit der Klausel EXCEPTIONS INTO können Verstöße gegen den Constraint, die beispielsweise seine Aktivierung verhindern, in einer eigenen Tabelle protokolliert werden. Diese Tabelle namens EXCEPTIONS muss

vorher durch das Skript ?\rdbms\admin\utlexcpt.sql angelegt werden.

Mit einer Abfrage auf EXCEPTIONS lässt sich die ROWID der Zeilen bestimmen, die gegen den Constraint verstößen.

Zur Aktivierung /Deaktivierung eines Constraints ist auch der Befehl

```
ALTER TABLE [Schema].Tabelle
ENABLE [ VALIDATE | NOVALIDATE ] | DISABLE
CONSTRAINT Constraintname
;
```

verfügbar. Bei Deaktivierung kann am Ende CASCADE ergänzt werden, um abhängige Constraints ebenfalls zu deaktivieren.

Für als DEFERRABLE definierte Constraints kann ihr Status geändert werden mit

```
SET CONSTRAINT Constraintname
ALL | IMMEDIATE | DEFERRED;
```

CONSTRAINT – Status ändern



- **Statusänderung:**

```
ALTER TABLE [Schema.]Tabelle
MODIFY CONSTRAINT Constraint Constraintstatus
[ EXCEPTIONS INTO exceptionstabelle ];
```

- **EXCEPTIONS Tabelle anlegen:**

- **Skript:** %ORACLE_HOME%\rdbms\admin\utlexcpt.sql

- **Aktivierung/Deaktivierung:**

```
ALTER TABLE [Schema.]Tabelle
ENABLE | DISABLE [ VALIDATE | NOVALIDATE ]
CONSTRAINT Constraint
;
```

Für einen, mehrere oder alle Constraints für die Dauer einer Transaktion oder

ALTER SESSION

SET CONSTRAINT[S] = IMMEDIATE | DEFERRED | DEFAULT;

für alle Constraints für die Dauer der Session. DEFAULT entspricht der als INITIALLY angegebenen Option im CREATE TABLE Befehl.

Beispiel:

Exceptionstabelle anlegen:

```
SQL> @%ORACLE_HOME%\RDBMS\admin\utlexcpt.sql
```

FOREIGN KEY Constraint der Tabelle emp auf die Tabelle dept (DEPTNO zu DEPTNO) ausschalten (DISABLE). Funktioniert genauso gut wenn man die Constraints löscht.

```
SQL> ALTER TABLE emp DISABLE CONSTRAINT fk_deptno;
```

Datensatz einfügen, welches Fremdschlüsselbeziehung verletzt (deptno = 77)

```
SQL> INSERT INTO emp (empno, ename, sal, deptno)
      VALUES (9999, 'Bejic', 1000, 77);
```

Nach dieser Änderung versuchen die Fremdschlüsselbeziehung wieder einzuschalten

```
SQL> ALTER TABLE emp ENABLE CONSTRAINT fk_deptno EXCEPTIONS into exceptions;
```

Klappt leider nicht, weil die Abteilung 77 nicht existiert.

FEHLER in Zeile 1:

ORA-02298: (MB2.FK_DEPTNO) kann nicht validiert werden – Übergeordnete Schlüssel nicht gefunden

Prüfen welche Datensätze Probleme verursachen, damit diese ggf. manuell repariert werden können.

```
SQL> SELECT
      ex.*,
      e.*
    FROM   emp e INNER JOIN exceptions ex
    ON e.rowid = ex.row_id;
```

ROW_ID	OWNER	TABLE_NAME	CONSTRAINT	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
AAAMqeAAEAAAANMAAO	SCOTT	EMP	FK_DEPTNO	9999	Bejic				1000		77

1 Zeile wurde ausgewählt.

7.6.8 Kaskadierende Constraints

Eine Spalte kann nicht gelöscht werden, wenn auf ihr ein Constraint liegt, der referenziert wird. Die Klausel CASCADE CONSTRAINTS bewirkt, dass neben der Spalte und dem darauf liegenden Constraint auch alle abhängigen Constraints mit gelöscht werden.

Beispiel:

```
SQL> ALTER TABLE emp  
      DROP COLUMN (empno) CASCADE CONSTRAINTS;  
  
SQL> ALTER TABLE dept  
      DROP COLUMN (deptno) CASCADE CONSTRAINTS;
```

Syntax:

```
ALTER TABLE [Schema.] Tabelle  
      DROP COLUMN (Spalte) CASCADE CONSTRAINTS;
```

Löschen von Spalten mit
CONSTRAINTs



▪ Syntax:

```
ALTER TABLE [Schema.] Tabelle  
      DROP COLUMN (Spalte) CASCADE CONSTRAINTS;
```

▪ Beispiel:

```
SQL> ALTER TABLE emp  
      DROP COLUMN (empno) CASCADE CONSTRAINTS;  
SQL> ALTER TABLE dept  
      DROP COLUMN (deptno) CASCADE CONSTRAINTS;
```


8

Datenschutz und DCL-Befehle

8.1	Konzept	8-3
8.2	Datenbankbenutzer und SCHEMA	8-5
8.3	Benutzerverwaltung	8-7
8.4	Befehle zur Benutzerverwaltung	8-9
8.4.1	ALTER USER Befehl	8-13
8.4.2	DROP USER Befehl	8-14
8.4.3	Data Dictionary Views zur Benutzerverwaltung	8-15
8.5	Privilegien	8-16
8.5.1	Systemprivilegien	8-16
8.5.2	Objektprivilegien	8-18
8.6	Befehle zur Rechteverwaltung (DCL)	8-22
8.6.1	Der GRANT Befehl für Systemprivilegien	8-22
8.6.2	Der GRANT Befehl für Objektprivilegien	8-24
8.6.3	Der REVOKE Befehl	8-25
8.6.4	Das Rollenkonzept	8-28
8.6.5	Der CREATE ROLE Befehl	8-31
8.6.6	Der ALTER ROLE Befehl	8-32
8.6.7	Der SET ROLE Befehl	8-32
8.6.8	Der DROP ROLE Befehl	8-34
8.6.9	Setzen von Default Rollen	8-34
8.6.10	Vordefinierte Rollen	8-36

8.7	Benutzerprofile	8-39
8.7.1	Der CREATE PROFILE Befehl	8-41
8.7.2	Der ALTER PROFILE Befehl	8-44
8.7.3	Der DROP PROFILE Befehl	8-44
8.7.4	Data Dictionary Views für Profiles.....	8-45

8 Datenschutz und DCL-Befehle

8.1 Konzept

In Bezug auf Datenbanken bedeutet Datenschutz oder Security immer den Schutz der gespeicherten Daten vor unberechtigtem Zugriff durch Dritte. Unterschieden wird zwischen dem Zugriff auf die eigentliche Datenbank und dem Zugriff auf bestimmte Daten. In der Datenbank existieren dazu verschiedene Benutzer, wobei jeder Benutzer den Zugriff auf seine Objekte bekommt. Gleichzeitig soll es aber auch Benutzern ermöglicht werden, Zugriff auf andere Daten und Datenstrukturen zu bekommen, die ihnen nicht gehören. Der Datenschutz unter ORACLE wird über verschiedene Mechanismen ermöglicht:

- **Zugriff auf die Datenbank**

Der Zugriff auf die Datenbank erfolgt über eine gültige Kombination von Benutzername und Passwort. Alle gültigen Namen sind mit einem verschlüsselten Passwort in der Datenbank abgelegt.

- **Zugriff auf Objekte**

Durch erfolgreiches Einloggen bekommt der Benutzer nun Zugriff auf alle ihm gehörenden Objekte. Die Sammlung der ihm gehörenden Objekte nennt man auch SCHEMA.

- **Zugriff auf fremde Objekte**

Zudem bekommt der Benutzer Zugriff auf alle Objekte, für die ihm Rechte übertragen worden sind.

- **Zugriff auf Systemebene**

Die Benutzerrechte auf Systemebene ermöglichen dem Benutzer den eigentlichen Zugriff auf die Datenbank. Mehr als 100 verschiedene Privilegien können an die einzelnen Benutzer vergeben werden. Dazu gehört beispielsweise die Benutzung bestimmter SQL-Kommandos.

Diese Rechte können entweder für den einzelnen Benutzer oder für ganze Gruppen von Benutzern (Rollen) vergeben werden. So kann eine Gruppe von Rechten mit einem einzelnen Kommando an eine Gruppe von Benutzern vergeben werden. Weiterhin können auch bestimmte Rechte, die z.B. für eine bestimmte Anwendung benötigt werden, zusammengefasst und mit einem einzelnen Kommando an einen Benutzer vergeben werden.

Konzept



- Gewährleistung nur berechtigter Zugriffe auf die Datenbank
 - Eingabe einer gültigen Benutzername/ Passwort-Kombination
- Zugriff auf eigene Objekte
 - Nach gültigem Login hat der Benutzer Zugriff auf sein SCHEMA (Sammlung ihm gehörender Objekte)
- Zugriff auf fremde Objekte
 - Vergabe von Rechten auf Objektebene

2.5.0122 © Integrata Cegos GmbH

ORACLE und SQL

2

Konzept



- Zugriff auf Systemebene
 - ermöglichen die Durchführung von DB-Operationen (SQL-Kommandos)
 - Vergabe von Rechten für jeden einzelnen Benutzer
 - Vergabe von Rechten über Rollen
- Profile
 - Beschränkung von Ressourcen für einen Benutzer/ Benutzer-Session

2.5.0122 © Integrata Cegos GmbH

ORACLE und SQL

3

8.2 Datenbankbenutzer und SCHEMA

Ohne eine gültige Benutzername/Passwort Kombination ist kein Zugriff auf eine Datenbank möglich. Die gültigen Benutzer sind innerhalb der Datenbank gespeichert. Die Verwaltung dieser Benutzer erfolgt über Strukturen innerhalb der Datenbank. Definiert werden Benutzer über SQL-Befehle. Die Informationen zu den einzelnen Benutzern sind im Data Dictionary der Datenbank abgelegt.

Wie schon zu Beginn erwähnt, werden die Objekte eines Benutzers über das SCHEMA verwaltet, wobei SCHEMA eine logische Sammlung von Objekten darstellt, die zu einem Benutzer gehören. Wird ein neuer Benutzer angelegt, so wird auch automatisch das dazugehörige SCHEMA angelegt. Jeder Benutzer kann nur ein SCHEMA besitzen, wobei die Verknüpfung der beiden über den Namen erfolgt, d.h. SCHEMA und Benutzer haben den gleichen Namen.

Loggt sich ein Benutzer ein, so wird automatisch die Verbindung zum SCHEMA aufgebaut und der Benutzer hat Zugriff zu allen ihm gehörenden Objekten.

SCHEMA und Benutzer werden häufig mit der gleichen Bedeutung benutzt.

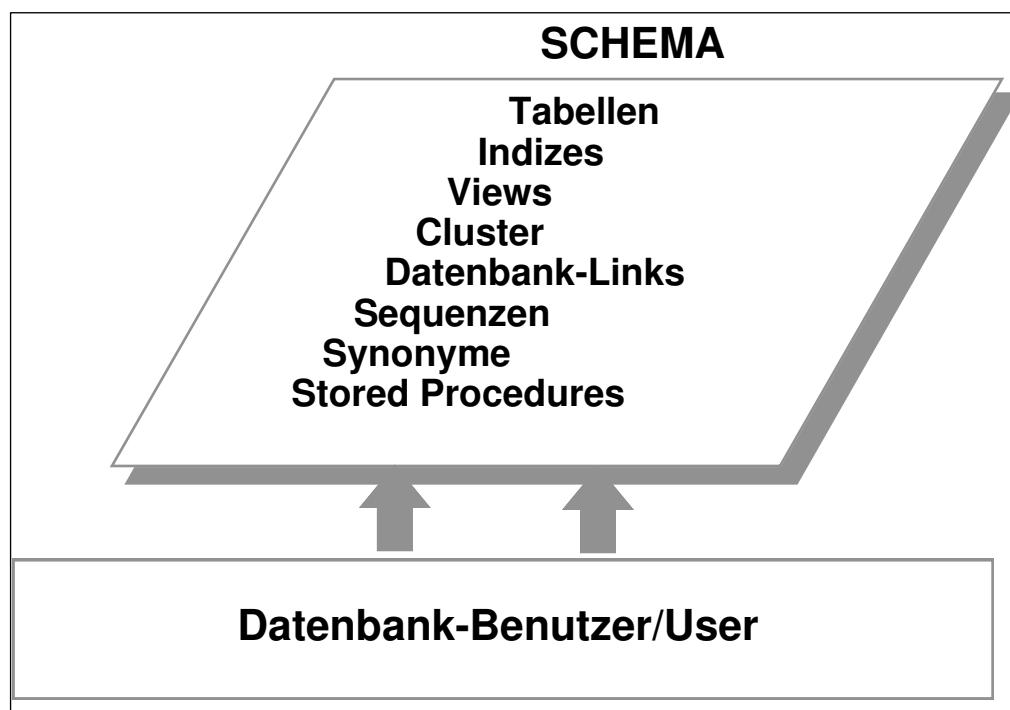


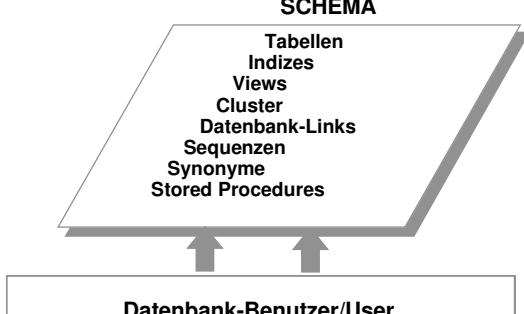
Abb. 8-1: Benutzer / SCHEMA

Datenbankbenutzer und Schema



- SCHEMA enthält alle Objekte, die zu einem Benutzer gehören
- SCHEMA und Benutzer haben den gleichen Namen
Das SCHEMA wird automatisch beim Anlegen eines neuen Benutzers erzeugt

SCHEMA



Tabellen
Indizes
Views
Cluster
Datenbank-Links
Sequenzen
Synonyme
Stored Procedures

Datenbank-Benutzer/User

2.5.0122 © Integrata Cegos GmbH ORACLE und SQL 4

8.3 Benutzerverwaltung

Die Verwaltung des Benutzers findet über die Security-Domäne statt. Diese umfasst alle Informationen, Rechte und Privilegien von jedem einzelnen Benutzer.

Die erste und wichtigste Information über den Benutzer ist seine Berechtigung, auf eine Datenbank zuzugreifen (Benutzername/ Passwort). Die Überprüfung des Passworts erfolgt im Normalfall auf ORACLE-Ebene. Benutzername und Passwort werden im Data Dictionary gespeichert. Die Überprüfung kann aber auch auf Systemebene (EXTERNALLY) über den Login-Namen des Benutzers erfolgen.

Benutzerverwaltung										
<ul style="list-style-type: none">▪ Security Domäne: beinhaltet alle Infos, Rechte und Privilegien eines Benutzers▪ Die Zugriffsberechtigung erfolgt über Benutzername und Passwort										
SECURITY Domäne										
<table border="1" style="width: 100%;"><tr><td style="width: 50%;">Benutzername/ Passwort</td><td style="width: 50%;">Zugriff auf Tablespaces</td></tr><tr><td>Privilegien und Rollen</td><td>Quoten für Tablespaces</td></tr><tr><td>Resource Limits</td><td>temporäre Tablespaces</td></tr><tr><td></td><td>default Tablespace</td></tr></table>		Benutzername/ Passwort	Zugriff auf Tablespaces	Privilegien und Rollen	Quoten für Tablespaces	Resource Limits	temporäre Tablespaces		default Tablespace	
Benutzername/ Passwort	Zugriff auf Tablespaces									
Privilegien und Rollen	Quoten für Tablespaces									
Resource Limits	temporäre Tablespaces									
	default Tablespace									
2.5.0122 © Integrata Cegos GmbH		ORACLE und SQL								
		5								

Folgende Bereiche gehören weiterhin zur Security-Domäne:

- Accessible Tablespaces
Tablespaces, in denen der Benutzer Speicherplatz anfordern kann.
- Quotas für accessible Tablespaces
Wie viel maximalen Platz darf der Benutzer in den einzelnen Tablespaces belegen. Mit QUOTA UNLIMITED kann der Benutzer den gesamten zur Verfügung stehenden Speicherplatz nutzen.
- Default Tablespace,
in dem alle Objekte abgelegt werden, sofern nicht explizit ein anderer Tablespace beim Erstellen angegeben wird. Ein Benutzer sollte in diesem Tablespace auch eine Quota besitzen.
- Temporärer Tablespace,
in dem die temporären Segmente des Benutzers abgelegt werden können.
- Welche Kommandos darf der Benutzer ausführen? Auf welche fremden Objekte darf er zugreifen (Privilegien und Rollen)?

Resource Limits für den Benutzer können über ein Profil gesetzt werden.

SECURITY Domäne	
Benutzername/ Passwort	Zugriff auf Tablespaces
Privilegien und Rollen	Quoten für Tablespaces
Resource Limits	temporäre Tablespaces
	default Tablespace

Abb. 8-2: Security Domäne

8.4 Befehle zur Benutzerverwaltung

Die im Folgenden aufgeführten SQL-Befehle dienen zur Verwaltung der Security-Domäne.

Um diese Befehle ausführen zu können, braucht man als Benutzer bestimmte Privilegien, die i. A. dem DBA vorbehalten sind.

CREATE/ ALTER/

DROP USER Mit diesen Befehlen werden Benutzer angelegt, verändert oder gelöscht. Mit dem CREATE USER-Befehl wird ein neuer Benutzer (und natürlich auch sein SCHEMA) angelegt und mit einem Passwort versehen. Mit ALTER USER kann man dem Benutzer ein neues Passwort vergeben. Des Weiteren können mit beiden Befehlen Default und Temporary Tablespace definiert werden und Quotas (limited / unlimited) auf diverse Tablespace gesetzt werden. Mit DROP USER wird ein bestehender Benutzer komplett aus der Datenbank gelöscht.

GRANT Mit diesem Befehl können Privilegien an einen Benutzer vergeben werden. Man unterscheidet Objekt- und Systemprivilegien. Ohne diese Privilegien, insbesondere Systemprivilegien, darf ein Benutzer nichts tun. Hat er kein CREATE SESSION-Privileg, so darf er sich nicht einmal in die Datenbank einloggen (Fehlermeldung: ORA-01045: user scott lacks CREATE SESSION privilege; logon denied).

REVOKE Mit diesem Befehl können Privilegien (Objekt und System), die mit GRANT vergeben worden sind, dem Benutzer wieder entzogen werden.

Befehle zur Benutzerverwaltung	
<ul style="list-style-type: none"> ▪ CREATE USER <ul style="list-style-type: none"> ▪ legt einen Benutzer an ▪ Benutzername/ Passwort wird gesetzt ▪ Default und Temporary Tablespaces definieren ▪ Quotas setzen ▪ ALTER USER <ul style="list-style-type: none"> ▪ anderes Passwort für den Benutzer vergeben ▪ Default und Temporary Tablespaces definieren ▪ Quotas setzen ▪ DROP USER <ul style="list-style-type: none"> ▪ Löschen eines Benutzers aus der DB ▪ GRANT <ul style="list-style-type: none"> ▪ Vergeben von Objekt- und Systemprivilegien ▪ REVOKE <ul style="list-style-type: none"> ▪ Entziehen von Objekt- und Systemprivilegien 	

2.5.0122 © Integrata Cegos GmbH

ORACLE und SQL

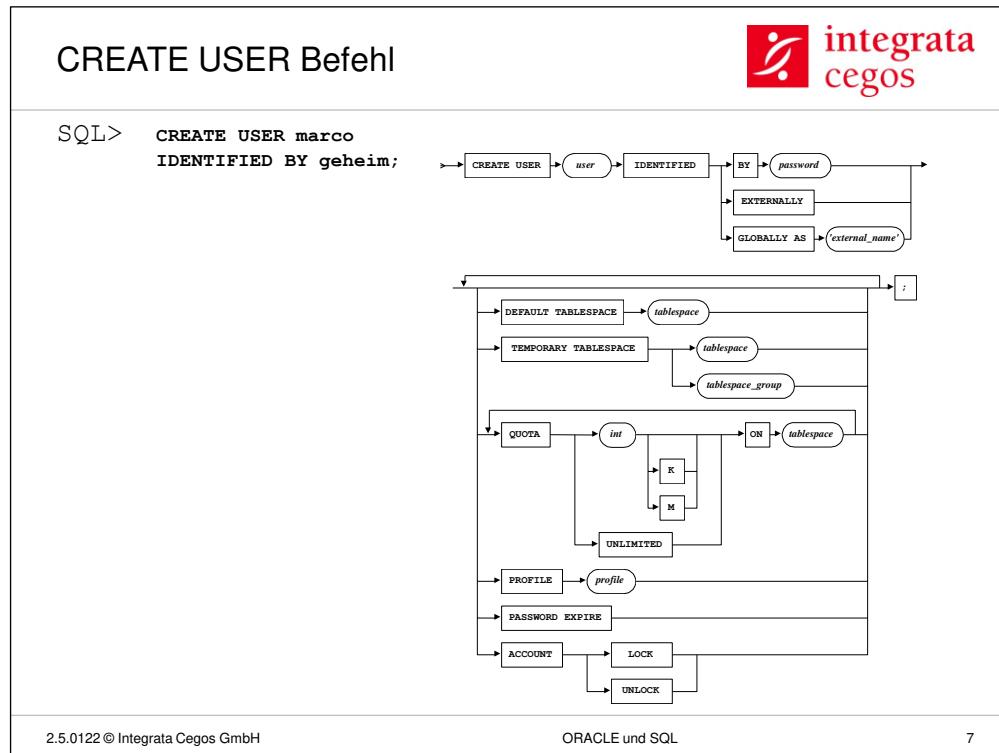
6

Der CREATE USER Befehl

Mit diesem Befehl kann ein neuer Benutzer angelegt werden. Erst danach können weitere Privilegien vergeben werden oder der Benutzer sich an die Datenbank anmelden.

- Es werden ein Name und ein gültiges Passwort für den Benutzer vergeben. Es sei denn, die Zugangsberechtigung wird außerhalb von ORACLE durchgeführt; in diesem Fall wird IDENTIFIED EXTERNALLY angegeben. Des Weiteren wird natürlich das SCHEMA des Benutzers angelegt.
- Default und temporary Tablespaces können definiert werden, wobei der Default Tablespace für den Benutzer eine limited/ unlimited Quota besitzen sollte. Für den Temporary Tablespace ist dies nicht nötig, da die temporären Segmente vom System aufgebaut werden. Ohne Angabe im CREATE USER-Befehl ist der Default vor Oracle 9i für beide Tablespaces der "SYSTEM"-Tablespace. Ab Oracle 9i ist das Default Tablespace USERS-Tablespace. Default temporäres Tablespace seit Version 9i ist TEMP-Tablespace.
- Mit QUOTA kann der Zugriff auf einen Tablespace erlaubt werden. Ein Limit kann angegeben werden in Bytes, Kilobytes(K) oder in Megabytes (M). Es sind auch Gigabytes (G) und Terabytes (T) als Einheit erlaubt. Soll ein Tablespace ohne Limit benutzt werden, so ist UNLIMITED zu setzen. Die QUOTA-Klausel kann mehrfach gesetzt werden, um den Zugriff auf mehrere Tablespaces zu definieren.

- Ein bereits bestehendes Profil kann dem neuen Benutzer mit der Klausel PROFILE zugeordnet werden.



Der CREATE USER-Befehl in seiner kürzesten Form sieht folgendermaßen aus:

```
CREATE USER Username IDENTIFIED BY Password;
```

```
SQL> CREATE USER marco
      IDENTIFIED BY sageheim#2018;
```

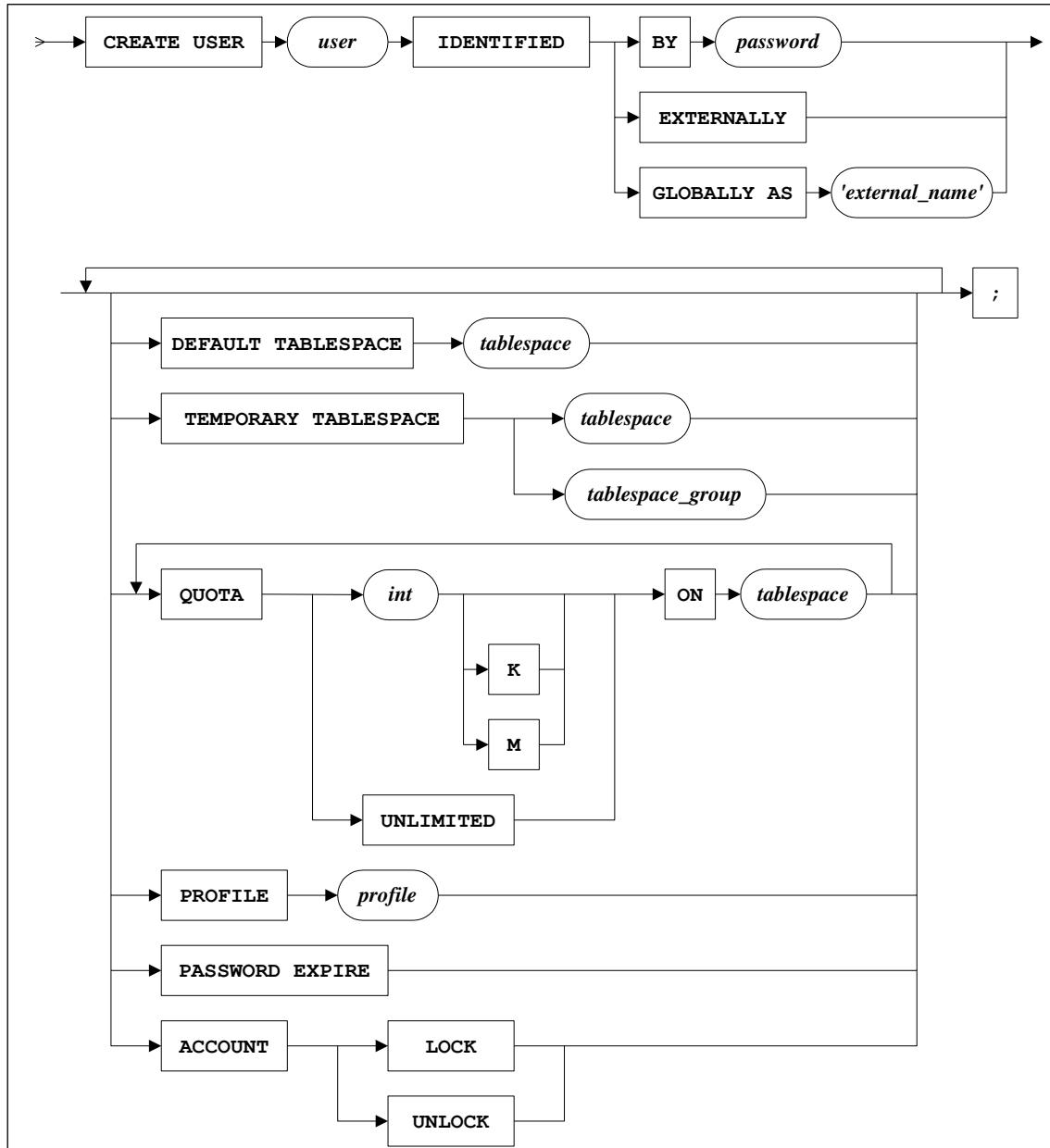


Abb. 8-3: Syntaxdiagramm des CREATE USER Befehls

8.4.1 ALTER USER Befehl

Mit diesem Befehl können Benutzerdefinitionen verändert werden. Die Syntax ist im Wesentlichen identisch zum CREATE USER-Befehl, nur ist z.B. IDENTIFIED BY nicht mehr obligatorisch.

Beispiel:

```
SQL> ALTER USER scott
      DEFAULT TABLESPACE      ts_2
      QUOTA UNLIMITED ON      ts_2;
```

Bei diesem Beispiel wurde dem Benutzer scott ein weiterer Tablespace zugeordnet, der auch gleichzeitig zum Default-Tablespace wurde. Diesen Befehl kann nur ein Benutzer mit DBA-Rechten absetzen.

Oft wird der Befehl zum Ändern des Passwortes benutzt:

Beispiel:

```
SQL> ALTER USER system IDENTIFIED BY imogdi;
```

Der Benutzer SYSTEM bekommt ein neues Passwort (für einen Bayern leichter zu merken!). Anders als z.B. unter UNIX findet keine Prüfung des vorhergehenden Passwortes statt. Diese Änderung kann auch vom Benutzer selbst (oder von einem Benutzer mit DBA-Rechten) durchgeführt werden.

Nachdem ein neuer Benutzer angelegt wurde (ohne Rechtevergabe) ist die Änderung seines Passworts das einzige, was ein User darf!

<p style="text-align: center;">ALTER USER Befehl</p> <pre>SQL> ALTER USER scott DEFAULT TABLESPACE ts_2 QUOTA UNLIMITED ON ts_2; SQL> ALTER USER system IDENTIFIED BY imogdi;</pre> <p style="margin-left: 40px;">▪ Syntax analog zu CREATE USER ▪ Alle Optionen aus CREATE USER Befehl können auch hier geändert werden</p>	 integrata cegos
<small>2.5.0122 © Integrata Cegos GmbH</small>	<small>ORACLE und SQL</small>
<small>8</small>	

8.4.2 DROP USER Befehl

Mit diesem Befehl wird ein Benutzer komplett aus der Datenbank gelöscht.

Mit der Klausel **CASCADE** werden außerdem alle noch existierenden Objekte zu diesem Benutzer gelöscht. Ohne die Angabe von CASCADE kann der Benutzer nur gelöscht werden, wenn keine zu dem Benutzer gehörigen Objekte mehr existieren. Ein Benutzer kann nicht gelöscht werden, solange er noch eingeloggt ist.

Beispiel 1:

```
SQL> DROP USER feldbusch;
```

Dieser Benutzer kann nur gelöscht werden, wenn keine ihm zugehörigen Objekte mehr existieren. Ansonsten erfolgt eine Fehlermeldung des Systems, und der Vorgang wird abgebrochen.

Beispiel 2:

```
SQL> DROP USER feldbusch CASCADE;
```

Der Benutzer und alle im gehörenden Objekte werden gelöscht. Dies kann natürlich im Zweifelsfall relativ lange dauern.

DROP USER Befehl



- Löscht einen Benutzer aus der Datenbank
- Kann lange dauern

- Syntax:

```
drop-user ::=  
    DROP USER Username [ CASCADE ] ;
```

- Beispiele:

```
SQL> DROP USER feldbusch;  
SQL> DROP USER feldbusch CASCADE;
```

8.4.3 Data Dictionary Views zur Benutzerverwaltung

Informationen über die angelegten Benutzer können mit folgenden Data Dictionary Views überwacht werden:

USER_USERS, ALL_USERS, DBA_USERS

DBA_USERS zeigt Benutzername, Passwort, Benutzer_id, Default und Temporary Tablespaces und das Profil aller Benutzer an und wann die Benutzer angelegt wurden.

ALL_USERS enthält nur Teile dieser Angaben.

USER_USERS zeigt nur die Daten für den aktuellen Benutzer an.

USER_TS_QUOTAS, DBA_TS_QUOTAS

zeigt an, welcher Benutzer auf welchen Tablespace über welche QUOTA (Angabe in Bytes und in Blöcken MAX_BLOCKS) verfügt und wie viel er davon belegt.

Hinweis: Auf die Views der DBA-Gruppe haben nur Benutzer mit DBA-Rechten Zugriff. Die Views der USER-Gruppe enthalten nur Informationen über den aktuell eingeloggten Benutzer.

Hinweis: Die QUOTA wird intern in der Datenbank in ORACLE-Blöcken (BLOCKS) abgelegt. Steht die QUOTA auf -1, so bedeutet das UNLIMITED.

Data Dictionary Views	
	
<ul style="list-style-type: none">▪ DBA_USERS, USER_USERS, zeigt an:<ul style="list-style-type: none">▪ USERNAME▪ USER_ID▪ PASSWORD (verschlüsselt, nur in DBA_USERS)▪ DEFAULT_TABLESPACE▪ TEMPORARY_TABLESPACE▪ CREATED▪ PROFILE▪ TABLESPACE_NAME▪ ALL_USERS zeigt nur an:<ul style="list-style-type: none">▪ USERNAME▪ USER_ID▪ CREATED▪ USERS_TS_QUOTAS, DBA_TS_QUOTAS zeigt an:<ul style="list-style-type: none">▪ BYTES▪ MAX_BYTES▪ BLOCKS▪ MAX_BLOCKS▪ QUOTA = -1 → UNLIMITED	

8.5 Privilegien

Privilegien sind Rechte, die ein bestimmter Benutzer hat. Man unterscheidet zwischen Systemprivilegien und Objektprivilegien. Rechte mit dem String "ANY" im Namen sollten immer mit Vorsicht vergeben werden, da dies Superuser-Rechte sind. Für gewöhnlich hat nur ein DBA solche Rechte.

8.5.1 Systemprivilegien

Systemprivilegien regeln den Zugriff auf die Datenbank. Sie werden normalerweise vom DBA vergeben, der selbst hohe Systemprivilegien besitzt.

Unter ORACLE gibt es über 100 verschiedene Systemprivilegien. Der Einfachheit halber kann man einige Rechte zu sog. Rollen zusammenfassen, um den Verwaltungsaufwand zu minimieren.

Mit allen Systemrechten sollte man sehr vorsichtig umgehen, so dass der damit verursachbare Schaden möglichst gering ist. Je nach Aufgabe braucht aber jeder User zumindest ein paar Systemprivilegien. CREATE SESSION ist zwingend erforderlich, denn dieses Recht besagt, dass sich der Benutzer bei der Datenbank anmelden darf. Soll ein User eigene Objekte erstellen, braucht er das entsprechende CREATE-Recht dafür, z.B. CREATE TABLE, CREATE VIEW...

Hinweis: In den meisten Fällen reicht es, wenn für den Endbenutzer das Systemprivileg CREATE SESSION und die Zugriffsrechte (Objektprivilegien) auf die benötigten Objekte fremder Schemata vergeben werden.

Informationen über seine eigenen System-Privilegien bekommt der Benutzer z. B. über die Data Dictionary-View USER_SYS_PRIVS oder SESSION_PRIVS. Ein DBA bekommt Informationen über die DD-View DBA_SYS_PRIVS.

Hinweis: Eine komplette Liste der System-Privilegien finden Sie im DBA-Guide. Sie wurde hier aus "Platzgründen" weggelassen.

Systemprivilegien



- System-Privilegien:
 - Über 100 Systemprivilegien
 - Systemprivilegien können einzeln oder in Gruppen vergeben werden.
 - *ANY*-Privilegien sind immer Superuserprivilegien
 - Es sollten nur Rechte vergeben werden, die ein Benutzer auch wirklich benötigt!
 - Jeder Benutzer benötigt das CREATE SESSION-Recht, um auf die Datenbank zugreifen zu können
- Data Dictionary Views:
 - USER_SYS_PRIVS
 - SESSION_PRIVS
 - DBA_SYS_PRIVS

8.5.2 Objektprivilegien

Objektprivilegien regeln den Zugriff auf Objekte der Datenbank. Der Eigentümer eines Objekts hat automatisch alle Rechte auf dieses Objekt, und nur er kann Rechte auf seine Objekte an andere vergeben bzw. andere berechtigen, diese Rechte weiterzugeben. Welche Objektprivilegien vergeben werden können, hängt von der Art des Objekts ab.

Objekte können sein:

- Tabellen
- Views
- Sequenzen
- Prozeduren
- Materialized Views (früher: snapshots)
- Synonyme.

Objektprivilegien können sein:

ALL [PRIVILEGES]	es können alle Rechte an einem Objekt auf einmal vergeben werden.
ALTER	Verändern von Tabellen, Sequenzen und Erstellen von Triggern auf eine Tabelle.
DELETE	Löschen von Datensätzen in einer Tabelle/View.
EXECUTE	Ausführen einer Prozedur.
INDEX	Erzeugen eines Indexes auf eine Tabelle.
INSERT	Einfügen von Datensätzen in eine Tabelle/View.
REFERENCES	Referenzierung einer fremden Tabelle durch einen FOREIGN KEY.
SELECT	Lesen von Sätzen (Zeilen) aus einer Tabelle, View oder einem Snapshot, Zugriff auf eine Sequenz. SELECT FOR UPDATE für Tabellen und Views und LOCK TABLE
READ	Ab 12.1.0.2 kann diese Recht vergeben werden. Es umfasst das Lesen von Sätzen (Zeilen) aus einer Tabelle, View oder einem Snapshot.
UPDATE	Verändern von Datensätzen in einer Tabelle oder View.

Objekte und Objektprivilegien

Objekt-privilegien	Objekte				
	Tabelle	View	Sequenz	Prozedur	Materialized View
ALTER	X		X		
DELETE	X	X			
EXECUTE				X	
INDEX	X				
INSERT	X	X			
REFERENCES	X				
SELECT	X	X	X		X
READ	X	X			X
UPDATE	X	X			

Objektprivilegien		Objekte				
Objektprivilegien	Tabelle	View	Sequenz	Procedure	Materialized View	
	ALTER	X		X		
	DELETE	X	X			
	EXECUTE				X	
	INDEX	X				
	INSERT	X	X			
	REFERENCES	X				
	SELECT	X	X	X		X
	UPDATE	X	X			
	READ	X	X			X

2.5.0122 © Integrata Cegos GmbH ORACLE und SQL 12

Data Dictionary Views für Objektprivilegien

Über diese Data Dictionary Views können Informationen über Privilegien, die auf Objekte vergeben wurden, abgerufen werden:

- ALL_TAB_PRIVS, USER_TAB_PRIVS, DBA_TAB_PRIVS
- ALL_TAB_PRIVS_REC, USER_TAB_PRIVS_REC
- ALL_TAB_PRIVS_MADE, USER_TAB_PRIVS_MADE
- ALL_COL_PRIVS_MADE, USER_COL_PRIVS_MADE
- ALL_COL_PRIVS, USER_COL_PRIVS, DBA_COL_PRIVS
- ALL_COL_PRIVS_REC, USER_COL_PRIVS_REC

Kurzbeschreibung

ALL woher ist ein Recht „gekommen“, bzw. wohin ist ein Recht „gegangen“

DBA steht für alle Rechte in der Datenbank, kann nur vom DBA gelesen werden

TAB Objektprivilegien

COL alle Objektprivilegien, die sich nur auf bestimmte Spalten einer Tabelle beziehen

PRIVS Privilegien**MADE** alle Rechte, die ein Benutzer vergeben hat**RECD** alle Rechte, die ein Benutzer bekommen hat (Abkürzung für Received).

Data Dictionary Views für Objektprivilegien		
ALL_TAB_PRIVS	ALL_TAB_PRIVS_MADE	
USER_TAB_PRIVS	USER_TAB_PRIVS_MADE	
DBA_TAB_PRIVS		
ALL_TAB_PRIVS_RECV	ALL_COL_PRIVS	
USER_TAB_PRIVS_RECV	USER_COL_PRIVS	
	DBA_COL_PRIVS	
ALL_COL_PRIVS_MADE	ALL_COL_PRIVS_RECV	
USER_COL_PRIVS_MADE	USER_COL_PRIVS_RECV	

8.6 Befehle zur Rechteverwaltung (DCL)

Mit ihnen werden System- und Objektprivilegien erteilt und entzogen.

Zu den DCL-Befehlen (Data Control Language) gehören:

- **GRANT** vergibt Rechte
- **REVOKE** entzieht Rechte

Hinweis: Ebenso wie alle DDL-Befehle beinhalten auch die DCL-Befehle ein implizites COMMIT.

8.6.1 Der GRANT Befehl für Systemprivilegien

Ohne die Vergabe von Rechten kann ein Benutzer innerhalb der Datenbank nichts tun. Er kann sich noch nicht einmal anmelden.

Das CREATE SESSION Privileg z. B. ermöglicht dem Benutzer das Starten einer sog. Session. Erst durch dieses Privileg kann der Benutzer sich in die Datenbank einloggen.

Mit dem GRANT-Befehl kann einzelnen Usern, allen Usern oder einer Rolle ein Recht zugewiesen werden. Außerdem kann einzelnen oder allen Usern eine Rolle zugewiesen werden, die dann diverse Rechte beinhaltet.

Beispiel Systemprivilegien:

```
SQL> GRANT create session TO thomminator;
```

```
SQL> GRANT create session TO dba_hiwi  
      WITH ADMIN OPTION;
```

Durch den Zusatz **WITH ADMIN OPTION** hat der Benutzer dba_hiwi das Recht, das ihm vergebene Systemprivileg auch an andere Benutzer weiterzugeben.

Um Systemprivilegien zu vergeben, muss man das GRANT ANY PRIVILEGE-Recht (Berechtigung zur Vergabe aller Privilegien) haben, DBA sein oder genau für die zu vergebenden Rechte die WITH ADMIN OPTION haben.

PUBLIC bedeutet, dass das Privileg an alle User vergeben wird.

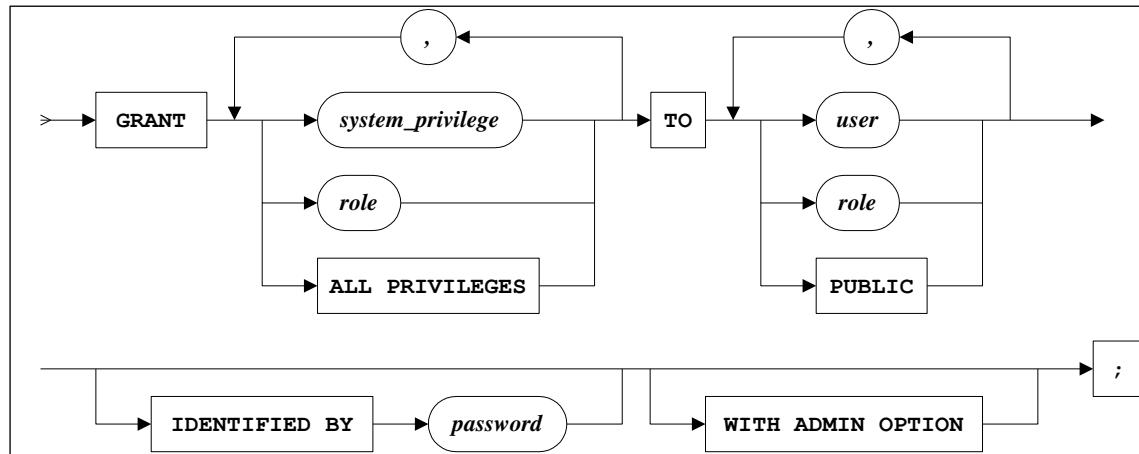


Abb. 8-4: GRANT Syntax für Systemprivilegien

GRANT Befehl für Systemprivilegien
 integrata
cegos

- GRANT Befehl vergibt System- und Objektprivilegien
- Syntax:

- Beispiel:

```

SQL> GRANT CREATE SESSION TO thomminator;
SQL> GRANT CREATE SESSION TO dba_hiwi
      WITH ADMIN OPTION;
    
```

2.5.0122 © Integrata Cegos GmbH ORACLE und SQL 14

8.6.2 Der GRANT Befehl für Objektprivilegien

Objektprivilegien beziehen sich, wie der Name schon sagt, auf Objekte der Datenbank.

Beispiel Objektprivilegien:

```
SQL> GRANT SELECT, UPDATE (job) ON emp
      TO scott, dba_hiwi
      WITH GRANT OPTION;
```

Hier wird das Recht an `scott` vergeben, die Inhalte der Spalte `job` in der Tabelle `EMP` zu selektieren und zu verändern. Er kann diese Rechte an andere Benutzer weitergeben durch die **WITH GRANT OPTION**.

Die Vergabe von Rechten auf Objektebene kann nur vom Eigentümer der Objekte oder von einem Benutzer durchgeführt werden, der diese Rechte mit der **WITH GRANT OPTION** bekommen hat.

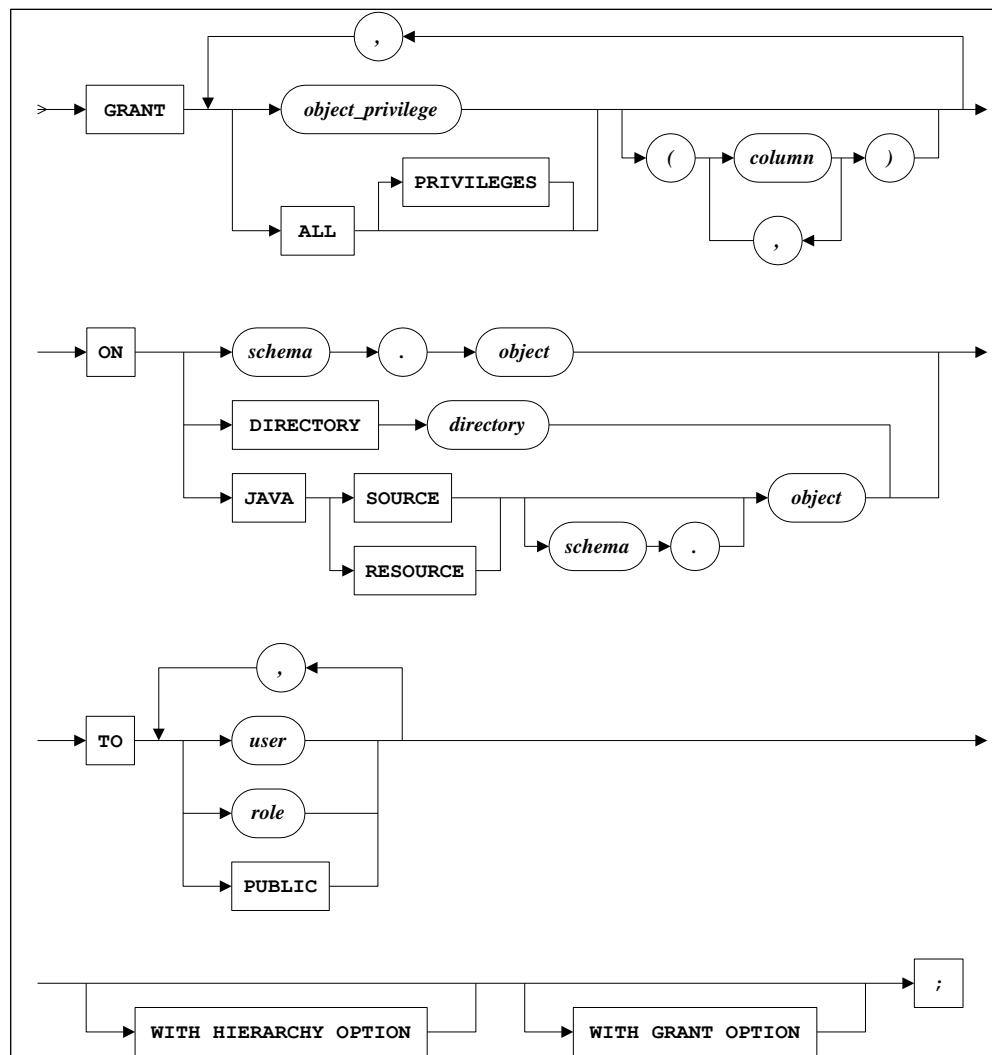
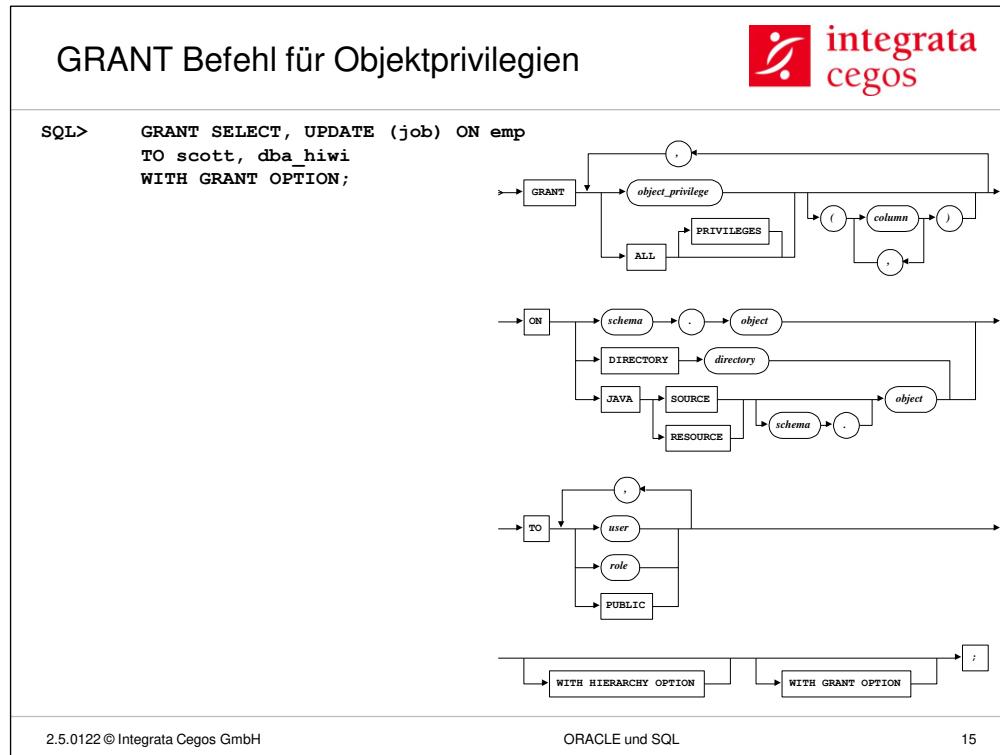


Abb. 8-5: GRANT Syntax für Objektprivilegien



8.6.3 Der REVOKE Befehl

Wie beim GRANT-Befehl können mit REVOKE sowohl Systemprivilegien als auch Objektprivilegien entzogen werden.

Alle Privilegien, die mit GRANT vergeben wurden, können mit REVOKE wieder entzogen werden.

8.6.3.1 REVOKE für Systemprivilegien

Beispiel für System-Privilegien:

```
SQL> REVOKE CREATE SESSION FROM praktikant;
```

Der Benutzer `praktikant` kann sich nun nicht mehr einloggen.

Um Systemprivilegien zu entziehen, muss man das GRANT ANY PRIVILEGE-Recht (Berechtigung zur Vergabe aller Privilegien) haben, DBA sein oder genau für die zu entziehenden Rechte die zu vergebenden Rechte mit der WITH ADMIN OPTION bekommen haben.

Der Entzug der Systemprivilegien wird nicht weitervererbt. Verändern sich die Rechte eines Benutzers (Grantor), der selbst an einen anderen Benutzer Rechte weitervergeben hat (Grantee), so bleiben die Rechte des Grantee erhalten, auch dann, wenn der Grantor seine Rechte verliert.

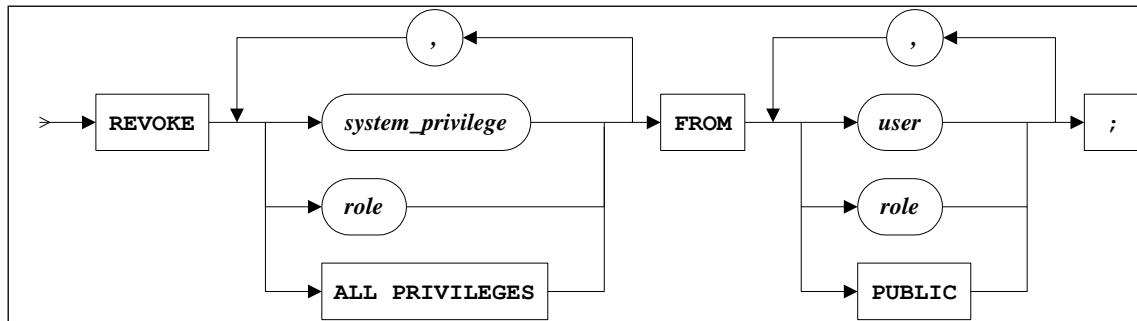


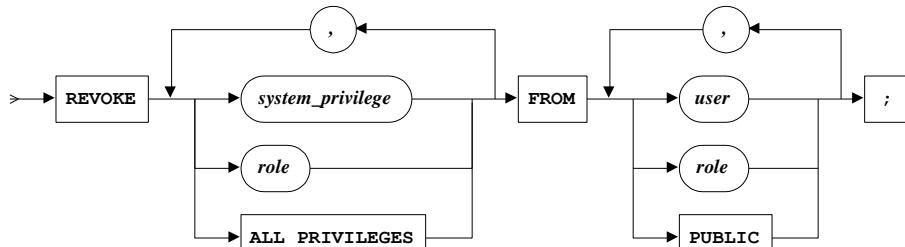
Abb. 8-6: REVOKE Syntax für Systemprivilegien

REVOKE Befehl für Systemprivilegien

- Entziehen der Privilegien
- Beispiel:

```
SQL> REVOKE CREATE SESSION FROM praktikant;
```

- Syntaxdiagramm des REVOKE Befehls für Systemprivilegien:



Das Diagramm ist identisch mit Abbildung 8-6, es zeigt die Struktur des REVOKE Befehls für Systemprivilegien.



2.5.0122 © Integrata Cegos GmbH

ORACLE und SQL

16

8.6.3.2 REVOKE für Objektprivilegien

Beispiel für Objekt-Privilegien:

```
SQL> REVOKE SELECT ON emp FROM praktikant;
```

Der Entzug von Objektprivilegien wird weitervererbt. Verliert ein Benutzer (Grantor) die Rechte, die er selbst an einen anderen Benutzer weitergegeben hat (Grantee), so verliert auch der Grantee diese Rechte.

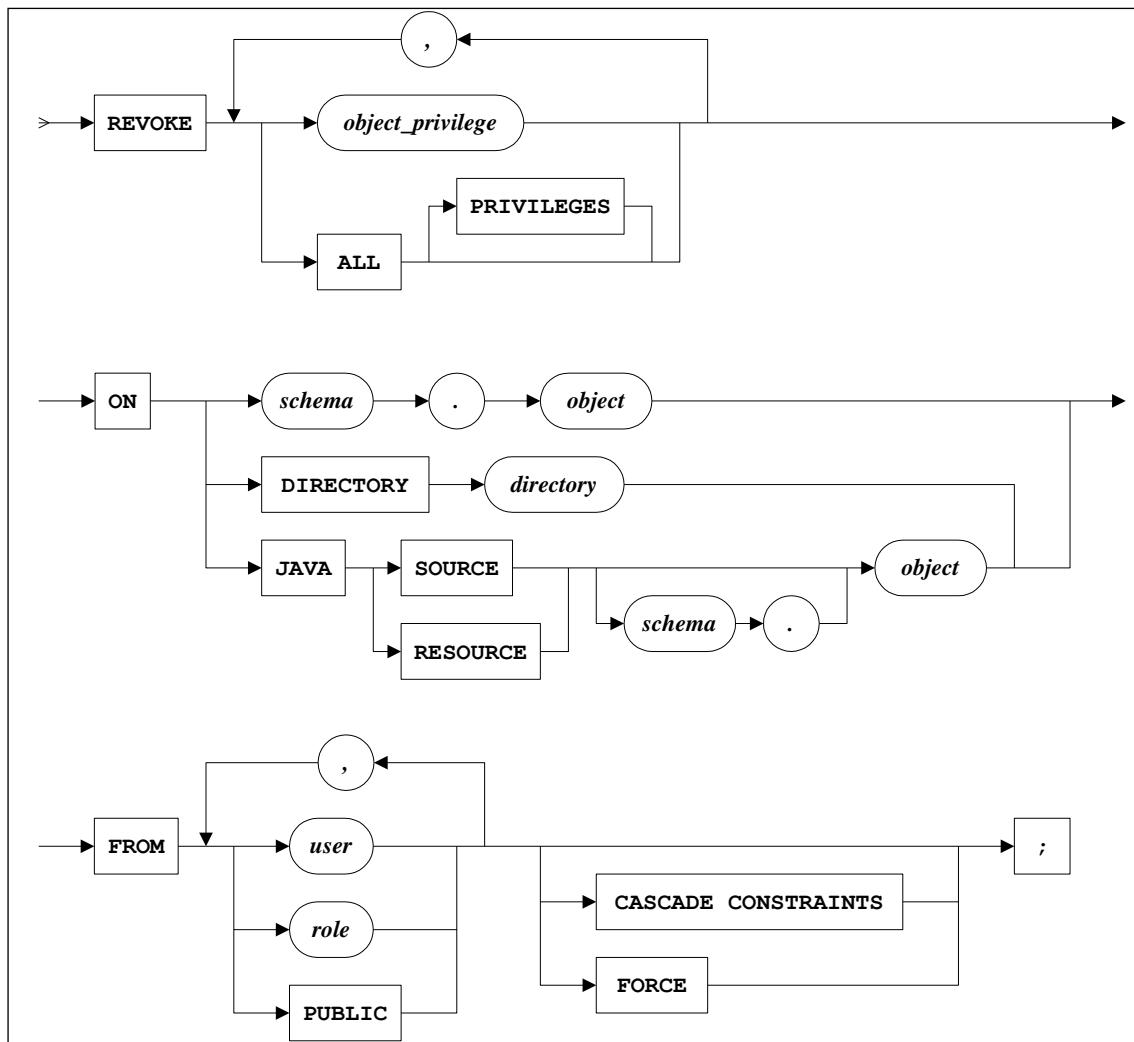


Abb. 8-7: REVOKE Syntax für Objektprivilegien

CASCADE CONSTRAINTS

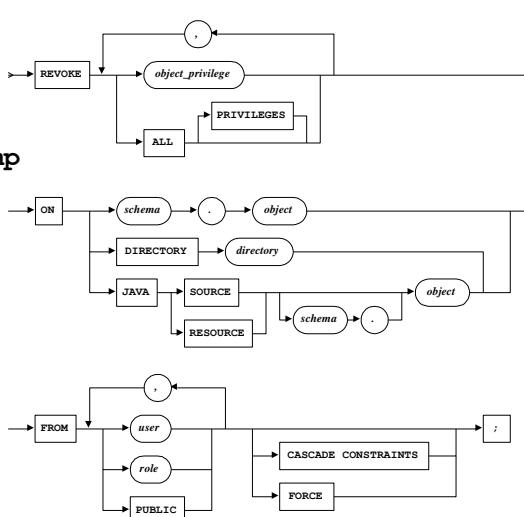
ist dann wichtig, wenn das Privileg REFERENCES zurückgenommen wird, da es alle auf diesem Recht basierenden FOREIGN KEY-Constraints löscht. Bei Nichtangabe dieser Klausel führt REVOKE unter Umständen zu einem Fehler, da diese Constraints nicht gelöscht werden.



REVOKE Befehl für Objektprivilegien

- Entziehen der Privilegien
- Beispiel:

```
SQL> REVOKE SELECT ON emp
      FROM praktikant;
```
- Syntaxdiagramm des REVOKE Befehls für Objektprivilegien:



2.5.0122 © Integrata Cegos GmbH ORACLE und SQL 17

8.6.4 Das Rollenkonzept

Um den Verwaltungsaufwand bei der Vergabe und dem Entzug von Rechten zu minimieren, gibt es die so genannten Rollen. Rollen sind im Prinzip System- und/ oder Objektprivilegien, die zu einer Gruppe zusammengefasst werden und unter einem Namen gemeinsam ansprechbar sind. Die Rollen werden in der Datenbank abgelegt.

Rollen können auch rekursiv auf anderen Rollen basieren. Es dürfen nur keine Schleifen entstehen, wie zum Beispiel: Rolle A basiert auf Rolle B und umgekehrt. Solch ein Phänomen wird allerdings auch von ORACLE abgefangen.

Eine erzeugte Rolle kann dann an einen Benutzer weitergegeben werden. Dies erfolgt in 3 Schritten:

- 1) Erzeugen der Rolle: Namensvergabe (und evtl. Passwort) und abspeichern im Data Dictionary.
- 2) Granten der Privilegien an die Rolle
- 3) Granten der Rolle an die Benutzer.

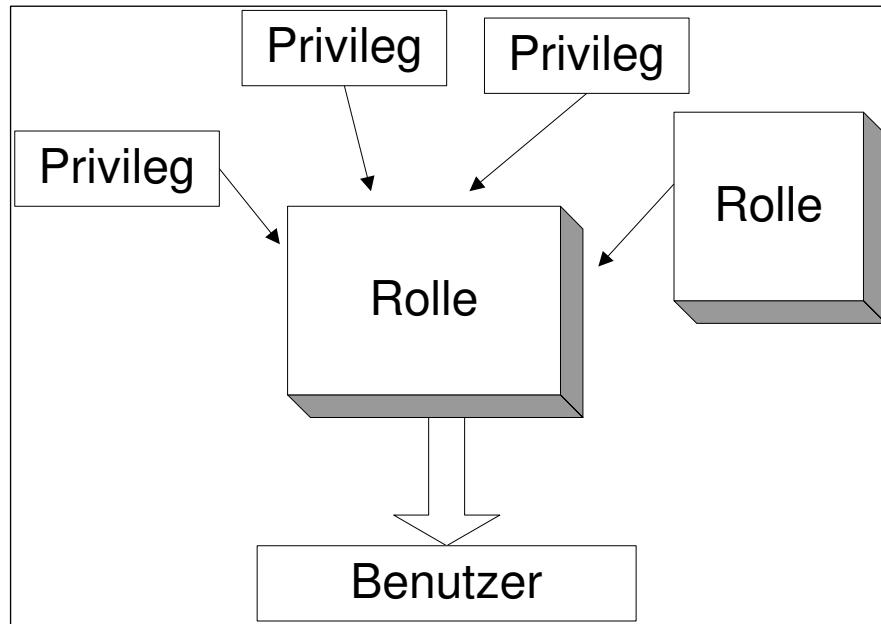


Abb. 8-8: Rollen

Sowohl System- als auch Objektprivilegien können in einer Datenbank-Rolle zusammengefasst werden. Es sollten allerdings solche Privilegien zusammengefasst werden, die auch inhaltlich in einer gewissen Beziehung zueinander stehen.

Das Rollenkonzept


- Erzeugen der Rolle:
Namensvergabe (und evtl. Passwort) und Abspeichern im Data Dictionary
- Granten der Privilegien an die Rolle
- Granten der Rolle an die Benutzer

```

graph TD
    P1[Privileg] --> R[Rolle]
    P2[Privileg] --> R
    P3[Privileg] --> R
    R --> B[Benutzer]
  
```

2.5.0122 © Integrata Cegos GmbH ORACLE und SQL 18

Beispiel:

- Zusammenfassen aller Privilegien, die für das Arbeiten mit einer Applikation notwendig sind.
- Zusammenfassen aller Systemprivilegien für einen Benutzer.

Rollen werden nicht explizit nur einem Benutzer zugeordnet sondern sind allgemein. Auch wird eine Rolle nicht gelöscht, wenn der Benutzer, der sie angelegt hat, aus der Datenbank gelöscht wird.

Rollen können daher auch an beliebig viele Benutzer oder andere Rollen vergeben werden.

Hat ein Benutzer eine Rolle zur Verfügung gestellt bekommen, so kann er diese Rolle durch den Befehl `SET ROLE` ein- und ausschalten. Für das Einschalten einer Rolle kann die Abfrage eines Passwortes eingebaut werden.

Vorteile einer Rolle:

- Vereinfachung der Security-Verwaltung
- Mehrere Benutzer haben genau die gleichen Rechte (Gruppen)
- Rechte können einfacher verändert werden, da nur die Rolle verändert werden muss und nicht die Rechte eines jeden Benutzers.

Rollen	
<ul style="list-style-type: none"> ▪ beinhalten Objekt- und Systemprivilegien ▪ sind allgemeingültig, nicht explizit einem Benutzer zugeordnet ▪ sollten eine inhaltliche Beziehung haben ▪ werden an Benutzer oder an andere Rollen vergeben ▪ Benutzer können eine Rolle ein- und ausschalten (<code>SET ROLE</code>) ▪ Vorteile von Rollen: <ul style="list-style-type: none"> ▪ Vereinfachung der Security-Verwaltung ▪ Mehrere Benutzer haben genau die gleichen Rechte (Gruppen) ▪ Rechte können einfacher verändert werden, da nur die Rolle verändert werden muss und nicht die Rechte eines jeden Benutzers 	

8.6.5 Der CREATE ROLE Befehl

Mit diesem Befehl kann eine Rolle angelegt werden. Der Name der Rolle muss innerhalb der Datenbank eindeutig sein. Der Benutzer, der sie anlegt, muss die entsprechenden Rechte besitzen (CREATE ROLE-Privileg oder ALTER ANY ROLE-Privileg oder GRANT mit WITH ADMIN OPTION).

Beispiel 1:

Anlegen einer Rolle ohne Zugriffsschutz

```
SQL> CREATE ROLE applik_1;
```

Beispiel 2:

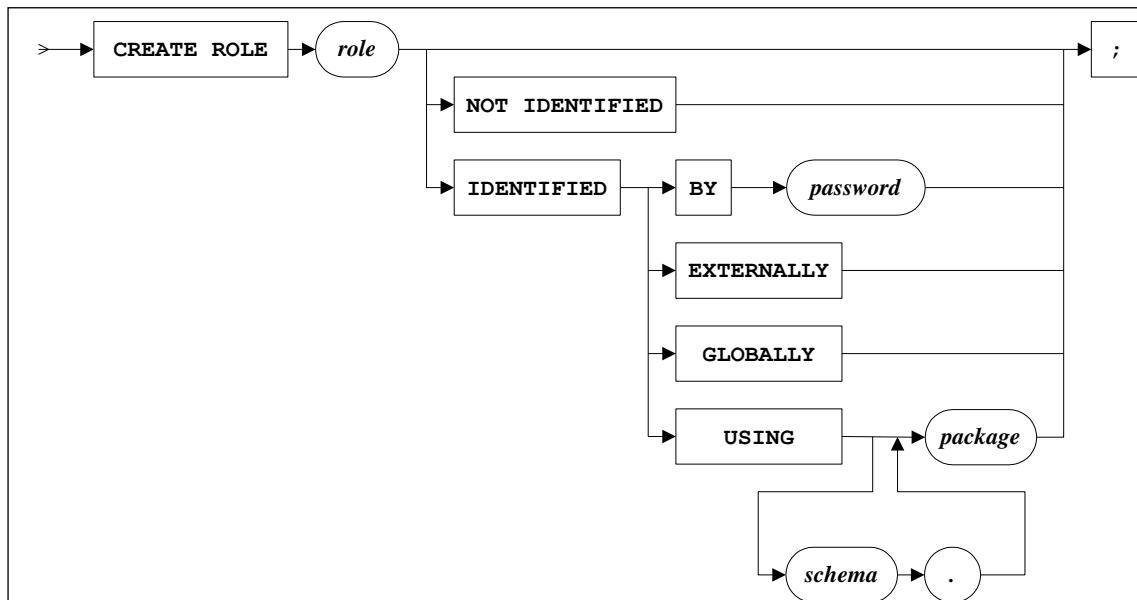
Anlegen einer Rolle mit Zugriffsschutz

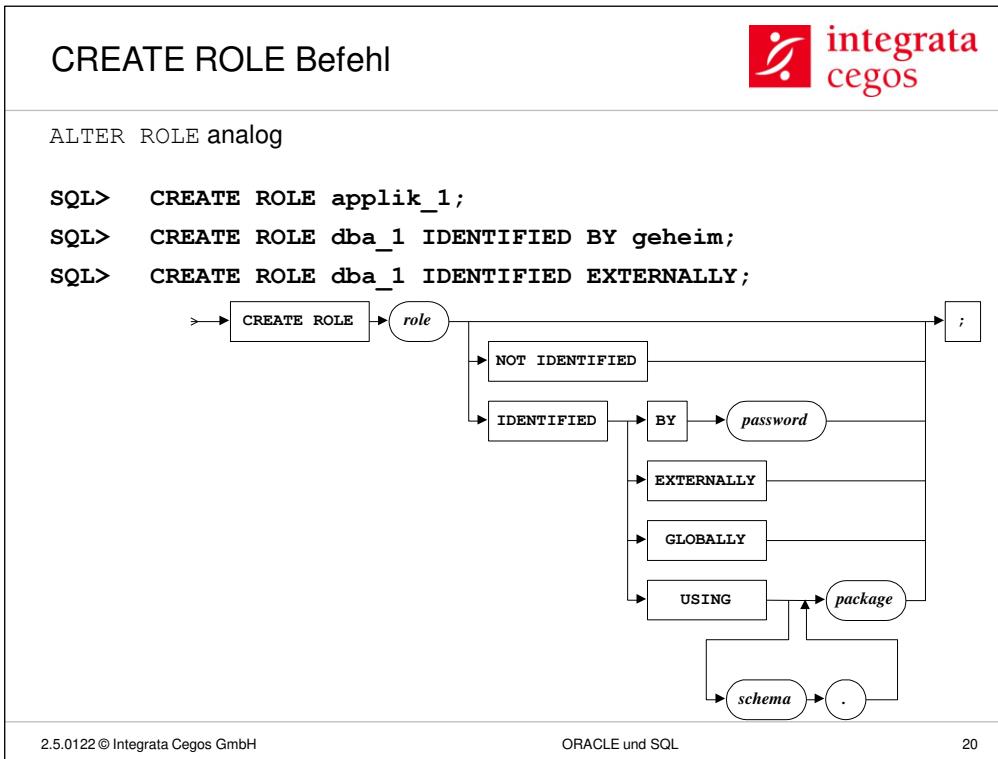
```
SQL> CREATE ROLE dba_1 IDENTIFIED BY geheim;
```

Beispiel 3:

Anlegen einer Rolle mit externer Identifizierung

```
SQL> CREATE ROLE dba_1 IDENTIFIED EXTERNALLY;
```





8.6.6 Der ALTER ROLE Befehl

Dieser Befehl hat die gleiche Syntax wie der CREATE ROLE und wird daher nicht genauer beschrieben.

8.6.7 Der SET ROLE Befehl

Ein Benutzer sollte immer nur die Rechte zur Verfügung haben, die er für seine aktuelle Arbeit benötigt. So können weitestgehend Fehler, wie z.B. Datenverlust, vermieden werden.

Werden einem Benutzer Privilegien direkt zugeordnet, so kann er diese nicht ohne weiteres abschalten. Benutzt man Rollen, so kann der Benutzer selbst die benötigten Rechte setzen und zurücknehmen. Dies geschieht über den SET ROLE Befehl. Es werden immer nur die angegebenen Rollen aktiviert und alle bestehenden ausgeschaltet.

Beispiel 1:

```
SQL> SET ROLE app_1 IDENTIFIED BY hugo;
```

Beispiel 2:

```
SQL> SET ROLE ALL EXCEPT app_1;
```

Hier werden nun alle Rollen aktiviert außer app_1. Benutzt man die ALL-Klausel, so kann kein Passwort angegeben werden, außer das Passwort wird extern abgefragt.

Beispiel 3:

```
SQL> SET ROLE NONE;
```

Alle Rollen werden ausgeschaltet.

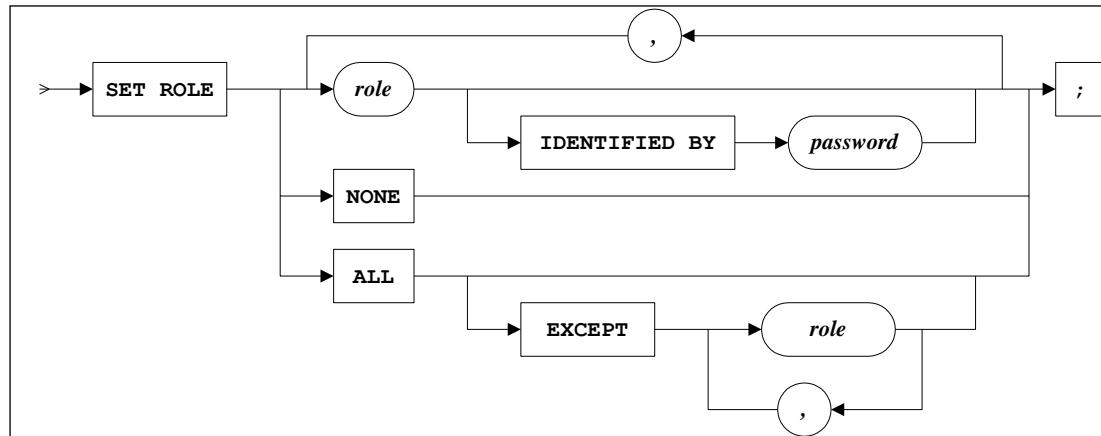
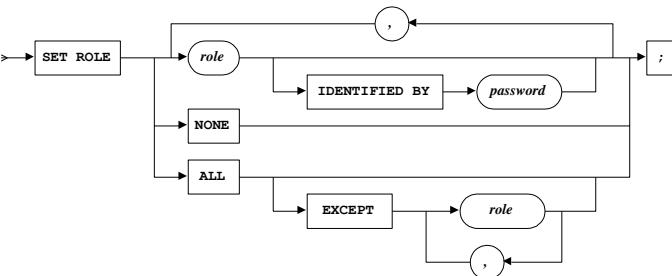


Abb. 8-9: SET ROLE Syntax

SET ROLE	 integrata cegos	
<pre>SQL> SET ROLE app_1 IDENTIFIED BY hugo; SQL> SET ROLE ALL EXCEPT app_1; SQL> SET ROLE NONE;</pre>		
<ul style="list-style-type: none"> ▪ Rolle löschen <pre>SQL> DROP ROLE app_1;</pre>		
		
2.5.0122 © Integrata Cegos GmbH	ORACLE und SQL	21

8.6.8 Der `DROP ROLE` Befehl

Wenn eine Rolle gelöscht wird, so entfernt ORACLE diese Rolle von allen Benutzern und Rollen, denen diese Rolle zugewiesen wurde und löscht sie danach aus der Datenbank.

8.6.9 Setzen von Default Rollen

Für die typischen Arbeiten eines Benutzers kann man eine oder mehrere Default Rollen setzen. Diese Default Rollen werden mit dem `ALTER USER` Befehl aktiviert. Mit dem `CREATE USER` Befehl ist dies nicht möglich.

Beispiel:

```
SQL> ALTER USER thomminator DEFAULT ROLE app_1;
```

Ein Passwort wird beim Setzen der `DEFAULT ROLE` nicht abgefragt.

Die maximale Anzahl von aktiven Rollen in einer Session wird durch den INIT.ORA-Parameter `MAX_ENABLED_ROLES` gesetzt.

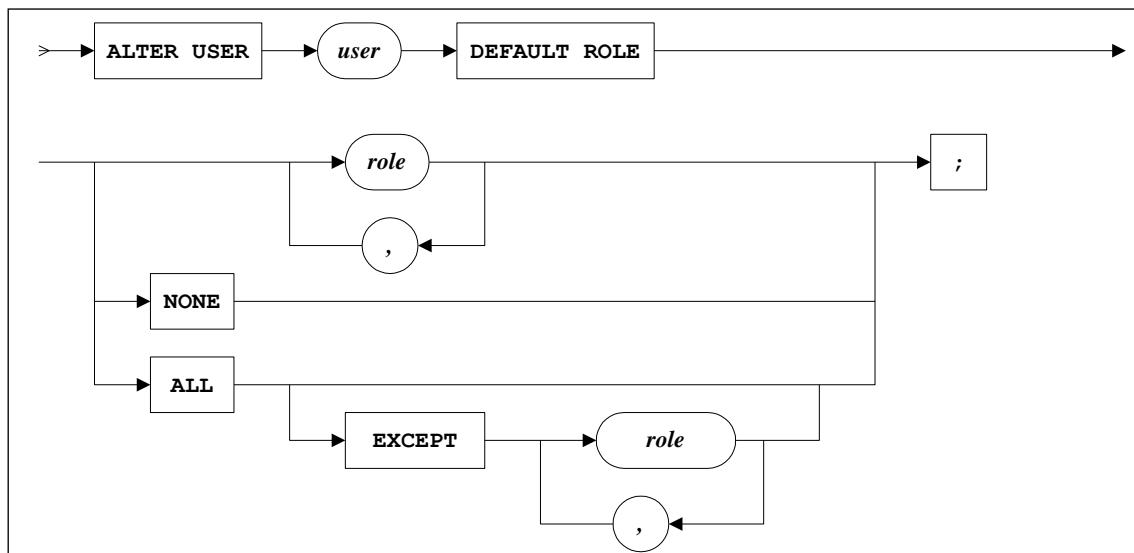
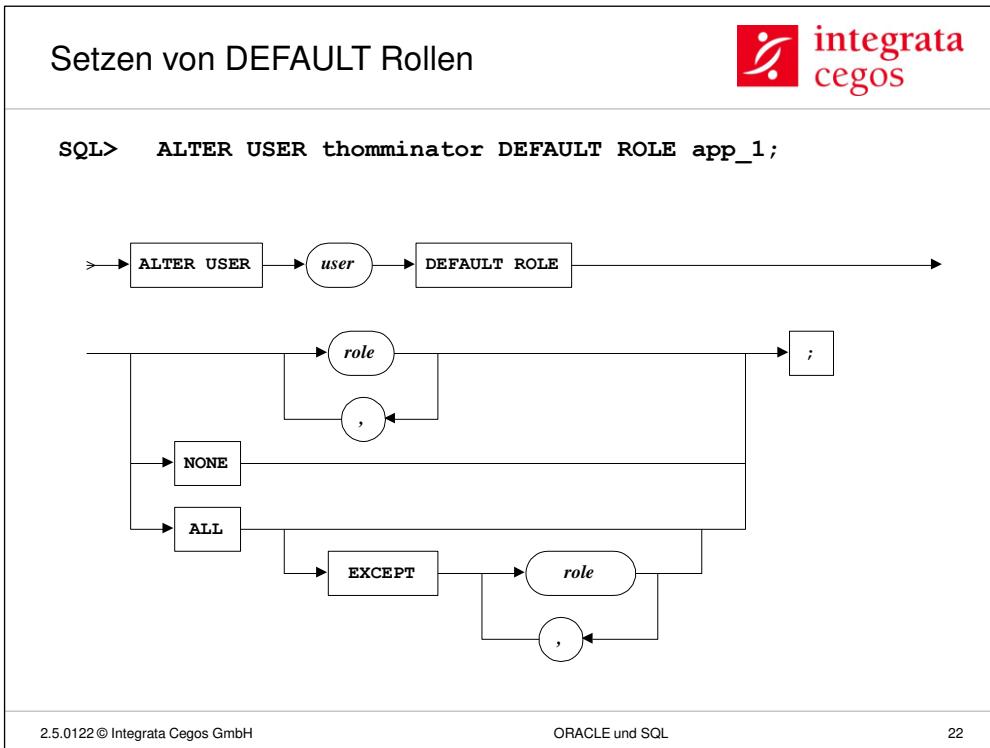


Abb. 8-10: `DEFAULT ROLE` Syntax



8.6.10 Vordefinierte Rollen

Unter ORACLE6 gab es nur die Privilegien CONNECT, RESOURCE und DBA. Da diese seit ORACLE7 so nicht mehr existieren, sondern durch viele Einzelrechte ersetzt worden sind, wurden aus Kompatibilitätsgründen vordefinierte Rollen angelegt, welche diese Privilegien nachbilden. Bei ORACLE 12c sind es inzwischen 57 vordefinierte Rollen.

Hinweis: Anmerkung von ORACLE:

Es ist nicht gewährleistet, dass diese Rollen in jedem zukünftigen ORACLE-Release zur Verfügung stehen werden. Deswegen sollte weitestgehend auf die Nutzung dieser Rollen verzichtet werden.

Data Dictionary Views für Roles

DBA_ROLES	Alle Rollen, die in der Datenbank existieren.
DBA_ROLE_PRIVS	Rollen, die Usern und Rollen zugewiesen sind.
USER_ROLE_PRIVS	Dem aktuellen Benutzer zugewiesene Rollen.
ROLE_ROLE_PRIVS	Rollen, die Rollen zugewiesen sind.
ROLE_SYS_PRIVS	Systemprivilegien, die Rollen zugewiesen sind.
ROLE_TAB_PRIVS	Objektprivilegien, die Rollen zugewiesen sind.
SESSION_ROLES	Vom aktuellen Benutzer aktivierte Rollen.

Einige vordefinierte Rollen

Rolle	enthaltene Privilegien
CONNECT	CREATE SESSION -- ALTER SESSION -- CREATE DATABASE LINK -- CREATE SYNONYM -- CREATE SEQUENCE -- CREATE TABLE -- CREATE VIEW -- CREATE CLUSTER
RESOURCE	CREATE TABLE CREATE CLUSTER CREATE SEQUENCE CREATE PROCEDURE CREATE TRIGGER CREATE TYPE
DBA	Alle Systemprivilegien mit der WITH ADMIN OPTION
EXP_FULL_DATABASE	SELECTANY TABLE BACKUP ANY TABLE EXECUTE ANY PROCEDURE EXECUTE ANY TYPE
IMP_FULL_DATABASE	insgesamt 50 Rechte, die auch in DBA enthalten sind, aber ohne ADMIN OPTION

Vordefinierte Rollen		
Rolle	enthaltene Privilegien	
CONNECT	CREATE SESSION	
RESOURCE	CREATE TABLE CREATE CLUSTER CREATE SEQUENCE CREATE PROCEDURE CREATE TRIGGER CREATE TYPE	
DBA	Alle Systemprivilegien mit der WITH ADMIN OPTION	
EXP_FULL_DATABASE	SELECT ANY TABLE BACKUP ANY TABLE EXECUTE ANY PROCEDURE EXECUTE ANY TYPE	
IMP_FULL_DATABASE	insgesamt 50 Rechte, die auch in DBA enthalten sind, aber ohne ADMIN OPTION	

2.5.0122 © Integrata Cegos GmbH ORACLE und SQL 23

8.7 Benutzerprofile

Mit Benutzerprofilen können auf der Benutzerebene verschiedene Resource-Limits definiert werden. Das hat den Vorteil, dass die Nutzung der Datenbank und ihrer Systemressourcen besser verteilt werden können.

Es gibt zwei Limit-Ebenen:

Session-Level: gilt pro gültigem Login. Meldet sich ein Benutzer auf verschiedenen Terminals mehrmals an, so werden mehrere unabhängige Sessions gestartet. Wird das Session Limit überschritten, so wird die Session abgebrochen und die offene Transaktion zurückgesetzt.

Call Level: Resource pro Call. Jedoch ist ein SQL-Befehl nicht gleichzeitig ein CALL. Jeder SQL-Befehl kann mehrere Calls absetzen (z. B. SELECT mit FETCH). Wird das Limit für den CALL überschritten, so wird der Befehl abgebrochen und zurückgesetzt.

Ein Profil wird einem Benutzer durch den CREATE/ALTER USER Befehl zugeordnet (siehe Beispiel).

Default Profil ist sinnigerweise das Profil DEFAULT.

Wird einem Benutzer kein explizites Profil zugeordnet, so hat er automatisch das Profil DEFAULT. Werden Werte in anderen Profilen nicht gesetzt, so gelten für diese Werte auch die Defaultwerte aus dem DEFAULT-Profil.

Wird eine neue Datenbank angelegt, so stehen alle Profil-Werte auf UNLIMITED.

Beispiel:

```
SQL> ALTER USER hugo PROFILE profi;
```

Benutzerprofile



- DB-Resource-Limits für einen Benutzer setzen
- Es gibt zwei Limit-Ebenen:
 - Session Level (pro Anmeldung)
 - Call Level (pro SQL-Befehl)
- Wird einem Benutzer kein explizites Profil zugeordnet, so hat er automatisch das `DEFAULT`-Profil
- Defaultmäßig stehen alle Werte des Profils auf `UNLIMITED`

Beispiel:

```
SQL> ALTER USER hugo PROFILE profi;
```

8.7.1 Der CREATE PROFILE Befehl

Mit dem CREATE PROFILE Befehl kann ein neues Profil angelegt werden.

Folgende Ressourcen können gesetzt werden:

SESSIONS_PER_USER	Anzahl gleichzeitiger Sessions pro Benutzer.
CPU_PER_SESSION	Maximale CPU-Zeit pro Session (1/100 sec).
CPU_PER_CALL	Maximale CPU-Zeit pro Call (1/100 sec).
CONNECT_TIME	Maximale Zeit, die ein Benutzer in Minuten angemeldet sein darf. Beim Überschreiten der Zeit wird der Benutzer abgemeldet.
IDLE_TIME	Maximale Zeit eines angemeldeten Benutzers in Minuten, die er untätig sein darf. Beim Überschreiten der Zeit wird der Benutzer abgemeldet.
INACTIVE_ACCOUNT_TIME	Ab 12.2: Maximale Zeitspanne in tagen, die ein Benutzer inaktiv sein kann, bevor sein Account gesperrt wird.
LOGICAL_READS_PER_SESSION	Maximale Anzahl von Datenblöcken, die in einer Session gelesen werden dürfen.
LOGICAL_READS_PER_CALL	Maximale Anzahl von Datenblöcken, die von einem CALL gelesen werden dürfen.
PRIVATE_SGA	Maximale Anzahl Bytes, welche die Session in der SGA für einen Benutzer belegen kann. Sinnvoll für Multi-Threaded Server.
COMPOSITE_LIMIT	Erweitert die Flexibilität über die Gewichtung von CPU PER SESSION, CONNECT TIME, READS PER SESSION und PRIVATE SGA. Die Gewichtung kann durch den ALTER RESOURCE COST-Befehl gesetzt werden.

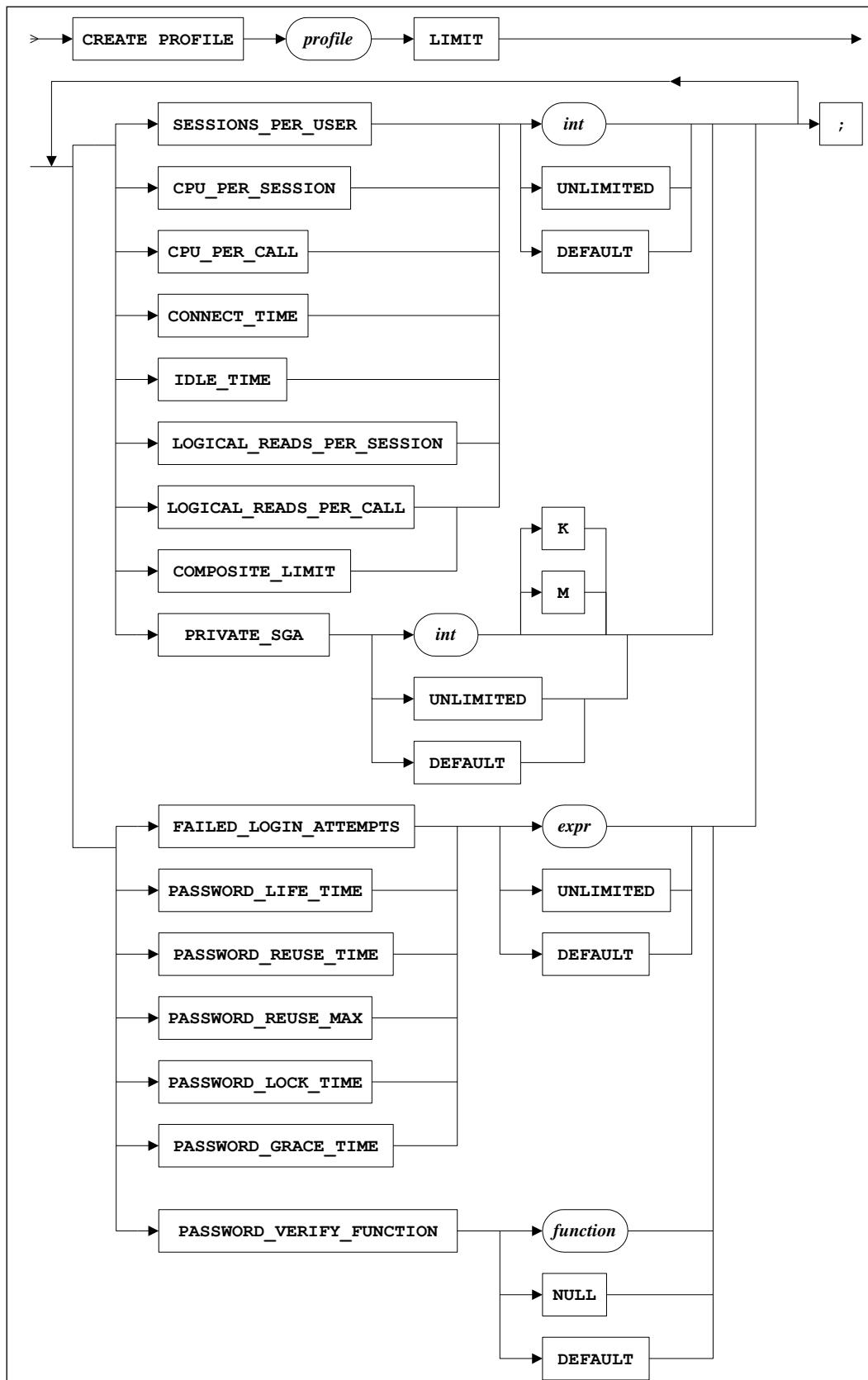
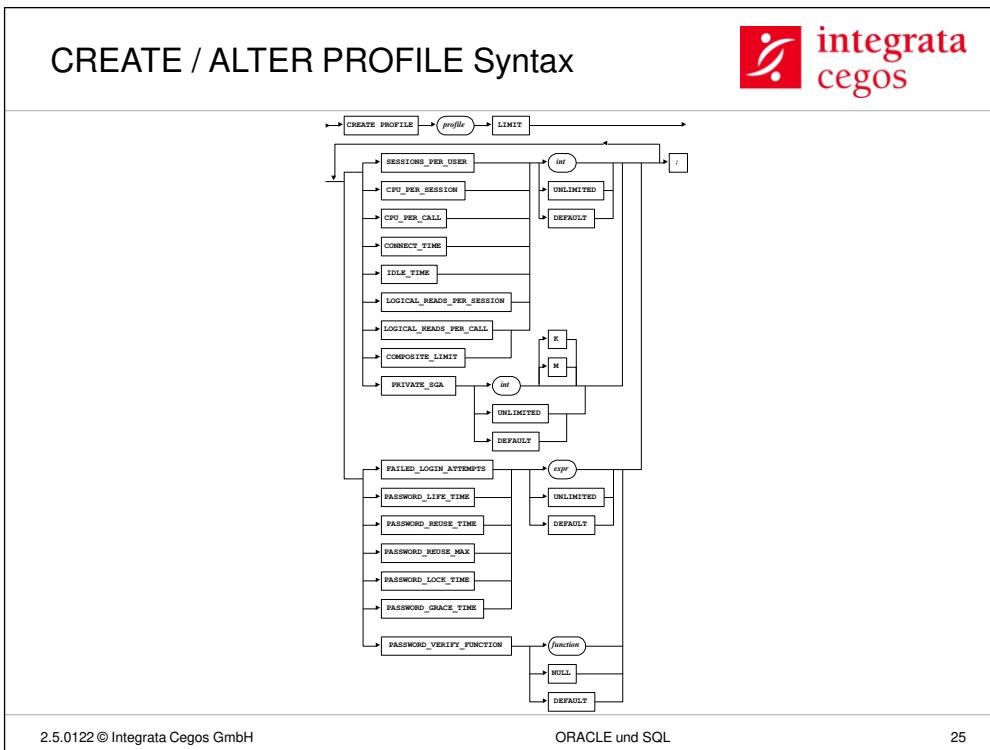


Abb. 8-11: CREATE PROFILE Syntax

Beispiel:

```
SQL> CREATE PROFILE profi
      SESSIONS_PER_USER          UNLIMITED
      CPU_PER_SESSION             UNLIMITED
      CPU_PER_CALL                DEFAULT
      LOGICAL_READS_PER_SESSION   UNLIMITED
      LOGICAL_READS_PER_CALL     100000;
```



8.7.2 Der ALTER PROFILE Befehl

Mit diesem Befehl können Limits verändert, hinzugefügt oder aufgehoben werden. Die Syntax ist identisch zu CREATE PROFILE.

Die Änderungen wirken jedoch erst beim nächsten Login des Benutzers, da das Profil bereits beim Starten einer Session durchlaufen wird.

Beispiel:

```
SQL> ALTER PROFILE profi LIMIT
      CONNECT_TIME 720
      IDLE_TIME     60;
```

8.7.3 Der DROP PROFILE Befehl

Mit diesem Befehl können Profile gelöscht werden.

Durch die Option CASCADE wird den Benutzern, die das zu löschen Profil besitzen, das Profil DEFAULT zugeordnet.

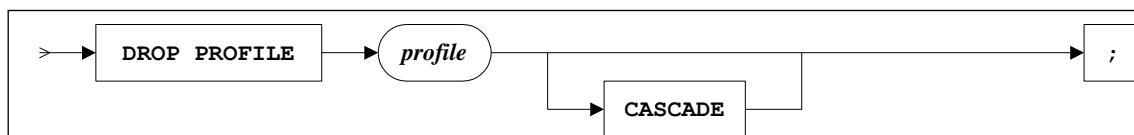


Abb. 8-12: Syntax DROP PROFILE

Beispiel:

```
SQL> DROP PROFILE profi CASCADE;
```

Hinweis: CASCADE ist notwendig, wenn das Profil noch einem Benutzer zugewiesen ist. Dieser erhält dann das DEFAULT-Profil.

8.7.4 Data Dictionary Views für Profiles

- DBA_PROFILES: es werden alle Profile mit ihren Limits angezeigt.
- DBA_USERS: in der Spalte Profiles werden alle Default-Profile der Benutzer angezeigt.

CREATE PROFILE und ALTER PROFILE Befehl

integrata cegos

▪ Beispiel

- **CREATE PROFILE:**

```
CREATE PROFILE profi
  SESSIONS_PER_USER      UNLIMITED
  CPU_PER_SESSION        UNLIMITED
  CPU_PER_CALL           DEFAULT
  LOGICAL_READS_PER_SESSION UNLIMITED
  LOGICAL_READS_PER_CALL   100000;
```


▪ Beispiel

- **ALTER PROFILE:**

```
ALTER PROFILE profi
  CONNECT_TIME            720
  IDLE_TIME                60;
```

2.5.0122 © Integrata Cegos GmbH ORACLE und SQL 26

DROP PROFILE

integrata cegos

SQL> DROP PROFILE profi;
SQL> DROP PROFILE profi CASCADE;

CASCADE ist notwendig, wenn das Profil noch einem Benutzer zugewiesen ist.
Dieser erhält dann das DEFAULT-Profil.

```

graph LR
    A[DROP PROFILE] --> B((profile))
    B --> C[;]
    B --> D[CASCADE]

```

▪ Data Dictionary Views für Profile

- DBA_PROFILES:
es werden alle Profile mit ihren Limits angezeigt.
- DBA_USERS:
in der Spalte Profiles werden alle Default-Profile der Benutzer angezeigt.

2.5.0122 © Integrata Cegos GmbH ORACLE und SQL 27

9

Analytische Funktionen

9.1	Aggregatsfunktionen als analytische Funktion.....	9-3
9.2	LAG und LEAD	9-7
9.3	Rekursive Berechnungen	9-10
9.4	Ranking Funktionen (RANK, DENSE_RANK)	9-11
9.5	LISTAGG.....	9-13

9 Analytische Funktionen

9.1 Aggregatsfunktionen als analytische Funktion

Analytische Funktionen stehen schon seit mehreren Jahren zu Verfügung und bieten vielfältige Möglichkeiten, die sich mit einfachen SELECTS so nicht realisieren lassen. Wir werden in den nachfolgenden Beispielen die Vorteile von analytischen Funktionen aufzeigen.

Dazu zunächst ein einfaches Beispiel auf die bekannte EMP Tabelle:

Es soll deptno,ename, job, sal und sum(sal) über Zeilen angezeigt werden. Zur Erinnerung, sobald eine Aggregatsfunktion im select enthalten ist, müssen alle nicht aggregierten Spalten in Group By aufgenommen werden. Mit Group By ist hier also keine Lösung möglich.

Lösung:

	DEPTNO	ENAME	JOB	SAL	GESAMTSUMME
1	20	SMITH	CLERK	850	29075
2	30	ALLEN	SALESMAN	1600	29075
3	30	WARD	SALESMAN	1250	29075
4	20	JONES	MANAGER	2975	29075
5	30	MARTIN	SALESMAN	1250	29075
6	30	BLAKE	MANAGER	2850	29075
7	10	CLARK	MANAGER	2450	29075
8	20	SCOTT	ANALYST	3000	29075
9	10	KING	PRESIDENT	5000	29075
10	30	TURNER	SALESMAN	1500	29075
11	20	ADAMS	CLERK	1100	29075
12	30	JAMES	CLERK	950	29075
13	20	FORD	ANALYST	3000	29075
14	10	MILLER	CLERK	1300	29075

```

SELECT
    deptno,
    ename,
    job,
    sal,
    SUM(sal) OVER() AS Gesamtsumme
FROM
    emp;

```

Die gezeigte Lösung lässt sich leicht variiert auch zur Ausgabe von sum(sal) pro deptno benutzen

```
SELECT
    deptno,
    ename,
    job,
    sal,
    SUM(sal) OVER(partition by deptno) AS SummeProAbt
FROM
    emp;
```

	DEPTNO	ENAME	JOB	SAL	SUMMEPROABT
1	10	CLARK	MANAGER	2450	8750
2	10	KING	PRESIDENT	5000	8750
3	10	MILLER	CLERK	1300	8750
4	20	JONES	MANAGER	2975	10925
5	20	FORD	ANALYST	3000	10925
6	20	ADAMS	CLERK	1100	10925
7	20	SMITH	CLERK	850	10925
8	20	SCOTT	ANALYST	3000	10925
9	30	WARD	SALESMAN	1250	9400
10	30	TURNER	SALESMAN	1500	9400
11	30	ALLEN	SALESMAN	1600	9400
12	30	JAMES	CLERK	950	9400
13	30	BLAKE	MANAGER	2850	9400
14	30	MARTIN	SALESMAN	1250	9400

SUM(sal) OVER(partition by deptno) AS SummeProAbt

Durch OVER(partition by deptno) erhalten Sie nicht mehr die Gesamtsumme sondern die Summe pro deptno, analog group by deptno, nur nicht mit den angesprochenen Einschränkungen.

Sie können die Ausgabe jederzeit um weitere aggregierende analytische Funktionen erweitern, beispielsweise um das Durchschnittsgehalt pro Abteilung.

```
SELECT
    deptno,
    ename,
    job,
    sal,
    round(AVG(sal)
        OVER(PARTITION BY deptno), 0) AS schnittproabt,
    SUM(sal)
        OVER(PARTITION BY deptno)           AS summeproabt
FROM
    emp;
```

The screenshot shows a database query results window titled "Abfrageergebnis". The window includes standard toolbar icons for copy, paste, refresh, and exit. Below the toolbar, it says "Alle Zeilen abgerufen: 14 in 0,008 Sekunden". The result set is a table with the following columns and data:

	DEPTNO	ENAME	JOB	SAL	SCHNITTPROABT	SUMMEROABT
1	10	CLARK	MANAGER	2450	2917	8750
2	10	KING	PRESIDENT	5000	2917	8750
3	10	MILLER	CLERK	1300	2917	8750
4	20	JONES	MANAGER	2975	2185	10925
5	20	FORD	ANALYST	3000	2185	10925
6	20	ADAMS	CLERK	1100	2185	10925
7	20	SMITH	CLERK	850	2185	10925
8	20	SCOTT	ANALYST	3000	2185	10925
9	30	WARD	SALESMAN	1250	1567	9400
10	30	TURNER	SALESMAN	1500	1567	9400
11	30	ALLEN	SALESMAN	1600	1567	9400
12	30	JAMES	CLERK	950	1567	9400
13	30	BLAKE	MANAGER	2850	1567	9400
14	30	MARTIN	SALESMAN	1250	1567	9400

Die folgende Grafik zeigt wie die Begriffe Partition, Window und Current Row bei analytischen Funktionen zu verstehen sind

DEPTNO	ENAME	HIREDATE	SAL	COUNT_SAL
10	MILLER	23-JAN-82	1300	1
10	CLARK	09-JUN-81	2450	1
10	KING	17-NOV-81	5000	1
20	SMITH	17-DEC-80	800	2
20	ADAMS	12-JAN-81	1100	2
20	JONES	02-APR-81	2975	3
20	SCOTT	09-DEC-82	3000	3
20	FORD	03-DEC-81	3000	3
30	JAMES	03-DEC-81	950	5
30	WARD	22-FEB-81	1250	5
30	MERCIN	28-SEP-81	1250	5
30	TURNER	08-SEP-81	1500	5
30	ADAMS	20-FEB-81	1600	5
30	BLAKE	01-MAY-81	2850	5

Partition

Window

Current
Row

9.2 LAG und LEAD

Mit den Analytischen Funktionen LAG und LEAD lässt sich ausgehend von der CURRENT ROW auf Werte in vorhergehenden Zeilen (LAG) und auf nachfolgende Werte (LEAD) zugreifen.

Beispiel LAG

```
SELECT  
    empno,  
    ename,  
    job,  
    sal,  
    LAG(sal, 1) OVER(ORDER BY empno) AS vorwert  
FROM  
    emp;
```

Ergebnis:

	EMPNO	ENAME	JOB	SAL	VORWERT
1	7369	SMITH	CLERK	850	(null)
2	7499	ALLEN	SALESMAN	1600	850
3	7521	WARD	SALESMAN	1250	1600
4	7566	JONES	MANAGER	2975	1250
5	7654	MARTIN	SALESMAN	1250	2975
6	7698	BLAKE	MANAGER	2850	1250
7	7782	CLARK	MANAGER	2450	2850
8	7788	SCOTT	ANALYST	3000	2450
9	7839	KING	PRESIDENT	5000	3000
10	7844	TURNER	SALESMAN	1500	5000
11	7876	ADAMS	CLERK	1100	1500
12	7900	JAMES	CLERK	950	1100
13	7902	FORD	ANALYST	3000	950
14	7934	MILLER	CLERK	1300	3000

LAG(sal, 1) OVER(ORDER BY empno) AS vorwert

In diesem Beispiel wird auf die direkte Vorgängerzeile LAG(sal,1) zugegriffen. Analog kann durch Ersetzen der 1 durch 2 oder 3 auf Zeilenwerte die 2 oder 3 Zeilen davor lagen zugegriffen werden.

Beispiel LEAD:

```
SELECT  
    empno,  
    ename,  
    job,  
    sal,  
    LEAD(sal, 1) OVER(ORDER BY empno) AS Nachfolger  
FROM  
    emp;
```

Ergebnis:

	EMPNO	ENAME	JOB	SAL	NACHFOLGER
1	7369	SMITH	CLERK	850	1600
2	7499	ALLEN	SALESMAN	1600	1250
3	7521	WARD	SALESMAN	1250	2975
4	7566	JONES	MANAGER	2975	1250
5	7654	MARTIN	SALESMAN	1250	2850
6	7698	BLAKE	MANAGER	2850	2450
7	7782	CLARK	MANAGER	2450	3000
8	7788	SCOTT	ANALYST	3000	5000
9	7839	KING	PRESIDENT	5000	1500
10	7844	TURNER	SALESMAN	1500	1100
11	7876	ADAMS	CLERK	1100	950
12	7900	JAMES	CLERK	950	3000
13	7902	FORD	ANALYST	3000	1300
14	7934	MILLER	CLERK	1300	(null)

Häufig werden LAG und LEAD auch benutzt um Differenzen zum Vorgänger/Nachfolger ausgeben zu lassen.

Beispiel:

```
SELECT  
    empno,  
    ename,  
    job,  
    sal,  
    LAG(sal, 1) OVER(ORDER BY empno) as Vorwert,  
    sal - LAG(sal, 1) OVER(ORDER BY empno) as DiffVorwert,  
    LEAD(sal, 1) OVER(ORDER BY empno) AS Nachfolger,  
    LEAD(sal, 1) OVER(ORDER BY empno) - sal as DiffNachfolger  
FROM  
    emp;
```

Ergebnis:

	EMPNO	ENAME	JOB	SAL	VORWERT	DIFFVORWERT	NACHFOLGER	DIFFNACHFOLGER
1	7369	SMITH	CLERK	850	(null)	(null)	1600	750
2	7499	ALLEN	SALESMAN	1600	850	750	1250	-350
3	7521	WARD	SALESMAN	1250	1600	-350	2975	1725
4	7566	JONES	MANAGER	2975	1250	1725	1250	-1725
5	7654	MARTIN	SALESMAN	1250	2975	-1725	2850	1600
6	7698	BLAKE	MANAGER	2850	1250	1600	2450	-400
7	7782	CLARK	MANAGER	2450	2850	-400	3000	550
8	7788	SCOTT	ANALYST	3000	2450	550	5000	2000
9	7839	KING	PRESIDENT	5000	3000	2000	1500	-3500
10	7844	TURNER	SALESMAN	1500	5000	-3500	1100	-400
11	7876	ADAMS	CLERK	1100	1500	-400	950	-150
12	7900	JAMES	CLERK	950	1100	-150	3000	2050
13	7902	FORD	ANALYST	3000	950	2050	1300	-1700
14	7934	MILLER	CLERK	1300	3000	-1700	(null)	(null)

9.3 Rekursive Berechnungen

Hierzu ein einfaches Beispiel:

	ENAME	SAL	SUMAUFGELAUFEN
1	SMITH	850	850
2	JAMES	950	1800
3	ADAMS	1100	2900
4	WARD	1250	4150
5	MARTIN	1250	5400
6	MILLER	1300	6700
7	TURNER	1500	8200
8	ALLEN	1600	9800
9	CLARK	2450	12250
10	BLAKE	2850	15100
11	JONES	2975	18075
12	SCOTT	3000	21075
13	FORD	3000	24075
14	KING	5000	29075

In diesem Beispiel wird die SAL Spalte aufsteigend ausgegeben und daneben die Spalte Sumaufgelaufen fortlaufend je nach Zeile von Anfang an aufsummiert.

```

SELECT
    ename,
    sal,
    SUM(sal) OVER (ORDER BY sal
                    ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT
                    ROW) AS Sumaufgelaufen
FROM
    emp;

SUM(sal) OVER (ORDER BY sal
                ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT
                ROW) AS Sumaufgelaufen

```

SUM(SAL) wird in diesem Beispiel so berechnet, dass die Current Row sich zeilenweise über den Datenbestand schiebt und für jede Zeile die

Summe immer zwischen erster Zeile (unbounded preceding) und der aktuellen Zeile (Current Row) gebildet wird.

Dieses Intervall für die Berechnung kann den eigenen Wünschen angepasst werden – beispielweise rows between 1 preceding and 1 following)

9.4 Ranking Funktionen (RANK, DENSE_RANK)

Als analytische Funktionen können nicht nur die klassischen Gruppenfunktion (SUM, MIN, MAX, AVG, COUNT) verwendet werden, sondern es gibt auch weitere Funktionen, die speziell für die Verwendung als analytische Funktionen konzipiert wurden.

Dazu gehört die Familie der Ranking Funktionen mit denen Ranglisten oder Top-n Abfragen formuliert werden können. Die Funktionen RANK und DENSE_RANK berechnen die Ranglisten-Nummer für jede Zeile. Der Unterschied zwischen beiden Funktionen besteht darin, wie nach mehreren Geichplazierten die Nummerierung fortgesetzt wird.

Beispiel:

```
SELECT
    deptno,
    ename,
    sal,
    RANK() OVER (ORDER by sal desc) as RANK,
    DENSE_RANK()    OVER    (ORDER    by    sal    desc)   as
DENSE_RANK
FROM
    emp;
```

Ergebnis:

	DEPTNO	ENAME	SAL	RANK	DENSE_RANK
1	10	KING	5000	1	1
2	20	FORD	3000	2	2
3	20	SCOTT	3000	2	2
4	20	JONES	2975	4	3
5	30	BLAKE	2850	5	4
6	10	CLARK	2450	6	5
7	30	ALLEN	1600	7	6
8	30	TURNER	1500	8	7
9	10	MILLER	1300	9	8
10	30	WARD	1250	10	9
11	30	MARTIN	1250	10	9
12	20	ADAMS	1100	12	10
13	30	JAMES	950	13	11
14	20	SMITH	850	14	12

RANK nummeriert olympisch – 2x Silber gibt es keine Bronze

DENSE_RANK fährt nach gleichen Werten einfach mit der nächsten Platzziffer fort. Siehe Zeile 4 bzw. Zeile 12.

9.5 LISTAGG

Hier ein kleines Beispiel der LISTAGG Funktion, die in diesem Fall eine Komma separierte Liste der Namen aller Mitarbeiter pro Abteilung ausgibt.

```
SELECT
    deptno,
    LISTAGG (ename, ',') WITHIN GROUP (order by ename)
    As Namensliste
FROM
    emp
    GROUP BY deptno
    ORDER by deptno;
```

Ergebnis:

	DEPTNO	NAMENSLISTE
1	10	CLARK, KING, MILLER
2	20	ADAMS, FORD, JONES, SCOTT, SMITH
3	30	ALLEN, BLAKE, JAMES, MARTIN, TURNER, WARD

10

Anhang A

10.1	Systemarchitektur	10-3
10.2	Die Komponenten der ORACLE-Datenbank	10-5
10.3	Die logische und physische ORACLE-Struktur.....	10-8
10.3.1	Logische Datenbankstruktur	10-8
10.3.2	Physische Datenbankstruktur	10-9
10.4	Informationen zu ORACLE – Dateien auf Betriebssystemebene	10-11

10 Anhang A

10.1 Systemarchitektur

Anwendungssysteme, die mit Hilfe eines Datenbanksystems entwickelt wurden, kann man in drei Funktionsebenen einteilen:

- **Anwendungsprogramme:** Programme, welche die entsprechenden Anwendungsfunktionen zur Verfügung stellen
- **Datenbankmanagementsystem (DBMS):** Datenbank-Programme, welche die spezifischen Datenbank-Operationen bereitstellen, wie beispielsweise:
 - Lesen, Ändern, Einfügen und Löschen von Daten
 - Sperrmechanismen
 - Backup und Recovery-Verfahren, usw.
- **Datenbank:** Plattspeicherbereiche, auf denen die eigentlichen Daten abgelegt werden.

Das ORACLE-System kann ebenfalls in diese drei Ebenen aufgeteilt werden, denn auch hier kommunizieren die Anwendungsprogramme mit dem Datenbanksystem.

Ein ORACLE-Server kann weiter untergliedert werden in:

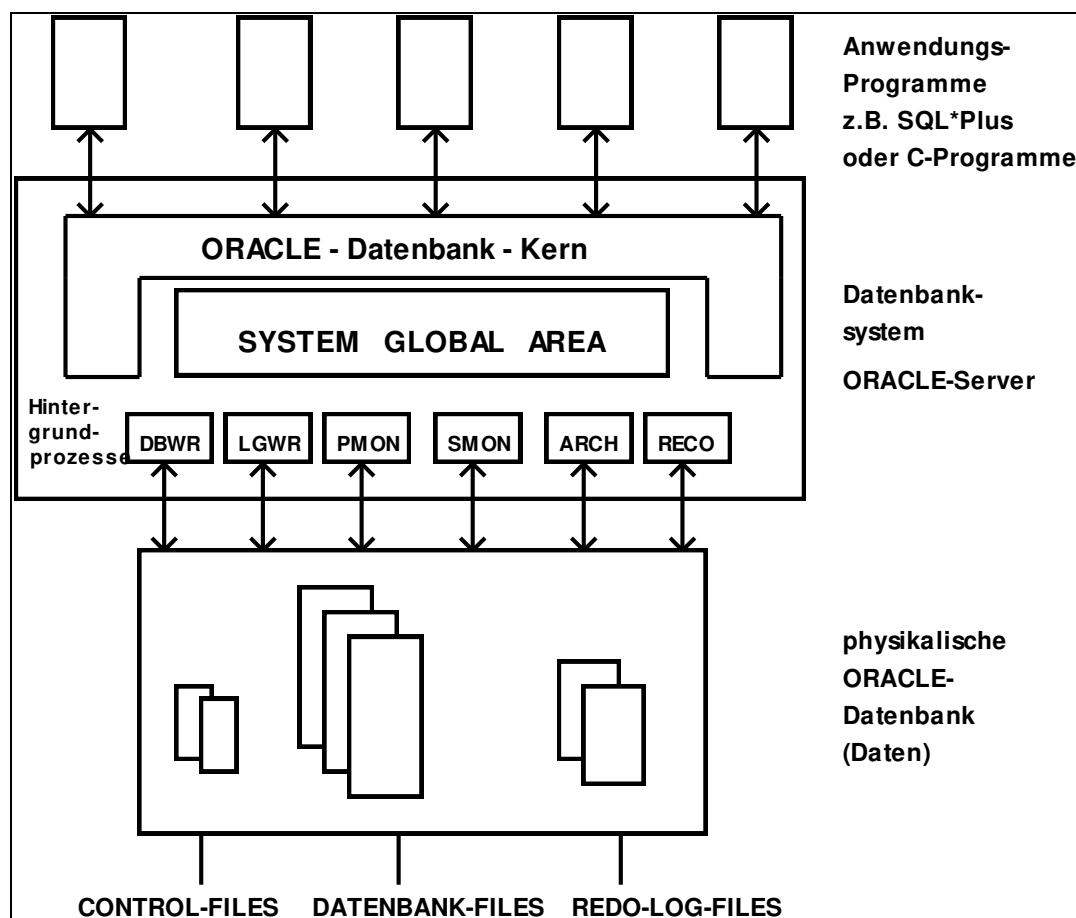
- Datenbank-Kern (ORACLE-Kernel). Dieser Teil ist verantwortlich für die gesamte Steuerung und Durchführung aller Datenbank-Operationen.
- System Global Area (SGA). Es ist ein Hauptspeicherbereich, in dem u.a. alle Datenbankblöcke abgelegt werden (Datenbank-Cache). Die SGA dient als Kommunikationsleiste zwischen ORACLE-Server und den Anwendungsprogrammen.
- Verschiedene Hintergrundprozesse. Neben Dateien und Speicherbereichen spielen die Prozesse eine zentrale Rolle bei dem Betrieb einer ORACLE-Datenbank.

Die eigentliche physikalische ORACLE-Datenbank besteht aus drei Dateiarten:

- **Datenbankdateien.** Sie enthalten alle Datenbankstrukturen und die dazugehörigen Daten.
- **Redolog-Dateien.** Sie enthalten Daten, die benötigt werden, um ein u. U. notwendiges Datenbank-Recovery durchzuführen.
- **Control-Dateien.** Sie enthalten Daten, die zur Verwaltung des ORACLE-Servers notwendig sind.

Der Begriff "ORACLE-Server" bezeichnet das Datenbank (management) System. Oracle mit der RAC Option ermöglicht den Zugriff von mehreren Instanzen auf einer physikalischen Datenbank.

ORACLE – Architektur



10.2 Die Komponenten der ORACLE-Datenbank

Die **SGA** (System Global Area) ist ein prozessübergreifender Speicherbereich, auf den alle Benutzerprozesse und Hintergrundprozesse, die dieser Datenbank zugeordnet sind, zugreifen. In der SGA sind der Database Buffer Pool, der Redo Log Buffer und der Shared Pool enthalten.

Bei einem **Leseprozess** durch einen Benutzer sind folgende Prozesse beteiligt:

- der Benutzerprozess
- die Datendateien
- die SGA.

Die aus der Datendatei gelesenen Datenbank-Blöcke werden durch den Benutzerprozess in den Datenbank-Puffer der SGA eingelesen.

Bei einem **Schreibprozess** werden die veränderten oder neu angelegten Datenblöcke durch den Hintergrundprozess DATABASE WRITER (DBW0) irgendwann in die Datendateien geschrieben. Dies hat den Vorteil, dass der Benutzerprozess nicht darauf warten muss, bis alle Daten wieder auf die Platte geschrieben sind. Somit wird gewährleistet, dass durch umfangreiche Manipulationsvorgänge keine Aktivitätsspitzen vorkommen. Obwohl ein Database-Writer genügt, können aus Performance-Gründen weitere Database-Writer gestartet werden (DBW1,, DBW100).

Der **DBWR**-Prozess sorgt für das Zurückschreiben von Daten aus der SGA, die lange nicht verändert wurden. Somit wird freier Speicherplatz für Daten geschaffen, die vielen Änderungen durch den oder die Benutzerprozesse unterworfen sind.

Zudem sind in den Datendateien Bereiche reserviert, die als Rollback-Segmente bezeichnet werden. In ihnen werden die geänderten Bytes eines Datenblockes gespeichert. Dadurch besteht die Möglichkeit ausgeführte Befehle rückgängig zu machen. Diese Rollback-Segmente werden wie die Daten-Segmente durch die SGA- und DBWR-Prozesse verwaltet.

Um auch nach einem Systemabsturz die Wiederherstellung einer konsistenten Datenbank zu gewährleisten, sind noch folgende Komponenten notwendig:

- der Redo Log Buffer in der SGA
- der Log Writer-Hintergrundprozess (LGWR)
- die Redo Log-Dateien.

Der **Redo Log Buffer** der SGA hat die Aufgabe, alle neuen oder geänderten Daten (nur die entsprechenden Bytes) aufzunehmen. Die Einträge im Redo Log Buffer sind auch in der online Redo Log Datei enthalten, um im Falle einer Recovery die Datenbank wiederherstellen zu können.

Der **LGWR**-Prozess ist dafür zuständig, diese relativ kleinen Redo Log Buffer auf die Platte in die Redo Log-Dateien zu schreiben. Es sind immer mindestens zwei Redo Log-Dateien vorhanden, die abwechselnd beschrieben werden. Der Archivierungsprozess **arch_xx** ist nun dafür zuständig, die gefüllten Redo Log-Dateien in ein Archivierungsverzeichnis zu kopieren und die Redo Log-Dateien mit einer neuen, höheren Nummer zu versehen und sie wieder freizugeben. Der Archivierungsprozess ist nur aktiv, wenn die Datenbank sich im Archivmodus befindet.

In der SGA befindet sich außerdem der Database Buffer, in dem sich aus Performance-Gründen die zuletzt benutzten Blöcke der Daten aus der Datenbank befinden. Dieser Buffer enthält nicht geänderte und geänderte Daten.

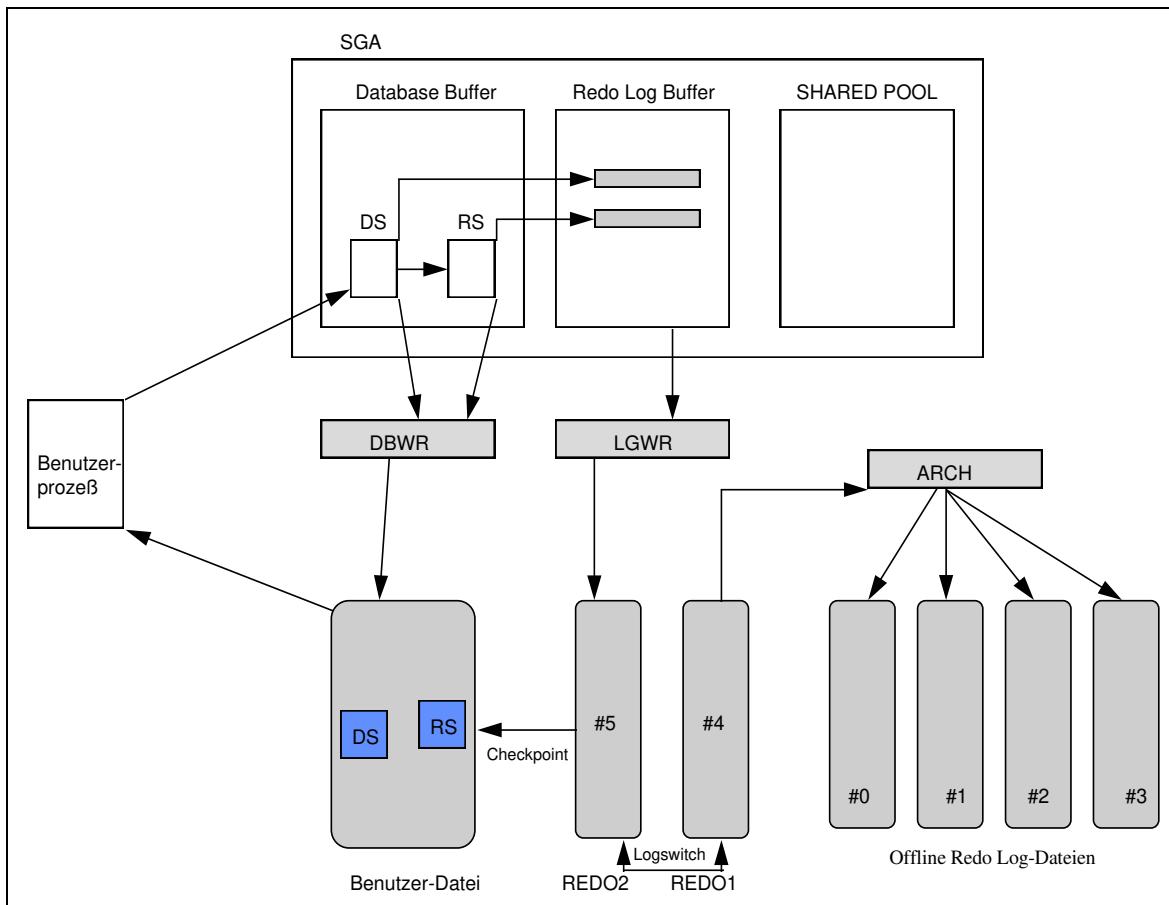
Zu bestimmten Zeiten werden die Daten vom Database-Writer in die Datenbank zurückgeschrieben. Diesen Zeitpunkt nennt man Checkpoint. Der Checkpoint-Prozess benachrichtigt den Database-Writer und aktualisiert die Control- und Redo Logfiles mit dem höchsten Checkpoint.

Weiter befindet sich in der SGA der **SHARED POOL**. Er enthält den ROW CACHE bzw. den DATA DICTIONARY CACHE und die SHARED SQL AREAS. Der **ROW CACHE** enthält Informationen aus dem Data Dictionary (Tabellenstrukturen, Spaltendefinitionen...), **SHARED SQL AREAS** enthalten geparsste SQL-Kommandos.

Der Systemmonitor (**SMON**) unterstützt eine Instanz-Recovery beim Instanz-Start und gibt ggf. temporäre Speicherbereiche frei.

Der Prozessmonitor (**PMON**) räumt auf, falls ein Benutzerprozess abbricht.

Die Komponenten der Oracle-Datenbank



10.3 Die logische und physische ORACLE-Struktur

Eine ORACLE-Datenbank hat sowohl eine logische als auch eine physische Struktur. Der Vorteil dieser Trennung ist, dass der physikalische Speicher der Daten verwaltet werden kann, ohne dass der Zugang und der Zugriff auf die logische Speicherstruktur beeinträchtigt wird.

10.3.1 Logische Datenbankstruktur

Unter den logischen Strukturen einer ORACLE-Datenbank versteht man die verschiedenen Objekte der Datenbank, wie Tabellen, Indizes, Cluster, die einem Schema, d.h. einem "Benutzer", zugeordnet sind. Die Objekte sind als Segment in einer logischen Speichereinheit, dem Tablespace, in Form von einem oder mehreren Extents gespeichert.

Definitionen:

Datenbankblock Die kleinste Unterteilung einer Datenbank ist der Datenbankblock. Diese Größe wird beim Anlegen der Datenbank spezifiziert.

Extent Ein Extent besteht aus einer Anzahl aufeinander folgenden Datenbankblöcken.

Segment Ein Segment ist eine Menge von Extents für eine logische Struktur. Man unterscheidet Daten-, Index-, Rollbacksegmente und temporäre Segmente.

Tablespace Eine Datenbank wird in mehrere logische Speicher- einheiten unterteilt, die sogenannten Tablespaces.

Schema Unter diesem Begriff wird einerseits der logische Rahmen verstanden, in dem ein Benutzer (User) seine Objekte anlegen kann. Andererseits ist es die Gesamtheit der bereits bestehenden Datenbankobjekte eines Benutzers, wie Tabellen, Views, Sequenzen, Prozeduren Synonyme, Indizes. Jeder Benutzer ist einem Schema zugeordnet, wobei der Name des Schemas gleich dem Namen des Benutzers ist.

Den einzelnen Tablespaces können ein oder mehrere Dateien zugeordnet werden. Innerhalb eines Tablespaces können beliebig viele Datenbankobjekte erzeugt werden.

Zusätzlich enthält jede ORACLE-Datenbank einen System-Tablespace, der alle systemrelevanten Informationen enthält. Dies sind:

- Data-Dictionary. Es umfasst ORACLE-Tabellen, die das System für alle Datenbankoperationen benötigt.
- System-Rollback-Segment. Hierbei handelt es sich um eine ORACLE interne Struktur zur Zwischenspeicherung noch nicht festgesetzter Transaktionen.
- Tabellen von ORACLE-Werkzeugen SQL*Forms oder SQL*Report Writer

Ein Datenbankadministrator (DBA) kann mit Hilfe von SQL-Befehlen und des Tablespace-Konzeptes eine Datenbank strukturieren.

10.3.2 Physische Datenbankstruktur

Die physischen Strukturen dagegen werden durch die Dateien des Betriebssystems gebildet, welche die Datenbank ausmachen. Jede ORACLE Datenbank besteht aus drei verschiedenen Dateitypen, von denen jeder spezielle Daten enthält, die für bestimmte Datenbankfunktionen notwendig sind:



Datenbankdateien

Es gibt eine oder mehrere Datenbankdateien. Sie enthalten alle Datenbankobjekte, die das ORACLE-System verwalten kann. Dazu gehören:

- ORACLE Data Dictionary
- vom Benutzer angelegte Datenstrukturen
- Zugriffsstrukturen
- prozedurale Objekte
- die eigentlichen Daten



Redo Log-Dateien

Es gibt mindestens zwei Redo Log-Dateien. Hier werden alle Datenänderungen, die innerhalb einer Transaktion durchgeführt werden, protokolliert und abgespeichert. Dies ist notwendig, um die Konsistenz der Datenbank im Fehlerfall wiederherzustellen.

Zu einer ORACLE-Datenbank gehören auch archivierte Redo Log-Dateien. Dies sind vollständig beschriebene Redo Log-Dateien, die zu Sicherungszwecken auf eine andere Platte oder ein anderes Speichermedium kopiert werden.



Control-Dateien

Es gibt eine oder mehrere Control-Dateien. Sie enthalten Grundstrukturen und Grundinformationen über eine ORACLE-Datenbank:

- Zeitpunkt der Datenbankerstellung
- Namen aller DB-Dateien
- Namen aller Redo Log-Dateien und deren Sequenznummern
- Zeitpunkt des letzten Checkpoints und zugehörige Redo Log-Dateien
- Datenbank-Strukturänderungen, wie z.B. das Vergrößern der Datenbank durch zusätzliche Dateien

Obwohl das ORACLE-Datenbanksystem aus einer Reihe von Dateien besteht, stellen diese nur eine untergeordnete Strukturierungseinheit dar. Die Datenbankdateien sind lediglich Speicherlieferant. ORACLE prägt ihnen seine eigene, für die Verwaltung großer Datenmengen optimale – logische – Struktur auf.

Redo Log- und Control-Dateien sind wichtig für notwendige Recovery- und Verwaltungsfunktionen.

10.4 Informationen zu ORACLE – Dateien auf Betriebssystemebene

1. DBA_TABLESPACES

Die Tabelle DBA_TABLESPACES kann zur Ermittlung aller Tablespaces und der Darstellung seiner Eigenschaften herangezogen werden. Sie besitzt folgende Struktur:

<u>Spaltenname</u>	<u>Bedeutung</u>
TABLESPACE_NAME	Name des Tablespaces
PCT_INCREASE	Voreingestellte prozentuale Vergrößerung eines Extents
STATUS	Status (Verfügbarkeit) des Tablespaces

2. DBA_DATA_FILES

Die Tabelle DBA_DATA_FILES gibt Aufschluss über zugeordnete Datenbankdateien und deren Charakteristika. Sie besitzt folgende Struktur:

<u>Spaltenname</u>	<u>Bedeutung</u>
FILE_NAME	Name der Datenbankdatei
FILE_ID	Interne Dateinummer
TABLESPACE_NAME	Name des zugeordneten Tablespaces
BYTES	Größe der Datei in Bytes
BLOCKS	Größe der Datei in Datenbank-Blöcken
STATUS	Status (Verfügbarkeit) der Datenbankdatei

11

Anhang B SQL*Plus

11.1	Editierbefehle.....	11-3
11.2	Austauschvariablen	11-4
11.3	Neuerungen.....	11-4

11 Anhang B SQL*Plus

11.1 Editierbefehle

Mit dem Zeichen * wird die aktuelle Zeile markiert. Standardmäßig ist dies die letzte Zeile. In der folgenden Tabelle steht *n* (und *m*) für eine beliebige positive Zahl.

Kommando	Abkürzung	Beschreibung
APPEND <i>text</i>	A <i>text</i>	fügt einen Text am Ende der Zeile hinzu
CHANGE / <i>alt/neu</i>	C / <i>alt/neu</i>	ändert alten Text in neuen Text
CHANGE / <i>text</i>	C / <i>text</i>	löscht Text aus der Zeile
CLEAR BUFFER	CL BUFF	löscht Pufferinhalt
DEL	-	löscht die aktuelle Zeile
DEL <i>n</i>		löscht die Zeile <i>n</i>
DEL <i>m n</i>		löscht die Zeilen <i>m</i> bis <i>n</i>
INPUT	I	fügt eine oder mehr Zeilen hinzu
INPUT <i>text</i>	I <i>text</i>	fügt 1 Zeile mit angegebenen Text hinzu
LIST	L	listet alle Zeilen des Puffers auf
LIST <i>n</i>	L <i>n</i> oder <i>n</i>	listet die Zeile <i>n</i> auf
LIST *	L *	listet die aktuelle Zeile auf
LIST LAST	L LAST	listet die letzte Zeile auf
LIST <i>m n</i>	L <i>m n</i>	listet die Zeilen <i>m</i> bis <i>n</i> auf

11.2 Austauschvariablen

Unter SQL*Plus können in SQL-Anweisungen sogenannte lokale Variablen verwendet werden. Eine Form der lokalen Variablen ist die Austauschvariable oder Substitutionsvariable. Sie wird in einem SQL-Befehl mit dem Zeichen "&" angesprochen.

- Wurde vorher keine Variable dieses Namens definiert, so wird der Benutzer automatisch durch eine Standardmeldung zur Eingabe eines Wertes aufgefordert. Kommt die gleiche Variable mehrfach vor, so kann man die Mehrfachaufrufforderung zur Eingabe eines Wertes umgehen, indem man dem Variablenamen beim ersten Auftreten statt des "&" ein "&&&" voranstellt.
- Mit dem Befehl **ACCEPT** *Variablenname* **PROMPT** *Text* kann eine Variable deklariert werden, und der Benutzer wird mit dem angegebenen Prompt zur Eingabe eines Wertes aufgefordert, der dann der Variablen fest zugewiesen bleibt.
- Mit dem Befehl **DEFINE** *Variablenname* = *Wert* kann eine Variable deklariert und gleichzeitig mit einem Wert belegt werden. Eine Benutzereingabe ist dann nicht mehr nötig.

DEFINE zeigt den Wert aller Variablen an, **DEFINE** *Variablenname* dagegen den Wert einer einzelnen Variablen. Mit **UNDEFINE** kann eine Variable wieder gelöscht werden. Wird sie nicht gelöscht, so bleibt sie bis zum Ende einer Sitzung definiert.

11.3 Neuerungen

SQL*Plus bietet jetzt plattformunabhängig die Möglichkeit, die zuvor ausgeführten Befehle erneut auszugeben.

Diese Funktionalität ähnelt dem Shell-History-Befehl, der auf den Befehlszeilen-Shells der UNIX-Plattform verfügbar ist.

Syntax:

```
SHOW HISTORY  
SET HIST[ORY] { ON | OFF | n }  
HIST[ORY] [n {RUN|EDIT|DEL[ETE]}] |  
[CLEAR| LIST]
```

12

Anhang C Syntaxdiagramme

12.1	Vorbemerkung	12-3
12.2	CREATE TABLE / ALTER TABLE.....	12-3
12.3	CREATE INDEX / ALTER INDEX.....	12-10
12.4	CREATE VIEW.....	12-13
12.5	ALTER SEQUENCE.....	12-15
12.6	SELECT-Anweisung.....	12-17
12.7	DML-Befehle.....	12-20
12.7.1	INSERT.....	12-20
12.7.2	UPDATE	12-21
12.7.3	DELETE.....	12-23

12 Anhang C Syntaxdiagramme

12.1 Vorbemerkung

Die hier aufgeführten Syntaxdiagramme sind nur als Gedächtnisstützen aufzufassen, da sie aufgrund der enormen Komplexität der vollständigen ORACLE Syntax im Interesse der Lesbarkeit vereinfacht werden mussten. Trotz der Vereinfachung sind wesentliche Aspekte jedoch berücksichtigt, ausführlichere Informationen sind in der ORACLE Dokumentation zu finden.

12.2 CREATE TABLE / ALTER TABLE

<i>schema</i>	ist das Schema, in dem die Tabelle angelegt wird. Wird kein Schema angegeben, so wird die Tabelle in dem Schema des Benutzers angelegt, der den Create Table Befehl ausführt.
<i>table</i>	ist der Name der anzulegenden Tabelle. Dieser Name muss eindeutig innerhalb eines Schemas sein.
<i>column</i>	definiert die Namen der Spalten einer Tabelle. Eine Tabelle muss mindestens eine Spalte und darf maximal 1000 Spalten besitzen.
<i>type</i>	spezifiziert den Datentyp der Spalte.
DEFAULT	Bei einem INSERT-Befehl ohne Angabe eines Wertes für diese Spalte, wird dieser DEFAULT-Wert eingesetzt.
<i>constraint</i>	definiert Regeln für die Spalte dieser Tabelle, die zur referentiellen Integrität dienen.
<i>tablespace</i>	definiert den Tablespace, in dem die Tabelle gespeichert werden soll.
<i>storage</i>	definiert STORAGE-Werte, die zu Performance-Verbesserungen bei großen Tabellen führen.

LOGGING	gewährleistet die Protokollierung von Änderungsaktionen in einer RedoLog-Datei (Standard). Die Datenbank muss dazu im ARCHIVELOG Modus betrieben werden..
NOLOGGING	die Protokollierung von Änderungen der Tabelle in einer RedoLog-Datei wird unterdrückt.
PARALLEL	definiert den Parallelitätsgrad für die Erstellung einer Tabelle.
ENABLE	schaltet die Integritätsüberprüfung für diese Tabelle ein (Standard).
DISABLE	schaltet die Integritätsüberprüfung für diese Tabelle aus.
PCTFREE	definiert den Speicherfreibereich in % (0 bis 99) eines Datenblockes für zukünftige Änderungen in dieser Tabelle.
PCTUSED	definiert den minimalen Belegungsgrad in % (0 bis 99) eines Datenblockes für eine Tabelle. (Vorgabewert: 40)
INITRANS	definiert eine Initialisierungszahl für Transaktionsinträge in einem Block. Dieser Wert steht standardmäßig auf 1.

Hinweis: Oracle 10g ignoriert MAXTRANS-Angaben und nimmt stillschweigend einen Wert von 255 dafür an, ohne einen Fehler auszugeben.

AS <i>query</i>	durch diese Option werden die Spalten der anzulegenden Tabelle genauso definiert, wie sie im SELECT-Befehl auf eine andere Tabelle angegeben sind. Wird diese Option benutzt, so muss beim Spaltelement nur der Name der Spalte angegeben werden, Datentypen und Größen werden aus der selektierten Tabelle übernommen.
CACHE	speichert Tabellen spezifische Datenblöcke in den most recently used Speicherbereich der SGA.
NOCACHE	speichert Tabellen spezifische Datenblöcke in den least recently used Speicherbereich der SGA (Standard).
ADD	mit ADD können Spalten in die Tabelle eingefügt werden.
MODIFY	ändert die Spaltendefinition

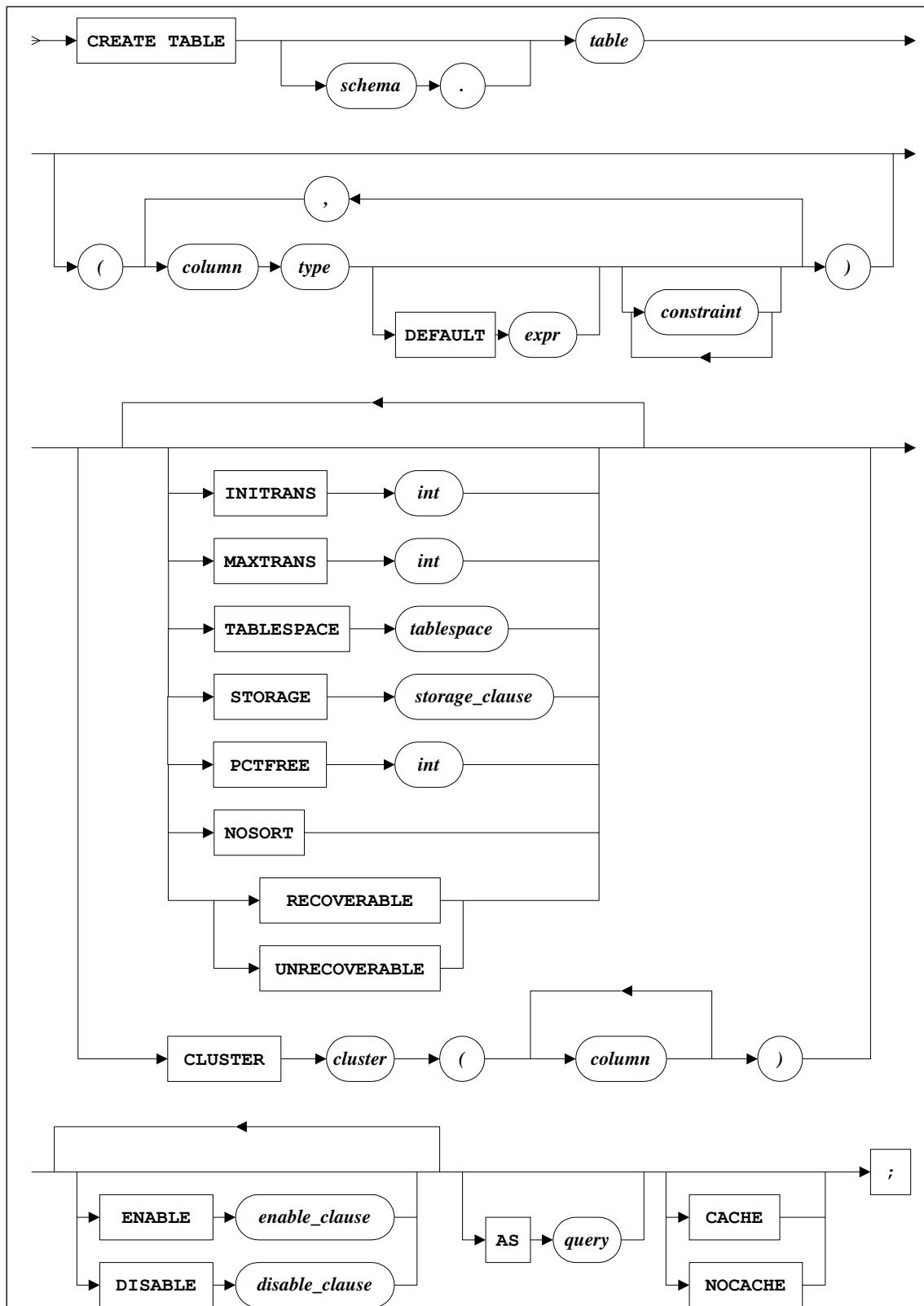


Abb. 12-1: CREATE TABLE Syntax (vereinfacht)

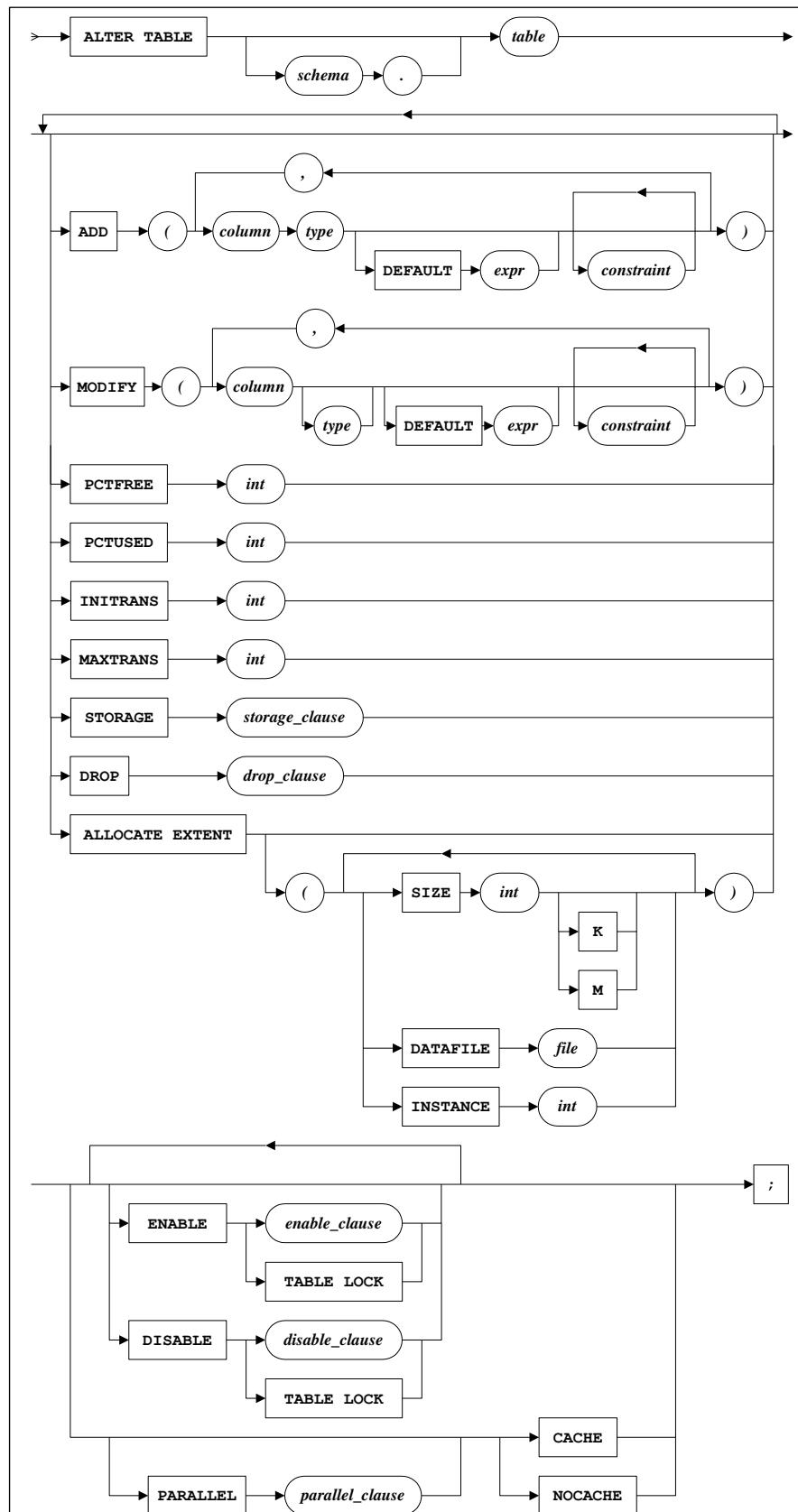


Abb. 12-2: ALTER TABLE Syntax (vereinfacht)

Objekt-Tabellen

Seit Oracle 8 lassen sich komplexe Objekttypen erstellen, so dass Objekte in Objekte eingebettet und an diese Objekte Methoden binden bzw. kapseln lassen.

Z. B.:

```
CREATE TYPE dept_t AS OBJECT
  (dname VARCHAR2(100),
   address VARCHAR2(200));
CREATE TABLE dept OF dept_t;
```

Nested Tables

Das Anlegen von Spalten einer Tabelle mit einem Type TABLE legt implizit im Speicher eine Tabelle für jede eingebettete Tabelle an. Diese eingebettete Tabelle liegt im gleichen Tablespace wie die Haupttabelle.

Z. B.:

```
CREATE TABLE employee (empno NUMBER, name CHAR(31),
  projects PROJ_TABLE_TYPE)
  NESTED TABLE projects STORE AS nested_proj_table;
```

Partitionierte Tabellen

Partitionierung von Tabellen und Indizes bedeutet die Aufteilung von großen Tabellen und Indizes in kleinere Einheiten, die separat verwaltet und angesprochen werden können. Anstelle der Verwaltung einer Tabelle als ein großes monolithisches Objekt, beruht die Partitionierung auf der sogenannten "divide and conquer" Technik, die eine skalierbare Performance für große Datenmengen zur Folge hat. Wie viele kleinere Tabellen hier entstehen, hängt wohl mehr von der organisatorischen Kraft der DBAs ab, da die ORACLE Obergrenze von 64.000 Partitionen pro Tabelle oder Index viel Spielraum lässt.

Die Partitionierung führt zu folgenden Vorteilen:

- Reduzierung der Zeit für Verwaltungsaufgaben (kleinere Verwaltungseinheiten)
- Verbesserung der Performance (durch Erhöhung der Parallelität)
- Verbesserung der Verfügbarkeit (durch Reduzierung der Folgen eines Ausfalls)

Beispiele:

```
CREATE TABLE verkauf
  (rechnungsnr number,
   jahr          number(4) NOT NULL,
   monat        number(2) NOT NULL,
   tag          number(2) NOT NULL)
PARTITION BY RANGE (jahr, monat, tag)
  PARTITION verkauf_1
    VALUES LESS THAN (1999, 01, 01)
      TABLESPACE tsl,
  PARTITION verkauf_2
    VALUES LESS THAN (1999, 07, 01)
      TABLESPACE ts2);
```

```
CREATE TABLE
  STORAGE (INITIAL 100M NEXT 100M)
  PARTITION BY HASH (hiredate)
  ( partition p1998 tablespace user_dat1,
    partition p1999 tablespace user_dat2,
    partition p2000 tablespace user_dat3);
```

```
CREATE TABLE
  STORAGE (INITIAL 100M NEXT 100M)
  PARTITION BY HASH (hiredate)
  PARTITIONS 4 STORE IN
  ( user_dat1, user_dat2, user_dat3, user_dat4);
```

12.3 CREATE INDEX / ALTER INDEX

Oracle unterstützt folgende Typen: B-Baum Indizes, B-Baum Cluster Indizes, Hash Cluster Indizes, Reverse Key Indizes, Bitmap Indizes.

Ähnlich wie bei Tabellen existiert auch für Indizes die Möglichkeit, Partitionen zu bilden. Grundsätzlich erfolgt die Definition von Partitionen ebenfalls anhand von Syntaxerweiterungen.

UNIQUE	Der Wert des Indexes ist eindeutig innerhalb der Tabelle.
BITMAP	Für jeden in der Spalte auftretenden Wert wird ein Vektor angelegt.
<i>schema</i>	Name des Schemas, in dem der Index angelegt wird.
<i>index</i>	Name des Indexes. Dieser muss innerhalb der zu einem Benutzer zugehörigen Objekte eindeutig sein.
<i>table</i>	Name der Tabelle, für die der Index angelegt wird.
<i>column</i>	Name der Tabellenspalte/n, auf die der Index gelegt werden soll. Maximal können 32 Spalten definiert werden. Spalten, die als LONG oder LONG RAW definiert wurden, können nicht angegeben werden.
ASC	Index wird aufsteigend sortiert (Standard).
DESC	Index wird absteigend sortiert.
CLUSTER	bezeichnet den Cluster, für den der Index angelegt wird.
INITTRANS	definiert eine Initialisierungszahl für Transaktionseinträge in einem Block. Dieser Wert steht standardmäßig auf 1 und sollte auch so belassen werden.
TABLESPACE	definiert den Tablespace, in dem der Index gespeichert werden soll.
STORAGE	definiert einen STORAGE-Wert, der zu Performance-Verbesserungen führt.
PCTFREE	definiert den Speicherfreibereich in % (0 bis 99) eines Datenblockes für zukünftige Änderungen.
NOSORT	weist das DBMS darauf hin, dass die Zeilen in der Tabelle bereits in aufsteigender Reihenfolge sortiert sind. Somit kann diese Option die Zeit, die zur Erzeugung des Indexes benötigt wird, enorm reduzieren. NOSORT sollte allerdings nur sofort nach erstmaligem Laden von Daten durchgeführt werden. Nur dann kann garantiert werden, dass die Daten in korrekter Reihenfolge vorliegen.
PARALLEL	bezeichnet den Grad der Parallelität, mit dem der Index angelegt werden soll

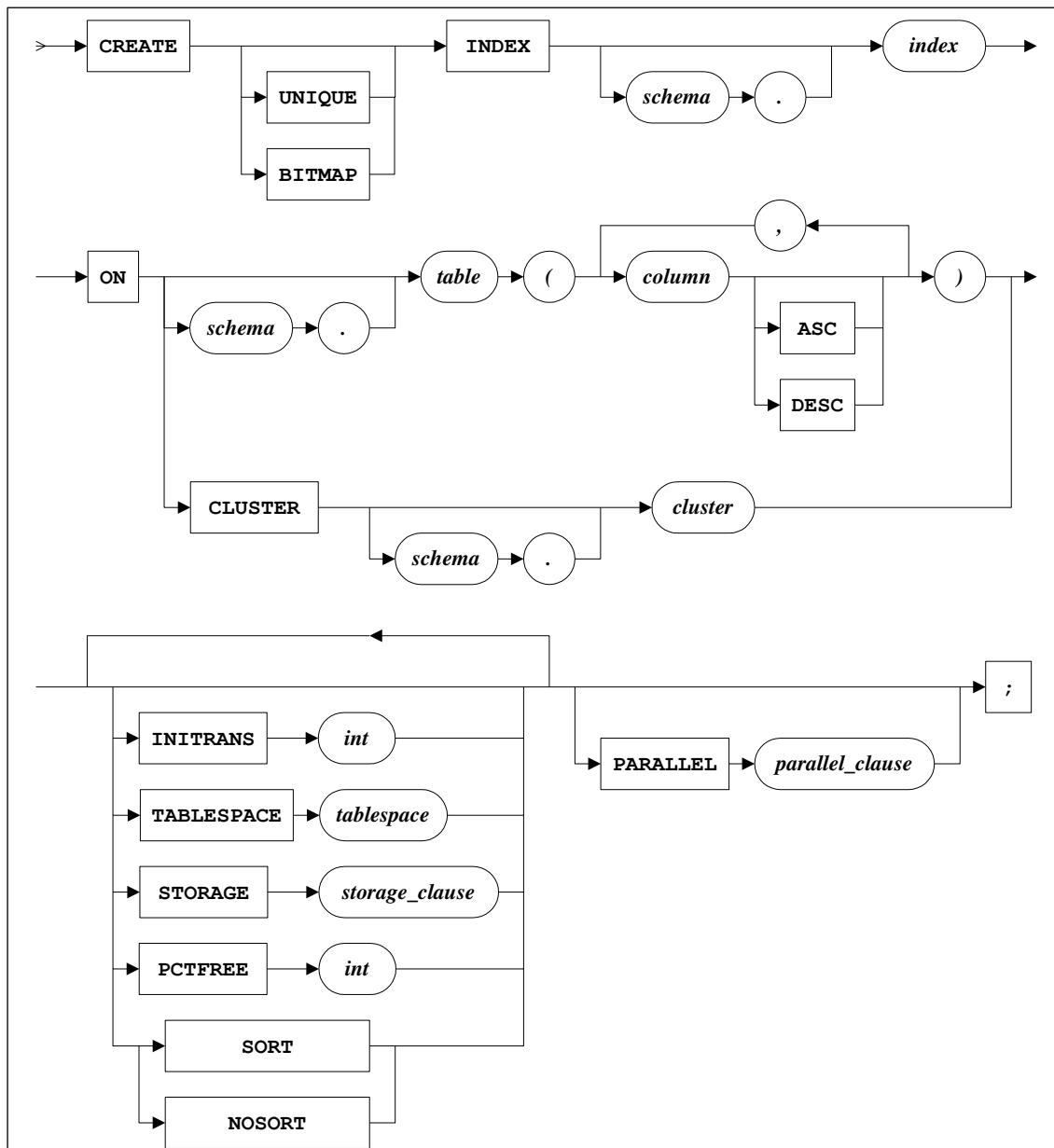


Abb. 12-3: CREATE INDEX Syntax (vereinfacht)

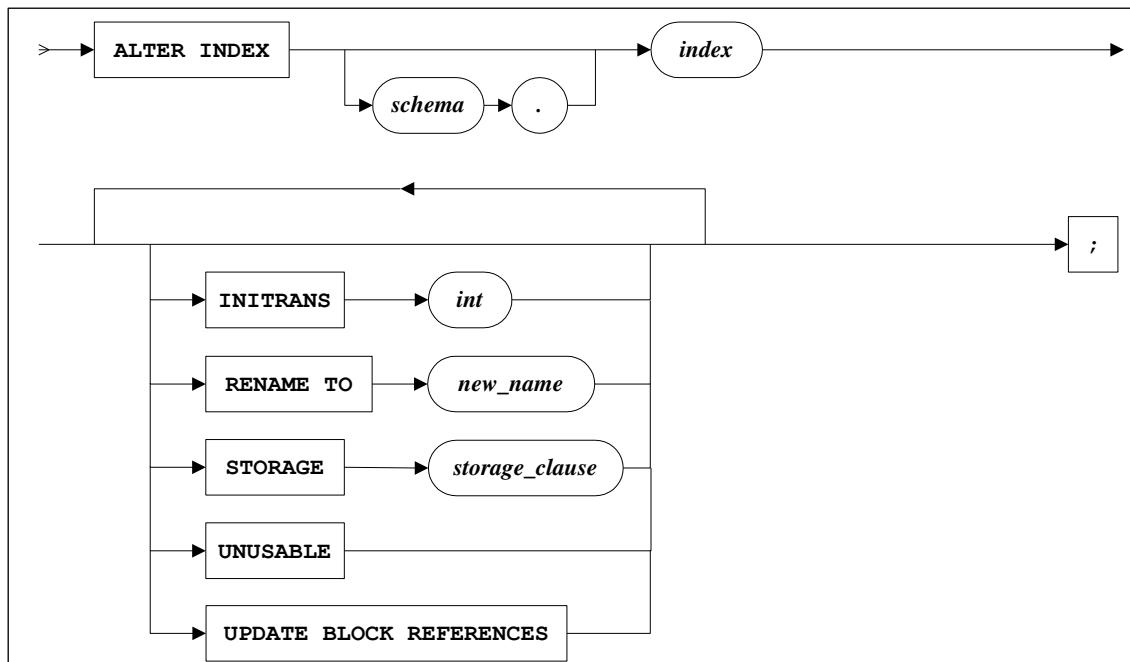


Abb. 12-4: ALTER INDEX Syntax (vereinfacht)

12.4 CREATE VIEW

OR REPLACE	Mit der OR REPLACE Angabe wird eine eventuell vorhandene View ersetzt.
FORCE	Die View wird auch erstellt, wenn die entsprechende Tabelle nicht vorhanden ist.
NO FORCE	Die View kann nur erstellt werden, wenn die entsprechenden Tabellen vorhanden sind. (Defaultwert)
<i>schema</i>	Ist das Schema, in dem die Tabelle angelegt wird.
<i>view</i>	Name der neuen View.
<i>alias</i>	Definiert Alias-Namen für die Spalten der View. Werden keine Alias-Namen angegeben, so werden die Spaltennamen der zugrundeliegenden Tabellen verwendet.
AS <i>subquery</i>	wählt aus bestehenden Tabellen und Views mit Hilfe des SELECT-Statements die Spalten und Sätze aus, die den neuen View bilden sollen. Werden im View Spalten benannt, so muss die Anzahl der Spalten in der Ergebnistabelle des SELECT mit der Anzahl der im View genannten Spalten übereinstimmen. Die SELECT-Anweisung darf keine ORDER BY- oder FOR UPDATE- Klauseln enthalten.
WITH READ ONLY	die View wird auf read only gesetzt.
WITH CHECK OPTION	Sätze, die Sie über einen View eingeben oder ändern, werden auf die Einhaltung der in der SELECT-Anweisung definierten Bedingung (WHERE-Klausel) überprüft. Sätze, welche die Bedingung nicht erfüllen, werden abgewiesen. Ist die Option nicht angegeben, können auch Sätze eingegeben werden, welche die Bedingung nicht erfüllen. Diese sind jedoch anschließend nicht mehr über den View zugreifbar.
CONSTRAINT	gibt Namen für Constraint an.

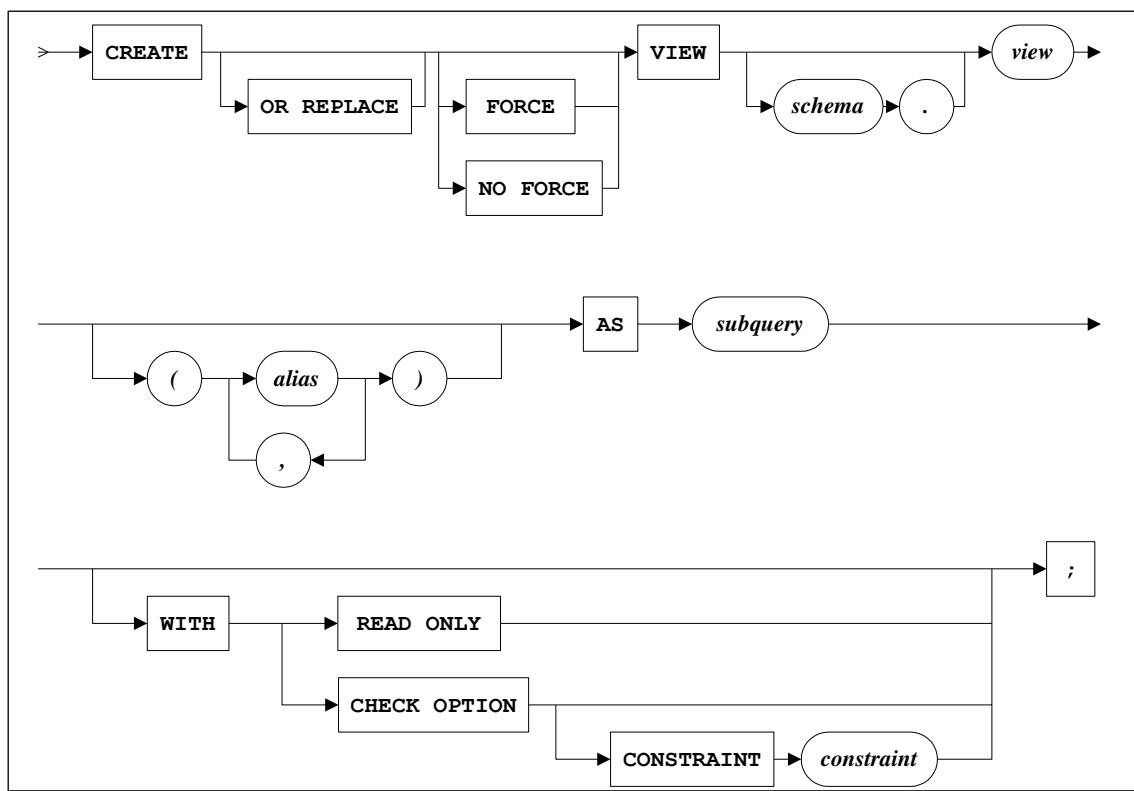


Abb. 12-5: CREATE VIEW Syntax (vereinfacht)

12.5 ALTER SEQUENCE

INCREMENT BY *n* spezifiziert das Intervall zwischen den einzelnen Sequenznummern. Das Intervall kann positive wie auch negative Werte annehmen (nicht 0, Default 1)

MINVALUE <i>n</i>	gibt die kleinste Sequenznummer an
NOMINVALUE	Default, Minimumvorgabe ist 1
MAXVALUE <i>n</i>	gibt die größte Sequenznummer an
NOMAXVALUE	Default, Maximalvorgabe ist 10^{27}
CYCLE	mit dieser Option wird beim Erreichen der größten bzw. der kleinsten Sequenznummer wieder mit dem Startwert begonnen.
NOCYCLE	mit diesem Parameter ist ein erneutes Erstellen von Sequenznummern nach Erreichen des Maximalwertes nicht möglich (Default)
CACHE <i>n</i>	CACHE gibt an, wie viele Sequenznummern im Speicher gehalten werden. Minimum: 2. (Default: CACHE 20)
NOCACHE	Es werden keine Sequenznummern im Speicher gehalten.

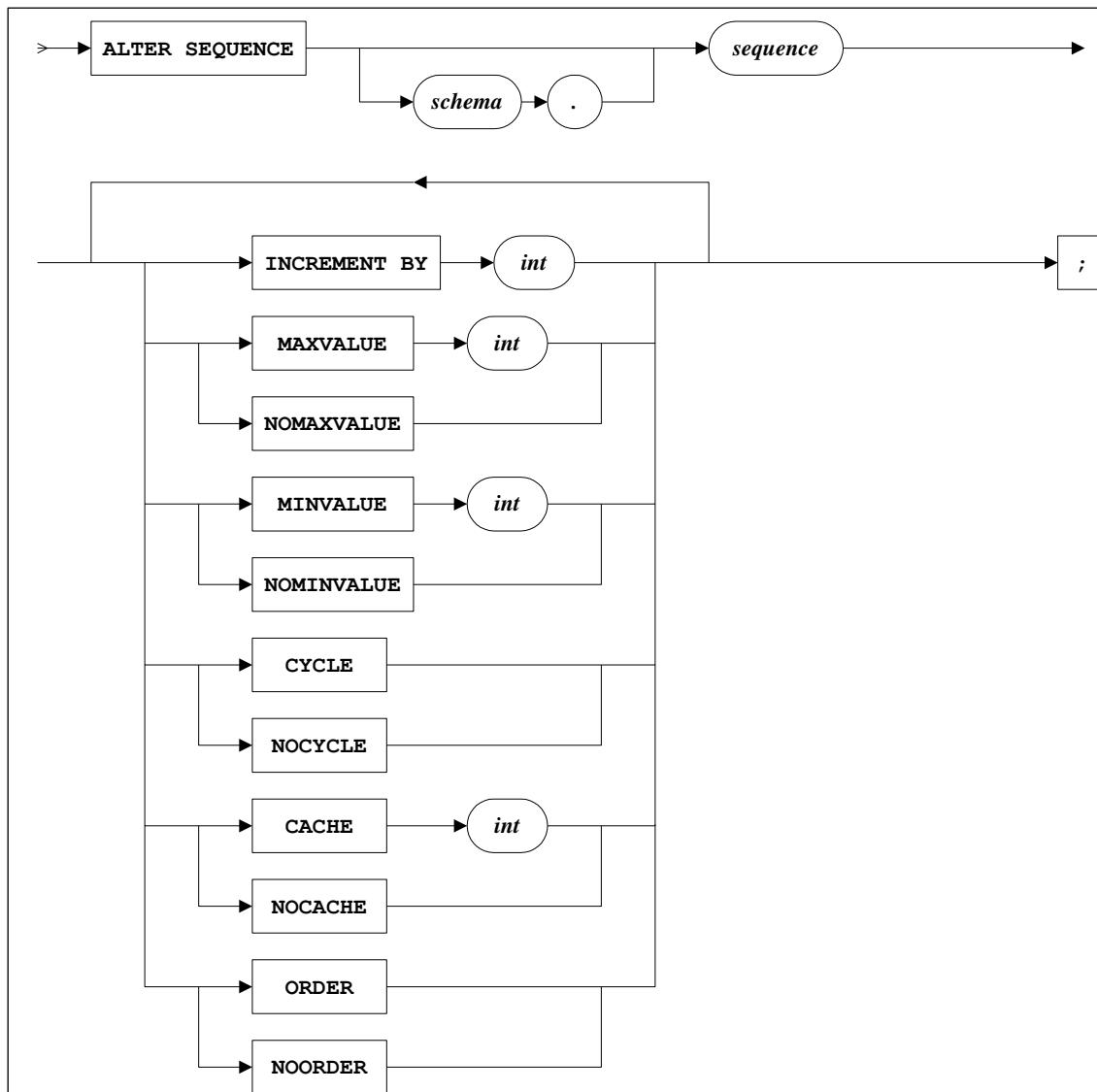


Abb. 12-6: ALTER SEQUENCE Syntax

12.6 SELECT-Anweisung

DISTINCT	Doppelte Sätze werden entfernt.
ALL	Doppelte Sätze in der Ergebnistabelle bleiben erhalten (Default)
*	alle Spalten der angegebenen Tabelle/n werden selektiert.
<i>table.*</i>	Selektiert alle Spalten der angegebenen Tabelle.
<i>view.*</i>	Selektiert alle Spalten der angegebenen View.
<i>snapshot.*</i>	Selektiert alle Spalten der angegebenen Materialized View (früher: Snapshot).
<i>expr</i>	Selektiert nach dem angegebenen Ausdruck.
<i>alias</i>	Anderer Name für eine Spalte.
<i>schema</i>	gibt das Schema an, in dem das zu selektierende Objekt vorhanden ist.
FROM	Angabe der oder des Objekte(s), aus denen selektiert werden soll(en):
<i>table</i>	Name der zu selektierenden Tabelle.
<i>view</i>	Name der zu selektierenden View.
<i>snapshot</i>	Name des zu selektierenden Materialized View (früher: Snapshot).
<i>subquery</i>	Beschreibung eines weiteren SELECT-Befehls.
WHERE <i>bedingung</i>	wird verwendet, um Suchkriterien zu definieren und um Verknüpfungen zwischen mehreren Tabellen zu definieren.
START WITH	gibt die gefundenen Elemente in geordneter Weise aus.
CONNECT BY	Hierarchische Abfrage
GROUP BY	Gruppierung von Tabellensätzen.
HAVING <i>bedingung</i>	Angabe von Gruppenbedingungen.
ORDER BY	Angabe von Sortierkriterien.

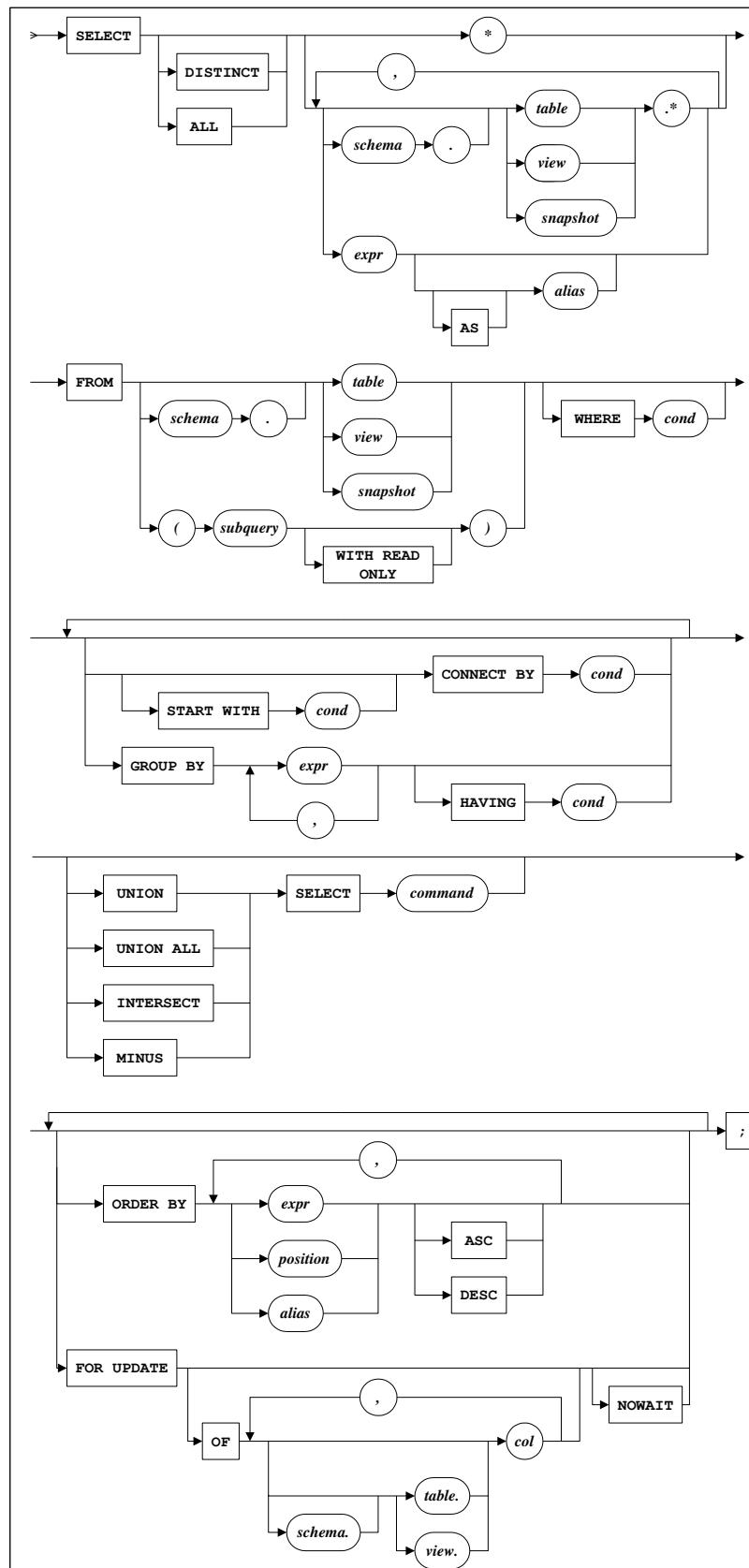


Abb. 12-7: SELECT Syntax (vereinfacht)

Die folgenden Operatoren verbinden die Ergebnisse von zwei SELECTs zu einer Menge

- UNION** eliminiert doppelte Ergebnisse aus beiden Mengen.
- UNION ALL** gibt doppelte Ergebnisse auch doppelt zurück
- INTERSECT** gibt nur die Ergebnisse zurück, die in beiden SELECTs vorkommen.
- MINUS** gibt die Ergebnisse aus dem ersten SELECT zurück, nachdem vorher die Ergebnisse eliminiert wurden, die auch vom zweiten SELECT zurückgegeben wurden.

12.7 DML-Befehle

12.7.1 INSERT

- schema* bezeichnet das Schema, welches das Objekt beinhaltet.
table steht für den Namen der Tabelle
view steht alternativ für den Namen der View
subquery1 bezeichnet eine Subquery, die wie eine View behandelt wird.
@dblink ist der Name für die Tabelle einer Remote-Datenbankanwendung.
column ist eine Spalte des Objekts (Tabelle, View, etc.)
VALUES bezeichnet eine bestimmte Anzahl von Werten, die eingefügt werden sollen.
subquery2 ist eine Subquery, die als Befehl Datensätze selektiert und zurückgibt, die wiederum in die angegebene Tabelle des **INSERT**-Befehls eingefügt werden.

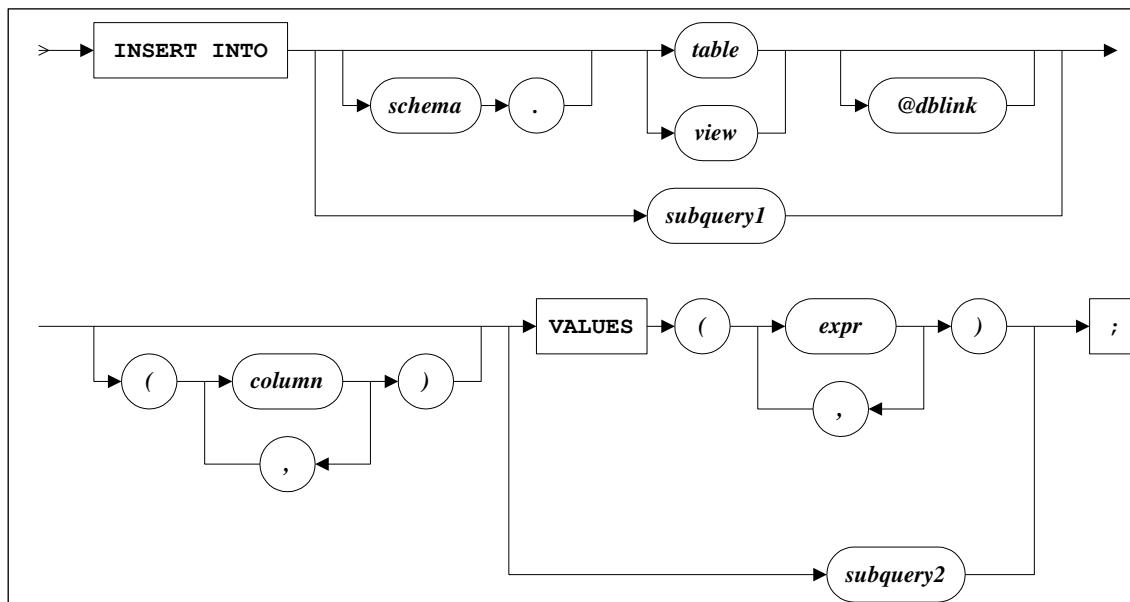


Abb. 12-8: **INSERT** Syntax (vereinfacht)

12.7.2 UPDATE

<i>schema</i>	bezeichnet das Schema, welches das Objekt beinhaltet.
<i>table</i>	steht für den Namen der Tabelle
<i>view</i>	steht alternativ für den Namen der View
<i>subquery1</i>	bezeichnet eine Subquery, die wie eine View behandelt wird.
<i>@dblink</i>	ist der Name für die Tabelle einer Remote-Datenbankanwendung.
<i>alias</i>	bezeichnet einen Aliasnamen.
WHERE	gibt an, welche Sätze geändert werden sollen. Ist die WHERE-Bedingung nicht angegeben, werden alle Sätze der Tabelle geändert.
<i>column = expr</i>	jeder angegebenen Spalte wird einzeln ein neuer Wert zugewiesen.
<i>subquery2</i>	diese Subquery gibt den entsprechenden Wert zurück, der als Ausdruck verwendet wird.
<i>subquery3</i>	diese Subquery gibt den entsprechenden Wert zurück, der als Ausdruck verwendet wird.

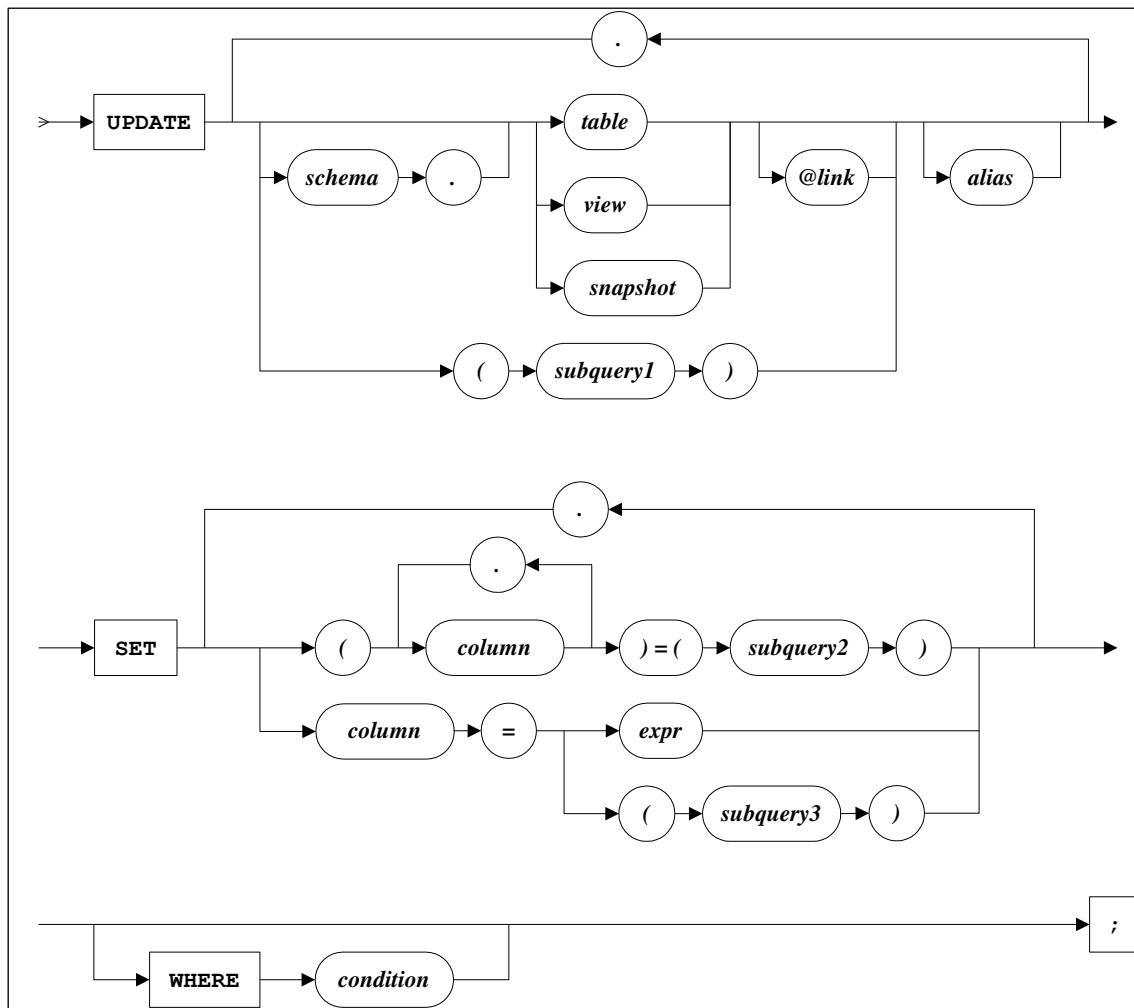


Abb. 12-9: UPDATE Syntax (vereinfacht)

12.7.3 DELETE

- schema* bezeichnet das Schema, welches das Objekt beinhaltet.
table steht für den Namen der Tabelle
view steht alternativ für den Namen der View
subquery bezeichnet eine Subquery, die zu löschen Daten selektiert.
@dblink ist der Name für die Tabelle einer Remote-Datenbankanwendung.
alias bezeichnet einen Aliasnamen.
WHERE gibt an, welche Sätze gelöscht werden sollen. Ist die WHERE-Bedingung nicht angegeben, werden alle Sätze der Tabelle gelöscht. Von der Bedingung hängt ab, welche Sätze gelöscht werden. Ein Satz wird nur gelöscht, wenn er die angegebene Bedingung erfüllt.

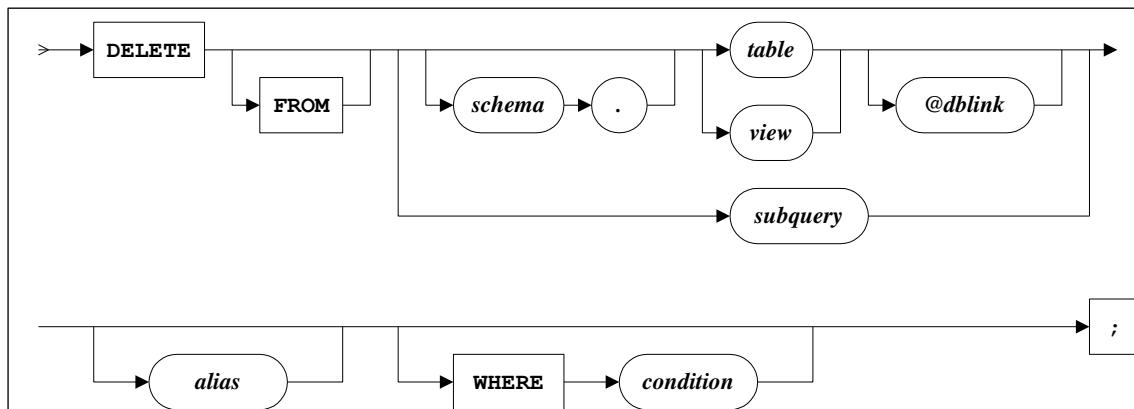


Abb. 12-10: DELETE Syntax (vereinfacht)

13

Literaturverzeichnis

13 Literaturverzeichnis

Allgemein:

[1]

Jeffrey E. F. Friedl: **Reguläre Ausdrücke**,
deutsche Übersetzung von Andreas Karrer,
473 Seiten,
O'Reilly Verlag GmbH & Co. KG
ISBN: 3-89721-349-4

SQL allgemein:

[2]

Chris J. Date, Hugh Darwen: **SQL – Der Standard.** SQL/92 mit Erweiterungen CLI und PSM,
582 Seiten
Addison-Wesley Verlag
ISBN: 3-8273-1345-7

Datenbanken allgemein:

[3]

Thomas Conolly, Carolyn Begg, Anne Strachan: **Datenbanksysteme**,
1077 Seiten
Addison-Wesley Verlag, 2002,
ISBN: 3-8273-2013-5

[4]

Alfons Kemper, Andre Eickler: **Datenbanksysteme**,
672 Seiten,
Oldenbourg Verlag,
ISBN: 3-486-57690-9

Datenbanken speziell:**[5]**

Ramez A. Elmasri, Shamkant Navathe: **Grundlagen von Datenbanksystemen**,
1103 Seiten,
Pearson Studium und Addison-Wesley, 3. Auflage, 2002,
ISBN: 3-8273-7021-3

ORACLE:**[6]**

Mark Gurry, Wolfgang Gabriel (Übersetzer): **Oracle SQL Tuning**,
120 Seiten,
Broschiert, O'Reilly & Associates,
ISBN: 3897212420

[7]

Oracle DBA Checklisten. Kurz und gut,
83 Seiten,
Broschiert, O'Reilly & Associates,
ISBN: 3897212366

[8]

Lutz Fröhlich: **ORACLE 19c/20c Das umfassende Praxishandbuch**,
876 Seiten
MITP,
ISBN: 978-3-7475-0057-6

[9]

Jonathan Gennick, Peter Linsley: **Oracle Regular Expressions**,
Pocket Reference,
60 Seiten,
O'Reilly & Associates,
ISBN: 0-596-00601-2
Beurteilung: Reguläre Ausdrücke in ORACLE. Kurz und knapp.

[10]

Steven Feuerstein, Bill Pribyl, Chip Dawes: **Oracle PL/SQL kurz und gut,**

98 Seiten,

Broschiert, O'Reilly & Associates,

ISBN: 389721217X

Steven Feuerstein, Bill Pribyl: **Oracle PL/SQL Programmierung**

1058 Seiten

Broschiert, O'Reilly & Associates,

ISBN: 3-89721-184-X

[11]

Daniel Warner: **Advanced SQL,**

533 Seiten,

Franzis Verlag,

ISBN: 978-3-7723-7170-7

Gesamtindex

*

* 3-8

/

/ -Befehl 2-7

1

1:1 1-21

1:n 1-21

12 Regeln 1-14

A

Abfrage 1-5

AFIEDT.BUF 2-13

Ähnlichkeitsprädikat 3-37

Alias 3-17, 3-18, 12-17

ALL 3-41, 5-41, 12-17

ALL_VIEWS 1-43

ALL-ANY-SOME Prädikate 3-41

ALTER INDEX 7-27, 12-10

ALTER PROFILE 8-44

ALTER ROLE 8-32

ALTER SEQUENCE 7-47, 12-15

ALTER TABLE 12-3

ALTER TABLE 7-12

ALTER USER 8-13

ALTER VIEW 7-36

Analytische Funktionen 9-3

ANY 3-41, 5-39

arithmetische Funktionen 4-3

arithmetische Operatoren 3-27

ASC 3-44

Attribut 1-6, 1-21, 1-27

Austauschvariablen 11-4

B

Befehls-Dateien 2-13

Befehls-Funktionen 2-13

Befehlspuffer 2-7

Benutzerprofile 8-39

Benutzerverwaltung 8-7

Befehle 8-9

beschreibende mengenorientierte Sprache
1-8

BETWEEN 3-35

Beziehungsintegrität 1-28

Beziehungstypen 1-21, 1-23, 1-30

BINARY_DOUBLE 3-49

BINARY_FLOAT 3-49, 3-51

C

CHAR 3-49

CHECK 7-57, 7-67

Codd 1-3

Codds 12 Regeln 1-14

COLUMN 2-17

COMMIT 6-14

CONNECT 2-17

CONNECT_BY_ISCYCLE 5-57, 5-59

CONNECT_BY_ISLEAF 5-57, 5-59

CONNECT_BY_ROOT 5-57, 5-59

Constraints 7-57

Constraintstatus 7-63, 7-68

Control-Datei 10-10

Control-Dateien 10-10

CREATE INDEX 7-26, 12-10

CREATE PROFILE 8-41

CREATE ROLE 8-31

CREATE SEQUENCE 7-42

CREATE SYNONYM 7-54

CREATE TABLE 7-5, 12-3

CREATE USER 8-10

CREATE VIEW 7-32, 12-13

D

Data Control Language 8-22

Data Definition Language 7-3

Data Dictionary 1-41, 10-9

Externe Ebene 1-41

Interne Ebene 1-41

Data Dictionary Views

ALL_USERS 8-15

- DBA_DATA_FILES 10-11
 DBA_SYS_PRIVS 8-16
 DBA_TABLESPACES 10-11
 DBA_TS_QUOTAS 8-15
 DBA_USERS 8-15
 Objektprivilegien 8-20
 SESSION_PRIVS 8-16
 USER_SYS_PRIVS 8-16
 USER_TS_QUOTAS 8-15
 USER_USERS 8-15
 Data Manipulation Language 3-3
 DATE 3-49
 Datenbank 1-5
 - Architektur 10-3
 - Kern 10-3
 - relationales Modell 1-3
 Datenbankbenutzer 8-5
 Datenbankdateien 10-9
 Datenbankdesign 1-19
 Datenbankmanagementsystem 1-8, 1-9
 Datendefinitionssprache 7-3
 Datenintegrität 1-9, 1-28
 Datenschutz 8-3
 Datentypen 1-5, 3-49
 - BFILE 3-50
 - BINARY_DOUBLE 3-49
 - BINARY_FLOAT 3-49, 3-51
 - BLOB 3-50
 - CHAR(n) 3-49
 - CLOB 3-50
 - DATE 3-49
 - INTERVAL DAY TO SECOND 3-50
 - INTERVAL YEAR TO MONTH 3-50
 - LOB 3-50
 - LONG 3-50
 - LONG RAW 3-50
 - LONG VARCHAR 3-50
 - NCHAR 3-50
 - NCLOB 3-50
 - NUMBER(n,m) 3-49
 - NVARCHAR2(n) 3-50
 - RAW(n) 3-50
 - ROWID 3-50
 - TIMESTAMP[(n)] 3-49, 3-50, 3-51
 - TIMESTAMP[(n)] WITH LOCAL TIME ZONE 3-51
 - TIMESTAMP[(n)] WITH TIME ZONE 3-51
 - VARCHAR(n) 3-49
 Datenunabhängigkeit 1-12
- Datumsfunktionen 4-13
 DB2 1-3
 DBA_VIEWS 1-43
 DBMS 1-8
 DCL 1-38, 8-22
 DDL 1-38, 7-3
 DEFAULT ROLE 8-34
 Default-Werte 6-7
 DEFINE_EDITOR 2-14
 DELETE 6-10, 12-23
 DENSE_RANK 9-11
 DESC 3-44
 DESCRIBE 2-17
 Direkte Vergleichsprädikate 3-34
 DISTINCT 3-14, 3-15, 12-17
 DML 1-38
 Drei-Ebenen-Architektur 1-11
 DROP INDEX 7-29
 DROP PROFILE 8-44
 DROP ROLE 8-34
 DROP SEQUENCE 7-48
 DROP SYNONYM 7-55
 DROP TABLE 7-15
 DROP USER 8-14
 - CASCADE 8-14
 DROP VIEW 7-37
- E**
- EBNF 3-4
 ed 2-7
 Editierfunktionen 2-7
 EERM 1-20
 Enhanced Entity-Relationship Modell 1-20
 Entitätsintegrität 7-57
 Entitätstypen 1-21, 1-23, 1-30
 Entity-Relationship Modell 1-20, 1-21
 ERM 1-20, 1-25, 1-27
 - ORACLE-Notation 1-26
 Erweiterte Backus-Naur-Form 3-4
 EXISTS 3-43, 5-38
- F**
- FALSCH 3-30
 Feld 1-5
 FK 1-29
 FOREIGN KEY 1-29, 7-57, 7-65
 Fremdschlüssel 1-29
 FROM 3-8, 3-9

G

GET Dateiname 2-13
glogin.sql 2-9
GRANT
 Objektprivilegien 8-24
 Systemprivilegien 8-22
GROUP BY 12-17
 CUBE 4-37
 ROLLUP 4-35

H

HAVING 12-17
HELP 2-17
HELP INDEX 2-7
Hierarchische Abfragen 5-49

I

IDENTIFIED EXTERNALLY 8-10
implizites COMMIT 7-3, 8-22
IN 3-39, 5-36
Index 7-24
INIT.ORA 8-34
Inline View 7-35
INSERT 6-5, 12-20
interne Ebene 1-11
INTERSECT 5-60, 12-19
Intervallprädikat 3-35
IS NULL 3-38

J

Join
 Cross-Join 5-25
 Equi-Join 5-21
 Kartesisches Produkt 5-4
 Natural-Join 5-25
 Self-Join 5-23
 USING-Klausel 5-27

K

KANN-Beziehung 1-25
Kardinalität der Beziehung 1-21
Konvertierungsfunktionen 4-10
konzeptionelle Ebene 1-11
konzeptuelles Design 1-19
künstliche Schlüssel 1-29

L

LAG 9-7
LEAD 9-7
LIKE 3-37
LIST 2-7
logische Datenunabhängigkeit 1-12
Logische Struktur 10-8
logisches Design 1-19
Lokale Variablen 11-4

M

m:n 1-21
mengenorientiert 1-3
Mengenvergleichsoperatoren 3-39, 3-41, 3-43
MERGE 6-12
Metazeichen 4-25
MINUS 5-60, 12-19
MUSS-Beziehung 1-25

N

n:1 1-21
Nested Tables 12-8
NOCYCLE 5-57, 5-58
Normalformen 1-32
Normalisierung 1-20, 1-32
NOT NULL 7-57, 7-64
NULL 1-5, 3-22, 3-27, 3-30, 3-38
NULL-Test Prädikat 3-38
NUMBER 3-49
NVL 3-23, 3-25, 3-27

O

Objektprivilegien 8-18
 ALL 8-18
 ALTER 8-18
 DELETE 8-18
 EXECUTE 8-18
 INDEX 8-18
 INSERT 8-18
 REFERENCES 8-18
 SELECT 8-18
 UPDATE 8-18
Objekt-Tabellen 12-8
Operatoren 3-20
Oracle SQL Developer 2-19
ORDER BY 3-44, 12-17

P

Partitionierung 12-8
 physische Datenunabhängigkeit 1-12
 physische Ebene 1-11
 Physische Struktur 10-9
 physisches Design 1-19
 Pivot 5-45
 PK 1-29
 Platzhaltersymbole 3-37
 Primär- und Fremdschlüsselprinzip 1-29
 Primärschlüssel 1-29
 PRIMARY KEY 1-29, 7-57, 7-64
 Pseudospalte 5-59

Q

Qualifizierte Angabe 3-12
 Query 1-5
 Quota 8-8, 8-10, 8-15

R

RANK 9-11
 RDB 1-9
 RDBMS 1-9
 Redo-Log-Dateien 10-9
 Referentielle Integrität 7-57, 7-65
 REGEXP_COUNT 4-23
 REGEXP_INSTR 4-19
 REGEXP_LIKE 4-21
 REGEXP_REPLACE 4-22
 REGEXP_SUBSTR 4-20
 regular expressions 4-17
 Reguläre Ausdrücke 4-17
 Rekursion 5-58
 Rekursive Abfragen 5-49
 Relation 1-6
 Relationale Algebra 1-36
 relationale Datenbankmanagementsysteme 1-9
 relationales Modell 1-3, 1-27, 1-30
 Relationen 1-3, 1-27
 RENAME 7-18
 REVOKE
 Objektprivilegien 8-27
 Systemprivilegien 8-25
 ROLLBACK 6-14
 ROLLBACK TO SAVEPOINT 6-15
 Rollen 8-28
 Setzen von Default Roles 8-34

Vordefinierte 8-36
 RUN 2-7

S

SAV[E] Dateiname 2-13
 SAVEPOINT 6-15
 SCHEMA 8-5
 schwacher Entitätstyp 1-23
 Security 8-3
 Domäne 8-7
 Verwaltung 8-30
 SELECT 3-3, 3-8, 3-10, 3-17, 12-17
 select-spezifikation 3-5
 select-statement 3-5, 5-60
 Semantische Datenintegrität 7-57
 SEQUEL 1-3
 Sequenzen 7-42
 SET 2-9
 SET ROLE 8-32
 SET TRANSACTION 6-16
 ISOLATION LEVEL READ COMMITTED 6-16
 ISOLATION LEVEL SERIALIZABLE 6-16
 READ ONLY 6-16
 READ WRITE 6-16
 SET-Klausel 6-8
 SGA 10-3
 SHOW 2-9
 Sicht 1-5
 SOME 3-41, 5-39
 sortieren 3-44
 Spalte 1-3, 1-5
 Spaltenalias 3-17
 Spaltenauswahl 3-10
 Spaltenreihenfolge 3-10
 SPOOL 2-15
 APPEND 2-15
 SQL 1-38
 SQL*Plus 2-3
 Merkmale 2-3
 starten 2-4
 Umgebungsparameter
 ECHO on/off 2-9
 FEED[BACK] on /off/n 2-9
 HEA[DING] on/off 2-10
 HEADS[EP] on/off/"zeichen" 2-10
 LIN[ESIZE] zahl 2-10
 LONG zahl 2-10

NEWP[AGE] zahl 2-10
NULL "zeichen" 2-10
NUM[WIDTH] zahl 2-10
PAGES[IZE] zahl 2-10
PAU[SE] on/off/"text" 2-10
SPA[CE] zahl 2-11
SQLP[ROMPT] "text" 2-11
SQLT[ERMINATOR] "zeichen"/on/ off 2-11
TERM[OUT] on/off 2-11
TI[ME] on/off 2-11
TIMI[NG] on/off 2-11
UND[ERLINE] on/off "zeichen" 2-11
VER[IFY] on/off 2-11

Umgebungsparameter 2-9
verlassen 2-5
starker Entitätstyp 1-23
START 2-13
Structured Query Language 1-38
SYNONYM 7-54
Syntaxdiagramm 3-4
System Global Area 10-3
System R 1-3
Systemarchitektur 10-3
Systemprivilegien 8-16
System-Rollback-Segment 10-9

T

Tabelle 1-3, 1-5
Tabellenalias 3-19
Table 1-6
Tablespace 10-8

System-Tablespace 10-9
Transaktion 1-9, 6-14
TRUNCATE TABLE 7-16
Tuple 1-6

U

unendliche Schleife 5-58
UNION 5-60, 12-19
UNION ALL 5-60
UNIQUE 7-57, 7-64
Unpivot 5-47
UPDATE 6-8, 12-21
USER_VIEWS 1-43

V

VALUES-Klausel 6-5, 6-6
VARCHAR2 3-49
Vektor 3-39
Vergleichsprädikate 3-32
Verkettungsoperator 3-28
View 1-5, 7-30

W

WAHR 3-30
WHERE 3-30, 4-39, 12-17
wildcard 3-37

Z

Zeichenkettenfunktionen 4-7
Zeile 1-3, 1-5, 1-6
zusammengesetzte Schlüssel 1-29

