

**Hardware and Software**  
Engineered to Work Together



# Oracle Database 12c: SQL Workshop I

Schulungsunterlagen – Band I

D80190DE11

Production 1.1 | Dezember 2014 | D88602

Learn more from Oracle University at [oracle.com/education/](http://oracle.com/education/)

**Autor**  
Dimpi Rani Sarmah

**Technischer Inhalt und  
Überarbeitung**

Nancy Greenberg  
Swarnapriya Shridhar  
Bryan Roberts  
Laszlo Czinkoczki  
KimSeong Loh  
Brent Dayley  
Jim Spiller  
Christopher Wensley  
Manish Pawar  
Clair Bennett  
Yanti Chang  
Joel Goodman  
Gerlinde Frenzen  
Madhavi Siddireddy

**Redaktion**  
Smita Kommini  
Aju Kumar

**Herausgeber**  
Pavithran Adka  
Giri Venugopal

**Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.**

Diese Software und zugehörige Dokumentation werden im Rahmen eines Lizenzvertrages zur Verfügung gestellt, der Einschränkungen hinsichtlich Nutzung und Offenlegung enthält und durch Gesetze zum Schutz geistigen Eigentums geschützt ist. Sofern nicht ausdrücklich in Ihrem Lizenzvertrag vereinbart oder gesetzlich geregelt, darf diese Software weder ganz noch teilweise in irgendeiner Form oder durch irgendein Mittel zu irgendeinem Zweck kopiert, reproduziert, übersetzt, gesendet, verändert, lizenziert, übertragen, verteilt, ausgestellt, ausgeführt, veröffentlicht oder angezeigt werden. Reverse Engineering, Disassemblierung oder Dekompilierung der Software ist verboten, es sei denn, dies ist erforderlich, um die gesetzlich vorgesehene Interoperabilität mit anderer Software zu ermöglichen.

Die hier angegebenen Informationen können jederzeit und ohne vorherige Ankündigung geändert werden. Wir übernehmen keine Gewähr für deren Richtigkeit. Sollten Sie Fehler oder Unstimmigkeiten finden, bitten wir Sie, uns diese schriftlich mitzuteilen.

Wird diese Software oder zugehörige Dokumentation an die Regierung der Vereinigten Staaten von Amerika bzw. einen Lizenznehmer im Auftrag der Regierung der Vereinigten Staaten von Amerika geliefert, gilt Folgendes:

**U.S. GOVERNMENT END USERS:**

Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

Oracle und Java sind eingetragene Marken der Oracle Corporation und/oder ihrer verbundenen Unternehmen. Andere Namen und Bezeichnungen können Marken ihrer jeweiligen Inhaber sein.

# Inhaltsverzeichnis

## 1 Einführung

Ziele	1-2
Lektionsagenda	1-3
Kursziele	1-4
Kursagenda	1-5
In diesem Kurs verwendete Anhänge und Übungen	1-7
Lektionsagenda	1-8
Oracle Database 12c: Kernbereiche	1-9
Oracle Database 12c	1-10
Oracle Fusion Middleware	1-12
Oracle Cloud	1-14
Oracle Cloud-Services	1-15
Cloud-Deployment-Modelle	1-16
Lektionsagenda	1-17
Relationale und objektrelationale Datenbankmanagementsysteme	1-18
Datenspeicherung auf verschiedenen Medien	1-19
Relationale Datenbanken – Konzept	1-20
Relationale Datenbanken – Definition	1-21
Datenmodelle	1-22
Entity-Relationship-Modell	1-23
Entity-Relationship-Modellierung – Konventionen	1-25
Beziehungen zwischen mehreren Tabellen	1-27
Relationale Datenbanken – Terminologie	1-29
Lektionsagenda	1-31
Datenbanken mit SQL abfragen	1-32
Im Kurs verwendete SQL-Anweisungen	1-33
SQL-Entwicklungsumgebungen	1-34
Lektionsagenda	1-35
Schema Human Resources (HR)	1-36
In diesem Kurs verwendete Tabellen	1-37
Lektionsagenda	1-38
Oracle Database – Dokumentation	1-39
Zusätzliche Ressourcen	1-40
Zusammenfassung	1-41
Übungen zu Lektion 1 – Überblick	1-42

## **2 Daten mit der SQL SELECT-Anweisung abrufen**

Ziele 2-2

Lektionsagenda 2-3

Einfache `SELECT`-Anweisungen 2-4

Alle Spalten wählen 2-5

Bestimmte Spalten wählen 2-6

SQL-Anweisungen schreiben 2-7

Standardwerte für Spaltenüberschriften 2-8

Lektionsagenda 2-9

Arithmetische Ausdrücke 2-10

Arithmetische Operatoren 2-11

Operatorpriorität 2-12

Nullwerte definieren 2-13

Nullwerte in arithmetischen Ausdrücken 2-14

Lektionsagenda 2-15

Spaltenaliasnamen definieren 2-16

Spaltenaliasnamen 2-17

Lektionsagenda 2-18

Verkettungsoperator 2-19

Literale Zeichenfolgen 2-20

Literale Zeichenfolgen – Beispiel 2-21

Operator `⋈` für alternative Anführungszeichen 2-22

Mehrfach vorhandene Zeilen 2-23

Lektionsagenda 2-24

Tabellenstrukturen anzeigen 2-25

Befehl `DESCRIBE` 2-26

Quiz 2-27

Zusammenfassung 2-28

Übungen zu Lektion 2 – Überblick 2-29

## **3 Daten einschränken und sortieren**

Ziele 3-2

Lektionsagenda 3-3

Zeilen durch Auswahl begrenzen 3-4

Auszugebende Zeilen einschränken 3-5

Klausel `WHERE` 3-6

Zeichenfolgen und Datumsangaben 3-7

Vergleichsoperatoren 3-8

Vergleichsoperatoren – Beispiel 3-9

Bereichsbedingungen mit dem Operator <code>BETWEEN</code>	3-10
Operator <code>IN</code> – Beispiel	3-11
Muster mit dem Operator <code>LIKE</code> vergleichen	3-12
Platzhalterzeichen kombinieren	3-13
<code>NULL</code> -Bedingungen	3-14
Bedingungen mit logischen Operatoren definieren	3-15
Operator <code>AND</code>	3-16
Operator <code>OR</code>	3-17
Operator <code>NOT</code>	3-18
Lektionsagenda	3-19
Prioritätsregeln	3-20
Lektionsagenda	3-22
Klausel <code>ORDER BY</code>	3-23
Daten sortieren	3-24
Lektionsagenda	3-26
SQL-Klausel zur Zeilenbeschränkung	3-27
SQL-Klausel zur Zeilenbeschränkung in Abfragen	3-28
SQL-Klausel zur Zeilenbeschränkung – Beispiel	3-29
Lektionsagenda	3-30
Substitutionsvariablen	3-31
Substitutionsvariable <code>&amp;</code>	3-33
Zeichen- und Datumswerte mit Substitutionsvariablen	3-35
Spaltennamen, Ausdrücke und Text angeben	3-36
Substitutionsvariable <code>&amp;&amp;</code>	3-37
Substitutionsvariablen in SQL*Plus	3-38
Lektionsagenda	3-39
Befehl <code>DEFINE</code>	3-40
Befehl <code>VERIFY</code>	3-41
Quiz	3-42
Zusammenfassung	3-43
Übungen zu Lektion 3 – Überblick	3-44

#### **4 Ausgabe mit Single-Row-Funktionen anpassen**

Ziele	4-2
Lektionsagenda	4-3
SQL-Funktionen	4-4
Zwei Typen von SQL-Funktionen	4-5
Single-Row-Funktionen	4-6
Lektionsagenda	4-8
Zeichenfunktionen	4-9

Funktionen zur Umwandlung der Groß-/Kleinschreibung von Zeichenfolgen 4-11  
Funktionen zur Umwandlung der Groß-/Kleinschreibung von Zeichenfolgen –  
    Beispiel 4-12  
Funktionen zum Bearbeiten von Zeichen 4-13  
Funktionen zum Bearbeiten von Zeichen – Beispiel 4-14  
Lektionsagenda 4-15  
Funktionen verschachteln 4-16  
Funktionen verschachteln – Beispiel 4-17  
Lektionsagenda 4-18  
Numerische Funktionen 4-19  
Funktion ROUND 4-20  
Funktion TRUNC 4-21  
Funktion MOD 4-22  
Lektionsagenda 4-23  
Mit Datumswerten arbeiten 4-24  
Datumsformat RR 4-25  
Funktion SYSDATE 4-27  
Funktionen CURRENT\_DATE und CURRENT\_TIMESTAMP 4-28  
Mit Datumswerten rechnen 4-29  
Arithmetische Operatoren für Datumswerte 4-30  
Lektionsagenda 4-31  
Funktionen zum Bearbeiten von Datumswerten 4-32  
Datumsfunktionen 4-33  
Funktionen ROUND und TRUNC für Datumswerte 4-34  
Quiz 4-35  
Zusammenfassung 4-36  
Übungen zu Lektion 4 – Überblick 4-37

## **5 Konvertierungsfunktionen und bedingte Ausdrücke**

Ziele 5-2  
Lektionsagenda 5-3  
Konvertierungsfunktionen 5-4  
Implizite Datentypkonvertierung 5-5  
Explizite Datentypkonvertierung 5-7  
Lektionsagenda 5-9  
Funktion TO\_CHAR in Verbindung mit Datumswerten 5-10  
Datumsformatmasken – Elemente 5-11  
Funktion TO\_CHAR in Verbindung mit Datumswerten 5-14  
Funktion TO\_CHAR in Verbindung mit Zahlenwerten 5-15  
Funktionen TO\_NUMBER und TO\_DATE 5-18

Funktionen `TO_CHAR` und `TO_DATE` in Verbindung mit dem Datumsformat `RR` 5-20

Lektionsagenda 5-21

Allgemeine Funktionen 5-22

Funktion `NVL` 5-23

Funktion `NVL` – Beispiel 5-24

Funktion `NVL2` – Beispiel 5-25

Funktion `NULLIF` – Beispiel 5-26

Funktion `COALESCE` – Beispiel 5-27

Lektionsagenda 5-29

Bedingte Ausdrücke 5-30

`CASE`-Ausdrücke 5-31

`CASE`-Ausdrücke – Beispiel 5-32

Searched `CASE`-Ausdrücke 5-33

Funktion `DECODE` 5-34

Funktion `DECODE` – Beispiel 5-35

Quiz 5-37

Zusammenfassung 5-38

Übungen zu Lektion 5 – Überblick 5-39

## **6 Mit Gruppenfunktionen Berichte aus aggregierten Daten erstellen**

Ziele 6-2

Lektionsagenda 6-3

Gruppenfunktionen 6-4

Typen von Gruppenfunktionen – Wiederholung 6-5

Gruppenfunktionen – Syntax 6-6

Funktionen `AVG` und `SUM` 6-7

Funktionen `MIN` und `MAX` 6-8

Funktion `COUNT` 6-9

Schlüsselwort `DISTINCT` 6-10

Gruppenfunktionen und Nullwerte 6-11

Lektionsagenda 6-12

Datengruppen erstellen 6-13

Datengruppen erstellen – Syntax der Klausel `GROUP BY` 6-14

Klausel `GROUP BY` 6-15

Nach mehreren Spalten gruppieren 6-17

Klausel `GROUP BY` über mehrere Spalten 6-18

Unzulässige Abfragen mit Gruppenfunktionen 6-19

Gruppenergebnisse einschränken 6-21

Gruppenergebnisse mit der Klausel `HAVING` einschränken 6-22

Klausel `HAVING` 6-23  
Lektionsagenda 6-25  
Gruppenfunktionen verschachteln 6-26  
Quiz 6-27  
Zusammenfassung 6-28  
Übungen zu Lektion 6 – Überblick 6-29

## **7 Daten aus mehreren Tabellen mit Joins anzeigen**

Ziele 7-2  
Lektionsagenda 7-3  
Daten aus mehreren Tabellen abrufen 7-4  
Typen von Joins 7-5  
Tabellen mithilfe der SQL:1999-Syntax verknüpfen 7-6  
Lektionsagenda 7-7  
Natural Joins erstellen 7-8  
Datensätze mit Natural Joins abrufen 7-9  
Joins mit der Klausel `USING` erstellen 7-10  
Spaltennamen verknüpfen 7-11  
Datensätze mit der Klausel `USING` abrufen 7-12  
Mehrdeutige Spaltennamen eindeutig kennzeichnen 7-13  
Tabellenaliasnamen mit der Klausel `USING` 7-14  
Joins mit der Klausel `ON` erstellen 7-15  
Datensätze mit der Klausel `ON` abrufen 7-16  
3-Way Joins erstellen 7-17  
Zusätzliche Bedingungen in Joins anwenden 7-18  
Lektionsagenda 7-19  
Tabellen mit sich selbst verknüpfen 7-20  
Self Joins mit der Klausel `ON` 7-21  
Lektionsagenda 7-22  
Non Equij Joins 7-23  
Datensätze mit Non-Equi Joins abrufen 7-24  
Lektionsagenda 7-25  
Mit `OUTER`-Joins Datensätze ohne direkte Übereinstimmung zurückgeben 7-26  
`INNER`- und `OUTER`-Joins – Vergleich 7-27  
`LEFT OUTER JOINS` 7-28  
`RIGHT OUTER JOINS` 7-29  
`FULL OUTER JOINS` 7-30  
Lektionsagenda 7-31  
Kartesische Produkte 7-32  
Kartesische Produkte generieren 7-33



Cross Joins erstellen 7-34  
Quiz 7-35  
Zusammenfassung 7-36  
Übungen zu Lektion 7 – Überblick 7-37

## **8 Unterabfragen in Abfragen**

Ziele 8-2  
Lektionsagenda 8-3  
Problemstellungen mit Unterabfragen lösen 8-4  
Unterabfragen – Syntax 8-5  
Unterabfragen 8-6  
Unterabfragen – Regeln und Richtlinien 8-7  
Typen von Unterabfragen 8-8  
Lektionsagenda 8-9  
Single-Row-Unterabfragen 8-10  
Single-Row-Unterabfragen ausführen 8-11  
Gruppenfunktionen in Unterabfragen – Beispiel 8-12  
Unterabfragen in `HAVING`-Klauseln 8-13  
Wo liegt der Fehler in dieser Anweisung? 8-14  
Innere Abfrage gibt keine Zeilen zurück 8-15  
Lektionsagenda 8-16  
Multiple-Row-Unterabfragen 8-17  
Operator `ANY` in Multiple-Row-Unterabfragen – Beispiel 8-18  
Operator `ALL` in Multiple-Row-Unterabfragen – Beispiel 8-19  
Multiple-Column-Unterabfragen 8-20  
Multiple-Column-Unterabfragen – Beispiel 8-21  
Lektionsagenda 8-22  
Nullwerte in Unterabfragen 8-23  
Quiz 8-25  
Zusammenfassung 8-26  
Übungen zu Lektion 8 – Überblick 8-27

## **9 Mengenoperatoren**

Ziele 9-2  
Lektionsagenda 9-3  
Mengenoperatoren 9-4  
Mengenoperatoren – Richtlinien 9-5  
Oracle-Server und Mengenoperatoren 9-6  
Lektionsagenda 9-7  
In dieser Lektion verwendete Tabellen 9-8

Lektionsagenda 9-12  
Operator UNION 9-13  
Operator UNION – Beispiel 9-14  
Operator UNION ALL 9-15  
Operator UNION ALL – Beispiel 9-16  
Lektionsagenda 9-17  
Operator INTERSECT 9-18  
Operator INTERSECT – Beispiel 9-19  
Lektionsagenda 9-20  
Operator MINUS 9-21  
Operator MINUS – Beispiel 9-22  
Lektionsagenda 9-23  
SELECT-Anweisungen abgleichen 9-24  
SELECT-Anweisungen abgleichen – Beispiel 9-25  
Lektionsagenda 9-26  
ORDER BY-Klauseln in Mengenoperationen 9-27  
Quiz 9-28  
Zusammenfassung 9-29  
Übungen zu Lektion 9 – Überblick 9-30

## **10 Tabellen mit DML-Anweisungen verwalten**

Ziele 10-2  
Lektionsagenda 10-3  
Data Manipulation Language 10-4  
Tabellen neue Zeilen hinzufügen 10-5  
Anweisung INSERT – Syntax 10-6  
Neue Zeilen einfügen 10-7  
Zeilen mit Nullwerten einfügen 10-8  
Spezielle Werte einfügen 10-9  
Spezielle Datums- und Uhrzeitwerte einfügen 10-10  
Skripte erstellen 10-11  
Zeilen aus anderen Tabellen kopieren 10-12  
Lektionsagenda 10-13  
Daten in Tabellen ändern 10-14  
UPDATE-Anweisungen – Syntax 10-15  
Zeilen einer Tabelle aktualisieren 10-16  
Zwei Spalten mithilfe einer Unterabfrage aktualisieren 10-17  
Zeilen auf Basis einer anderen Tabelle aktualisieren 10-18  
Lektionsagenda 10-19  
Zeilen aus Tabellen entfernen 10-20

DELETE-Anweisungen	10-21
Zeilen aus Tabellen löschen	10-22
Zeilen auf Basis einer anderen Tabelle löschen	10-23
TRUNCATE-Anweisungen	10-24
Lektionsagenda	10-25
Datenbanktransaktionen	10-26
Datenbanktransaktionen – Beginn und Ende	10-27
COMMIT- und ROLLBACK-Anweisungen – Vorteile	10-28
Explizite Steueranweisungen für Transaktionen	10-29
Änderungen bis zu einer Markierung zurückrollen	10-30
Implizite Transaktionsverarbeitung	10-31
Zustand der Daten vor COMMIT oder ROLLBACK	10-33
Zustand der Daten nach COMMIT	10-34
Daten festschreiben	10-35
Zustand der Daten nach ROLLBACK	10-36
Zustand der Daten nach ROLLBACK – Beispiel	10-37
Rollbacks auf Anweisungsebene	10-38
Lektionsagenda	10-39
Lesekonsistenz	10-40
Lesekonsistenz implementieren	10-41
Lektionsagenda	10-42
Klausel FOR UPDATE in SELECT-Anweisungen	10-43
Klausel FOR UPDATE – Beispiele	10-44
Quiz	10-46
Zusammenfassung	10-47
Übungen zu Lektion 10 – Überblick	10-48

## **11 Data Definition Language – Einführung**

Ziele	11-2
Lektionsagenda	11-3
Datenbankobjekte	11-4
Benennungsregeln	11-5
Lektionsagenda	11-6
CREATE TABLE-Anweisungen	11-7
Tabellen erstellen	11-8
Lektionsagenda	11-9
Datetime-Datentypen	11-12
Option DEFAULT	11-13
Lektionsagenda	11-14
Constraints hinzufügen	11-15

Constraints – Richtlinien	11-16
Constraints definieren	11-17
Constraint NOT NULL	11-19
Constraint UNIQUE	11-20
Constraint PRIMARY KEY	11-22
Constraint FOREIGN KEY	11-23
Constraint FOREIGN KEY – Schlüsselwörter	11-25
Constraint CHECK	11-26
CREATE TABLE – Beispiel	11-27
Constraints verletzen	11-28
Lektionsagenda	11-30
Tabellen mithilfe von Unterabfragen erstellen	11-31
Lektionsagenda	11-33
ALTER TABLE-Anweisungen	11-34
Spalten hinzufügen	11-36
Spalten ändern	11-37
Spalten löschen	11-38
Option SET UNUSED	11-39
Schreibgeschützte Tabellen	11-41
Lektionsagenda	11-42
Tabellen löschen	11-43
Quiz	11-44
Zusammenfassung	11-45
Übungen zu Lektion 11 – Überblick	11-46

## **A Tabellenbeschreibungen**

## **B SQL Developer**

Ziele	B-2
Was ist Oracle SQL Developer?	B-3
SQL Developer – Spezifikationen	B-4
SQL Developer 3.2 – Benutzeroberfläche	B-5
Datenbankverbindungen erstellen	B-7
Datenbankobjekte durchsuchen	B-10
Tabellenstrukturen anzeigen	B-11
Dateien durchsuchen	B-12
Schemaobjekte erstellen	B-13
Neue Tabellen erstellen – Beispiel	B-14
SQL Worksheet	B-15
SQL-Anweisungen ausführen	B-19

SQL-Skripte speichern B-20  
Gespeicherte Skriptdateien ausführen – Methode 1 B-21  
Gespeicherte Skriptdateien ausführen – Methode 2 B-22  
SQL-Code formatieren B-23  
Snippets B-24  
Snippets – Beispiel B-25  
Papierkorb B-26  
Prozeduren und Funktionen debuggen B-27  
Datenbankberichte B-28  
Benutzerdefinierte Berichte erstellen B-29  
Suchmaschinen und externe Tools B-30  
Voreinstellungen festlegen B-31  
SQL Developer-Layout zurücksetzen B-33  
Data Modeler in SQL Developer B-34  
Zusammenfassung B-35

## **C SQL\*Plus**

Ziele C-2  
SQL und SQL\*Plus – Interaktion C-3  
SQL-Anweisungen und SQL\*Plus-Befehle – Vergleich C-4  
SQL\*Plus – Überblick C-5  
Bei SQL\*Plus anmelden C-6  
Tabellenstrukturen anzeigen C-7  
SQL\*Plus – Bearbeitungsbefehle C-9  
LIST, n und APPEND C-11  
Befehl CHANGE C-12  
SQL\*Plus – Dateibefehle C-13  
Befehle SAVE und START C-14  
Befehl SERVEROUTPUT C-15  
SQL\*Plus-Befehl SPOOL C-16  
Befehl AUTOTRACE C-17  
Zusammenfassung C-18

## **D Häufig verwendete SQL-Befehle**

Ziele D-2  
Einfache SELECT-Anweisungen D-3  
SELECT-Anweisungen D-4  
WHERE-Klauseln D-5  
ORDER BY-Klauseln D-6  
GROUP BY-Klauseln D-7

Data Definition Language	D-8
CREATE TABLE-Anweisungen	D-9
ALTER TABLE-Anweisungen	D-10
DROP TABLE-Anweisungen	D-11
GRANT-Anweisungen	D-12
Typen von Berechtigungen	D-13
REVOKE-Anweisungen	D-14
TRUNCATE TABLE-Anweisungen	D-15
Data Manipulation Language	D-16
INSERT-Anweisungen	D-17
UPDATE-Anweisungen – Syntax	D-18
DELETE-Anweisungen	D-19
Anweisungen zur Transaktionskontrolle	D-20
COMMIT-Anweisungen	D-21
ROLLBACK-Anweisungen	D-22
SAVEPOINT-Anweisungen	D-23
Joins	D-24
Typen von Joins	D-25
Mehrdeutige Spaltennamen eindeutig kennzeichnen	D-26
Natural Joins	D-27
Equi Joins	D-28
Datensätze mit Equi Joins abrufen	D-29
Zusätzliche Suchbedingungen mit den Operatoren AND und WHERE	D-30
Datensätze mit Non-Equi Joins abrufen	D-31
Datensätze mit USING-Klauseln abrufen	D-32
Datensätze mit ON-Klauseln abrufen	D-33
Left Outer Joins	D-34
Right Outer Joins	D-35
Full Outer Joins	D-36
Self Joins – Beispiel	D-37
Cross Joins	D-38
Zusammenfassung	D-39

# 1

## Einführung

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

# Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- Ziele des Kurses definieren
- Features von Oracle Database 12c auflisten
- Wichtigste Features von Oracle Cloud beschreiben
- Theoretische und physische Aspekte relationaler Datenbanken diskutieren
- Implementierung von relationalen (RDBMS) und objektrelationalen Datenbankmanagementsystemen (ORDBMS) auf dem Oracle-Server beschreiben
- Für diesen Kurs verwendbare Entwicklungsumgebungen bestimmen
- In diesem Kurs verwendete Datenbank und zugehöriges Schema beschreiben

**ORACLE**

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In dieser Lektion werden relationale Datenbankmanagementsysteme (RDBMS) vorgestellt. Außerdem erhalten Sie eine Einführung in die Entwicklungsumgebungen Oracle SQL Developer und SQL\*Plus, die zur Ausführung von SQL-Anweisungen und für Formatierungs- und Berichterstellungsaufgaben verwendet werden.



# Lektionsagenda

- **Kursziele, Agenda und Anhänge dieses Kurses**
- Oracle Database 12c und zugehörige Produkte – Überblick
- Relationales Datenbankmanagement – Überblick zu Konzepten und Begriffen
- SQL und zugehörige Entwicklungsumgebungen – Einführung
- In diesem Kurs verwendetes Schema Human Resources (HR) und verwendete Tabellen
- SQL-Dokumentation und zusätzliche Ressourcen zu Oracle Database 12c

**ORACLE**

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Kursziele

Nach Ablauf dieses Kurses haben Sie folgende Ziele erreicht:

- Hauptkomponenten von Oracle Database bestimmen
- Mithilfe von `SELECT`-Anweisungen Zeilen- und Spaltendaten aus Tabellen abrufen
- Berichte aus sortierten und eingeschränkten Daten erstellen
- Mithilfe von SQL-Funktionen benutzerdefinierte Daten generieren und abrufen
- Mithilfe komplexer Abfragen Daten aus mehreren Tabellen abrufen
- DML-(Data Manipulation Language-)Anweisungen zur Aktualisierung von Daten in Oracle Database ausführen
- DDL-(Data Definition Language-)Anweisungen zur Erstellung und Verwaltung von Schemaobjekten ausführen

**ORACLE**

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Dieser Kurs vermittelt eine Einführung in die Technologie von Oracle Database. Sie lernen die Grundkonzepte relationaler Datenbanken sowie die leistungsstarke Programmiersprache SQL kennen und eignen sich grundlegende SQL-Kenntnisse an, um Abfragen für einzelne oder mehrere Tabellen durchzuführen, Daten in Tabellen zu bearbeiten, Datenbankobjekte zu erstellen und Metadaten abzufragen.

# Kursagenda

- 1. Tag:
  - Einführung
  - Daten mit der SQL-Anweisung `SELECT` abrufen
  - Daten einschränken und sortieren
  - Ausgabe mithilfe von Single-Row-Funktionen anpassen
- 2. Tag:
  - Konvertierungsfunktionen und bedingte Ausdrücke
  - Mit Gruppenfunktionen Berichte aus aggregierten Daten erstellen
  - Mit Joins Daten aus mehreren Tabellen anzeigen
  - Unterabfragen in Abfragen verwenden

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

# Kursagenda

- 3. Tag:
  - Mengenoperatoren
  - Tabellen mithilfe von DML-Anweisungen verwalten
  - Data Definition Language (DDL) – Einführung

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## **In diesem Kurs verwendete Anhänge und Übungen**

- Anhang A: Tabellenbeschreibungen
- Anhang B: SQL Developer
- Anhang C: SQL\*Plus
- Anhang D: Häufig verwendete SQL-Befehle
- Übungen
  - Übungen und Lösungen
  - Zusätzliche Übungen und Lösungen

**ORACLE**

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

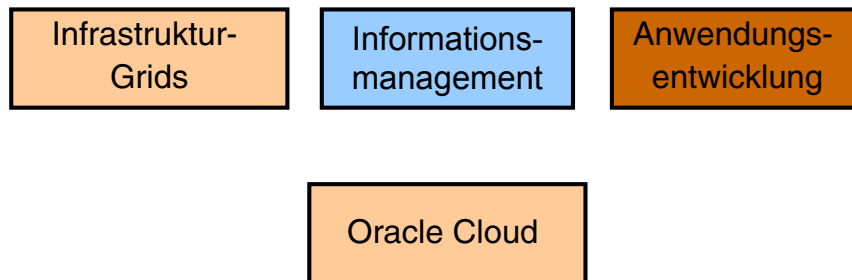
# Lektionsagenda

- Kursziele, Agenda und Anhänge dieses Kurses
- **Oracle Database 12c und zugehörige Produkte – Überblick**
- Relationales Datenbankmanagement – Überblick zu Konzepten und Begriffen
- SQL und zugehörige Entwicklungsumgebungen – Einführung
- In diesem Kurs verwendetes Schema Human Resources (HR) und verwendete Tabellen
- SQL-Dokumentation und zusätzliche Ressourcen zu Oracle Database 12c

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

# Oracle Database 12c – Kernbereiche



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Oracle Database 12c bietet umfassende Features in folgenden Kernbereichen:

- **Infrastruktur-Grids:** Mit der Infrastruktur-Grid-Technologie von Oracle lassen sich kostengünstige Server und Speicher in Pools zusammenfassen und daraus Systeme bilden, die höchste Servicequalität hinsichtlich Verwaltbarkeit, High Availability und Performance bieten. Oracle Database 12c konsolidiert und erweitert die Vorteile von Grid Computing. Neben der vollständigen Unterstützung von Grid Computing verfügt Oracle Database 12c über einzigartige Change Assurance-Funktionen, mit denen sich Änderungen gezielt und kostengünstig verwalten lassen.
- **Informationsmanagement:** Oracle Database 12c erweitert die bestehenden Informationsmanagementfunktionen in den Bereichen Content-Management, Informationsintegration und ILM (Information Lifecycle Management). Oracle unterstützt das Content-Management für erweiterte Datentypen wie XML (Extensible Markup Language), Text, räumliche Daten, Multimediainhalte, Bildgebungsverfahren in der Medizin sowie semantische Technologien.
- **Anwendungsentwicklung:** Mit Oracle Database 12c lassen sich alle gängigen Umgebungen zur Anwendungsentwicklung verwenden und verwalten, darunter PL/SQL, Java/JDBC, .NET und Windows, PHP, SQL Developer sowie Application Express.
- **Oracle Cloud:** Oracle Cloud ist eine Enterprise-Cloud für Unternehmen, die integrierte Cloudservices für Anwendungen und Plattformen auf Basis von Best-in-Class-Produkten und offenen Java- und SQL-Standards bereitstellt.

# Oracle Database 12c



Verwaltbarkeit  
High Availability  
Performance  
Sicherheit  
Informationsintegration

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Organisationen müssen in der Lage sein, mehrere Terabyte an Informationen zu unterstützen, auf die Benutzer von Geschäftsanwendungen rund um die Uhr schnell und sicher zugreifen können. Voraussetzung dafür sind zuverlässige Datenbanksysteme, die bei einem beliebigen Fehler rasch wiederhergestellt werden können. Bei der Entwicklung von Oracle Database 12c wurden folgende Kriterien zugrunde gelegt, um Organisationen bei der einfachen Verwaltung von Infrastruktur-Grids und der Bereitstellung hochwertiger Services zu unterstützen:

- **Verwaltbarkeit:** DBAs können mit bestimmten Features für Change Assurance, automatische Verwaltung und Fehlerdiagnose ihre Produktivität erhöhen, Kosten senken, Fehler minimieren und die Servicequalität maximieren. Zu diesen nützlichen Features für ein verbessertes Management gehören unter anderem die Funktionen für die Datenbankwiedergabe und das automatische SQL-Tuning, SQL Performance Analyzer und die Echtzeitüberwachung des Datenbankbetriebs.

Enterprise Manager Database Express 12c ist ein webbasiertes Verwaltungstool für Oracle-Datenbanken, mit dem sich die Datenbankperformance deutlich einfacher diagnostizieren lässt. Dabei werden alle für die Performance der Datenbank relevanten Bildschirme in einer einzigen konsolidierten View (**Database Performance Hub**) zusammengeführt. So können DBAs zentral aktuelle Echtzeit- und Verlaufsdaten zur Datenbankperformance einsehen und verschiedene Dimensionen wie Datenbanklast, überwachtes SQL und PL/SQL sowie Active Session History (ASH) für den gewählten Zeitraum auf einer einzelnen Seite untersuchen.



- **High Availability (HA):** Mit den HA-Features lässt sich die Gefahr von Ausfallzeiten und Datenverlusten senken. Dies verbessert den Onlinebetrieb und ermöglicht schnellere Datenbankupgrades.
- **Performance:** Mit SecureFiles, Komprimierung für OLTP (Online Transaction Processing), Optimierungen mit Real Application Clustern (RAC), Result-Caches usw. lässt sich die Performance von Datenbanken erheblich verbessern. Oracle Database 12c ermöglicht Organisationen die Verwaltung großer skalierbarer transaktionaler Systeme und Data Warehouse-Systeme, die sich durch schnellen Datenzugriff und kostengünstigen modularen Speicher auszeichnen.
- **Sicherheit:** Oracle Database 12c unterstützt Organisationen durch eindeutige, sichere Konfigurationen sowie Funktionen für Datenverschlüsselung, Datenmaskierung und anspruchsvolles Auditing beim Schutz ihrer Informationen. Damit steht eine sichere und skalierbare Plattform für den zuverlässigen und schnellen Zugriff auf alle Typen von Informationen mithilfe von Industriestandschnittstellen zur Verfügung.
- **Informationsintegration:** Oracle Database 12c bietet viele Features zur besseren Integration von Daten im Unternehmen und unterstützt auch erweiterte ILM-Funktionen. So lassen sich Datenänderungen in der Datenbank besser verwalten.

# Oracle Fusion Middleware

Oracle Fusion Middleware setzt sich aus marktführenden, standardbasierten und kundenerprobten Softwareprodukten zusammen. Das breite Spektrum an Tools und Services reicht von Java EE- und Entwicklertools bis hin zu Integration Services, Business Intelligence, Collaboration und Content-Management.



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Oracle Fusion Middleware ist eine umfassende und gut integrierte Produktfamilie, die lückenlose Unterstützung für die Entwicklung, das Deployment und die Verwaltung serviceorientierter Architekturen (SOA) bietet. SOA erleichtert die Entwicklung modularer Businessservices, die einfach integriert und wiederverwendet werden können. Dies senkt die Entwicklungs- und Wartungskosten und bietet eine höhere Servicequalität. Die integrierbare Architektur von Oracle Fusion Middleware gewährleistet Investitionsschutz für alle vorhandenen Anwendungen, Systeme und Technologien. Dank der unzerstörbaren Kerntechnologie werden Unterbrechungen durch geplante und nicht geplante Ausfälle minimiert.

Zur Oracle Fusion Middleware-Familie gehören unter anderem:

- **Anwendungsserver:** Java EE, Webservices
- **SOA und Prozessmanagement:** BPEL Process Manager, SOA Governance
- **Entwicklungstools:** Oracle Application Development Framework, JDeveloper, SOA Suite
- **Business Intelligence:** Oracle Business Activity Monitoring, Oracle Data Integrator
- **Unternehmensmanagement:** Enterprise Manager
- **Identitätsmanagement:** Oracle Identity Management
- **Content-Management:** Oracle Content Database Suite
- **Benutzerinteraktion:** Portal, Rich Internet Apps

# Oracle Enterprise Manager Cloud Control

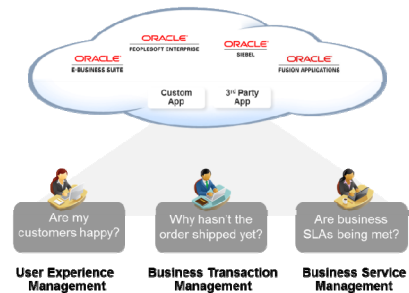
- Vollständigen Satz von Cloudservices erstellen und verwalten
- Sämtliche Phasen im Cloudlebenszyklus verwalten
- Gesamten Cloudstack verwalten
- Integrität aller Komponenten überwachen
- Geschäftsprobleme erkennen, verstehen und lösen



Gesamter Lebenszyklus



Kompletter Stack



Vollständige Integration

Selfservice-IT | Einfach und automatisch | Geschäftsorientiert

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Das Verwaltungstool Enterprise Manager Cloud Control stellt Überwachungs- und Verwaltungsfunktionen für Oracle-Komponenten und Komponenten anderer Hersteller bereit. Als umfassende, integrierte und geschäftsorientierte Cloudmanagementlösung in einem einzelnen Produkt wird es auch als "Total Cloud Control" bezeichnet.

Mit Enterprise Manager Cloud Control können Sie:

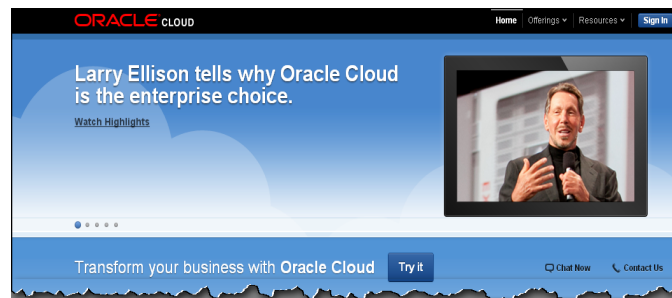
- einen vollständigen Satz von Cloudservices erstellen und verwalten: Infrastructure-as-a-Service, Database-as-a-Service, Platform-as-a-Service usw.
- sämtliche Phasen im Cloudlebenszyklus verwalten
- den gesamten Cloudstack verwalten: Application-to-Disk-Management, einschließlich Engineered Systems (Exa-Serie) und mit integrierten Supportmöglichkeiten
- die Integrität aller Komponenten, der zugehörigen Hosts und der wichtigsten unterstützten Geschäftsprozesse überwachen
- Geschäftsprobleme durch die einheitliche und korrelierte Verwaltung von Benutzererfahrung, Geschäftstransaktionen und Businessservices über alle als Package integrierte bzw. angepasste Anwendungen erkennen, verstehen und lösen

# Oracle Cloud

Oracle Cloud ist eine Enterprise-Cloud für den Unternehmensbetrieb, die sich aus zahlreichen unterschiedlichen Services mit gemeinsamen Merkmalen zusammensetzt:

- On-Demand-Selfservice
- Ressourcen-Pooling
- Schnelle Elastizität
- Messbarer Service
- Breiter Netzwerkzugriff

[www.cloud.oracle.com](http://www.cloud.oracle.com)



**ORACLE**

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Oracle Cloud ist eine Enterprise-Cloud für den Unternehmensbetrieb, die integrierte Cloudservices für Anwendungen und Plattformen auf Basis von Best-in-Class-Produkten und offenen Java- und SQL-Standards bereitstellt. Die beiden größten Vorteile von Cloud-Computing bestehen hinsichtlich Geschwindigkeit und Kosten.

Die in Oracle Cloud bereitgestellten Anwendungen und Datenbanken sind portierbar und können ohne Weiteres in und aus Private Clouds und lokalen Umgebungen verschoben werden.

- Alle Cloudservices lassen sich über eine Selfservice-Schnittstelle bereitstellen. Benutzer können ihre Cloudservices auf einer integrierten Entwicklungs- und Deployment-Plattform mit entsprechenden Tools zur schnellen Erweiterung und Erstellung neuer Services erhalten.
- Oracle Cloud-Services setzen auf Oracle Exalogic Elastic Cloud und Oracle Exadata Database Machine auf und bilden gemeinsam eine Plattform für extreme Performance, Redundanz und Skalierbarkeit.

Die fünf wesentlichen Merkmale von Oracle Cloud-Services sind:

- **On-Demand-Selfservice:** Gesteuerte Bereitstellung, Überwachung und Verwaltung
- **Ressourcen-Pooling:** Gemeinsame Nutzung und Abstraktionsebene zwischen Services und ihren Consumern
- **Schnelle Elastizität:** Rasche, anforderungsgerechte Auf- oder Abwärtsskalierung
- **Messbarer Service:** Nutzungsmessung für interne Rückbelastung (Private Cloud) oder externe Abrechnung (Public Cloud)
- **Breiter Netzwerkzugriff:** Browserbasierter Zugriff über jedes Netzwerkdevice

# Oracle Cloud-Services

Oracle Cloud stellt drei Typen von Services bereit:

- Software-as-a-Service (SaaS)
- Platform-as-a-Service (PaaS)
- Infrastructure-as-a-Service (IaaS)



**ORACLE**

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit dem Begriff SaaS werden in der Regel Anwendungen bezeichnet, die Endbenutzern über das Internet zur Verfügung gestellt werden. Ein Beispiel für ein solches SaaS-Angebot ist Oracle CRM On Demand, das je nach Vorliebe des Kunden als mehrmandantenfähige oder Einzelmandantenoption erhältlich ist.

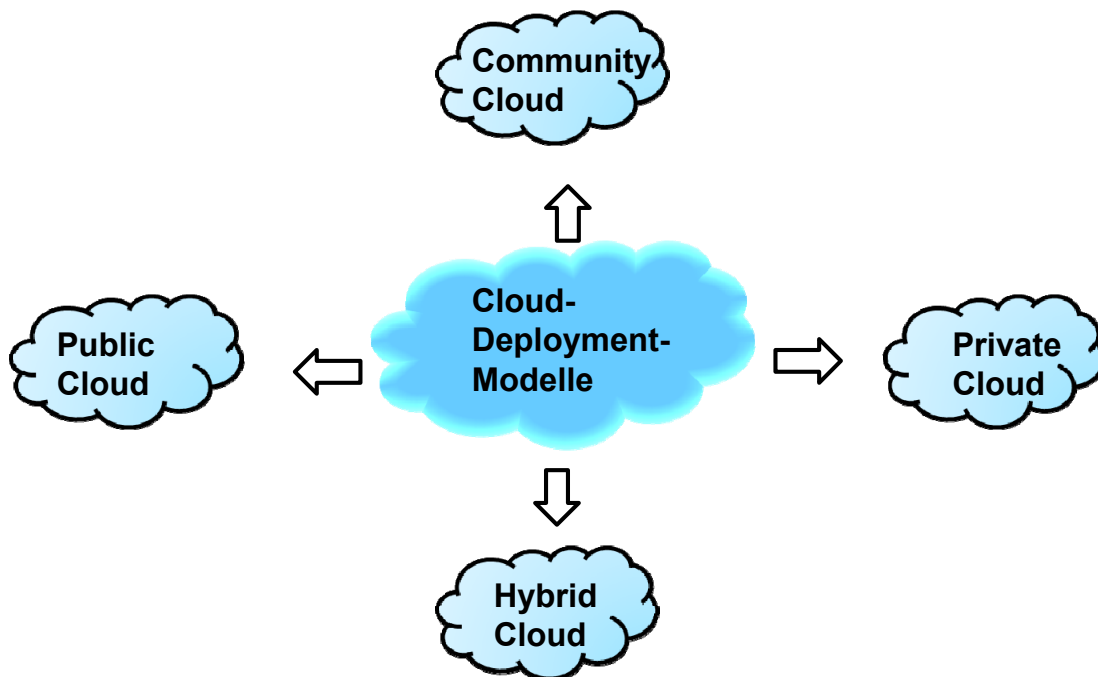
PaaS bezieht sich im Allgemeinen auf eine Plattform zur Anwendungsentwicklung und -bereitstellung, die Entwicklern als Service zur Verfügung gestellt wird und ihnen die rasche Entwicklung und Bereitstellung von SaaS-Anwendungen für Endbenutzer erlaubt. Typischerweise umfasst die Plattform Datenbanken, Middleware und Entwicklungstools, die allesamt als Service über das Internet bereitgestellt werden.

IaaS bezieht sich auf Hardware (Server, Speicher und Netzwerk), die Benutzern als Service überlassen wird. Dieser Service schließt in der Regel die zugehörige Software sowie Betriebssysteme, Virtualisierung, Clustering usw. ein. IaaS-Beispiele in der Public Cloud: Amazon Elastic Compute Cloud (EC2) und Amazon Simple Storage Service (S3)

Die Datenbankcloud wird in einer Private Cloud-Umgebung eines Unternehmens als PaaS-Modell erstellt und ermöglicht den elastisch skalierbaren und messbaren, anforderungsgerechten Zugriff auf Datenbankservices im Selfservice-Prinzip. Die Datenbankcloud bietet überzeugende Vorteile hinsichtlich Kosten, Servicequalität und Flexibilität. Datenbanken können innerhalb einer virtuellen Maschine (VM) auf einer IaaS-Plattform bereitgestellt werden.

Sie können Datenbankclouds schnell auf Oracle Exadata bereitstellen, das eine vorintegrierte und optimierte Hardwareplattform für OLTP- und DW-Workloads bietet.

# Cloud-Deployment-Modelle



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

- **Private Cloud:** Einzelne Organisationen nutzen Private Clouds, die in der Regel in privaten Data Centern gesteuert, verwaltet und gehostet werden. Allerdings können Hosting und Betrieb auch extern an einen Drittanbieter vergeben werden. Beispiel für eine von einem externen Anbieter gehostete Private Cloud: Amazon Virtual Private Cloud
- **Public Cloud:** Mehrere Organisationen (Mandanten) nutzen gemeinsam eine Private Cloud, die von einem Drittanbieter gehostet und verwaltet wird. Beispiele: Amazon Elastic Compute Cloud (EC2), IBM Blue Cloud, Sun Cloud, Google AppEngine usw.
- **Community Cloud:** Eine Gruppe zusammengehöriger Organisationen, die eine gemeinsame Umgebung für das Cloud-Computing verwenden möchten, nutzt eine Community Cloud, die entweder von den teilnehmenden Organisationen selbst oder einem Drittanbieter verwaltet wird. Das Hosting erfolgt intern oder extern. Beispiele: Communitys können aus den verschiedenen Bereichen des Militärs, aus allen Universitäten einer bestimmten Region oder aus allen Lieferanten eines großen Herstellers gebildet werden.
- **Hybrid Cloud:** Organisationen, die sowohl Private als auch Public Clouds für eine einzelne Anwendung nutzen möchten, entscheiden sich für eine Hybrid Cloud. Dieses Modell wird sowohl intern als auch über einen externen Anbieter verwaltet. Beispiel: Eine Organisation nutzt einen Public Cloud-Service wie Amazon Simple Storage Service (Amazon S3) für Archivdaten, speichert betrieblich genutzte Kundendaten aber weiterhin intern (in einer Private Cloud).

# Lektionsagenda

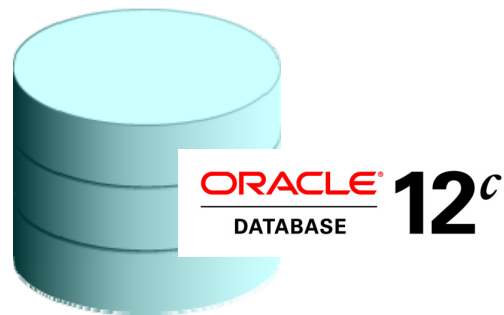
- Kursziele, Agenda und Anhänge dieses Kurses
- Oracle Database 12c und zugehörige Produkte – Überblick
- **Relationales Datenbankmanagement – Überblick zu Konzepten und Begriffen**
- SQL und zugehörige Entwicklungsumgebungen – Einführung
- In diesem Kurs verwendetes Schema Human Resources (HR) und verwendete Tabellen
- SQL-Dokumentation und zusätzliche Ressourcen zu Oracle Database 12c

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

# Relationale und objektrelationale Datenbankmanagementsysteme

- Relationales Modell und objektrelationales Modell
- Benutzerdefinierte Datentypen und Objekte
- Uneingeschränkt kompatibel zu relationalen Datenbanken
- Unterstützung für Multimediaobjekte und Large Objects
- Qualitativ hochwertige Datenbankserverfunktionen



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Der Oracle-Server unterstützt sowohl das relationale als auch das objektrelationale Datenbankmodell.

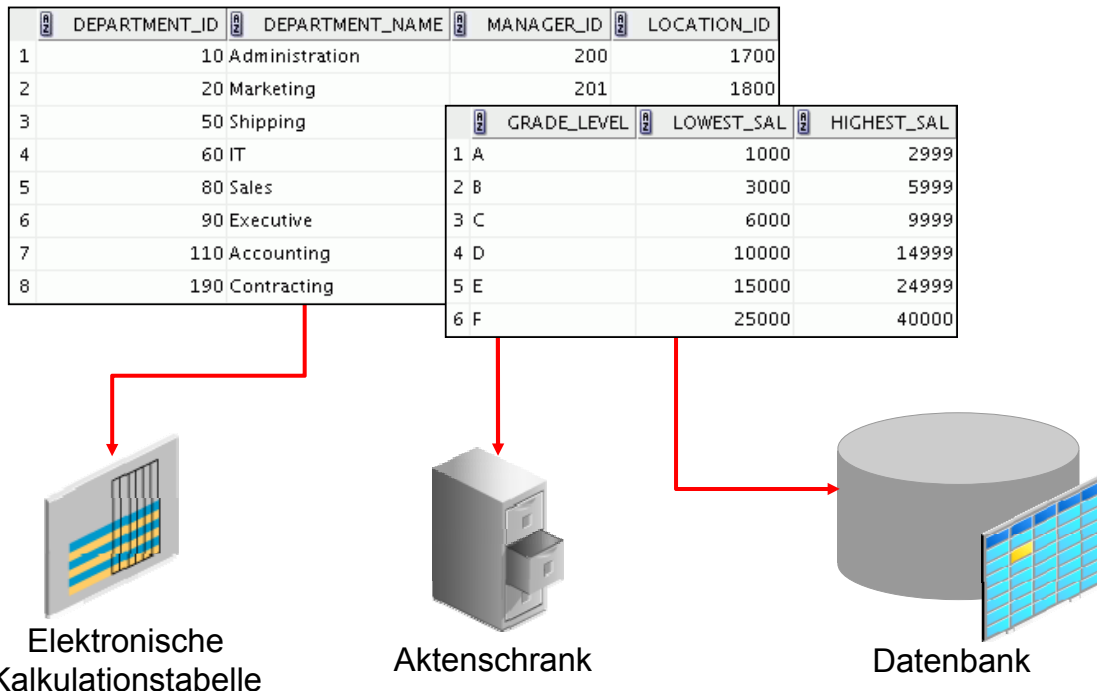
Die Datenmodellierungsfunktionen werden um die Unterstützung des objektrelationalen Datenbankmodells erweitert, zu dessen Vorteilen die objektorientierte Programmierung, die Verwendung komplexer Datentypen und Geschäftsobjekte sowie die vollständige Kompatibilität zur relationalen Welt zählen.

Der Oracle-Server enthält verschiedene Funktionen, um Performance und Funktionalität von OLTP-Anwendungen zu steigern. Hierzu zählen etwa die verbesserte gemeinsame Nutzung von Laufzeitdatenstrukturen, größere Puffercaches und verzögerbare Constraints. Data Warehouse-Anwendungen profitieren von Verbesserungen wie der parallelen Ausführung von INSERT-, UPDATE- und DELETE-Vorgängen, der Partitionierung und der Optimierung paralleler Abfragen. Das Oracle-Modell unterstützt verteilte Client/Server- und webbasierte Anwendungen in Multi-Tier-Umgebungen.

Weitere Informationen über das relationale und objektrelationale Modell finden Sie im Handbuch *Oracle Database Concepts for 12c Database*.



# Datenspeicherung auf verschiedenen Medien



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Jede Organisation hat bestimmte Informationsanforderungen. Bibliotheken verwalten Listen der Mitglieder, Bücher, Rückgabetermine und Überziehungsgebühren. Unternehmen müssen Informationen über Mitarbeiter, Abteilungen und Gehälter speichern. Diese Informationseinheiten werden als *Daten* bezeichnet.

Organisationen können Daten auf verschiedenen Medien und in unterschiedlichen Formaten vorhalten, etwa als gedruckte Dokumente in einem Aktenschrank oder als elektronische Daten in Kalkulationstabellen und Datenbanken.

Eine *Datenbank* ist eine organisierte Zusammenstellung von Informationen.

Um Datenbanken zu verwalten, benötigen Sie ein Datenbankmanagementsystem (DBMS). Ein DBMS ist ein Programm, das Daten in der Datenbank auf Anforderung speichert, abrufen und ändert. Es gibt vier Haupttypen von Datenbanken: *hierarchische Datenbanken*, *Netzwerkdatenbanken*, *relationale Datenbanken* und (seit einigen Jahren) *objektrelationale Datenbanken*.

# Relationale Datenbanken – Konzept

- Dr. E. F. Codd entwickelte das relationale Modell für Datenbanksysteme im Jahr 1970.
- Es bildet die Grundlage für das relationale Datenbankmanagementsystem (RDBMS).
- Das relationale Modell besteht aus folgenden Komponenten:
  - Zusammenstellung von Objekten oder Relationen
  - Gruppe von Operatoren, mit denen die Relationen bearbeitet werden
  - Datenintegrität für Genauigkeit und Konsistenz

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Grundlagen des relationalen Modells wurden im Juni 1970 erstmalig von Dr. E. F. Codd in einer Veröffentlichung mit dem Titel *A Relational Model of Data for Large Shared Data Banks* beschrieben, in der er das relationale Modell für Datenbanksysteme vorstellte.

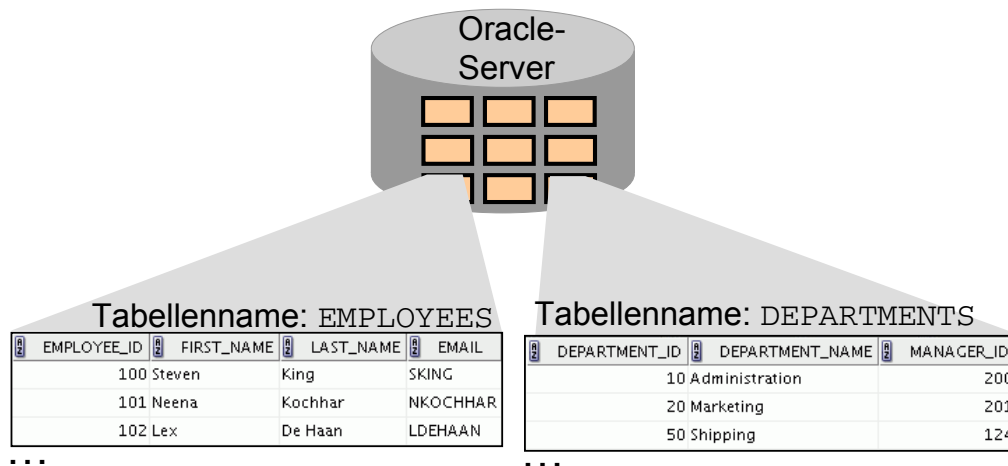
Bis dahin wurden meist hierarchische Modelle und Netzwerkmodelle oder selbst einfache Flat File-Datenstrukturen verwendet. Relationale Datenbankmanagementsysteme (RDBMS) konnten sich vor allem wegen ihrer einfachen Handhabung und ihrer flexiblen Struktur schnell etablieren. Innovative Anbieter wie Oracle ergänzten das RDBMS außerdem durch eine Suite leistungstarker Tools zur Anwendungsentwicklung sowie Benutzeroberflächenprodukte und boten damit eine Komplettlösung an.

## Komponenten des relationalen Modells

- Zusammenstellungen von Objekten oder Relationen zum Speichern der Daten
- Eine Gruppe von Operatoren, mit denen die Relationen bearbeitet werden können, um weitere Relationen zu erstellen
- Datenintegrität für Genauigkeit und Konsistenz

# Relationale Datenbanken – Definition

Eine relationale Datenbank besteht aus einer Sammlung von Relationen oder zweidimensionalen Tabellen, die durch den Oracle-Server gesteuert werden.



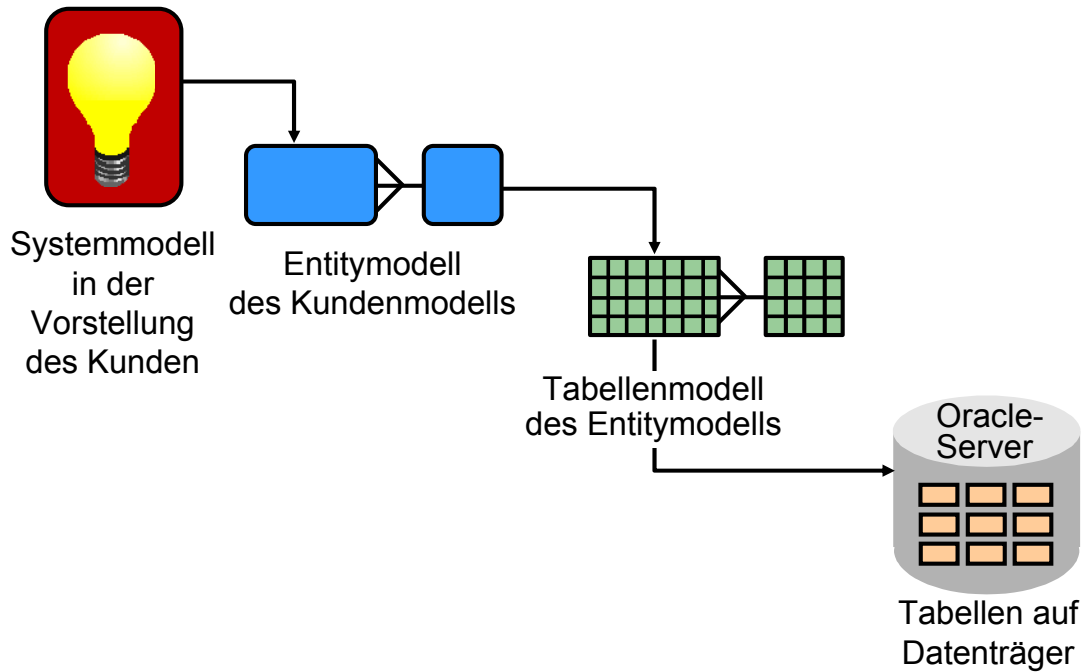
ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Relationale Datenbanken verwenden Relationen oder zweidimensionale Tabellen zum Speichern von Informationen.

Beispiel: Sie möchten Informationen über alle Mitarbeiter in Ihrem Unternehmen speichern. In einer relationalen Datenbank erstellen Sie mehrere Tabellen für die verschiedenen Arten von Mitarbeiterdaten, etwa eine Tabelle für Mitarbeiter (EMPLOYEES), eine Tabelle für Abteilungen (DEPARTMENTS) und eine Tabelle für Gehälter (SALARY).

# Datenmodelle



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Modelle stellen die Grundpfeiler für den Entwurf dar. Ingenieure erstellen ein Modell eines Autos, an dem sie alle Details ausarbeiten, bevor das Auto in Produktion geht. Auf dieselbe Weise entwickeln Systemdesigner Modelle, um Konzepte zu untersuchen und das Verständnis des Datenbankdesigns zu verbessern.

## Zweck von Modellen

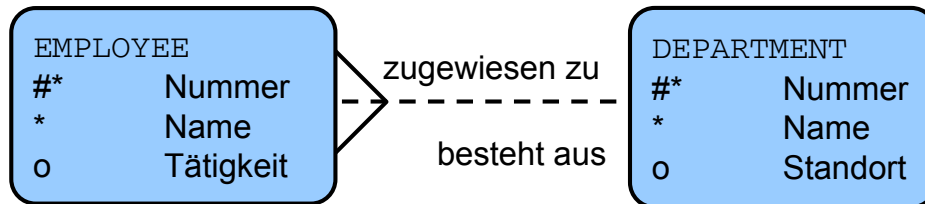
Modelle erleichtern die Vermittlung von Konzepten. Sie dienen folgenden Zwecken:

- Kommunikation
- Einordnung in Kategorien
- Beschreibung
- Spezifikation
- Untersuchung
- Weiterentwicklung
- Analyse
- Nachahmung

Ziel jedes Modells ist es, eine Vielzahl dieser Verwendungszwecke zu erfüllen, für Endbenutzer verständlich zu sein und Entwicklern ausreichend Details an die Hand zu geben, um daraus ein Datenbanksystem zu erstellen.

# Entity-Relationship-Modell

- Entity-Relationship-Diagramm aus Geschäftsspezifikationen oder -beschreibungen erstellen:



- Szenario:
  - "...einer Abteilung einen oder mehrere Mitarbeiter zuweisen..."
  - "...Einigen Abteilungen sind noch keine Mitarbeiter zugewiesen..."

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In einem effektiven System sind die Daten in verschiedene Kategorien oder Entitäts unterteilt. Ein ER-(Entity-Relationship-)Modell ist eine Darstellung der verschiedenen Entitäts in einem Unternehmen mit den zwischen ihnen bestehenden Beziehungen. ER-Modelle werden aus Geschäftsspezifikationen oder -beschreibungen abgeleitet und im Lebenszyklus der Systementwicklung während der Analysephase erstellt. In ER-Modellen sind die für den Geschäftsbetrieb benötigten Informationen von den im Unternehmen ausgeführten Aktivitäten getrennt. Auch bei wechselnden Aktivitäten bleiben die Informationstypen in der Regel unverändert. Daher bleiben normalerweise auch die Datenstrukturen konstant.

## Vorteile der ER-Modellierung

- Dokumentiert Informationen in einem verständlichen, präzisen Format
- Vermittelt ein klares Bild vom Umfang der Informationsanforderung
- Präsentiert das Datenbankdesign in einer leicht verständlichen, bildlichen Darstellung
- Stellt einen effizienten Rahmen für die Integration mehrerer Anwendungen bereit

## Schlüsselkomponenten

- **Entity:** Ein wichtiger Aspekt, über den Informationen bekannt sein müssen. Beispiele: Abteilungen, Mitarbeiter und Aufträge
- **Attribut:** Etwas, das eine Entity beschreibt oder qualifiziert. Die Entity EMPLOYEE kann z. B. über Attribute für Personalnummer, Name, Tätigkeit, Einstellungsdatum, Abteilungsnummer usw. verfügen. Jedes Attribut ist entweder obligatorisch oder optional. Dieser Status wird als *Optionalität* bezeichnet.
- **Beziehung:** Eine benannte Zuordnung zwischen Entitys, welche die Optionalität und den Grad einer Verbindung angibt. Beispiele: Mitarbeiter und Abteilungen oder Aufträge und Artikel

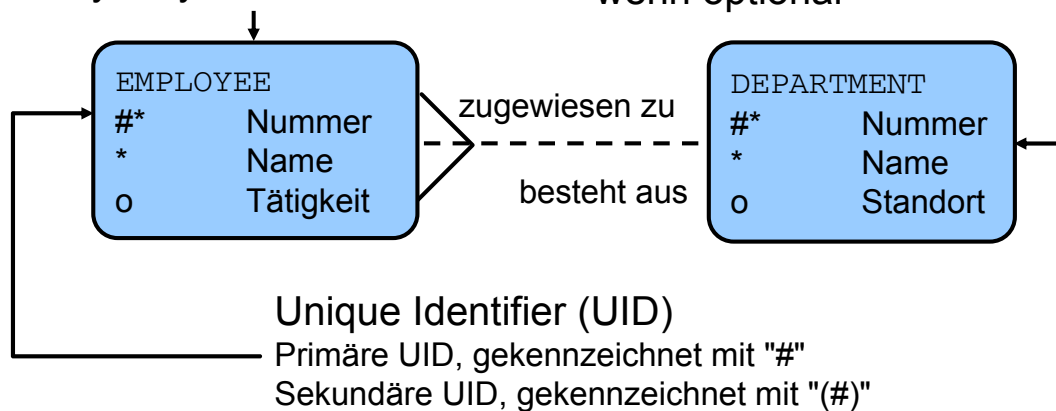
# Entity-Relationship-Modellierung – Konventionen

## Entity:

- Eindeutiger Name im Singular
- Großschreibung
- Abgerundetes Kästchen
- Synonym in Klammern

## Attribut:

- Name im Singular
- Kleinschreibung
- Gekennzeichnet mit "\*", wenn obligatorisch
- Gekennzeichnet mit "o", wenn optional



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Entitys

Für die Darstellung von Entitys in einem Modell gelten folgende Konventionen:

- Eindeutiger Entityname im Singular
- Entityname in Großbuchstaben
- Abgerundetes Kästchen
- Optionale Synonyme in Großbuchstaben in Klammern: ( )

## Attribute

Für die Darstellung von Attributen in einem Modell gelten folgende Konventionen:

- Kleingeschriebener Name im Singular
- Obligatorische Attribute (Werte, die bekannt sein *müssen*) sind mit einem Sternchen (\*) gekennzeichnet.
- Optionale Attribute (Werte, die bekannt sein *können*) sind mit dem Buchstaben "o" gekennzeichnet.

## Beziehungen

Jede Richtung der Beziehung verfügt über:

- **ein Label:** z. B. *unterrichtet von* oder *zugewiesen zu*
- **eine Optionalität:** *muss* oder *kann*
- **einen Grad:** *genau ein/e/r* oder *ein/e/r* oder *mehrere*

Symbol	Beschreibung
Gestrichelte Linie	Optionales Element, Beziehung "kann" existieren
Durchgezogene Linie	Obligatorisches Element, Beziehung "muss" existieren
Krähenfuß	Grad der Beziehung "ein oder mehrere"
Einfache Linie	Grad der Beziehung "genau ein"

**Hinweis:** Der Begriff *Kardinalität* ist ein Synonym für *Grad*.

Jede Quellentity {kann | muss} in Relation mit {genau einer | einer oder mehreren} Zielentity(s) stehen.

**Hinweis:** Der Konvention nach wird im Uhrzeigersinn gelesen.

## Unique Identifier

Ein eindeutiger Bezeichner (Unique Identifier, UID) ist eine beliebige Kombination aus Attributen und/oder Beziehungen und dient dazu, die einzelnen Vorkommen einer Entity zu unterscheiden. Jedes Vorkommen einer Entity muss eindeutig identifizierbar sein.

- Kennzeichnen Sie jedes Attribut, das Teil der UID ist, mit dem Nummernzeichen "#".
- Sekundäre UIDs werden mit einem Nummernzeichen in Klammern gekennzeichnet: "(#)".



## Beziehungen zwischen mehreren Tabellen

- Jede Datenzeile in einer Tabelle kann durch einen Primärschlüssel eindeutig identifiziert werden.
- Daten aus mehreren Tabellen lassen sich über Fremdschlüssel logisch zueinander in Beziehung setzen.

Tabellenname: EMPLOYEES

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPARTMENT_ID
100	Steven	King	90
101	Neena	Kochhar	90
102	Lex	De Haan	90
103	Alexander	Hunold	60
104	Bruce	Ernst	60
107	Diana	Lorentz	60
124	Kevin	Mourgos	50
141	Trenna	Rajs	50
142	Curtis	Davies	50

↑ Primärschlüssel

↑ Fremdschlüssel

Tabellenname: DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting	(null)	1700

↑ Primärschlüssel

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Jede Tabelle enthält Daten, die genau eine Entity beschreiben. Beispiel: Die Tabelle `EMPLOYEES` enthält Informationen über Mitarbeiter. Am oberen Rand jeder Tabelle sind Datenkategorien aufgelistet, unter denen die einzelnen Werte angegeben sind. Mithilfe des Tabellenformats können Sie Informationen schnell darstellen, verstehen und verwenden.

Weil Daten über verschiedene Entitäts in unterschiedlichen Tabellen gespeichert sind, müssen Sie möglicherweise zwei oder mehr Tabellen kombinieren, um bestimmte Fragen zu beantworten. Beispiel: Sie möchten den Standort der Abteilung wissen, in der ein Mitarbeiter arbeitet. In diesem Fall benötigen Sie Informationen aus der Tabelle `EMPLOYEES` (die Daten über die Mitarbeiter enthält) und der Tabelle `DEPARTMENTS` (die Informationen über die Abteilungen enthält). Mit einem RDBMS können Sie die Daten einer Tabelle über Fremdschlüssel zu den Daten einer anderen Tabelle in Beziehung setzen. Ein Fremdschlüssel ist eine Spalte oder eine Gruppe von Spalten, die auf einen Primärschlüssel in derselben oder einer anderen Tabelle verweist.

Durch die Fähigkeit, Beziehungen zwischen Daten in unterschiedlichen Tabellen herzustellen, können Sie Informationen in separaten, verwaltbaren Einheiten organisieren. Personaldaten lassen sich beispielsweise logisch von den Abteilungsdaten trennen, indem Sie diese in separaten Tabellen speichern.

## **Richtlinien für Primärschlüssel und Fremdschlüssel**

- Ein Primärschlüssel kann keine doppelten Werte enthalten.
- Primärschlüssel können generell nicht geändert werden.
- Fremdschlüssel basieren auf Datenwerten und sind rein logische (nicht physische) Zeiger.
- Ein Fremdschlüsselwert muss einem vorhandenen Primärschlüsselwert oder einem eindeutigen Schlüsselwert entsprechen bzw. Null sein.
- Ein Fremdschlüssel muss auf eine Primärschlüsselspalte oder eine Spalte mit einem eindeutigen Schlüssel verweisen.

# Relationale Datenbanken – Terminologie

2	3	4	5	6		
1	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	COMMISSION_PCT	DEPARTMENT_ID
	100	Steven	King	24000	(null)	90
	101	Neena	Kochhar	17000	(null)	90
	102	Lex	De Haan	17000	(null)	90
	103	Alexander	Hunold	9000	(null)	60
	104	Bruce	Ernst	6000	(null)	60
	107	Diana	Lorentz	4200	(null)	60
	124	Kevin	Mourgos	5800	(null)	50
	141	Trenna	Rajs	3500	(null)	50
	142	Curtis	Davies	3100	(null)	50
	143	Randall	Matos	2600	(null)	50
	144	Peter	Vargas	2500	(null)	50
	149	Eleni	Zlotkey	10500	0.2	80
	174	Ellen	Abel	11000	0.3	80
	176	Jonathan	Taylor	8600	0.2	80
	178	Kimberely	Grant	7000	0.15	(null)
	200	Jennifer	Whalen	4400	(null)	10
	201	Michael	Hartstein	13000	(null)	20
	202	Pat	Fay	6000	(null)	20
	205	Shelley	Higgins	12000	(null)	110
	206	William	Gietz	8300	(null)	110

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Eine relationale Datenbank kann eine oder mehrere Tabellen enthalten. Die grundlegende Speicherstruktur eines RDBMS ist die *Tabelle*. Tabellen enthalten alle nötigen Daten über Personen oder Objekte aus der realen Welt, etwa Mitarbeiter, Rechnungen oder Kunden.

Die Folie zeigt den Inhalt der *Tabelle* bzw. *Relation* `EMPLOYEES`. Die Zahlen geben Folgendes an:

1. Eine einzelne *Zeile* (oder ein *Tupel*) mit allen für einen bestimmten Mitarbeiter benötigten Daten. Jede Zeile in einer Tabelle sollte durch einen Primärschlüssel identifiziert sein, sodass doppelten Zeilen ausgeschlossen sind. Die Reihenfolge der Zeilen hat keine Bedeutung. Sie können die Zeilenreihenfolge beim Datenabruf festlegen.
2. Eine *Spalte* oder ein Attribut mit der Personalnummer. Die Personalnummer kennzeichnet einen Mitarbeiter in der Tabelle `EMPLOYEES` in *eindeutiger* Weise. In diesem Beispiel ist die Spalte `EMPLOYEE_ID` der *Primärschlüssel*. Ein Primärschlüssel muss einen Wert enthalten, und dieser Wert muss eindeutig sein.
3. Eine Spalte, die keinen Schlüsselwert enthält. Eine Spalte stellt eine bestimmte Art von Daten in einer Tabelle dar. Im vorliegenden Beispiel gibt die Spalte die Gehälter der einzelnen Mitarbeiter an. Die Reihenfolge der Spalten ist beim Speichern der Daten nicht von Bedeutung. Sie können die Spaltenreihenfolge beim Datenabruf festlegen.

4. Eine Spalte, in der die Abteilungsnummer gespeichert wird und die auch ein *Fremdschlüssel* ist. Ein Fremdschlüssel ist eine Spalte, die definiert, wie verschiedene Tabellen zueinander in Beziehung stehen. Ein Fremdschlüssel verweist auf einen Primärschlüssel oder einen eindeutigen Schlüssel in derselben oder einer anderen Tabelle. Im Beispiel oben ermöglicht `DEPARTMENT_ID` die eindeutige Identifizierung einer Abteilung aus der Tabelle `DEPARTMENTS`.
5. Ein *Feld* bildet den Schnittpunkt zwischen einer Zeile und einer Spalte. Jedes Feld kann nur einen Wert enthalten.
6. Ein Feld kann leer sein (Nullwert). In der Tabelle `EMPLOYEES` enthält das Feld `COMMISSION_PCT` (Provision) nur für die Mitarbeiter einen Wert, denen die Rolle Sales Representative zugeordnet ist.

# Lektionsagenda

- Kursziele, Agenda und Anhänge dieses Kurses
- Oracle Database 12c und zugehörige Produkte – Überblick
- Relationales Datenbankmanagement – Überblick zu Konzepten und Begriffen
- **SQL und zugehörige Entwicklungsumgebungen – Einführung**
- In diesem Kurs verwendetes Schema Human Resources (HR) und verwendete Tabellen
- SQL-Dokumentation und zusätzliche Ressourcen zu Oracle Database 12c

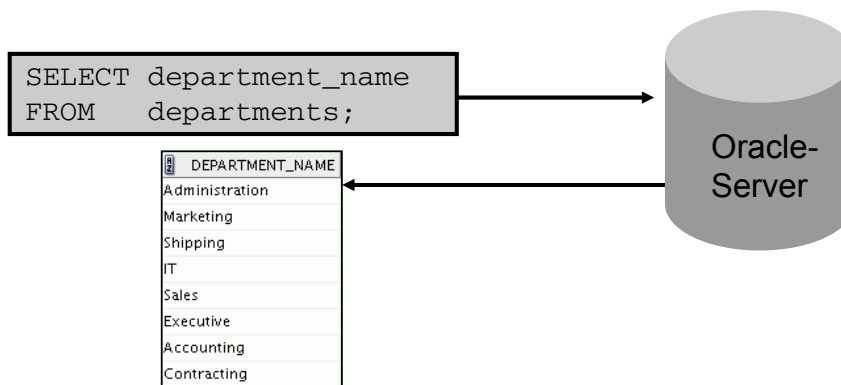
ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

# Datenbanken mit SQL abfragen

SQL (Structured Query Language) ist:

- die ANSI-Standardsprache für den Betrieb relationaler Datenbanken
- effizient, einfach zu lernen und zu verwenden
- funktional vollständig (Mit SQL lassen sich Daten in Tabellen definieren, abrufen und bearbeiten.)



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In einer relationalen Datenbank geben Sie keinen Zugriffspfad auf die Tabellen an, und Sie brauchen nicht zu wissen, wie die Daten physisch angeordnet sind.

Um auf die Datenbank zuzugreifen, führen Sie eine SQL-Anweisung aus. SQL (Structured Query Language) ist die ANSI-(American National Standards Institute-)Standardsprache für den Betrieb relationaler Datenbanken und entspricht auch dem ISO-Standard (SQL 1999).

SQL stellt Anweisungen bereit, die allen Programmen und Benutzern den Zugriff auf die Daten in einer Oracle-Datenbank ermöglichen. Mit Anwendungsprogrammen und Oracle-Tools können Benutzer häufig auf die Datenbank zugreifen, ohne SQL direkt verwenden zu müssen. Diese Anwendungen müssen jedoch bei der Ausführung der Benutzeranforderung SQL verwenden.

SQL bietet Anweisungen für vielfältige Aufgaben, darunter:

- Daten abfragen
- Zeilen in Tabellen einfügen, aktualisieren und löschen
- Objekte erstellen, ersetzen, ändern und löschen
- Zugriff auf Datenbanken und ihre Objekte steuern
- Konsistenz und Integrität der Datenbank gewährleisten

SQL vereint alle vorstehend genannten Aufgaben in einer konsistenten Sprache und ermöglicht Ihnen, auf logischer Ebene mit den Daten zu arbeiten.

# Im Kurs verwendete SQL-Anweisungen

SELECT INSERT UPDATE DELETE MERGE	Data Manipulation Language (DML)
CREATE ALTER DROP RENAME TRUNCATE COMMENT	Data Definition Language (DDL)
GRANT REVOKE	Data Control Language (DCL)
COMMIT ROLLBACK SAVEPOINT	Transaktionskontrolle

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## SQL-Anweisungen

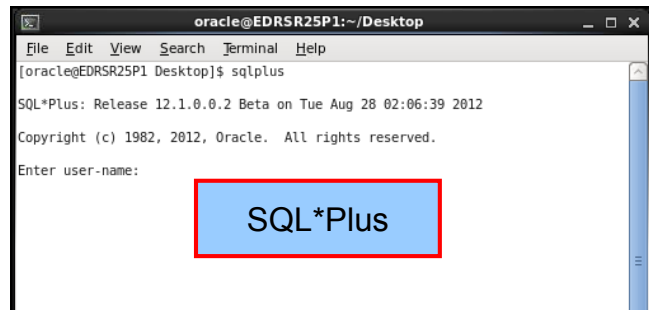
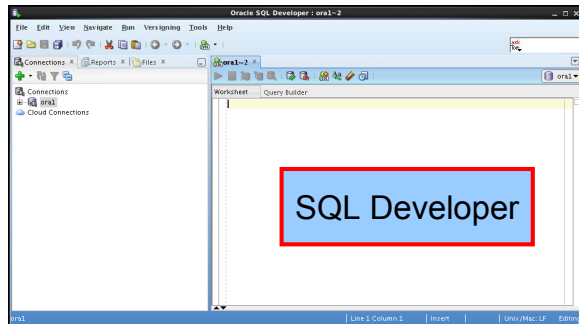
Die von Oracle unterstützten SQL-Anweisungen entsprechen den Industriestandards. Oracle garantiert die Erfüllung aktueller und künftiger Standards, indem führende Mitarbeiter aus dem Unternehmen aktiv in den SQL-Standardisierungsausschüssen der branchenweit anerkannten Organisationen ANSI und ISO (International Standards Organization) mitwirken. Sowohl ANSI als auch ISO haben SQL als Standardsprache für relationale Datenbanken anerkannt.

Anweisung	Beschreibung
SELECT INSERT UPDATE DELETE MERGE	Daten aus der Datenbank abrufen, neue Zeilen eingeben, vorhandene Zeilen ändern oder nicht benötigte Zeilen aus Tabellen in der Datenbank löschen. Diese Anweisungen werden unter dem Begriff <i>Data Manipulation Language</i> (DML) zusammengefasst.
CREATE ALTER DROP RENAME TRUNCATE COMMENT	Datenstrukturen in Tabellen einrichten, ändern oder aus Tabellen entfernen. Diese Anweisungen werden unter dem Begriff <i>Data Definition Language</i> (DDL) zusammengefasst.
GRANT REVOKE	Zugriffsrechte für die Oracle-Datenbank und die darin enthaltenen Strukturen erteilen oder widerrufen
COMMIT ROLLBACK SAVEPOINT	Mit DML-Anweisungen durchgeführte Änderungen verwalten. Änderungen an den Daten können zu logischen Transaktionen gruppiert werden.

# SQL-Entwicklungsumgebungen

In diesem Kurs sind zwei Entwicklungsumgebungen verfügbar:

- Das bevorzugte Tool ist Oracle SQL Developer.
- Daneben kann auch die SQL\*Plus-Befehlszeilenschnittstelle verwendet werden.



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## SQL Developer

Bei der Entwicklung dieses Kurses wurde als Tool zur Ausführung der in den Lektions- und Übungsbeispielen erörterten SQL-Anweisungen Oracle SQL Developer verwendet. SQL Developer stellt das Standardtool für diesen Kurs dar.

## SQL\*Plus

Alle in diesem Kurs behandelten SQL-Befehle können auch in der SQL\*Plus-Umgebung ausgeführt werden.

## Hinweise

- Informationen zur Verwendung von SQL Developer sowie eine Kurzanleitung zum Installationsprozess finden Sie in Anhang B.
- Informationen zur Verwendung von SQL\*Plus finden Sie in Anhang C.



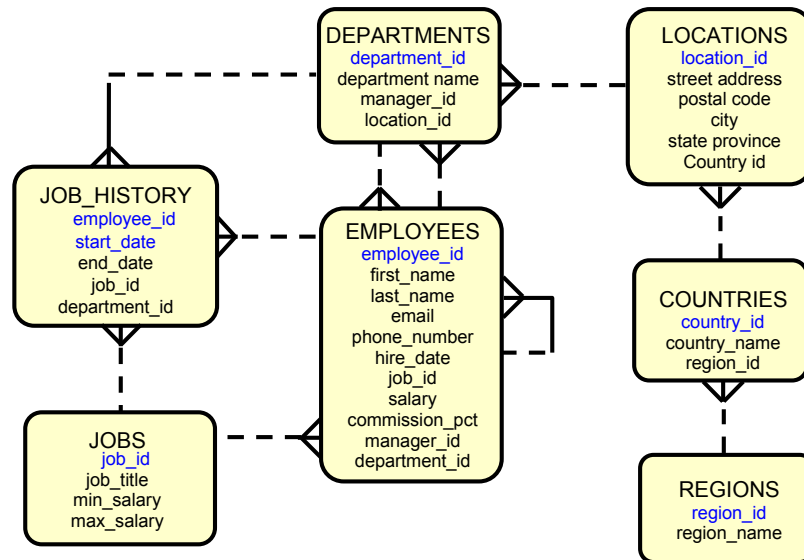
# Lektionsagenda

- Kursziele, Agenda und Anhänge dieses Kurses
- Oracle Database 12c und zugehörige Produkte – Überblick
- Relationales Datenbankmanagement – Überblick zu Konzepten und Begriffen
- SQL und zugehörige Entwicklungsumgebungen – Einführung
- **In diesem Kurs verwendetes Schema Human Resources (HR) und verwendete Tabellen**
- SQL-Dokumentation und zusätzliche Ressourcen zu Oracle Database 12c

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Schema Human Resources (HR)



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

### Schema Human Resources (HR) – Beschreibung

Das Schema Human Resources (HR) gehört zu den von Oracle bereitgestellten Beispielschemas, die in einer Oracle-Datenbank installiert werden können. In den Übungen dieses Kurses werden Daten aus dem Schema HR verwendet.

#### Tabellenbeschreibungen

- **REGIONS** enthält Zeilen, die für eine Region wie Nordamerika, Asien usw. stehen.
- **COUNTRIES** enthält Zeilen für Länder, die jeweils einer Region zugeordnet sind.
- **LOCATIONS** enthält die spezifische Adresse eines bestimmten Büros, Lagers oder Produktionsstandortes eines in einem bestimmten Land ansässigen Unternehmens.
- **DEPARTMENTS** zeigt Details zu den Abteilungen, in denen Mitarbeiter arbeiten. Die Abteilungen können jeweils eine Beziehung zum Abteilungsleiter in der Tabelle **EMPLOYEES** aufweisen.
- **EMPLOYEES** enthält Details zu den einzelnen Mitarbeitern in einer Abteilung. Nicht alle Mitarbeiter müssen einer Abteilung zugeordnet sein.
- **JOBS** enthält die Typen von Tätigkeiten, die einzelne Mitarbeiter ausüben können.
- **JOB\_HISTORY** enthält die Tätigkeitshistorie der Mitarbeiter. Wenn ein Mitarbeiter unter Beibehaltung derselben Tätigkeit die Abteilung wechselt oder innerhalb einer Abteilung eine andere Tätigkeit übernimmt, wird eine neue Zeile mit Informationen zur bisherigen Tätigkeit des Mitarbeiters in diese Tabelle eingefügt.

# Im Kurs verwendete Tabellen

EMPLOYEES

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY
1	100	Steven	King	SKING	515.123.4567	17-JUN-03	AD_PRES	24000
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-05	AD_VP	17000
3	102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-01	AD_VP	17000
4	103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-06	AC_MGR	12008
5	104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-07	IT_PROG	6000
6	107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-07	IT_PROG	4200
7	124	Kevin	Mourgos	KMOURGOS	650.123.5234	16-NOV-07	ST_MAN	5800
8	141	Trenna	Rajs	TRAJS	650.121.8009	17-OCT-03	ST_CLERK	3500
9	142	Curtis	Davies	CDAVIES	650.121.2994	29-JAN-05	ST_CLERK	3100
10	143	Randall	Matos	RMATOS	650.121.2874	15-MAR-06	ST_CLERK	2600
11	144	Peter	Vargas	PVARGAS	650.121.2004	09-JUL-06	ST_CLERK	2500
12	149	Eleni	Zlotkey	EZLOTKEY	011.44.1344.429018	29-JAN-08	SA_MAN	10500
13	174	Ellen	Abel	EABEL	011.44.1644.429267	11-MAY-04	SA_REP	11000
14	176	Jonathon	Taylor	JTAYLOR	011.44.1644.429265	24-MAR-06	SA_REP	8600
15	178	Kimberely	Grant	KGRANT	011.44.1644.429263	24-MAY-07	SA_REP	7000
16	200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-03	AD_ASST	4400
17	201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-04	MK_MAN	13000
18	202	Pat	Fay	PFAY	603.123.6666	17-AUG-05	MK_REP	6000
19	205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-02	AC_MGR	12008
20	206	William	Gietz	WGIEZT	515.123.8181	07-JUN-02	AC_ACCOUNT	8300

	GRADE_LEVEL	LOWEST_SAL	HIGHEST_SAL
1	A	1000	2999
2	B	3000	5999
3	C	6000	9999
4	D	10000	14999
5	E	15000	24999
6	F	25000	40000

JOB\_GRADES

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	50	Shipping	124	1500
4	60	IT	103	1400
5	80	Sales	149	2500
6	90	Executive	100	1700
7	110	Accounting	205	1700
8	190	Contracting	(null)	1700

DEPARTMENTS

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In diesem Kurs werden hauptsächlich folgende Tabellen verwendet:

- EMPLOYEES: Enthält Details zu allen Mitarbeitern
- DEPARTMENTS: Enthält Details zu allen Abteilungen
- JOB\_GRADES: Enthält Details zu den Gehältern für verschiedene Gehaltsstufen

Neben diesen Tabellen verwenden Sie auch andere Tabellen, die auf der vorhergehenden Folie aufgeführt sind, etwa die Tabellen LOCATIONS und JOB\_HISTORY.

**Hinweis:** Die Struktur und Daten aller Tabellen finden Sie in Anhang A.

# Lektionsagenda

- Kursziele, Agenda und Anhänge dieses Kurses
- Oracle Database 12c und zugehörige Produkte – Überblick
- Relationales Datenbankmanagement – Überblick zu Konzepten und Begriffen
- SQL und zugehörige Entwicklungsumgebungen – Einführung
- In diesem Kurs verwendetes Schema Human Resources (HR) und verwendete Tabellen
- SQL-Dokumentation und zusätzliche Ressourcen zu Oracle Database 12c

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Oracle Database – Dokumentation

- *Oracle Database New Features Guide*
- *Oracle Database Reference*
- *Oracle Database SQL Language Reference*
- *Oracle Database Concepts*
- *Oracle Database SQL Developer User's Guide*

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Dokumentationsbibliothek zu Oracle Database 12c finden Sie unter <http://st-doc.us.oracle.com/12/121/index.htm>.

## Zusätzliche Ressourcen

Weitere Informationen zu Oracle Database 12c:

- *Oracle Database 12c: New Features eStudies*
- *Oracle Learning Library:*
  - <http://www.oracle.com/goto/oll>
- *Oracle Cloud:*
  - <http://cloud.oracle.com>
- Onlinehomepage von SQL Developer:
  - [http://www.oracle.com/technology/products/database/sql\\_developer/index.html](http://www.oracle.com/technology/products/database/sql_developer/index.html)
- Tutorial zu SQL Developer:
  - <http://download.oracle.com/oll/tutorials/SQLDeveloper/index.htm>

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

# Zusammenfassung

In dieser Lektion haben Sie Folgendes gelernt:

- Ziele des Kurses definieren
- Features von Oracle Database 12c auflisten
- Wichtigste Features von Oracle Cloud beschreiben
- Theoretische und physische Aspekte relationaler Datenbanken diskutieren
- Implementierung von relationalen (RDBMS) und objektrelationalen Datenbankmanagementsystemen (ORDBMS) auf dem Oracle-Server beschreiben
- Für diesen Kurs verwendbare Entwicklungsumgebungen bestimmen
- In diesem Kurs verwendete Datenbank und zugehöriges Schema beschreiben

**ORACLE**

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Relationale Datenbankmanagementsysteme bestehen aus Objekten oder Relationen. Sie werden über Vorgänge verwaltet und unterliegen Datenintegritäts-Constraints.

Oracle entwickelt Produkte und Services, die Ihre Anforderungen an ein RDBMS erfüllen. Zu den wichtigsten Produkten zählen:

- Oracle Database, um Informationen mithilfe von SQL zu speichern und zu verwalten
- Oracle Fusion Middleware, um modulare integrierbare und wiederverwendbare Businessservices zu entwickeln, bereitzustellen und zu verwalten
- Oracle Enterprise Manager Grid Control, um Administrationsaufgaben in einer Grid-Umgebung über mehrere Systeme hinweg zu verwalten und zu automatisieren

## SQL

Der Oracle-Server unterstützt ANSI-Standard-SQL und enthält Erweiterungen. SQL ist die Sprache, in der Sie mit dem Server kommunizieren, um auf Daten zuzugreifen sowie Daten zu bearbeiten und zu steuern.

# Übungen zu Lektion 1 – Überblick

Diese Übungen behandeln folgende Themen:

- Oracle SQL Developer starten
- Neue Datenbankverbindung erstellen
- HR-Tabellen durchsuchen

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In dieser Übung führen Sie folgende Aufgaben aus:

- Oracle SQL Developer starten und eine neue Datenbankverbindung für den Account `ora1` erstellen
- Mit Oracle SQL Developer Datenobjekte im Account `ora1` untersuchen. Der Account `ora1` enthält die Tabellen des Schemas `HR`.

Die Übungsdateien befinden sich im Verzeichnis

`/home/oracle/labs/sql1/labs`

Wenn Sie zum Speichern von Übungsdateien aufgefordert werden, speichern Sie die Dateien unter diesem Pfad.

Die Übungen können weitere Aufgaben enthalten, die mit der Wendung "Falls Sie noch Zeit haben" oder "Wenn Sie eine weitere Herausforderung suchen" beginnen. Bearbeiten Sie diese Aufgaben nur, wenn Sie alle anderen Übungen in der verfügbaren Zeit abgeschlossen haben und Ihre Kenntnisse weiter testen möchten.

Bearbeiten Sie die Übungen langsam und präzise. Sie können mit dem Speichern und Ausführen von Befehlsdateien experimentieren. Wenn Sie Fragen haben, wenden Sie sich an den Dozenten.

**Hinweis:** Für alle schriftlichen Übungen wird Oracle SQL Developer als Entwicklungsumgebung verwendet. Die Verwendung von Oracle SQL Developer wird empfohlen, Sie können in diesem Kurs aber auch SQL\*Plus verwenden.



# 2

## Daten mit der SQL-Anweisung `SELECT` abrufen

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- Funktionsmöglichkeiten von `SELECT`-Anweisungen auflisten
- Einfache `SELECT`-Anweisungen ausführen

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Um Daten aus der Datenbank zu extrahieren, verwenden Sie die SQL-Anweisung `SELECT`. In bestimmten Fällen müssen Sie einschränken, welche Spalten angezeigt werden. In dieser Lektion wird die Anweisung `SELECT` beschrieben, die Sie zur Ausführung dieser Aktionen benötigen. Darüber hinaus erfahren Sie, wie Sie `SELECT`-Anweisungen erstellen, die Sie mehrfach wiederverwenden können.

# Lektionsagenda

- SQL-Anweisungen vom Typ `SELECT` – Funktionen
- Arithmetische Ausdrücke und Nullwerte in `SELECT`-Anweisungen
- Spaltenaliasnamen
- Verkettungsoperator, literale Zeichenfolgen, alternativer Operator für Anführungszeichen und Schlüsselwort `DISTINCT`
- Befehl `DESCRIBE`

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

# Einfache SELECT-Anweisungen

```
SELECT  * | {[DISTINCT] column [alias], ...}
FROM    table;
```

- SELECT gibt die anzuzeigenden Spalten an.
- FROM gibt die Tabelle an, die diese Spalten enthält.

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Eine SELECT-Anweisung muss in ihrer einfachsten Form folgende Elemente enthalten:

- Eine SELECT-Klausel, die bestimmt, welche Spalten angezeigt werden.
- Eine FROM-Klausel zur Angabe der Tabelle mit den in der Klausel SELECT aufgeführten Spalten

Für die Syntax gilt:

SELECT	Ist eine Liste aus einzelnen oder mehreren Spalten
*	Wählt alle Spalten
DISTINCT	Unterdrückt doppelte Werte
column expression	Wählt die angegebene Spalte oder den Ausdruck
alias	Gibt den gewählten Spalten eine andere Überschrift
FROM table	Gibt an, welche Tabelle die Spalten enthält

**Hinweis:** Im gesamten Kurs werden die Begriffe *Schlüsselwort*, *Klausel* und *Anweisung* wie folgt verwendet:

- Ein *Schlüsselwort* bezieht sich auf ein einzelnes SQL-Element. Beispiele für Schlüsselwörter: SELECT und FROM
- Eine *Klausel* ist ein Teil einer SQL-Anweisung. Beispiel: SELECT employee\_id, last\_name
- Eine *Anweisung* ist eine Kombination aus zwei oder mehr Klauseln. Beispiel: SELECT \* FROM employees

## Alle Spalten wählen

```
SELECT *  
FROM departments;
```

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	50	Shipping	124	1500
4	60	IT	103	1400
5	80	Sales	149	2500
6	90	Executive	100	1700
7	110	Accounting	205	1700
8	190	Contracting	(null)	1700

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie können alle Datenspalten einer Tabelle anzeigen, indem Sie nach dem Schlüsselwort `SELECT` ein Sternchen (\*) angeben. Im Beispiel auf der Folie enthält die Tabelle `DEPARTMENTS` vier Spalten: `DEPARTMENT_ID`, `DEPARTMENT_NAME`, `MANAGER_ID` und `LOCATION_ID`. Die Tabelle enthält acht Zeilen – eine pro Abteilung.

Alternativ können Sie nach dem Schlüsselwort `SELECT` die Namen aller Tabellenspalten angeben. Mit der folgenden SQL-Anweisung zeigen Sie (wie im Beispiel auf der Folie) alle Spalten und alle Zeilen der Tabelle `DEPARTMENTS` an:

```
SELECT department_id, department_name, manager_id, location_id  
FROM departments;
```

**Hinweis:** In SQL Developer können Sie die SQL-Anweisung in ein SQL Worksheet eingeben und auf das Symbol **Execute Statement** klicken oder F9 drücken, um die Anweisung auszuführen. Die Ausgabe wird in der Registerkarte **Results** angezeigt, wie auf der Folie dargestellt.

## Bestimmte Spalten wählen

```
SELECT department_id, location_id  
FROM departments;
```

	DEPARTMENT_ID	LOCATION_ID
1	10	1700
2	20	1800
3	50	1500
4	60	1400
5	80	2500
6	90	1700
7	110	1700
8	190	1700

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mithilfe der Anweisung `SELECT` können Sie die Anzeige auch auf bestimmte Spalten der Tabelle beschränken. Dazu geben Sie die entsprechenden Spaltennamen durch Kommas getrennt an. Das Beispiel auf der Folie zeigt alle Abteilungs- und Standortnummern aus der Tabelle `DEPARTMENTS` an.

Geben Sie in der Klausel `SELECT` die gewünschten Spalten in der Reihenfolge an, in der sie in der Ausgabe angezeigt werden sollen. Beispiel: Um zuerst die Standortnummer und rechts daneben die Abteilungsnummer anzuzeigen, geben Sie folgende Anweisung ein:

```
SELECT location_id, department_id  
FROM departments
```

## SQL-Anweisungen erstellen

- In SQL-Anweisungen wird nicht zwischen Groß- und Kleinschreibung unterschieden.
- SQL-Anweisungen können in einer Zeile oder in mehreren Zeilen eingegeben werden.
- Schlüsselwörter dürfen nicht abgekürzt oder auf zwei Zeilen verteilt werden.
- Jede Klausel wird normalerweise in eine eigene Zeile geschrieben.
- Einrückungen verbessern die Lesbarkeit.
- In SQL Developer können SQL-Anweisungen optional mit einem Semikolon (;) beendet werden. Ein Semikolon ist erforderlich, wenn Sie mehrere SQL-Anweisungen ausführen.
- In SQL\*Plus müssen Sie jede SQL-Anweisung mit einem Semikolon (;) beenden.

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

### SQL-Anweisungen erstellen

Mit folgenden einfachen Regeln und Richtlinien erstellen Sie gültige Anweisungen, die leicht zu lesen und zu bearbeiten sind:

- Sofern nicht ausdrücklich angegeben, wird in SQL-Anweisungen nicht zwischen Groß- und Kleinschreibung unterschieden.
- SQL-Anweisungen können in einer Zeile oder in mehreren Zeilen eingegeben werden.
- Schlüsselwörter dürfen nicht auf zwei Zeilen verteilt oder abgekürzt werden.
- Klauseln werden aus Gründen der besseren Lesbarkeit und einfacheren Bearbeitung normalerweise jeweils in eine eigene Zeile geschrieben.
- Einrückungen machen den Code leserfreundlicher.
- Schlüsselwörter werden üblicherweise in Großbuchstaben geschrieben. Alle anderen Wörter (etwa Tabellen- und Spaltennamen) werden in Kleinbuchstaben angegeben.

## SQL-Anweisungen ausführen

Um die jeweiligen Befehle im SQL Worksheet auszuführen, klicken Sie in SQL Developer auf das Symbol **Run Script** oder drücken F5. Sie können auch auf das Symbol **Execute Statement** klicken oder F9 drücken, um eine SQL-Anweisung im SQL Worksheet auszuführen. Über das Symbol **Execute Statement** führen Sie diejenige Anweisung im Bereich **Enter SQL Statement** aus, auf der sich der Cursor befindet. Mit dem Symbol **Run Script** werden dagegen alle Anweisungen im Bereich **Enter SQL Statement** ausgeführt. Über das Symbol **Execute Statement** zeigen Sie die Ausgabe der Abfrage in der Registerkarte **Results** an, während mit dem Symbol **Run Script** die SQL\*Plus-Ausgabe emuliert und in der Registerkarte **Script Output** angezeigt wird.

In SQL\*Plus beenden Sie die SQL-Anweisung mit einem Semikolon. Um den Befehl auszuführen, drücken Sie anschließend die EINGABETASTE.



## Standardwerte für Spaltenüberschriften

- SQL Developer:
  - Standardausrichtung für Überschriften: Linksbündig
  - Standardanzeige von Überschriften: Großschreibung
- SQL\*Plus:
  - Ausrichtung der Überschriften von Zeichen- und Datumsspalten: Linksbündig
  - Ausrichtung der Überschriften von numerischen Spalten: Rechtsbündig
  - Standardanzeige von Überschriften: Großschreibung

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In SQL Developer werden Spaltenüberschriften in Großbuchstaben und linksbündig angezeigt.

```
SELECT last_name, hire_date, salary
FROM   employees;
```

Sie können die angezeigte Spaltenüberschrift durch einen Alias ersetzen. Spaltenaliasnamen werden im weiteren Verlauf dieser Lektion behandelt.

# Lektionsagenda

- SQL-Anweisungen vom Typ `SELECT` – Funktionen
- **Arithmetische Ausdrücke und Nullwerte in `SELECT`-Anweisungen**
- Spaltenaliasnamen
- Verkettungsoperator, literale Zeichenfolgen, alternativer Operator für Anführungszeichen und Schlüsselwort `DISTINCT`
- Befehl `DESCRIBE`

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

# Arithmetische Ausdrücke

Mit arithmetischen Operatoren Ausdrücke erstellen, die Daten vom Typ NUMBER und DATE enthalten

Operator	Beschreibung
+	Addieren
-	Subtrahieren
*	Multiplizieren
/	Dividieren

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Eventuell möchten Sie die Art der Datenanzeige ändern, Berechnungen durchführen oder mögliche Szenarios durchspielen. All dies ist mit arithmetischen Ausdrücken möglich. Ein arithmetischer Ausdruck kann Spaltennamen, konstante numerische Werte und die arithmetischen Operatoren enthalten.

## Arithmetische Operatoren

Auf der Folie sind die in SQL verfügbaren arithmetischen Operatoren aufgelistet. Sie können die arithmetischen Operatoren in jeder Klausel einer SQL-Anweisung mit Ausnahme der Klausel `FROM` verwenden.

**Hinweis:** In Verbindung mit den Datentypen `DATE` und `TIMESTAMP` können Sie nur die Operatoren für Addition und Subtraktion verwenden.

# Arithmetische Operatoren

```
SELECT last_name, salary, salary + 300  
FROM employees;
```

	LAST_NAME	SALARY	SALARY+300
1	King	24000	24300
2	Kochhar	17000	17300
3	De Haan	17000	17300
4	Hunold	9000	9300
5	Ernst	6000	6300
6	Lorentz	4200	4500
7	Mourgos	5800	6100
8	Rajs	3500	3800
9	Davies	3100	3400
10	Matos	2600	2900

...

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf der Folie wird der Additionsoperator verwendet, um für alle Mitarbeiter eine Gehaltserhöhung von \$ 300 zu berechnen. Die Ausgabe enthält die Spalte `SALARY+300`.

Beachten Sie, dass es sich bei der berechneten Spalte `SALARY+300` nicht um eine neue Spalte in der Tabelle `EMPLOYEES` handelt. Sie dient lediglich der Anzeige. Standardmäßig wird der Name einer neuen Spalte aus der Berechnung abgeleitet, mit der die Spalte generiert wurde – in diesem Fall `salary+300`.

**Hinweis:** Der Oracle-Server ignoriert Leerzeichen vor und nach dem arithmetischen Operator.

## Prioritätsregeln

- Multiplikation und Division haben Vorrang vor Addition und Subtraktion.
- Operatoren derselben Priorität werden von links nach rechts ausgewertet.
- Klammern werden verwendet, um die standardmäßige Priorität außer Kraft zu setzen oder die Anweisung zu verdeutlichen.

# Operatorpriorität

```
SELECT last_name, salary, 12*salary+100
FROM employees;
```

1

	LAST_NAME	SALARY	12*SALARY+100
1	King	24000	288100
2	Kochhar	17000	204100
3	De Haan	17000	204100
4	Hunold	9000	108100

...

```
SELECT last_name, salary, 12*(salary+100)
FROM employees;
```

2

	LAST_NAME	SALARY	12*(SALARY+100)
1	King	24000	289200
2	Kochhar	17000	205200
3	De Haan	17000	205200
4	Hunold	9000	109200

...

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Das erste Beispiel auf der Folie zeigt Nachname, Monatsgehalt und jährliche Vergütung der Mitarbeiter. Zur Berechnung der jährlichen Vergütung wird das Monatsgehalt mit 12 multipliziert und dann ein einmaliger Bonus von \$ 100 addiert. Die Multiplikation wird dabei vor der Addition ausgeführt.

**Hinweis:** Durch die Verwendung von Klammern können Sie die standardmäßige Berechnungsreihenfolge verdeutlichen oder die Lesbarkeit verbessern. Der Ausdruck auf der Folie kann zum Beispiel auch folgendermaßen formuliert werden:  $(12 * \text{salary}) + 100$ . Das Ergebnis bleibt dasselbe.

## Klammersetzung

Sie können die Prioritätsregeln umgehen, indem Sie mithilfe von Klammern angeben, in welcher Reihenfolge die Operatoren ausgeführt werden sollen.

Das zweite Beispiel auf der Folie zeigt Nachname, Monatsgehalt und jährliche Vergütung der Mitarbeiter. Die jährliche Vergütung wird wie folgt berechnet: Zum Monatsgehalt wird ein monatlicher Bonus von \$ 100 addiert. Das Ergebnis wird anschließend mit 12 multipliziert. Aufgrund der Klammern hat die Addition Vorrang vor der Multiplikation.

## Nullwerte definieren

- Ein Nullwert steht für einen nicht verfügbaren, nicht zugewiesenen, unbekannten oder nicht anwendbaren Wert.
- Ein Nullwert ist nicht dasselbe wie die Zahl Null oder ein Leerzeichen.

```
SELECT last_name, job_id, salary, commission_pct  
FROM employees;
```

	LAST_NAME	JOB_ID	SALARY	COMMISSION_PCT
1	King	AD_PRES	24000	(null)
2	Kochhar	AD_VP	17000	(null)
3	De Haan	AD_VP	17000	(null)

...

12	Zlotkey	SA_MAN	10500	0.2
13	Abel	SA_REP	11000	0.3
14	Taylor	SA_REP	8600	0.2
15	Grant	SA_REP	7000	0.15

...

18	Fay	MK_REP	6000	(null)
19	Higgins	AC_MGR	12008	(null)
20	Gietz	AC_ACCOUNT	8300	(null)

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wenn der Datenwert einer bestimmten Spalte für eine Zeile fehlt, wird von einem Nullwert gesprochen.

Spalten mit einem Nullwert können in `SELECT`-Abfragen gewählt werden und auch Teil eines arithmetischen Ausdrucks sein. Jeder arithmetische Ausdruck mit Nullwerten ergibt `NULL`.

Spalten aller Datentypen können Nullwerte enthalten. Bestimmte Constraints (`NOT NULL` und `PRIMARY KEY`) verhindern jedoch, dass Nullwerte in der Spalte verwendet werden.

Im Beispiel auf der Folie zeigt die Spalte `COMMISSION_PCT` der Tabelle `EMPLOYEES`, dass nur Sales Manager oder Sales Representatives Provisionen verdienen können. Die anderen Mitarbeiter sind nicht provisionsberechtigt. Ein Nullwert spiegelt diese Tatsache wider.

**Hinweis:** SQL Developer gibt Nullwerte standardmäßig mit dem Literal `NULL` an. Sie können jedoch einen Wert einstellen, der für Sie mehr Aussagekraft hat. Wählen Sie hierzu im Menü **Tools** die Option **Preferences**, und blenden Sie im Dialogfeld **Preferences** den Knoten **Database** ein. Klicken Sie auf **Advanced Parameters**, und geben Sie im rechten Bereich für **Display Null value As** den entsprechenden Wert ein.

## Nullwerte in arithmetischen Ausdrücken

Bei arithmetischen Ausdrücken, die einen Nullwert enthalten, ist das Auswertungsergebnis ein Nullwert.

```
SELECT last_name, 12*salary*commission_pct  
FROM employees;
```

	LAST_NAME	12*SALARY*COMMISSION_PCT
1	King	(null)
2	Kochhar	(null)
3	De Haan	(null)

...

12	Zlotkey	25200
13	Abel	39600
14	Taylor	20640
15	Grant	12600

...

17	Hartstein	(null)
18	Fay	(null)
19	Higgins	(null)
20	Gietz	(null)

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sobald eine Spalte in einem arithmetischen Ausdruck einen Nullwert enthält, ist das Ergebnis ebenfalls ein Nullwert. Beispiel: Wenn Sie versuchen, einen Wert durch 0 zu dividieren, erhalten Sie einen Fehler. Wenn Sie jedoch eine Zahl durch einen Nullwert dividieren, ist das Ergebnis ein Nullwert, also unbekannt.

Im Beispiel auf der Folie erhält der Mitarbeiter King keine Provision. Da die Spalte `COMMISSION_PCT` im arithmetischen Ausdruck einen Nullwert enthält, ist das Ergebnis ebenfalls ein Nullwert.

Weitere Informationen finden Sie in der *Oracle Database SQL Language Reference* für Oracle Database 12c im Abschnitt "Basic Elements of Oracle SQL".

## Lektionsagenda

- SQL-Anweisungen vom Typ `SELECT` – Funktionen
- Arithmetische Ausdrücke und Nullwerte in `SELECT`-Anweisungen
- **Spaltenaliasnamen**
- Verkettungsoperator, literale Zeichenfolgen, alternativer Operator für Anführungszeichen und Schlüsselwort `DISTINCT`
- Befehl `DESCRIBE`

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.



## Spaltenaliasnamen definieren

Ein Spaltenalias:

- benennt eine Spaltenüberschrift um
- ist nützlich bei Berechnungen
- wird direkt nach dem Spaltennamen angegeben (Optional kann zwischen dem Spaltennamen und dem Alias das Schlüsselwort `AS` angegeben werden.)
- muss in doppelte Anführungszeichen gesetzt werden, wenn er Leer- oder Sonderzeichen enthält oder wenn die Groß-/Kleinschreibung beachtet werden soll

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Bei der Anzeige von Abfrageergebnissen verwendet SQL Developer normalerweise den Namen der gewählten Spalte als Spaltenüberschrift. Diese Überschrift ist möglicherweise nicht sehr aussagekräftig und daher schwer verständlich. Sie können eine Spaltenüberschrift ändern, indem Sie einen Spaltenalias angeben.

Geben Sie den Alias in der `SELECT`-Liste nach der Spalte ein, und verwenden Sie dabei ein Leerzeichen als Trennzeichen. Standardmäßig werden Aliasnamen in Überschriften in Großbuchstaben angezeigt. Wenn der Alias Leer- oder Sonderzeichen enthält (z. B. `-`, `!` oder `_`) oder wenn die Groß-/Kleinschreibung beachtet werden soll, setzen Sie den Alias in doppelte Anführungszeichen (`" "`).

# Spaltenaliasnamen

```
SELECT last_name AS name, commission_pct comm
FROM employees;
```

	NAME	COMM
1	King	(null)
2	Kochhar	(null)
3	De Haan	(null)
4	Hunold	(null)

...

```
SELECT last_name "Name", salary*12 "Annual Salary"
FROM employees;
```

	Name	Annual Salary
1	King	288000
2	Kochhar	204000
3	De Haan	204000
4	Hunold	108000

...

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im ersten Beispiel werden die Namen und Provisionsprozentsätze aller Mitarbeiter angezeigt. Beachten Sie, dass das optionale Schlüsselwort `AS` vor dem Spaltenalias angegeben wurde. Das Ergebnis der Abfrage ist dasselbe, und zwar unabhängig davon, ob das Schlüsselwort `AS` verwendet wird. Die Spaltenaliasnamen `name` und `comm` wurden in der SQL-Anweisung in Kleinbuchstaben angegeben. Die Spaltenüberschriften im Ergebnis der Abfrage werden jedoch in Großbuchstaben angezeigt. Wie auf der vorhergehenden Folie erwähnt, werden für Spaltenüberschriften standardmäßig Großbuchstaben verwendet.

Im zweiten Beispiel werden die Nachnamen und Jahresgehälter aller Mitarbeiter angezeigt. Da `Annual Salary` ein Leerzeichen enthält, wurde es in doppelte Anführungszeichen gesetzt. Die Spaltenüberschrift in der Ausgabe ist mit dem Spaltenalias identisch.

# Lektionsagenda

- SQL-Anweisungen vom Typ `SELECT` – Funktionen
- Arithmetische Ausdrücke und Nullwerte in Anweisungen vom Typ `SELECT`
- Spaltenaliasnamen
- Verkettungsoperator, literale Zeichenfolgen, alternativer Operator für Anführungszeichen und Schlüsselwort `DISTINCT`
- Befehl `DESCRIBE`

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

# Verkettungsoperator

Ein Verkettungsoperator:

- verknüpft Spalten und Zeichenfolgen mit anderen Spalten
- wird durch zwei senkrechte Striche ( || ) dargestellt
- erstellt eine Ergebnisspalte, die einen Zeichenausdruck enthält

```
SELECT last_name || job_id AS "Employees"
FROM employees;
```

Employees
1 AbelSA_REP
2 DaviesST_CLERK
3 De HaanAD_VP
4 ErnstIT_PROG
5 FayMK_REP
6 GietzAC_ACCOUNT
7 GrantSA_REP
8 HartsteinMK_MAN

...

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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

Im Beispiel werden `LAST_NAME` und `JOB_ID` verkettet und mit dem Alias `Employees` bezeichnet. Beachten Sie, dass Nachname des Mitarbeiters und Tätigkeitscode kombiniert und in der Ausgabe in einer einzigen Spalte angezeigt werden.

Das Schlüsselwort `AS` vor dem Alias verbessert die Lesbarkeit der Klausel `SELECT`.

## Nullwerte und Verkettungsoperator

Wenn Sie einen Nullwert mit einer Zeichenfolge verketten, ist das Ergebnis eine Zeichenfolge.

`LAST_NAME || NULL` ergibt `LAST_NAME`.

## Literale Zeichenfolgen

- Literale sind Zeichen, Zahlen oder Datumswerte in einer `SELECT`-Anweisung.
- Literale Datums- und Zeichenwerte müssen in Hochkommas gesetzt werden.
- Jede Zeichenfolge wird einmal pro zurückgegebener Zeile ausgegeben.

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Literale sind Zeichen, Zahlen oder Datumswerte, die in der `SELECT`-Liste aufgeführt, aber keine Spaltennamen oder Spaltenaliasnamen sind. Sie werden für jede zurückgegebene Zeile ausgegeben. Literale Zeichenfolgen in einem beliebigen Textformat können im Abfrageergebnis enthalten sein. Sie werden in der `SELECT`-Liste wie eine Spalte behandelt.

Datums- und Zeichenliterale *müssen* in einfache Anführungszeichen ( ' ' ) gesetzt werden, Zahlenliterale nicht.

## Literale Zeichenfolgen – Beispiel

```
SELECT last_name || ' is a ' || job_id  
        AS "Employee Details"  
FROM    employees;
```

	Employee Details
1	Abel is a SA_REP
2	Davies is a ST_CLERK
3	De Haan is a AD_VP
4	Ernst is a IT_PROG
5	Fay is a MK_REP
6	Gietz is a AC_ACCOUNT
7	Grant is a SA_REP
8	Hartstein is a MK_MAN
9	Higgins is a AC_MGR
10	Hunold is a IT_PROG
11	King is a AD_PRES

...

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf der Folie werden die Nachnamen und Tätigkeitscodes aller Mitarbeiter angezeigt. Die Spalte hat die Überschrift **Employee Details**. Beachten Sie die Leerzeichen zwischen den Hochkommas in der Anweisung `SELECT`. Die Leerzeichen verbessern die Lesbarkeit der Ausgabe.

Im folgenden Beispiel werden die Nachnamen und Gehälter der einzelnen Mitarbeiter mit einem Literal verkettet, um den zurückgegebenen Zeilen mehr Aussagekraft zu verleihen:

```
SELECT last_name || ': 1 Month salary = ' || salary Monthly  
FROM    employees;
```

## Operator $\mathfrak{q}$ für alternative Anführungszeichen

- Eigenes Begrenzungszeichen anstelle von Anführungszeichen angeben
- Beliebiges Begrenzungszeichen wählen
- Lesbarkeit und Verwendbarkeit verbessern

```
SELECT department_name ||  $\mathfrak{q}$ '[ Department's Manager Id: ]'  
      || manager_id  
      AS "Department and Manager"  
FROM departments;
```

Department and Manager	
1	Administration Department's Manager Id: 200
2	Marketing Department's Manager Id: 201
3	Shipping Department's Manager Id: 124
4	IT Department's Manager Id: 103
5	Sales Department's Manager Id: 149
6	Executive Department's Manager Id: 100
7	Accounting Department's Manager Id: 205
8	Contracting Department's Manager Id:

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Viele SQL-Anweisungen verwenden Zeichenlitterale in Ausdrücken und Bedingungen. Wenn das Literal selbst ein Hochkomma enthält, können Sie mit dem Operator  $\mathfrak{q}$  ein eigenes Begrenzungszeichen wählen.

Sie können ein beliebiges Begrenzungszeichen wählen: Einzel- oder Mehrbytezeichen oder eines der Zeichenpaare [ ], { }, ( ) oder < >.

Im Beispiel auf der Folie enthält die Zeichenfolge ein Hochkomma, das normalerweise als Begrenzungszeichen der Zeichenfolge interpretiert wird. Mit dem Operator  $\mathfrak{q}$  werden anstelle von Anführungszeichen eckige Klammern ( [ ] ) als Begrenzungszeichen verwendet. Die Zeichenfolge zwischen den eckigen Klammern wird als literale Zeichenfolge interpretiert.

## Mehrfach vorhandene Zeilen

Standardmäßig werden bei Abfragen alle Zeilen angezeigt, auch mehrfach vorhandene Zeilen.

1

```
SELECT department_id  
FROM employees;
```

	DEPARTMENT_ID
1	90
2	90
3	90
4	60
5	60
6	60
7	50
8	50

...

2

```
SELECT DISTINCT department_id  
FROM employees;
```

	DEPARTMENT_ID
1	(null)
2	90
3	20
4	110
5	50
6	80
7	60
8	10

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wenn nicht anders angegeben, zeigt SQL als Ergebnis einer Abfrage alle Zeilen an, ohne mehrfach vorhandene Zeilen auszublenden. Im ersten Beispiel auf der Folie werden alle Abteilungsnummern aus der Tabelle `EMPLOYEES` angezeigt. Beachten Sie, dass die Abteilungsnummern mehrfach angezeigt werden.

Um mehrfach vorhandene Zeilen aus dem Ergebnis auszublenden, geben Sie in der Klausel `SELECT` unmittelbar nach dem Schlüsselwort `SELECT` das Schlüsselwort `DISTINCT` an. Im zweiten Beispiel auf der Folie enthält die Tabelle `EMPLOYEES` 20 Zeilen, dabei aber nur sieben verschiedene Abteilungsnummern.

Sie können mehrere Spalten nach dem Qualifier `DISTINCT` angeben. Der Qualifier `DISTINCT` wirkt sich auf alle gewählten Spalten aus. Als Ergebnis werden alle eindeutigen Spaltenkombinationen angezeigt.

```
SELECT DISTINCT department_id, job_id  
FROM employees;
```



# Lektionsagenda

- SQL-Anweisungen vom Typ `SELECT` – Funktionen
- Arithmetische Ausdrücke und Nullwerte in `SELECT`-Anweisungen
- Spaltenaliasnamen
- Verkettungsoperator, literale Zeichenfolgen, alternativer Operator für Anführungszeichen und Schlüsselwort `DISTINCT`
- **Befehl `DESCRIBE`**

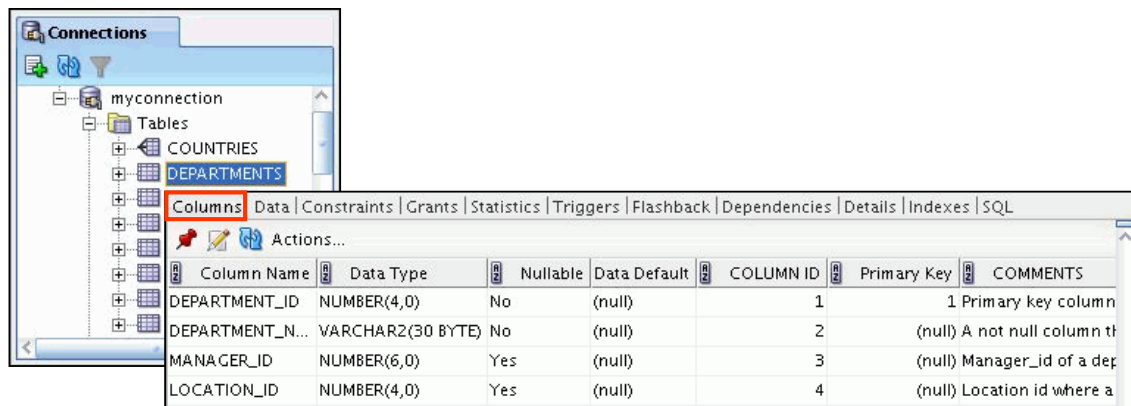
ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Tabellenstrukturen anzeigen

- Struktur einer Tabelle mit dem Befehl `DESCRIBE` anzeigen
- Oder die Tabelle im Baum **Connections** markieren und die Tabellenstruktur in der Registerkarte **Columns** anzeigen

```
DESC[RIBE] tablename
```



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Struktur einer Tabelle zeigen Sie mit dem Befehl `DESCRIBE` an. Der Befehl zeigt die Spaltennamen und Datentypen an und gibt an, ob eine Spalte Daten enthalten *muss* (das heißt, ob für die Spalte ein Constraint vom Typ `NOT NULL` existiert).

In der Syntax ist *table name* der Name beliebiger vorhandener Tabellen, Views und Synonyme, auf die der Benutzer zugreifen kann.

Mit der grafischen Benutzeroberfläche von SQL Developer können Sie im Baum **Connections** die Tabelle markieren und in der Registerkarte **Columns** die Tabellenstruktur anzeigen.

**Hinweis:** Der SQL\*Plus-Befehl `DESCRIBE` (kurz: `DESC`) wird auch von SQL Developer unterstützt.

## Befehl DESCRIBE

```
DESCRIBE employees
```

```
DESCRIBE Employees
Name          Null      Type
-----
EMPLOYEE_ID   NOT NULL  NUMBER(6)
FIRST_NAME    VARCHA2(20)
LAST_NAME     NOT NULL  VARCHA2(25)
EMAIL         NOT NULL  VARCHA2(25)
PHONE_NUMBER  VARCHA2(20)
HIRE_DATE     NOT NULL  DATE
JOB_ID        NOT NULL  VARCHA2(10)
SALARY        NUMBER(8,2)
COMMISSION_PCT NUMBER(2,2)
MANAGER_ID    NUMBER(6)
DEPARTMENT_ID NUMBER(4)
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf der Folie werden mit dem Befehl `DESCRIBE` Informationen über die Struktur der Tabelle `EMPLOYEES` angezeigt.

In der Ergebnisanzeige gibt *Null* an, dass die Werte für diese Spalte unter Umständen unbekannt sind. `NOT NULL` gibt an, dass eine Spalte Daten enthalten muss. *Type* zeigt den Datentyp einer Spalte an.

Die Datentypen werden in der folgenden Tabelle beschrieben:

Datentyp	Beschreibung
NUMBER ( <i>p</i> , <i>s</i> )	Zahlenwert mit maximal <i>p</i> Ziffern und <i>s</i> Dezimalstellen
VARCHAR2 ( <i>s</i> )	Zeichenwert variabler Länge mit einer maximalen Größe von <i>s</i>
DATE	Datums- und Zeitwert zwischen dem 1. Januar 4712 v. Chr. und dem 31. Dezember 9999 n. Chr.

## Quiz

Welche SELECT-Anweisungen werden erfolgreich ausgeführt?

a. 

```
SELECT first_name, last_name, job_id, salary*12
AS Yearly Sal
FROM employees;
```

b. 

```
SELECT first_name, last_name, job_id, salary*12
"yearly sal"
FROM employees;
```

c. 

```
SELECT first_name, last_name, job_id, salary AS
"yearly sal"
FROM employees;
```

d. 

```
SELECT first_name+last_name AS name, job_id,
salary*12 yearly sal
FROM employees;
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

**Richtige Antwort: b und c**

# Zusammenfassung

In dieser Lektion haben Sie gelernt, `SELECT`-Anweisungen zu erstellen, die:

- alle Zeilen und Spalten einer Tabelle zurückgeben
- bestimmte Spalten einer Tabelle zurückgeben
- Spaltenaliasnamen verwenden, um aussagekräftigere Spaltenüberschriften anzuzeigen

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In dieser Lektion haben Sie gelernt, mithilfe von `SELECT`-Anweisungen Daten aus einer Datenbanktabelle abzurufen.

```
SELECT  *|{[DISTINCT] column [alias],...}  
FROM    table;
```

Für die Syntax gilt:

<code>SELECT</code>	Ist eine Liste aus einzelnen oder mehreren Spalten
<code>*</code>	Wählt alle Spalten
<code>DISTINCT</code>	Unterdrückt doppelte Werte
<code>column expression</code>	Wählt die angegebene Spalte oder den Ausdruck
<code>alias</code>	Gibt den gewählten Spalten eine andere Überschrift
<code>FROM table</code>	Gibt an, welche Tabelle die Spalten enthält

## Übungen zu Lektion 2 – Überblick

Diese Übungen behandeln folgende Themen:

- Alle Daten aus verschiedenen Tabellen wählen
- Tabellenstrukturen beschreiben
- Arithmetische Berechnungen ausführen und Spaltennamen angeben

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In dieser Übung erstellen Sie einfache `SELECT`-Abfragen. Dabei verwenden Sie die meisten der in dieser Lektion behandelten `SELECT`-Klauseln und -Vorgänge.

# 3

## Daten einschränken und sortieren

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- Von Abfragen ausgegebene Zeilen einschränken
- Von Abfragen ausgegebene Zeilen sortieren
- Ausgabe mithilfe von Substitutionsvariablen (&, &&) zur Laufzeit einschränken und sortieren

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Beim Abruf von Daten aus der Datenbank möchten Sie gegebenenfalls folgende Aufgaben ausführen:

- Anzuzeigende Datenzeilen einschränken
- Reihenfolge der anzuzeigenden Zeilen festlegen

In dieser Lektion werden die SQL-Anweisungen für die auf der Folie genannten Aktionen vorgestellt.



# Lektionsagenda

- Auszugebende Zeilen einschränken mit:
  - der Klausel `WHERE`
  - Vergleichsbedingungen mit den Operatoren `=`, `<=`, `BETWEEN`, `IN`, `LIKE` und `NULL`
  - logischen Bedingungen mit den Operatoren `AND`, `OR` und `NOT`
- Prioritätsregeln für Operatoren in Ausdrücken
- Zeilen mit der Klausel `ORDER BY` sortieren
- SQL-Klausel zur Zeilenbeschränkung in Abfragen
- Substitutionsvariablen
- Befehle `DEFINE` und `VERIFY`

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.


## Zeilen durch Auswahl begrenzen

EMPLOYEES

	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	100	King	AD_PRES	90
2	101	Kochhar	AD_VP	90
3	102	De Haan	AD_VP	90
4	103	Hunold	IT_PROG	60
5	104	Ernst	IT_PROG	60
6	107	Lorentz	IT_PROG	60

...

"Alle Mitarbeiter  
aus Abteilung  
90 abrufen"



	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	100	King	AD_PRES	90
2	101	Kochhar	AD_VP	90
3	102	De Haan	AD_VP	90

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf der Folie sollen alle Mitarbeiter aus Abteilung 90 angezeigt werden. Zurückgegeben werden deshalb nur die Zeilen, bei denen die Spalte `DEPARTMENT_ID` den Wert 90 enthält. Auf dieser Einschränkungsmethode basiert die SQL-Klausel `WHERE`.

## Auszugebende Zeilen einschränken

- Zurückzugebende Zeilen mit der Klausel `WHERE` einschränken:

```
SELECT  * | {[DISTINCT] column [alias],...}  
FROM    table  
[WHERE logical expression(s)];
```

- Die Klausel `WHERE` folgt auf die Klausel `FROM`.

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie können die von der Abfrage zurückgegebenen Zeilen mit der Klausel `WHERE` einschränken. Eine `WHERE`-Klausel enthält eine Bedingung, die erfüllt sein muss. Sie wird unmittelbar nach der Klausel `FROM` angegeben. Ist die Bedingung erfüllt, wird die entsprechende Zeile zurückgegeben.

Für die Syntax gilt:

`WHERE`

Schränkt die Abfrage auf die Zeilen ein, die eine Bedingung erfüllen

*logical expression*

Setzt sich zusammen aus Spaltennamen, Konstanten und einem Vergleichsoperator. Ein logischer Ausdruck gibt eine Kombination aus einzelnen oder mehreren Ausdrücken und booleschen Operatoren an und gibt den Wert `TRUE`, `FALSE` oder `KNOWN` zurück.

Die Klausel `WHERE` kann Werte in Spalten, Literalen, arithmetischen Ausdrücken und Funktionen vergleichen. Sie besteht aus drei Elementen:

- Spaltenname
- Vergleichsbedingung
- Spaltenname, Konstante oder Werteliste

## Klausel WHERE

```
SELECT employee_id, last_name, job_id, department_id
FROM   employees
WHERE  department_id = 90 ;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	100	King	AD_PRES	90
2	101	Kochhar	AD_VP	90
3	102	De Haan	AD_VP	90

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel oben ruft die Anweisung `SELECT` Personalnummer, Nachname, Tätigkeits-ID und Abteilungsnummer aller Mitarbeiter der Abteilung 90 ab.

**Hinweis:** Die Klausel `WHERE` darf keine Spaltenaliasnamen enthalten.

## Zeichenfolgen und Datumsangaben

- Zeichenfolgen und Datumswerte werden in Hochkommas gesetzt.
- Bei Zeichenwerten wird die Groß-/Kleinschreibung beachtet, Datumswerte sind formatabhängig.
- Das Standardanzeigeformat lautet DD-MON-RR.

```
SELECT last_name, job_id, department_id
FROM employees
WHERE last_name = 'Whalen';
```

LAST_NAME	JOB_ID	DEPARTMENT_ID
1 Whalen	AD_ASST	10

```
SELECT last_name
FROM employees
WHERE hire_date = '17-OCT-03';
```

LAST_NAME
1 Rajs

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Zeichenfolgen und Datumsangaben in `WHERE`-Klauseln müssen in Hochkommas ( ' ' ) gesetzt werden. Für numerische Konstanten gilt diese Vorgabe nicht.

Bei jeder Suche nach Zeichenwerten wird die Groß-/Kleinschreibung beachtet. Im folgenden Beispiel werden keine Zeilen zurückgegeben, da die Nachnamen in der Tabelle `EMPLOYEES` in gemischter Groß- und Kleinschreibung gespeichert sind:

```
SELECT last_name, job_id, department_id
FROM employees
WHERE last_name = 'WHALEN';
```

Oracle-Datenbanken speichern Datumsangaben in einem internen numerischen Format, das Jahrhundert, Jahr, Monat, Tag, Stunden, Minuten und Sekunden angibt. Das Standardformat für die Datumsanzeige ist DD-MON-RR.

**Hinweis:** Weitere Informationen zum RR-Format und zur Änderung des Standarddatumsformats finden Sie in der Lektion "Ausgabe mit Single-Row-Funktionen anpassen". Dort lernen Sie auch, wie Sie mit Single-Row-Funktionen wie `UPPER` und `LOWER` die Beachtung der Groß-/Kleinschreibung außer Kraft setzen.

# Vergleichsoperatoren

Operator	Bedeutung
=	Gleich
>	Größer als
>=	Größer/gleich
<	Kleiner als
<=	Kleiner/gleich
<>	Ungleich
BETWEEN ...AND...	Zwischen zwei Werten (einschließlich dieser Werte)
IN(set)	Entspricht einem Wert aus einer Werteliste
LIKE	Entspricht einem Zeichenmuster
IS NULL	Ist ein Nullwert

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Vergleichsoperatoren werden in Bedingungen verwendet, in denen ein Ausdruck mit einem anderen Wert oder Ausdruck verglichen wird. Sie werden in der Klausel `WHERE` im folgenden Format angegeben:

## Syntax

```
... WHERE expr operator value
```

## Beispiel

```
... WHERE hire_date = '01-JAN-05'  
... WHERE salary >= 6000  
... WHERE last_name = 'Smith'
```

Berücksichtigen Sie, dass in der Klausel `WHERE` keine Aliasnamen verwendet werden können.

**Hinweis:** Die Symbole `!=` und `^=` können ebenfalls die Bedingung *ungleich* darstellen.

## Vergleichsoperatoren – Beispiel

```
SELECT last_name, salary
FROM   employees
WHERE  salary <= 3000 ;
```

	LAST_NAME	SALARY
1	Matos	2600
2	Vargas	2500

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im ersten Beispiel auf der Folie ruft die Anweisung `SELECT` aus der Tabelle `EMPLOYEES` Nachname und Gehalt aller Mitarbeiter ab, deren Gehalt kleiner/gleich \$ 3.000 ist. In der Klausel `WHERE` wird ein expliziter Wert (3000) verwendet, der mit dem Gehaltswert in der Spalte `SALARY` der Tabelle `EMPLOYEES` verglichen wird.

## Bereichsbedingungen mit dem Operator BETWEEN

Mit dem Operator BETWEEN Zeilen anzeigen, die auf einem Wertebereich basieren:

```
SELECT last_name, salary
FROM   employees
WHERE  salary BETWEEN 2500 AND 3500 ;
```

Unterer Grenzwert

Oberer Grenzwert

	LAST_NAME	SALARY
1	Rajs	3500
2	Davies	3100
3	Matos	2600
4	Vargas	2500

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie können Zeilen auf der Basis eines Wertebereichs anzeigen, indem Sie den Operator BETWEEN verwenden. Der angegebene Bereich verfügt über einen unteren und einen oberen Grenzwert.

Die Anweisung SELECT auf der Folie gibt die Zeilen aller Mitarbeiter aus der Tabelle EMPLOYEES zurück, deren Gehalt zwischen \$ 2.500 und \$ 3.500 liegt.

Die mit dem Operator BETWEEN angegebene Werte sind inklusiv, das heißt enthalten auch die Grenzwerte. Sie müssen den unteren Grenzwert jedoch zuerst angeben.

Sie können den Operator BETWEEN auch für Zeichenwerte verwenden:

```
SELECT last_name FROM   employees
WHERE last_name BETWEEN 'King' AND 'Whalen 10
```



## Operator IN – Beispiel

Mit dem Operator IN auf Werte in einer Liste testen:

```
SELECT employee_id, last_name, salary, manager_id
FROM   employees
WHERE  manager_id IN (100, 101, 201) ;
```

	EMPLOYEE_ID	LAST_NAME	SALARY	MANAGER_ID
1	101	Kochhar	17000	100
2	102	De Haan	17000	100
3	124	Mourgos	5800	100
4	149	Zlotkey	10500	100
5	201	Hartstein	13000	100
6	200	Whalen	4400	101
7	205	Higgins	12008	101
8	202	Fay	6000	201

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Um in einer bestimmten Gruppe von Werten auf bestimmte Werte zu testen, verwenden Sie den Operator IN. Mit dem Operator IN definierte Bedingungen werden auch als *Zugehörigkeitsbedingungen* bezeichnet.

Das Beispiel auf der Folie zeigt Personalnummer, Nachname, Gehalt und Managerpersonalnummer aller Mitarbeiter, denen die Managerpersonalnummer 100, 101 oder 201 zugeordnet ist.

**Hinweis:** Die Wertegruppe kann in beliebiger Reihenfolge angegeben werden, zum Beispiel (201, 100, 101).

Der Operator IN kann für jeden Datentyp verwendet werden. Das folgende Beispiel gibt für alle Mitarbeiter der Tabelle EMPLOYEES eine Zeile zurück, deren Nachname in der Namensliste der Klausel WHERE angegeben ist:

```
SELECT employee_id, manager_id, department_id
FROM   employees
WHERE  last_name IN ('Hartstein', 'Vargas');
```

Zeichen- und Datumswerte müssen in der Liste in Hochkommas ( ' ' ) gesetzt werden.

## Muster mit dem Operator LIKE vergleichen

- Mit dem Operator LIKE eine Platzhaltersuche nach gültigen Zeichenfolgenwerten durchführen
- Die Suchkriterien können entweder literale Zeichen oder Zahlen enthalten:
  - % steht für kein oder mehrere Zeichen.
  - \_ steht für genau ein Zeichen.

```
SELECT first_name
FROM employees
WHERE first_name LIKE 'S%';
```

	FIRST_NAME
1	Shelley
2	Steven

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Nicht immer kennen Sie den exakten Wert, nach dem Sie suchen möchten. Mit dem Operator LIKE können Sie Zeilen wählen, die einem Zeichenmuster entsprechen. Die Suche nach einem Zeichenmuster wird als *Platzhaltersuche* bezeichnet. Zum Erstellen der Suchzeichenfolge können zwei Symbole verwendet werden.

Symbol	Beschreibung
%	Steht für eine beliebige Folge von keinem oder mehreren Zeichen
_	Steht für ein beliebiges einzelnes Zeichen

Die Anweisung SELECT auf der Folie gibt den Vornamen aller Mitarbeiter aus der Tabelle EMPLOYEES zurück, deren Vorname mit dem Buchstaben "S" beginnt. "S" wird als Großbuchstabe angegeben. Namen, die mit einem kleinen "s" beginnen, werden folglich ignoriert.

Der Operator LIKE kann als Shortcut für einige BETWEEN-Vergleiche verwendet werden. Im folgenden Beispiel werden Nachname und Einstellungsdatum aller Mitarbeiter angezeigt, die zwischen Januar 2005 und Dezember 2005 eingestellt wurden:

```
SELECT last_name, hire_date
FROM employees
WHERE hire_date LIKE '%05';
```

## Platzhalterzeichen kombinieren

- Für Mustervergleiche die beiden Platzhalterzeichen (% , \_) mit literalen Zeichen kombinieren:

```
SELECT last_name  
FROM   employees  
WHERE  last_name LIKE '_o%';
```

	LAST_NAME
1	Kochhar
2	Lorentz
3	Mourgos

- Um nach den Symbolen % und \_ selbst zu suchen, verwenden Sie den Identifier `ESCAPE`.

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie können die Symbole % und \_ beliebig mit literalen Zeichen kombinieren. Das Beispiel auf der Folie zeigt die Namen aller Mitarbeiter an, in deren Nachnamen "o" der zweite Buchstabe ist. Um nach den Zeichen % und \_ selbst zu suchen, verwenden Sie den Identifier `ESCAPE`.

# NULL-Bedingungen

Mit dem Operator IS NULL auf Nullwerte testen

```
SELECT last_name, manager_id
FROM   employees
WHERE  manager_id IS NULL ;
```

	LAST_NAME	MANAGER_ID
1	King	(null)

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Zu den NULL-Bedingungen zählen die Operatoren IS NULL und IS NOT NULL.

Der Operator IS NULL testet auf Nullwerte. Ein Nullwert bedeutet, dass der Wert nicht verfügbar, nicht zugewiesen, unbekannt oder nicht anwendbar ist. Der Operator = kann nicht verwendet werden, denn ein Nullwert kann nicht gleich oder ungleich einem anderen Wert sein. Im Beispiel auf der Folie werden Nachname (last\_name) und Manager-ID (manager\_id) aller Mitarbeiter abgerufen, denen kein Manager zugewiesen ist.

Weiteres Beispiel: Um Nachname, Tätigkeits-ID und Provision aller Mitarbeiter abzurufen, die *nicht* provisionsberechtigt sind, verwenden Sie folgende SQL-Anweisung:

```
SELECT last_name, job_id, commission_pct
FROM   employees
WHERE  commission_pct IS NULL;
```

...

## Bedingungen mit logischen Operatoren definieren

Operator	Bedeutung
AND	Gibt TRUE zurück, wenn <i>beide</i> Teilbedingungen wahr sind
OR	Gibt TRUE zurück, wenn <i>eine der beiden</i> Teilbedingungen wahr ist
NOT	Gibt TRUE zurück, wenn die Bedingung falsch ist

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Ein logischer Operator kombiniert das Ergebnis von zwei oder mehr Teilbedingungen zu einem einzelnen Ergebnis oder kehrt das Ergebnis einer einzelnen Bedingung um. Es wird nur dann eine Zeile zurückgegeben, wenn das Gesamtergebnis der Bedingung wahr ist.

In SQL sind drei logische Operatoren verfügbar:

- AND
- OR
- NOT

In allen bisherigen Beispielen wurde jeweils nur eine Bedingung in der Klausel `WHERE` angegeben. Mit den Operatoren `AND` und `OR` können Sie mehrere Bedingungen in einer einzelnen `WHERE`-Klausel verwenden.

## Operator AND

AND erfordert, dass beide Teilbedingungen wahr sind:

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary >= 10000
AND    job_id LIKE '%MAN%' ;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	149	Zlotkey	SA_MAN	10500
2	201	Hartstein	MK_MAN	13000

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel werden nur die Datensätze gewählt, für die beide Teilbedingungen wahr sind. Daher werden nur die Mitarbeiter zurückgegeben, deren Tätigkeits-ID die Zeichenfolge "MAN" enthält *und* die gleichzeitig \$ 10.000 oder mehr verdienen.

Bei der Zeichensuche wird immer die Groß-/Kleinschreibung beachtet. Es werden keine Zeilen zurückgegeben, wenn "MAN" nur in Kleinschreibung vorkommt. Außerdem müssen Zeichenfolgen in Anführungszeichen gesetzt werden.

### Wahrheitstabelle für AND

Die folgende Tabelle enthält die Ergebnisse aus der Kombination zweier Ausdrücke mit AND:

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

## Operator OR

OR erfordert, dass eine der beiden Teilbedingungen wahr ist:

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary >= 10000
OR     job_id LIKE '%MAN%';
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	100	King	AD_PRES	24000
2	101	Kochhar	AD_VP	17000
3	102	De Haan	AD_VP	17000
4	124	Mourgos	ST_MAN	5800
5	149	Zlotkey	SA_MAN	10500
6	174	Abel	SA_REP	11000
7	201	Hartstein	MK_MAN	13000
8	205	Higgins	AC_MGR	12008

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel werden alle Datensätze gewählt, für die mindestens eine der beiden Teilbedingungen wahr ist. Es werden die Mitarbeiter zurückgegeben, deren Tätigkeits-ID die Zeichenfolge "MAN" enthält *und/oder* die \$ 10.000 oder mehr verdienen.

### Wahrheitstabelle für OR

Die folgende Tabelle enthält die Ergebnisse aus der Kombination zweier Ausdrücke mit OR:

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

## Operator NOT

```
SELECT last_name, job_id
FROM   employees
WHERE  job_id
      NOT IN ('IT_PROG', 'ST_CLERK', 'SA_REP') ;
```

	LAST_NAME	JOB_ID
1	De Haan	AD_VP
2	Fay	MK_REP
3	Gietz	AC_ACCOUNT
4	Hartstein	MK_MAN
5	Higgins	AC_MGR
6	King	AD_PRES
7	Kochhar	AD_VP
8	Mourgos	ST_MAN
9	Whalen	AD_ASST
10	Zlotkey	SA_MAN

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf der Folie werden Nachname und Tätigkeits-ID aller Mitarbeiter angezeigt, deren Tätigkeits-ID *nicht* IT\_PROG, ST\_CLERK oder SA\_REP ist.

### Wahrheitstabelle für NOT

Die folgende Tabelle enthält die Ergebnisse aus der Anwendung des Operators NOT in einer Bedingung:

NOT	TRUE	FALSE	NULL
	FALSE	TRUE	NULL



# Lektionsagenda

- Auszugebende Zeilen beschränken mit:
  - der Klausel `WHERE`
  - Vergleichsbedingungen mit den Operatoren `=`, `<=`, `BETWEEN`, `IN`, `LIKE` und `NULL`
  - logischen Bedingungen mit den Operatoren `AND`, `OR` und `NOT`
- **Prioritätsregeln für Operatoren in Ausdrücken**
- Zeilen mit der Klausel `ORDER BY` sortieren
- SQL-Klausel zur Zeilenbeschränkung in Abfragen
- Substitutionsvariablen
- Befehle `DEFINE` und `VERIFY`

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

# Prioritätsregeln

Bedeutung	Operator
1	Arithmetische Operatoren
2	Verkettungsoperator
3	Vergleichsoperatoren
4	IS [NOT] NULL, LIKE, [NOT] IN
5	[NOT] BETWEEN
6	Ungleich
7	Logischer Operator NOT
8	Logischer Operator AND
9	Logischer Operator OR

Mithilfe von Klammern können Sie die Prioritätsregeln umgehen.

**ORACLE**

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Prioritätsregeln bestimmen die Reihenfolge, in der Ausdrücke ausgewertet und berechnet werden. Die Tabelle auf der Folie listet die Standardreihenfolge der Prioritäten auf. Sie können diese Standardreihenfolge jedoch ändern, indem Sie die Ausdrücke, die zuerst berechnet werden sollen, in Klammern setzen.

# Prioritätsregeln

```
SELECT last_name, department_id, salary
FROM employees
WHERE department_id = 60
OR department_id = 80
AND salary > 10000;
```

1

	LAST_NAME	DEPARTMENT_ID	SALARY
1	Hunold	60	9000
2	Ernst	60	6000
3	Lorentz	60	4200
4	Zlotkey	80	10500
5	Abel	80	11000

```
SELECT last_name, department_id, salary
FROM employees
WHERE (department_id = 60
OR department_id = 80)
AND salary > 10000;
```

2

	LAST_NAME	DEPARTMENT_ID	SALARY
1	Zlotkey	80	10500
2	Abel	80	11000

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## 1. Priorität des Operators AND – Beispiel

Dieses Beispiel enthält zwei Bedingungen:

- Die erste Bedingung lautet: Die Abteilungs-ID ist 80, *und* das Gehalt ist größer als \$ 10.000.
- Die zweite Bedingung lautet: Die Abteilungs-ID ist 60.

Die Anweisung `SELECT` lässt sich daher folgendermaßen lesen:

"Alle Mitarbeiter wählen, deren Abteilung 80 ist *und* die mehr als \$ 10.000 verdienen *oder* deren Abteilung 60 ist"

## 2. Klammersetzung – Beispiel

Dieses Beispiel enthält zwei Bedingungen:

- Die erste Bedingung lautet: Die Abteilungs-ID ist 80 *oder* 60.
- Die zweite Bedingung lautet: Das Gehalt ist größer als \$ 10.000.

Die Anweisung `SELECT` lässt sich daher folgendermaßen lesen:

"Mitarbeiter ausgeben, deren Abteilung 80 *oder* 60 ist *und* die mehr als \$ 10.000 verdienen"

# Lektionsagenda

- Auszugebende Zeilen beschränken mit:
  - der Klausel `WHERE`
  - Vergleichsbedingungen mit den Operatoren `=`, `<=`, `BETWEEN`, `IN`, `LIKE` und `NULL`
  - logischen Bedingungen mit den Operatoren `AND`, `OR` und `NOT`
- Prioritätsregeln für Operatoren in Ausdrücken
- **Zeilen mit der Klausel `ORDER BY` sortieren**
- SQL-Klausel zur Zeilenbeschränkung in Abfragen
- Substitutionsvariablen
- Befehle `DEFINE` und `VERIFY`

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Klausel ORDER BY

Abgerufene Zeilen mit der Klausel ORDER BY sortieren:

- ASC: Aufsteigende Reihenfolge, Standard
- DESC: Absteigende Reihenfolge

```
SELECT last_name, job_id, department_id, hire_date
FROM employees
ORDER BY hire_date ;
```

	LAST_NAME	JOB_ID	DEPARTMENT_ID	HIRE_DATE
1	De Haan	AD_VP		90 13-JAN-01
2	Gietz	AC_ACCOUNT		110 07-JUN-02
3	Higgins	AC_MGR		110 07-JUN-02
4	King	AD_PRES		90 17-JUN-03
5	Whalen	AD_ASST		10 17-SEP-03
6	Rajs	ST_CLERK		50 17-OCT-03

...

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In welcher Reihenfolge die Zeilen im Abfrageergebnis zurückgegeben werden, ist nicht festgelegt. Sie können die Zeilen mit der Klausel ORDER BY sortieren. Als Sortierbedingung können Sie einen Ausdruck, einen Alias oder eine Spaltenposition angeben. Sie können in der *order\_by\_clause* mehrere Ausdrücke angeben. Oracle Database sortiert die Zeilen zunächst auf Basis der Werte des ersten Ausdrucks. Bei gleichen Werten im ersten Ausdruck werden die Zeilen anschließend nach den Werten im zweiten Ausdruck sortiert usw.

### Syntax

```
SELECT          expr
FROM            table
[WHERE          condition(s)]
[ORDER BY {column, expr, numeric_position} [ASC|DESC]];
```

Für die Syntax gilt:

ORDER BY	Legt die Reihenfolge fest, in der die abgerufenen Zeilen angezeigt werden
ASC	Sortiert die Zeilen in aufsteigender Reihenfolge (Dies ist der Standardwert.)
DESC	Sortiert die Zeilen in absteigender Reihenfolge

Wird die Klausel ORDER BY nicht angegeben, ist die Sortierreihenfolge nicht festgelegt. In diesem Fall kann es vorkommen, dass die ausgegebenen Zeilen bei zwei gleichen Abfragen unterschiedlich sortiert sind. Verwenden Sie daher die Klausel ORDER BY, wenn Sie die Zeilen in einer bestimmten Reihenfolge anzeigen möchten.

# Sortieren

- In absteigender Reihenfolge sortieren:

```
SELECT last_name, job_id, department_id, hire_date
FROM employees
ORDER BY department_id DESC ;
```

1

- Nach Spaltenalias sortieren:

```
SELECT employee_id, last_name, salary*12 annsal
FROM employees
ORDER BY annsal ;
```

2

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Standardmäßig werden die Zeilen in aufsteigender Reihenfolge sortiert:

- Bei numerischen Werten wird der niedrigste Wert zuerst angezeigt. Beispiel: von 1 bis 999
- Bei Datumswerten wird das früheste Datum zuerst angezeigt. Beispiel: 01-JAN-92 vor 01-JAN-95
- Zeichenwerte werden in alphabetischer Reihenfolge angezeigt. Beispiel: "A" zuerst und "Z" zuletzt
- Nullwerte werden bei aufsteigender Reihenfolge zuletzt und bei absteigender Reihenfolge zuerst angezeigt.
- Sie können auch nach Spalten sortieren, die nicht in der SELECT-Liste enthalten sind.

## Beispiele

1. Um die Reihenfolge der angezeigten Zeilen umzukehren, geben Sie in der Klausel ORDER BY nach dem Spaltennamen das Schlüsselwort DESC an. Im Beispiel auf der Folie werden die Ergebnisse nach Abteilungsnummer sortiert.
2. Sie können in der Klausel ORDER BY auch einen Spaltenalias angeben. Im Beispiel auf der Folie werden die Daten nach dem Jahresgehalt sortiert.

**Hinweis:** Mit den Schlüsselwörtern NULLS FIRST und NULLS LAST geben Sie an, ob Zeilen mit Nullwerten in der Sortierreihenfolge an erster oder letzter Stelle angezeigt werden sollen.

# Sortieren

- Nach numerischer Position der Spalte sortieren:

```
SELECT last_name, job_id, department_id, hire_date
FROM employees
ORDER BY 3;
```

3

- Nach mehreren Spalten sortieren:

```
SELECT last_name, department_id, salary
FROM employees
ORDER BY department_id, salary DESC;
```

4

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Beispiele

3. Sie können angeben, an welcher Position in der Klausel `SELECT` sich die Spalte befindet, nach der sortiert werden soll. Im Beispiel auf der Folie werden die Ergebnisse nach `department_id` sortiert, da sich diese Spalte in der Klausel `SELECT` an dritter Position befindet.
4. Sie können Abfrageergebnisse nach mehreren Spalten sortieren. Dazu listen Sie die Spalten (oder die Nummern ihrer Positionen in der `SELECT`-Liste) durch Kommas getrennt in der Klausel `ORDER BY` auf. Die Ergebnisse werden zunächst nach der ersten Spalte, anschließend nach der zweiten und allen in der Klausel `ORDER BY` enthaltenen Spalten sortiert. Um einen Teil der Ergebnisse in absteigender Reihenfolge zu sortieren, geben Sie in der Klausel `ORDER BY` direkt nach dem Namen bzw. der Nummer der entsprechenden Spalte das Schlüsselwort `DESC` an. Das Ergebnis der auf der Folie gezeigten Abfrage wird in aufsteigender Reihenfolge nach `department_id` sowie in absteigender Reihenfolge nach `salary` sortiert.

# Lektionsagenda

- Auszugebende Zeilen beschränken mit:
  - der Klausel `WHERE`
  - Vergleichsbedingungen mit den Operatoren `=`, `<=`, `BETWEEN`, `IN`, `LIKE` und `NULL`
  - logischen Bedingungen mit den Operatoren `AND`, `OR` und `NOT`
- Prioritätsregeln für Operatoren in Ausdrücken
- Zeilen mit der Klausel `ORDER BY` sortieren
- **SQL-Klausel zur Zeilenbeschränkung in Abfragen**
- Substitutionsvariablen
- Befehle `DEFINE` und `VERIFY`

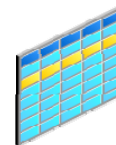
ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.



## SQL-Klausel zur Zeilenbeschränkung

- Mit `row_limiting_clause` die für Abfragen zurückzugebenden Zeilen beschränken
- Mit der Klausel `Top-N-Berichte` implementieren
- Mit dem Schlüsselwort `FETCH FIRST` Anzahl oder Prozentsatz der zurückzugebenden Zeilen angeben
- Mit dem Schlüsselwort `OFFSET` angeben, dass die zurückzugebenden Zeilen mit einer bestimmten Zeile nach der ersten Zeile der vollständigen Ergebnismenge beginnen sollen.
- Mit dem Schlüsselwort `WITH TIES` zusätzliche Zeilen berücksichtigen, die über dieselben Sortierschlüssel verfügen wie die letzte Zeile der eingeschränkten Ergebnismenge (`ORDER BY` muss in der Abfrage angegeben werden.)



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In Oracle Database 12c Release 1 wurde die SQL-Syntax `SELECT` um `row_limiting_clause` erweitert. Mit dieser Klausel lässt sich die Anzahl der in der Ergebnismenge zurückzugebenden Zeilen beschränken. Sie ist einfach verständlich und gleichzeitig aussagekräftig. Eine Beschränkung der zurückzugebenden Zeilen kann für Berichterstellung, Analyse, Datensuche oder andere Aufgaben sinnvoll sein. Abfragen, die Daten sortieren und dann die Zeilenausgabe beschränken, sind gängig. Sie werden häufig als Top-N-Abfragen bezeichnet. Top-N-Abfragen sortieren die Ergebnismenge und geben dann nur die ersten *n* Zeilen zurück.

Mit dem Schlüsselwort `FETCH FIRST` können Sie Anzahl oder Prozentsatz der zurückzugebenden Zeilen angeben. Mit dem Schlüsselwort `OFFSET` können Sie angeben, dass eine Zeile nach der ersten Zeile des vollständigen Ergebnissatzes als erste Zeile zurückgegeben werden soll. Das Schlüsselwort `WITH TIES` berücksichtigt Zeilen, die dieselben Sortierschlüssel verwenden wie die letzte Zeile der eingeschränkten Ergebnismenge. (`WITH TIES` erfordert die Angabe einer `ORDER BY`-Klausel.) Die Schlüsselwörter `FETCH FIRST` und `OFFSET` vereinfachen die Syntax und entsprechen dem ANSI-Standard für SQL.

Einschränkungen der SQL-Klausel zur Zeilenbeschränkung:

- Die Klausel kann nicht mit `for_update_clause` verwendet werden.
- Die Klausel kann nicht in Unterabfragen von `DELETE`- oder `UPDATE`-Anweisungen verwendet werden.
- In der `SELECT`-Liste sind die Sequence-Pseudospalten `CURRVAL` oder `NEXTVAL` nicht zulässig.

# SQL-Klausel zur Zeilenbeschränkung in Abfragen

Die `row_limiting_clause` folgt in `SELECT`-Anweisungen auf die Klausel `ORDER BY`.

Syntax:

```
SELECT ...  
  FROM ...  
  [ WHERE ... ]  
  [ ORDER BY ... ]  
  [OFFSET offset { ROW | ROWS }]  
  [FETCH { FIRST | NEXT } [{ row_count | percent PERCENT  
  }] { ROW | ROWS }  
  { ONLY | WITH TIES }]
```

ORACLE

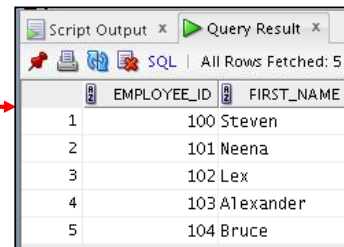
Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die `row_limiting_clause` folgt in `SELECT`-Anweisungen auf die Klausel `ORDER BY`. Beachten Sie, dass die Angabe von `ORDER BY` erforderlich ist, wenn die Zeilen aus Konsistenzgründen sortiert werden sollen.

- **OFFSET:** Mit dieser Klausel geben Sie an, wie viele Zeilen übersprungen werden sollen, bevor die Zeilenbeschränkung beginnt. Der Wert für Offset muss eine Zahl sein. Wenn Sie eine negative Zahl angeben, wird der Offset als 0 behandelt. Wenn Sie `NULL` oder eine Zahl angeben, die mindestens der Anzahl der von der Abfrage zurückgegebenen Zeilen entspricht, werden 0 Zeilen zurückgegeben.
- **ROW | ROWS:** Diese Schlüsselwörter sind austauschbar und werden zusammen aufgeführt, um semantische Klarheit zu schaffen.
- **FETCH:** Mit dieser Klausel geben Sie Anzahl oder Prozentsatz der zurückzugebenden Zeilen an.
- **FIRST | NEXT:** Diese Schlüsselwörter sind austauschbar und werden zusammen aufgeführt, um semantische Klarheit zu schaffen.
- **row\_count | percent PERCENT:** Mit `row_count` geben Sie die Anzahl der zurückzugebenden Zeilen an. Mit `PERCENT` geben Sie an, welcher Anteil der Gesamtzeilenzahl zurückgegeben werden soll. Der Wert für den Prozentsatz muss eine Zahl sein.
- **ONLY | WITH TIES:** Mit `ONLY` geben Sie die Anzahl oder Prozentzahl der zurückzugebenden Zeilen exakt an. Mit `WITH TIES` geben Sie vor, dass zusätzliche Zeilen zurückgegeben werden, die denselben Sortierschlüssel verwenden wie die zuletzt abgerufene Zeile. `WITH TIES` erfordert die Angabe von `order_by_clause`. Andernfalls werden keine zusätzlichen Zeilen zurückgegeben.

## SQL-Klausel zur Zeilenbeschränkung – Beispiele

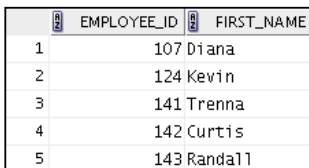
```
SELECT employee_id, first_name
FROM employees
ORDER BY employee_id
FETCH FIRST 5 ROWS ONLY;
```



Script Output x Query Result x  
SQL | All Rows Fetched: 5

	EMPLOYEE_ID	FIRST_NAME
1	100	Steven
2	101	Neena
3	102	Lex
4	103	Alexander
5	104	Bruce

```
SELECT employee_id, first_name
FROM employees
ORDER BY employee_id
OFFSET 5 ROWS FETCH NEXT 5 ROWS ONLY;
```



	EMPLOYEE_ID	FIRST_NAME
1	107	Diana
2	124	Kevin
3	141	Trenna
4	142	Curtis
5	143	Randall

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Das erste Codebeispiel gibt die fünf Mitarbeiter mit der niedrigsten Personalnummer (`employee_id`) zurück.

Das zweite Codebeispiel gibt die auf diese Gruppe folgende Fünfergruppe mit den niedrigsten Personalnummern zurück.

**Hinweis:** Wenn die Personalnummer `employee_id` fortlaufend nach dem Einstellungsdatum des Mitarbeiters zugeordnet wurde, geben diese SELECT-Beispiele nacheinander die zehn Mitarbeiter (1-5 und dann 6-10) mit der längsten Betriebszugehörigkeit aus.

# Lektionsagenda

- Auszugebende Zeilen beschränken mit:
  - der Klausel `WHERE`
  - Vergleichsbedingungen mit den Operatoren `=`, `<=`, `BETWEEN`, `IN`, `LIKE` und `NULL`
  - logischen Bedingungen mit den Operatoren `AND`, `OR` und `NOT`
- Prioritätsregeln für Operatoren in Ausdrücken
- Zeilen mit der Klausel `ORDER BY` sortieren
- SQL-Klausel zur Zeilenbeschränkung in Abfragen
- **Substitutionsvariablen**
- Befehle `DEFINE` und `VERIFY`

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

# Substitutionsvariablen



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Bislang wurden alle SQL-Anweisungen mit vordefinierten Spalten, Bedingungen und ihren Werten ausgeführt. Nun soll eine Abfrage erstellt werden, die nicht nur die Mitarbeiter mit der Tätigkeits-ID `SA_REP` auflistet, sondern Mitarbeiter mit verschiedenen Tätigkeiten. Sie können die Klausel `WHERE` so bearbeiten, dass bei jeder Ausführung des Befehls ein anderer Wert bereitgestellt wird. Es gibt aber auch eine einfachere Methode.

Um die gleiche Abfrage für verschiedene Werte auszuführen, verwenden Sie in der Klausel `WHERE` anstelle der genauen Werte eine Substitutionsvariable.

Sie können Berichte erstellen, die den Benutzer auffordern, eigene Werte anzugeben, um den zurückgegebenen Datenbereich über Substitutionsvariablen einzuschränken. Sie können *Substitutionsvariablen* in einer Befehlsdatei oder einer einzelnen SQL-Anweisung einbetten. Eine Variable ist eine Art Container, in dem die Werte vorübergehend gespeichert werden. Der gespeicherte Wert wird ersetzt, wenn die Anweisung ausgeführt wird.

# Substitutionsvariablen

- Zweck von Substitutionsvariablen:
  - Werte vorübergehend mit den Substitutionsvariablen & und && speichern
- Mit Substitutionsvariablen folgende Elemente ergänzen:
  - WHERE-Bedingungen
  - ORDER BY-Klauseln
  - Spaltenausdrücke
  - Tabellennamen
  - Ganze SELECT-Anweisungen

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit der Substitutionsvariablen & können Sie Werte vorübergehend speichern.

Sie können auch Variablen mit dem Befehl `DEFINE` vordefinieren. `DEFINE` erstellt Werte und weist sie Variablen zu.

## Eingeschränkte Datenbereiche – Beispiele

- Nur die Zahlen für das aktuelle Quartal oder einen festgelegten Zeitraum im Bericht anzeigen
- Nur die Daten im Bericht anzeigen, die für den anfordernden Benutzer relevant sind
- Nur die Mitarbeiter einer bestimmten Abteilung anzeigen

## Andere interaktive Effekte

Interaktive Effekte sind nicht auf die direkte Benutzerinteraktion mit der Klausel `WHERE` beschränkt. Dasselbe Prinzip kann auch verwendet werden, um andere Ziele zu erreichen:

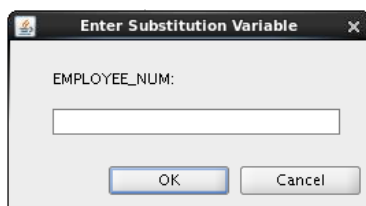
- Eingabewerte aus einer Datei abrufen, statt den Benutzer zur Eingabe aufzufordern
- Werte von einer SQL-Anweisung an eine andere übergeben

**Hinweis:** Sowohl SQL Developer als auch SQL\* Plus unterstützen Substitutionsvariablen und die Befehle `DEFINE`/`UNDEFINE`.

## Substitutionsvariable &

Variable mit einem Et-Zeichen (&) als Präfix verwenden, um den Benutzer zur Eingabe eines Wertes aufzufordern:

```
SELECT employee_id, last_name, salary, department_id
FROM   employees
WHERE  employee_id = &employee_num ;
```



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

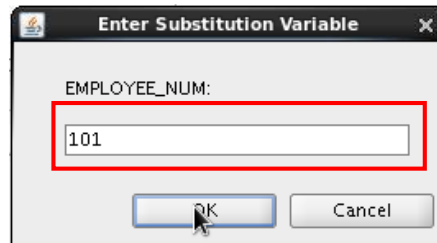
Benutzer, die einen Bericht ausführen, möchten häufig die Menge der dynamisch zurückgegebenen Daten beschränken. SQL\*Plus und SQL Developer bieten diese Flexibilität durch Benutzervariablen. Kennzeichnen Sie alle Variablen in Ihrer SQL-Anweisung mit dem Et-Zeichen (&). Sie müssen jedoch nicht für jede Variable einen Wert definieren.

Notation	Beschreibung
<i>&amp;user_variable</i>	Kennzeichnet eine Variable in einer SQL-Anweisung. Wenn die Variable nicht existiert, fordert SQL*Plus oder SQL Developer den Benutzer auf, einen Wert einzugeben. (Die neue Variable wird nach ihrer Verwendung verworfen.)

Das Beispiel auf der Folie erstellt eine SQL Developer-Substitutionsvariable für eine Personalnummer. Wenn die Anweisung ausgeführt wird, fordert SQL Developer den Benutzer auf, eine Personalnummer einzugeben, und zeigt dann Personalnummer, Nachname, Gehalt und Abteilungsnummer für diesen Mitarbeiter an.

Mit einem einzelnen Et-Zeichen (&) wird der Benutzer bei jeder Ausführung des Befehls aufgefordert, einen Wert einzugeben, falls die Variable nicht vorhanden ist.

## Substitutionsvariable &



	EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
1	101	Kochhar	17000	90

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wenn SQL Developer erkennt, dass die SQL-Anweisung ein Et-Zeichen (&) enthält, werden Sie aufgefordert, einen Wert für die in der SQL-Anweisung genannte Substitutionsvariable einzugeben.

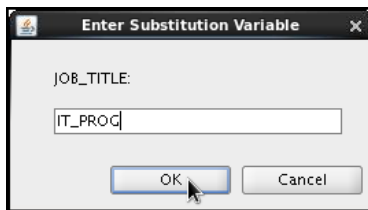
Nachdem Sie einen Wert eingegeben und auf die Schaltfläche **OK** geklickt haben, werden die Ergebnisse angezeigt.



## Zeichen- und Datumswerte mit Substitutionsvariablen

Hochkommas für Datums- und Zeichenwerte verwenden:

```
SELECT last_name, department_id, salary*12
FROM employees
WHERE job_id = '&job_title' ;
```



	LAST_NAME	DEPARTMENT_ID	SALARY*12
1	Hunold	60	108000
2	Ernst	60	72000
3	Lorentz	60	50400

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

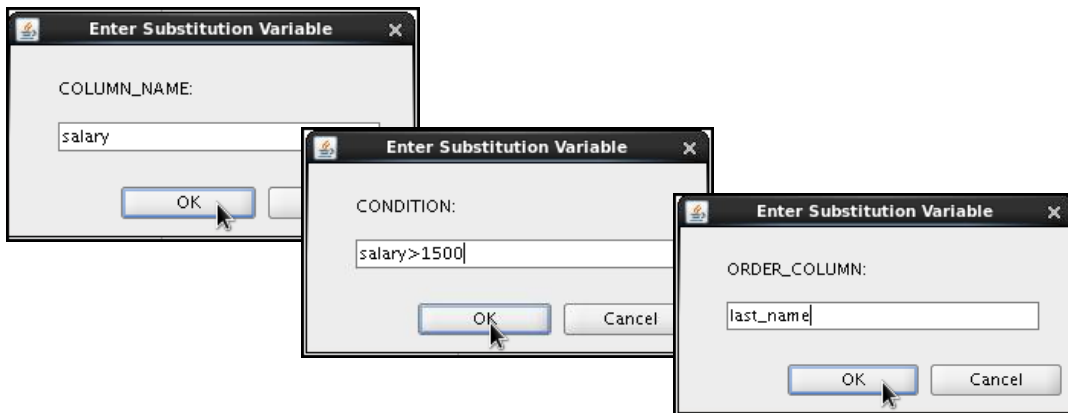
In einer WHERE-Klausel müssen Datums- und Zeichenwerte in Hochkommas gesetzt werden. Dieselbe Regel gilt für Substitutionsvariablen.

Setzen Sie die Variable in der SQL-Anweisung selbst in Hochkommas.

Die Folie zeigt eine Abfrage, die basierend auf dem Wert der Tätigkeits-ID für die SQL Developer-Substitutionsvariable Name, Abteilungsnummer und Jahresgehalt aller entsprechenden Mitarbeiter abrufen.

## Spaltennamen, Ausdrücke und Text angeben

```
SELECT employee_id, last_name, job_id, &column_name  
FROM employees  
WHERE &condition  
ORDER BY &order_column ;
```



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie können die Substitutionsvariablen nicht nur in der Klausel `WHERE` einer SQL-Anweisung verwenden, sondern auch als Ersatz für Spaltennamen, Ausdrücke und Text.

### Beispiel

Im Beispiel auf der Folie werden Personalnummer, Nachname, Tätigkeits-ID und jede weitere durch den Benutzer zur Laufzeit angegebene Spalte aus der Tabelle `EMPLOYEES` angezeigt. Sie werden für jede in der Anweisung `SELECT` enthaltene Substitutionsvariable aufgefordert, einen Wert einzugeben und dann auf die Schaltfläche **OK** zu klicken.

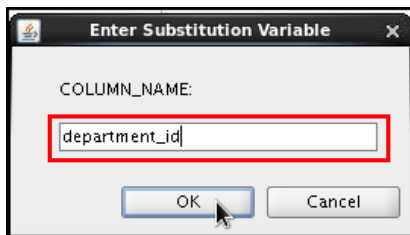
Wenn Sie die voranstehende Anweisung ausführen, ohne einen Wert für die Substitutionsvariablen einzugeben, wird eine Fehlermeldung angezeigt.

**Hinweis:** Substitutionsvariablen können an beliebigen Positionen der `SELECT`-Anweisung verwendet werden, jedoch nicht als erstes Wort an der Eingabeaufforderung.

## Substitutionsvariable &&

Zwei Et-Zeichen (&&) verwenden, wenn der Variablenwert wiederverwendet werden soll, ohne dass der Benutzer jedes Mal zur Eingabe aufgefordert wird:

```
SELECT  employee_id, last_name, job_id, &&column_name
FROM    employees
ORDER BY &column_name ;
```



	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	200	Whalen	AD_ASST	10
2	201	Hartstein	MK_MAN	20
3	202	Fay	MK_REP	20

...

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wenn Sie den Variablenwert wiederverwenden möchten, ohne dass der Benutzer jedes Mal zu einer Eingabe aufgefordert wird, verwenden Sie die Substitutionsvariable &&. Dem Benutzer wird die Eingabeaufforderung für den Wert dann nur einmal angezeigt. Im Beispiel auf der Folie wird der Benutzer nur einmal aufgefordert, den Wert für die Variable `column_name` einzugeben. Der vom Benutzer angegebene Wert (`department_id`) wird zum Anzeigen und Sortieren der Daten verwendet. Wenn Sie die Abfrage erneut ausführen, werden Sie nicht erneut zur Eingabe des Variablenwertes aufgefordert.

SQL Developer speichert den bereitgestellten Wert mit dem Befehl `DEFINE` und zieht ihn bei jeder Referenzierung des Variablennamens erneut heran. Nachdem die Benutzervariable festgelegt wurde, können Sie sie nur mit dem Befehl `UNDEFINE` löschen:

```
UNDEFINE column_name;
```

# Substitutionsvariablen in SQL\*Plus

```
oracle@EDRSR19P1:~/Desktop
File Edit View Search Terminal Help
SQL> SELECT employee_id, last_name, salary, department_id
FROM employees
WHERE employee_id = &employee_num ; 2 3
Enter value for employee num: 101
```

```
oracle@EDRSR19P1:~/Desktop
File Edit View Search Terminal Help
SQL> SELECT employee_id, last_name, salary, department_id
FROM employees
WHERE employee_id = &employee_num ; 2 3
Enter value for employee num: 101
old 3: WHERE employee_id = &employee_num
new 3: WHERE employee_id = 101

EMPLOYEE_ID LAST_NAME          SALARY DEPARTMENT_ID
-----
101 Kochhar          17000          90

SQL>
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Das Beispiel auf der Folie erstellt eine SQL\*Plus-Substitutionsvariable für eine Personalnummer. Wenn die Anweisung ausgeführt wird, fordert SQL\*Plus den Benutzer auf, eine Personalnummer einzugeben, und zeigt dann Personalnummer, Nachname, Gehalt und Abteilungsnummer für diesen Mitarbeiter an.

# Lektionsagenda

- Auszugebende Zeilen beschränken mit:
  - der Klausel `WHERE`
  - Vergleichsbedingungen mit den Operatoren `=`, `<=`, `BETWEEN`, `IN`, `LIKE` und `NULL`
  - logischen Bedingungen mit den Operatoren `AND`, `OR` und `NOT`
- SQL-Klausel zur Zeilenbeschränkung in Abfragen
- Prioritätsregeln für Operatoren in Ausdrücken
- Zeilen mit der Klausel `ORDER BY` sortieren
- Substitutionsvariablen
- **Befehle `DEFINE` und `VERIFY`**

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Befehl DEFINE

- Mit dem Befehl `DEFINE` einen Wert erstellen und diesen einer Variablen zuordnen
- Mit dem Befehl `UNDEFINE` eine Variable entfernen

```
DEFINE employee_num = 200  
  
SELECT employee_id, last_name, salary, department_id  
FROM employees  
WHERE employee_id = &employee_num;  
  
UNDEFINE employee_num
```

	EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
1	200	Whalen	4400	10

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf der Folie wird mit dem Befehl `DEFINE` eine Substitutionsvariable für eine Personalnummer erstellt. Zur Laufzeit werden dann Personalnummer, Name, Gehalt und Abteilungsnummer für den betreffenden Mitarbeiter angezeigt.

Da die Variable mit dem SQL Developer-Befehl `DEFINE` erstellt wird, erhält der Benutzer keine Aufforderung, einen Wert für die Personalnummer einzugeben. Stattdessen wird der definierte Variablenwert in der Anweisung `SELECT` automatisch ersetzt.

Die Substitutionsvariable `EMPLOYEE_NUM` ist in der Session so lange gültig, bis der Benutzer die Definition aufhebt oder die SQL Developer-Session beendet.

## Befehl VERIFY

Mit dem Befehl `VERIFY` die Substitutionsvariablen anzeigen, bevor und nachdem sie von SQL Developer durch Werte ersetzt werden:

```
SET VERIFY ON
SELECT employee_id, last_name, salary
FROM employees
WHERE employee_id = &employee_num;
```

Enter Substitution Variable

EMPLOYEE\_NUM:  
200

OK Cancel

Script Output

Task completed in 6.496 seconds

old:SELECT employee\_id, last\_name, salary  
FROM employees  
WHERE employee\_id = &employee\_num  
new:SELECT employee\_id, last\_name, salary  
FROM employees  
WHERE employee\_id = 200

EMPLOYEE_ID	LAST_NAME	SALARY
200	Whalen	4400

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Um die Änderungen in der SQL-Anweisung zu bestätigen, verwenden Sie den Befehl `VERIFY`. Mit `SET VERIFY ON` wird SQL Developer gezwungen, den Text eines Befehls anzuzeigen, nachdem Substitutionsvariablen durch Werte ersetzt wurden. Um die Ausgabe für `VERIFY` anzuzeigen, verwenden Sie im SQL Worksheet das Symbol **Run Script** (F5). SQL Developer zeigt den Text eines Befehls in der Registerkarte **Script Output** an, nachdem Substitutionsvariablen durch Werte ersetzt wurden (siehe Screenshot auf der Folie).

Das Beispiel auf der Folie zeigt den neuen Wert der Spalte `EMPLOYEE_ID` in der SQL-Anweisung und dahinter die Ausgabe.

### SQL\*Plus-Systemvariablen

SQL\*Plus verwendet zur Steuerung der Arbeitsumgebung verschiedene Systemvariablen. Eine dieser Variablen ist `VERIFY`. Um eine vollständige Liste aller Systemvariablen zu erhalten, geben Sie in der SQL\*Plus-Eingabeaufforderung den Befehl `SHOW ALL` ein.

## Quiz

Welche vier der folgenden Operatoren sind gültige Operatoren für die Klausel `WHERE`?

- a. `>=`
- b. `IS NULL`
- c. `!=`
- d. `IS LIKE`
- e. `IN BETWEEN`
- f. `<>`

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

**Richtige Antwort: a, b, c und f**



## Zusammenfassung

In dieser Lektion haben Sie Folgendes gelernt:

- Von Abfragen ausgegebene Zeilen einschränken
- Von Abfragen ausgegebene Zeilen sortieren
- Ausgabe mithilfe von Substitutionsvariablen (&, &&) zur Laufzeit einschränken und sortieren

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In dieser Lektion haben Sie gelernt, die von `SELECT`-Anweisungen zurückgegebenen Zeilen einzuschränken und zu sortieren. Außerdem wurde die Implementierung verschiedener Operatoren und Bedingungen behandelt.

Mithilfe der Substitutionsvariablen können Sie SQL-Anweisungen flexibler gestalten. Bei Abfragen können Benutzer damit zur Laufzeit aufgefordert werden, die Filterbedingung für die Zeilen anzugeben.

## Übungen zu Lektion 3 – Überblick

Diese Übungen behandeln folgende Themen:

- Daten wählen und Reihenfolge der angezeigten Zeilen ändern
- Zeilen mit der Klausel `WHERE` einschränken
- Zeilen mit der Klausel `ORDER BY` sortieren
- `SELECT`-Anweisungen durch Substitutionsvariablen flexibler gestalten

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In dieser Übung erstellen Sie weitere Berichte mit Anweisungen, die `WHERE`- und `ORDER BY`-Klauseln enthalten. Mithilfe von Substitutionsvariablen gestalten Sie SQL-Anweisungen allgemeiner und erleichtern ihre Wiederverwendung.

# 4

## Ausgabe mit Single-Row-Funktionen anpassen

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- Verschiedene Typen von Funktionen beschreiben, die in SQL verfügbar sind
- Zeichenfunktionen, numerische Funktionen und Datumsfunktionen in `SELECT`-Anweisungen verwenden

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Funktionen erhöhen die Leistungsfähigkeit des Hauptabfrageblocks und werden verwendet, um Datenwerte zu bearbeiten. Dies ist die erste von zwei Lektionen, die sich mit Funktionen befassen. Im Mittelpunkt stehen Single-Row-Funktionen wie Zeichen-, numerische und Datumsfunktionen.

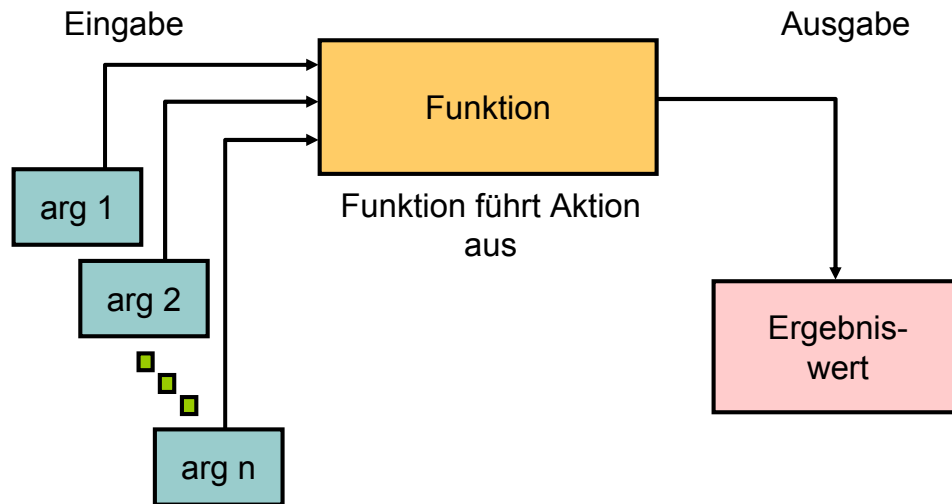
# Lektionsagenda

- Single-Row-SQL-Funktionen
  - Zeichenfunktionen
  - Funktionen verschachteln
  - Numerische Funktionen
  - Mit Datumswerten arbeiten
  - Datumsfunktionen

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

# SQL-Funktionen



ORACLE

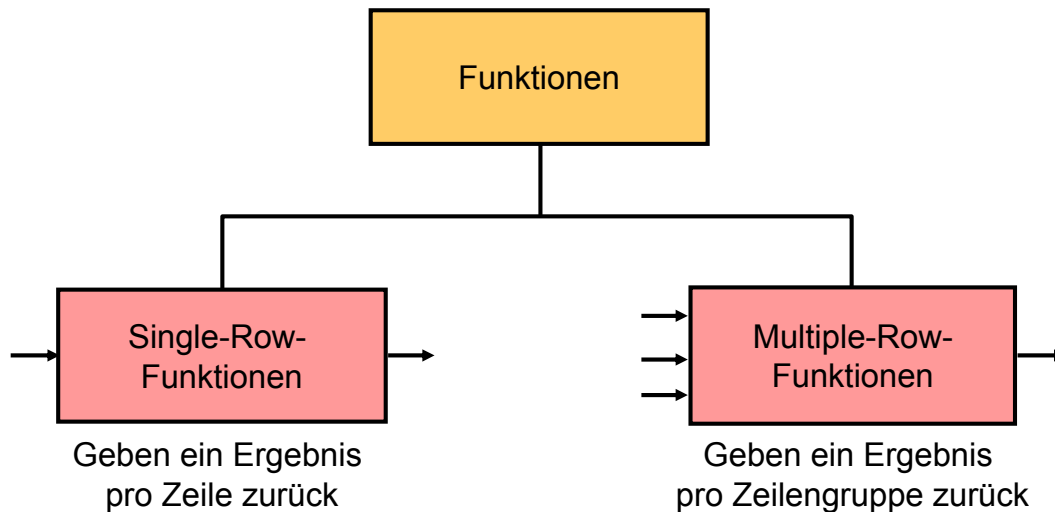
Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Funktionen sind ein äußerst leistungsstarkes Feature von SQL. Sie dienen folgenden Zwecken:

- Berechnungen mit Daten ausführen
- Einzelne Datenelemente ändern
- Ausgabe für Zeilengruppen ändern
- Datumswerte und Zahlen für die Anzeige formatieren
- Datentypen von Spalten konvertieren

SQL-Funktionen enthalten manchmal Argumente und geben immer einen Wert zurück.

# Zwei Typen von SQL-Funktionen



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Es gibt zwei Typen von Funktionen:

- Single-Row-Funktionen
- Multiple-Row-Funktionen

## Single-Row-Funktionen

Diese Funktionen bearbeiten nur einzelne Zeilen und geben ein Ergebnis pro Zeile zurück. Es gibt verschiedene Typen von Single-Row-Funktionen. In dieser Lektion werden die folgenden Funktionen behandelt:

- Zeichenfunktionen
- Numerische Funktionen
- Datumsfunktionen

## Multiple-Row-Funktionen

Diese Funktionen können Zeilengruppen bearbeiten und geben ein Ergebnis pro Zeilengruppe zurück. Sie werden auch als *Gruppenfunktionen* bezeichnet. (Gruppenfunktionen werden in der Lektion "Mit Gruppenfunktionen Berichte aus aggregierten Daten erstellen" behandelt.)

**Hinweis:** Weitere Informationen und eine vollständige Liste der verfügbaren Funktionen und der Syntax finden Sie in der *Oracle Database SQL Language Reference* für Oracle Database 12c im Abschnitt "Functions".

# Single-Row-Funktionen

Single-Row-Funktionen:

- bearbeiten Datenelemente
- akzeptieren Argumente und geben einen Wert zurück
- bearbeiten jede zurückgegebene Zeile
- geben ein Ergebnis pro Zeile zurück
- können den Datentyp ändern
- können verschachtelt werden
- akzeptieren Spalten oder Ausdrücke als Argumente

```
function_name [(arg1, arg2,...)]
```

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Single-Row-Funktionen dienen zum Bearbeiten von Datenelementen. Sie akzeptieren einzelne oder mehrere Argumente und geben für jede durch die Abfrage zurückgegebene Zeile einen Wert zurück. Ein Argument kann Folgendes sein:

- Vom Benutzer angegebene Konstante
- Variablenwert
- Spaltenname
- Ausdruck

Single-Row-Funktionen beinhalten folgende Features:

- Jede von der Abfrage zurückgegebene Zeile wird bearbeitet.
- Pro Zeile wird ein Ergebnis zurückgegeben.
- Der zurückgegebene Datenwert kann einen anderen Datentyp haben als der referenzierte Wert.
- Die Funktionen erwarten möglicherweise einzelne oder mehrere Argumente.
- Sie können in `SELECT`-, `WHERE`- und `ORDER BY`-Klauseln verwendet sowie verschachtelt werden.

Für die Syntax gilt:

*function\_name*

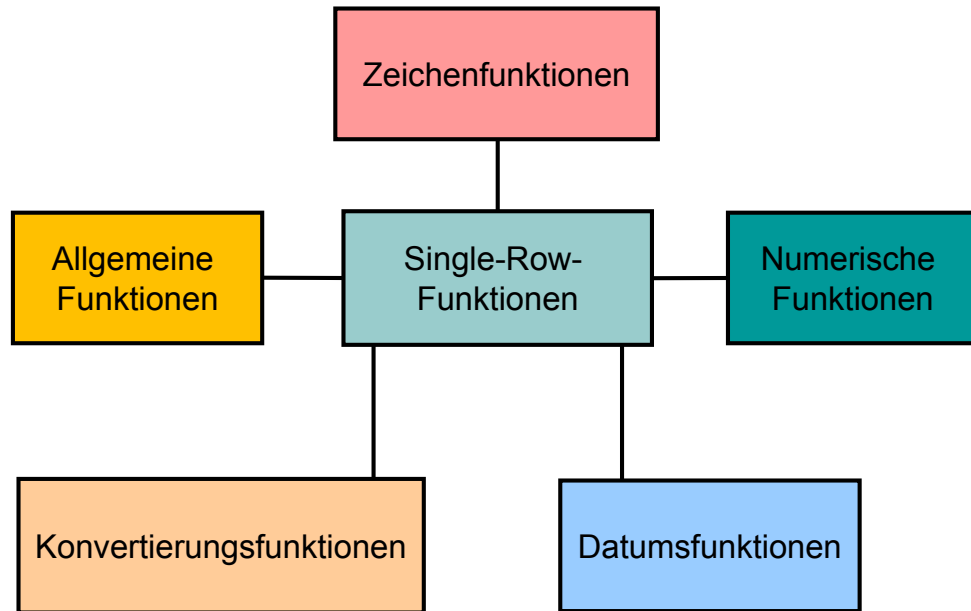
Ist der Name der Funktion

*arg1*, *arg2*

Sind die von der Funktion verwendeten Argumente. Hierbei kann es sich um einen Spaltennamen oder einen Ausdruck handeln.



# Single-Row-Funktionen



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In dieser Lektion werden folgende Single-Row-Funktionen behandelt:

- **Zeichenfunktionen:** Akzeptieren als Eingabe Zeichenwerte und geben Zeichenwerte und numerische Werte zurück
- **Numerische Funktionen:** Akzeptieren als Eingabe numerische Werte und geben numerische Werte zurück
- **Datumsfunktionen:** Bearbeiten Werte vom Datentyp `DATE`

In der Lektion "Konvertierungsfunktionen und bedingte Ausdrücke" werden folgende Single-Row-Funktionen behandelt:

- **Konvertierungsfunktionen:** Konvertieren einen Wert von einem Datentyp in einen anderen
- **Allgemeine Funktionen:** Diese Funktionen nehmen alle Datentypen an und können auch Nullwerte verarbeiten.

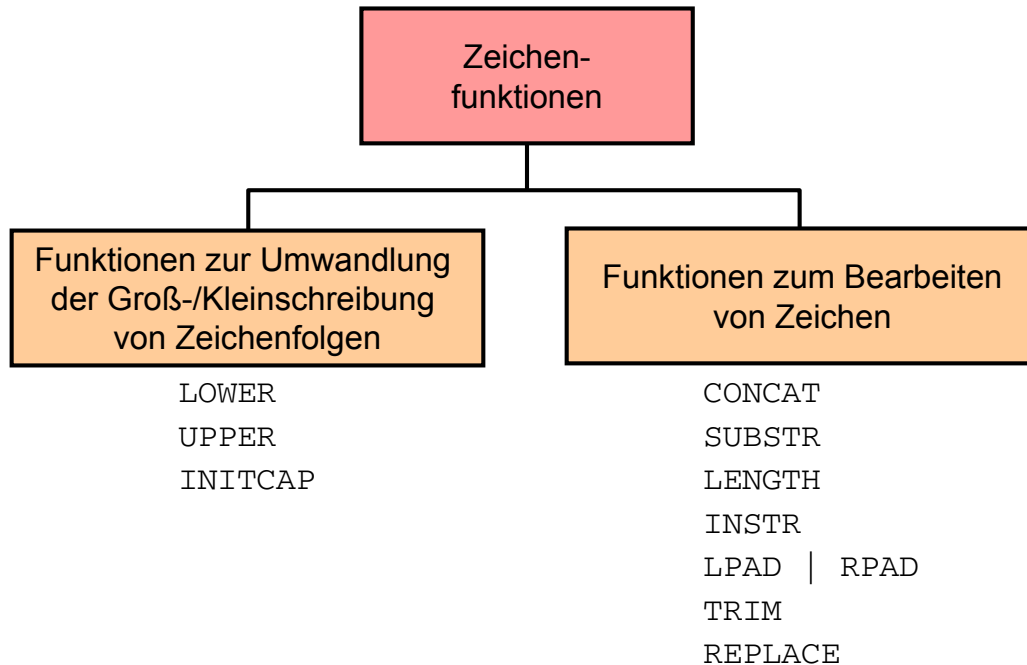
# Lektionsagenda

- Single-Row-SQL-Funktionen
- **Zeichenfunktionen**
- Funktionen verschachteln
- Numerische Funktionen
- Mit Datumswerten arbeiten
- Datumsfunktionen

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

# Zeichenfunktionen



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Single-Row-Zeichenfunktionen akzeptieren Zeichendaten als Eingabe und geben Zeichenwerte oder numerische Werte zurück. Zeichenfunktionen lassen sich folgendermaßen unterteilen:

- Funktionen zur Umwandlung der Groß-/Kleinschreibung von Zeichenfolgen
- Funktionen zum Bearbeiten von Zeichen

Funktion	Zweck
<code>LOWER(column   expression)</code>	Konvertiert alphanumerische Zeichenwerte in Kleinbuchstaben
<code>UPPER(column   expression)</code>	Konvertiert alphanumerische Zeichenwerte in Großbuchstaben
<code>INITCAP(column   expression)</code>	Konvertiert alphanumerische Zeichenwerte in Großschreibung, wobei der erste Buchstabe jedes Wortes groß- und alle anderen Buchstaben des Wortes kleingeschrieben werden
<code>CONCAT(column1   expression1, column2   expression2)</code>	Verkettet den ersten Zeichenwert mit dem zweiten Zeichenwert. Entspricht dem Verkettungsoperator (  )

**Hinweis:** In dieser Lektion werden nur einige der verfügbaren Funktionen behandelt.

Funktion	Zweck
<code>SUBSTR(column expression,m[,n])</code>	Gibt bestimmte Zeichen aus einem Zeichenwert zurück. Die zurückgegebene Teilzeichenfolge beginnt bei der Zeichenposition <i>m</i> und ist <i>n</i> Zeichen lang. (Wenn <i>m</i> negativ ist, wird am Ende des Zeichenwertes mit der Zählung begonnen. Wird <i>n</i> nicht angegeben, werden alle Zeichen bis zum Ende der Zeichenfolge zurückgegeben.)
<code>LENGTH(column expression)</code>	Gibt die Anzahl der Zeichen im Ausdruck zurück
<code>INSTR(column expression, 'string', [,m], [n] )</code>	Gibt die numerische Position einer benannten Zeichenfolge zurück. Optional können Sie mit <i>m</i> die Position angeben, an der die Suche beginnt, und mit <i>n</i> festlegen, welches Vorkommen der Zeichenfolge angezeigt werden soll. Standardmäßig haben <i>m</i> und <i>n</i> den Wert 1. Das bedeutet, die Suche beginnt am Anfang, und das erste Vorkommen der Zeichenfolge wird zurückgegeben.
<code>LPAD(column expression, n, 'string')</code> <code>RPAD(column expression, n, 'string')</code>	Gibt einen Ausdruck zurück, der bis zur Länge von <i>n</i> Zeichen linksbündig mit einem Zeichenausdruck aufgefüllt ist. Gibt einen Ausdruck zurück, der bis zur Länge von <i>n</i> Zeichen rechtsbündig mit einem Zeichenausdruck aufgefüllt ist.
<code>TRIM(leading trailing both, trim_character FROM trim_source)</code>	Ermöglicht es Ihnen, Zeichen am Anfang und/oder am Ende einer Zeichenfolge abzuschneiden. Wenn <i>trim_character</i> oder <i>trim_source</i> ein Zeichenliteral ist, müssen Sie es in Hochkommas setzen.
<code>REPLACE(text, search_string, replacement_string)</code>	Sucht in einem Textausdruck nach einer Zeichenfolge und ersetzt sie durch eine angegebene Ersatzzeichenfolge

## Funktionen zur Umwandlung der Groß-/Kleinschreibung von Zeichenfolgen

Diese Funktionen wandeln die Groß-/Kleinschreibung von Zeichenfolgen um:

Funktion	Ergebnis
LOWER('SQL Course')	sql course
UPPER('SQL Course')	SQL COURSE
INITCAP('SQL Course')	Sql Course

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

LOWER, UPPER und INITCAP sind die drei Funktionen zur Umwandlung der Groß-/Kleinschreibung von Zeichenfolgen.

- LOWER: Konvertiert Zeichenfolgen mit gemischter Groß- und Kleinschreibung bzw. Großschreibung in Kleinschreibung
- UPPER: Konvertiert Zeichenfolgen mit gemischter Groß- und Kleinschreibung bzw. Kleinschreibung in Großschreibung
- INITCAP: Konvertiert den ersten Buchstaben jedes Wortes in einen Großbuchstaben und alle anderen Buchstaben des Wortes in Kleinbuchstaben

```
SELECT 'The job id for '||UPPER(last_name)||' is '  
      ||LOWER(job_id) AS "EMPLOYEE DETAILS"  
FROM   employees;
```

## Funktionen zur Umwandlung der Groß-/ Kleinschreibung von Zeichenfolgen – Beispiel

Personalnummer, Namen und Abteilungsnummer für den Mitarbeiter Higgins anzeigen:

```
SELECT employee_id, last_name, department_id
FROM   employees
WHERE  last_name = 'higgins';
```

0 rows selected

```
SELECT employee_id, last_name, department_id
FROM   employees
WHERE  LOWER(last_name) = 'higgins';
```

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	205	Higgins	110

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf der Folie werden Personalnummer, Name und Abteilungsnummer des Mitarbeiters Higgins angezeigt.

Die WHERE-Klausel der ersten SQL-Anweisung gibt als Namen des Mitarbeiters `higgins` an. Da alle Daten in der Tabelle `EMPLOYEES` so gespeichert sind, dass jedes Wort mit einem Großbuchstaben beginnt, wird für den Namen `higgins` in der Tabelle kein Wert gefunden. Es werden keine Zeilen ausgewählt.

Die WHERE-Klausel der zweiten SQL-Anweisung gibt an, dass der Name des Mitarbeiters in der Tabelle `EMPLOYEES` mit `higgins` verglichen wird, wobei die Werte der Spalte `LAST_NAME` zu Vergleichszwecken in Kleinbuchstaben konvertiert werden. Da jetzt beide Namen in Kleinschreibung vorliegen, wird eine Übereinstimmung gefunden. Eine Zeile wird ausgewählt. Sie können die WHERE-Klausel auch folgendermaßen umschreiben, um dasselbe Ergebnis zu erzielen:

```
...WHERE last_name = 'Higgins'
```

Der Name wird in der Ausgabe so angezeigt, wie er in der Datenbank gespeichert ist. Um den Namen in Großbuchstaben anzuzeigen, verwenden Sie in der Anweisung `SELECT` die Funktion `UPPER`.

```
SELECT employee_id, UPPER(last_name), department_id
FROM   employees
WHERE  INITCAP(last_name) = 'Higgins'
```

# Funktionen zum Bearbeiten von Zeichen

Diese Funktionen bearbeiten Zeichenfolgen:

Funktion	Ergebnis
<code>CONCAT('Hello', 'World')</code>	HelloWorld
<code>SUBSTR('HelloWorld',1,5)</code>	Hello
<code>LENGTH('HelloWorld')</code>	10
<code>INSTR('HelloWorld', 'W')</code>	6
<code>LPAD(last_name,12, '-')</code>	*****24000
<code>RPAD(first_name, 12, '-')</code>	24000*****

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In dieser Lektion werden folgende Funktionen zum Bearbeiten von Zeichen behandelt: `CONCAT`, `SUBSTR`, `LENGTH`, `INSTR`, `LPAD` und `RPAD`.

- `CONCAT`: Verknüpft Werte (Sie können mit `CONCAT` maximal zwei Parameter verknüpfen.)
- `SUBSTR`: Extrahiert eine Zeichenfolge bestimmter Länge
- `LENGTH`: Zeigt die Länge einer Zeichenfolge als numerischen Wert an
- `INSTR`: Ermittelt die numerische Position eines benannten Zeichens
- `LPAD`: Gibt einen Ausdruck zurück, der bis zur Länge von *n* Zeichen linksbündig mit einem Zeichenausdruck aufgefüllt ist
- `RPAD`: Gibt einen Ausdruck zurück, der bis zur Länge von *n* Zeichen rechtsbündig mit einem Zeichenausdruck aufgefüllt ist

**Hinweis:** Sie können auch Funktionen wie `UPPER` und `LOWER` mit einer Substitutionsvariablen (&,&&) verwenden. Beispiel: Wenn Sie `UPPER('&job_title')` angeben, muss der Benutzer die Tätigkeitsbezeichnung nicht in einer bestimmten Schreibweise eingeben.

## Funktionen zum Bearbeiten von Zeichen – Beispiel

**1**

```
SELECT CONCAT(CONCAT(last_name, ''s job category is '), job_id)
"Job" FROM employees
WHERE SUBSTR(job_id, 4) = 'REP';
```

Job
1 Abel's job category is SA_REP
2 Fay's job category is MK_REP
3 Grant's job category is SA_REP
4 Taylor's job category is SA_REP

**2**

```
SELECT employee_id, CONCAT(first_name, last_name) NAME,
LENGTH (last_name), INSTR(last_name, 'a') "Contains 'a'?"
FROM employees
WHERE SUBSTR(last_name, -1, 1) = 'n';
```

EMPLOYEE_ID	NAME	LENGTH(LAST_NAME)	Contains 'a'?
1	102 LexDe Haan	7	5
2	200 JenniferWhalen	6	3
3	201 MichaelHartstein	9	2

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im ersten Beispiel auf der Folie werden Verknüpfungen aus dem Nachnamen und der Tätigkeits-ID für alle Mitarbeiter angezeigt, in deren Tätigkeits-ID beginnend ab der vierten Position die Zeichenfolge REP enthalten ist.

Die zweite SQL-Anweisung auf der Folie zeigt die Daten für alle Mitarbeiter an, deren Nachname auf den Buchstaben "n" endet.



# Lektionsagenda

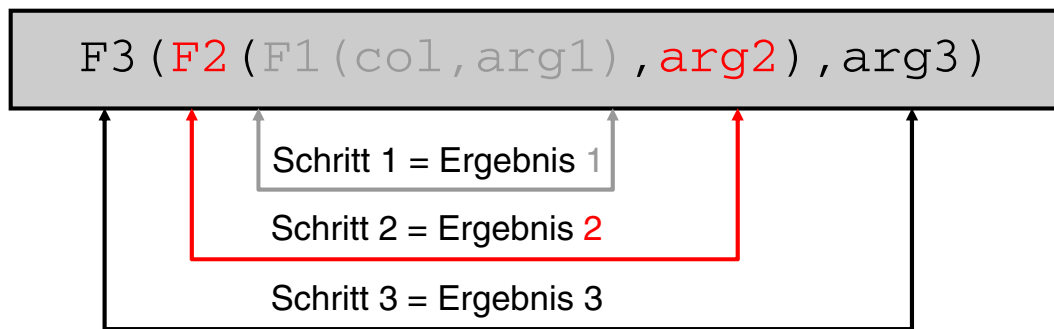
- Single-Row-SQL-Funktionen
- Zeichenfunktionen
- **Funktionen verschachteln**
- Numerische Funktionen
- Mit Datumswerten arbeiten
- Datumsfunktionen

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Funktionen verschachteln

- Single-Row-Funktionen können in beliebig viele Ebenen verschachtelt werden.
- Verschachtelte Funktionen werden von innen nach außen ausgewertet.



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Single-Row-Funktionen können in beliebig viele Ebenen verschachtelt werden. Verschachtelte Funktionen werden von innen nach außen ausgewertet. Das folgende Beispiel demonstriert die Flexibilität dieser Funktionen.

## Funktionen verschachteln – Beispiel

```
SELECT last_name,  
       UPPER(CONCAT(SUBSTR (LAST_NAME, 1, 8), '_US'))  
FROM   employees  
WHERE  department_id = 60;
```

	LAST_NAME	UPPER(CONCAT(SUBSTR(LAST_NAME,1,8),'_US'))
1	Hunold	HUNOLD_US
2	Ernst	ERNST_US
3	Lorentz	LORENTZ_US

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Das Beispiel auf der Folie zeigt die Nachnamen der Mitarbeiter aus Abteilung 60 an. Die Auswertung der SQL-Anweisung umfasst drei Schritte:

1. Die innere Funktion ruft die ersten acht Zeichen des Nachnamens ab.

Result1 = SUBSTR (LAST\_NAME, 1, 8)

2. Die äußere Funktion verkettet das Ergebnis mit \_US.

Result2 = CONCAT(Result1, '\_US')

3. Die äußerste Funktion konvertiert die Ergebnisse in Großschreibung.

Der gesamte Ausdruck wird zur Spaltenüberschrift, weil kein Spaltenalias angegeben wurde.

# Lektionsagenda

- Single-Row-SQL-Funktionen
- Zeichenfunktionen
- Funktionen verschachteln
- **Numerische Funktionen**
- Mit Datumswerten arbeiten
- Datumsfunktionen

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Numerische Funktionen

- **ROUND:** Rundet einen Wert auf eine bestimmte Dezimalstelle
- **TRUNC:** Schneidet einen Wert bis zu einer bestimmten Dezimalstelle ab
- **CEIL:** Gibt die kleinste Ganzzahl zurück, die größer oder gleich einer angegebenen Zahl ist
- **FLOOR:** Gibt die größte Ganzzahl zurück, die kleiner oder gleich einer angegebenen Zahl ist
- **MOD:** Gibt den Rest einer Division zurück

Funktion	Ergebnis
ROUND (45.926, 2)	45.93
TRUNC (45.926, 2)	45.92
CEIL (2.83)	3
FLOOR (2.83)	2
MOD (1600, 300)	100

**ORACLE**

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

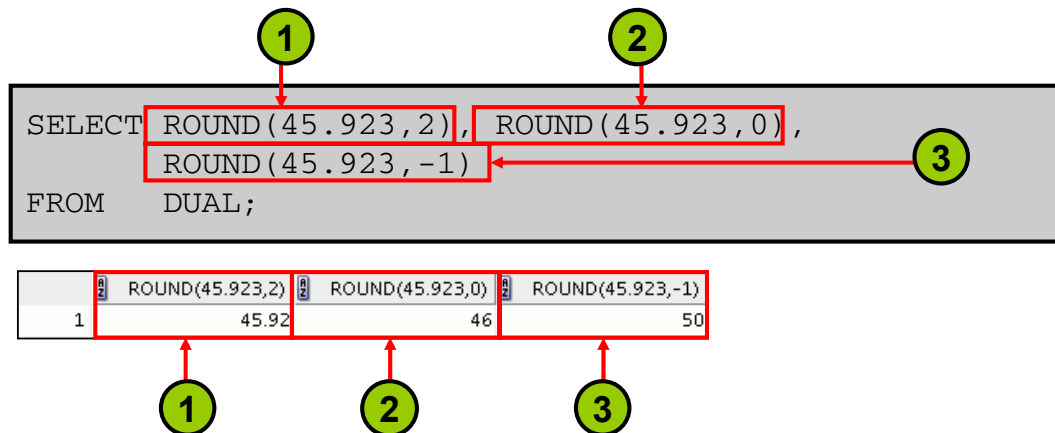
Numerische Funktionen akzeptieren numerische Eingabewerte und geben numerische Werte zurück. In diesem Abschnitt werden einige der numerischen Funktionen beschrieben.

Funktion	Zweck
ROUND ( <i>column</i>   <i>expression</i> , <i>n</i> )	Rundet die Spalte, den Ausdruck oder den Wert auf <i>n</i> Dezimalstellen. Ist kein Wert für <i>n</i> angegeben, wird auf einen ganzzahligen Wert gerundet. (Ist <i>n</i> negativ, wird die Zahl links vom Dezimalkomma gerundet.)
TRUNC ( <i>column</i>   <i>expression</i> , <i>n</i> )	Schneidet die Spalte, den Ausdruck oder den Wert auf <i>n</i> Dezimalstellen ab. Ist kein Wert für <i>n</i> angegeben, werden die Ziffern hinter dem Dezimalkomma abgeschnitten.
MOD ( <i>m</i> , <i>n</i> )	Gibt den Rest von <i>m</i> geteilt durch <i>n</i> zurück

**Hinweis:** Diese Liste enthält nur einige der verfügbaren numerischen Funktionen.

Weitere Informationen finden Sie in der *Oracle Database SQL Language Reference* für Oracle Database 12c im Abschnitt "Numeric Functions".

## Funktion ROUND



DUAL ist eine öffentliche Tabelle, in der Ergebnisse aus Funktionen und Berechnungen angezeigt werden.

ORACLE

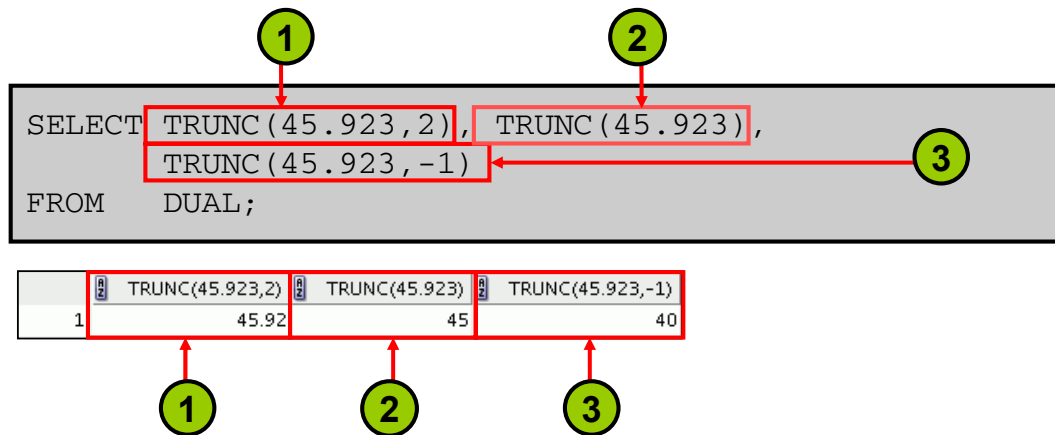
Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Funktion `ROUND` rundet eine Spalte, einen Ausdruck oder einen Wert auf  $n$  Dezimalstellen. Wenn das zweite Argument "0" ist oder kein Argument angegeben ist, wird der Wert auf null Dezimalstellen gerundet. Wenn das zweite Argument "2" ist, wird der Wert auf zwei Dezimalstellen nach dem Komma gerundet. Wenn das zweite Argument "-2" ist, wird der Wert auf die zweite Stelle vor dem Komma gerundet (auf den nächsten Zehnerwert).

### Tabelle DUAL

Eigentümer der Tabelle `DUAL` ist der Benutzer `SYS`. Alle Benutzer können jedoch auf die Tabelle zugreifen. Sie enthält die Spalte `DUMMY` und eine Zeile mit dem Wert `X`. Die Tabelle `DUAL` ist nützlich, wenn ein Wert (z. B. der Wert einer Konstanten, einer Pseudospalte oder eines Ausdrucks, der nicht aus einer Tabelle mit Benutzerdaten abgeleitet ist) nur ein einziges Mal zurückgegeben werden soll. Die Tabelle `DUAL` wird normalerweise verwendet, um die Vollständigkeit der Syntax von `SELECT`-Klauseln zu testen, weil `SELECT`- und `FROM`-Klauseln obligatorisch sind und verschiedene Berechnungen keine Werte aus echten Tabellen benötigen.

## Funktion TRUNC



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Funktion `TRUNC` schneidet eine Spalte, einen Ausdruck oder einen Wert auf  $n$  Dezimalstellen ab.

Bei der Funktion `TRUNC` werden ähnliche Argumente wie bei der Funktion `ROUND` verwendet. Wenn das zweite Argument "0" ist oder kein Argument angegeben wurde, werden die Ziffern hinter dem Dezimalkomma abgeschnitten. Wenn das zweite Argument "2" ist, wird der Wert nach der zweiten Dezimalstelle abgeschnitten. Ist das zweite Argument "-2", wird der Wert zwei Stellen vor dem Komma abgeschnitten. Wenn das zweite Argument "-1" lautet, wird der Wert eine Stelle vor dem Komma abgeschnitten.

## Funktion MOD

Alle Mitarbeiterdatensätze anzeigen, in denen die `employee_id` aus einer geraden Zahl besteht

```
SELECT employee_id as "Even Numbers", last_name  
FROM employees  
WHERE MOD(employee_id,2) = 0;
```

	Even Numbers	LAST_NAME
1	174	Abel
2	142	Davies
3	102	De Haan
4	104	Ernst
5	202	Fay
6	206	Gietz
7	178	Grant
8	100	King
9	124	Mourgos
10	176	Taylor
11	144	Vargas
12	200	Whalen

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Funktion `MOD` berechnet den Rest des ersten Arguments geteilt durch das zweite Argument. Das Beispiel auf der Folie zeigt alle Mitarbeiterdatensätze, in denen die `employee_id` aus einer geraden Zahl besteht.

**Hinweis:** Die Funktion `MOD` wird häufig verwendet, um zu ermitteln, ob ein Wert gerade oder ungerade ist.



# Lektionsagenda

- Single-Row-SQL-Funktionen
- Zeichenfunktionen
- Funktionen verschachteln
- Numerische Funktionen
- **Mit Datumswerten arbeiten**
- Datumsfunktionen

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Mit Datumswerten arbeiten

- Oracle Database speichert Datumswerte in einem internen numerischen Format: Jahrhundert, Jahr, Monat, Tag, Stunden, Minuten und Sekunden
- Das Standardanzeigeformat lautet DD-MON-RR.
  - Um Datumsangaben des 21. Jahrhunderts im 20. Jahrhundert zu speichern, reicht die Angabe der letzten beiden Ziffern der Jahreszahl.
  - Dies gilt auch umgekehrt, wenn Sie ein Datum aus dem 20. Jahrhundert im 21. Jahrhundert speichern.

```
SELECT last_name, hire_date  
FROM employees  
WHERE hire_date < '01-FEB-2008';
```

	LAST_NAME	HIRE_DATE
1	King	17-JUN-03
2	Kochhar	21-SEP-05

...

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Oracle Database speichert Datumswerte in einem internen numerischen Format, mit dem Jahrhundert, Jahr, Monat, Tag, Stunden, Minuten und Sekunden dargestellt werden.

Das Standardformat für die Anzeige und Eingabe eines Datums lautet DD-MON-RR. Gültige Oracle-Datumswerte liegen zwischen dem 1. Januar 4712 v. Chr. und dem 31. Dezember 9999 n. Chr.

Im Beispiel auf der Folie wird die Ausgabe der Spalte `HIRE_DATE` im Standardformat DD-MON-RR angezeigt. Datumswerte werden jedoch in der Datenbank nicht in diesem Format gespeichert. Stattdessen werden alle Komponenten von Datum und Uhrzeit gespeichert. Das Einstellungsdatum 17-JUN-03 wird also in der Form Tag, Monat und Jahr angezeigt, gespeichert werden jedoch auch die zugehörige *Uhrzeit* und das *Jahrhundert*. Das vollständige Datum wäre also June 17, 2003, 05:10:43 PM.

## Datumsformat RR

Aktuelles Jahr	Angegebenes Datum	Format RR	Format YY
1995	27-OCT-95	1995	1995
1995	27-OCT-17	2017	1917
2001	27-OCT-17	2017	2017
2001	27-OCT-95	1995	2095

		Wird die folgende zweistellige Jahreszahl eingegeben:	
		0–49	50–99
Lauten die letzten beiden Ziffern des aktuellen Jahres:	0–49	liegt das zurückgegebene Datum im aktuellen Jahrhundert	liegt das zurückgegebene Datum im Jahrhundert vor dem aktuellen Jahrhundert
	50–99	liegt das zurückgegebene Datum im Jahrhundert nach dem aktuellen Jahrhundert	liegt das zurückgegebene Datum im aktuellen Jahrhundert

**ORACLE**

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Das Datumsformat **RR** ähnelt dem Element **YY**, ermöglicht jedoch die Angabe unterschiedlicher Jahrhunderte. Wenn Sie das Datumsformatelement **RR** anstelle von **YY** verwenden, wird das Jahrhundert des Rückgabewertes in Abhängigkeit von der angegebenen zweistelligen Jahreszahl und den letzten beiden Ziffern des aktuellen Jahres gewählt. Die Tabelle auf der Folie gibt einen Überblick über das Verhalten des Elements **RR**.

Aktuelles Jahr	Angegebenes Datum	Rückgabewert (RR)	Rückgabewert (YY)
1994	27-OCT-95	1995	1995
1994	27-OCT-17	2017	1917
2001	27-OCT-17	2017	2017
2048	27-OCT-52	1952	2052
2051	27-OCT-47	2147	2047

Beachten Sie in der Tabelle oben die Werte in den letzten beiden Zeilen.

Diese Daten werden intern folgendermaßen gespeichert:

CENTURY	YEAR	MONTH	DAY	HOUR	MINUTE	SECOND
19	03	06	17	17	10	43

### **Jahrhunderte und das Jahr 2000**

Wenn Sie einen Datensatz mit einer Datumsspalte in eine Tabelle einfügen, werden die Informationen über das *Jahrhundert* aus der Funktion `SYSDATE` übernommen. Bei der Anzeige der Datumsspalte auf dem Bildschirm wird jedoch das Jahrhundert standardmäßig nicht angezeigt.

Der Datentyp `DATE` verwendet zwei Byte für die Angabe des Jahres, ein Byte für das Jahrhundert und ein Byte für das Jahr. Der Wert für das Jahrhundert wird stets eingefügt, unabhängig davon, ob er angegeben oder angezeigt wird. In diesem Fall bestimmt `RR` den Standardwert für das Jahrhundert beim `INSERT`.

## Funktion SYSDATE

Die Funktion SYSDATE gibt folgende Werte zurück:

- Datum
- Uhrzeit

```
SELECT sysdate
FROM dual;
```

	SYSDATE
1	24-AUG-12

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

SYSDATE ist eine Datumsfunktion, die das Systemdatum zurückgibt. Sie verwenden SYSDATE wie einen Spaltennamen. Sie können beispielsweise das Systemdatum anzeigen, indem Sie SYSDATE aus einer Tabelle wählen. SYSDATE wird normalerweise über eine öffentliche Tabelle mit dem Namen DUAL gewählt.

**Hinweis:** SYSDATE gibt das aktuelle Datum und die Uhrzeit zurück, die für das Betriebssystem eingestellt sind, auf dem sich die Datenbank befindet. Wenn Sie also an einem Rechner in Australien arbeiten und mit einer Remote-Datenbank in den Vereinigten Staaten (USA) verbunden sind, gibt die Funktion SYSDATE das Datum und die Uhrzeit für die Vereinigten Staaten zurück. In diesem Fall können Sie die Funktion CURRENT\_DATE verwenden, die das aktuelle Datum in der Zeitzone der Session zurückgibt.

## Funktionen CURRENT\_DATE und CURRENT\_TIMESTAMP

- CURRENT\_DATE gibt das aktuelle Datum aus der Benutzersession zurück.

```
SELECT SESSIONTIMEZONE, CURRENT_DATE FROM DUAL;
```

SESSIONTIMEZONE	CURRENT_DATE
1 Etc/Universal	26-MAY-14

- CURRENT\_TIMESTAMP gibt das aktuelle Datum und die Uhrzeit aus der Benutzersession zurück.

```
SELECT SESSIONTIMEZONE, CURRENT_TIMESTAMP FROM DUAL;
```

SESSIONTIMEZONE	CURRENT_TIMESTAMP
1 Etc/Universal	26-MAY-14 12.25.34.401622000 AM ETC/UNIVERSAL

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Funktionen CURRENT\_DATE und CURRENT\_TIMESTAMP geben das aktuelle Datum bzw. den aktuellen Zeitstempel zurück.

**Hinweis:** Die Funktion SESSIONTIMEZONE gibt den Wert der Zeitzone der aktuellen Session zurück. Der Rückgabebetyp ist eine Zeitzonendifferenz (ein Zeichentyp im Format ' [+ | ] TZH:TZM' ) oder der Name einer Zeitonenregion, je nachdem, wie der Benutzer den Wert der Zeitzone der Session in der letzten Anweisung ALTER SESSION angegeben hat. Das Beispiel auf der Folie zeigt, dass die Sessionzeitzone eine Differenz von -5 Stunden gegenüber UTC hat. Die Datenbankzeitzone unterscheidet sich von der Zeitzone der aktuellen Session.

## Mit Datumswerten rechnen

- Eine Zahl zu einem Datum addieren oder davon subtrahieren, um einen anderen Datumswert zu erhalten
- Zwei Datumswerte voneinander subtrahieren, um die Anzahl der dazwischenliegenden Tage zu ermitteln
- Stunden zu einem Datum addieren, indem die Anzahl der Stunden durch 24 dividiert wird

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Da die Datenbank Datumswerte als Zahlen speichert, können Sie mit den arithmetischen Operatoren Berechnungen wie Addition und Subtraktion durchführen. Sie können numerische Konstanten und Datumswerte addieren und subtrahieren.

Folgende Vorgänge sind möglich:

Operation	Ergebnis	Beschreibung
Datum + Zahl	Datum	Addiert eine Anzahl von Tagen zu einem Datum
Datum - Zahl	Datum	Subtrahiert ein Datum von einem anderen Datum
Datum - Datum	Anzahl von Tagen	Subtrahiert ein Datum von einem anderen Datum
Datum + Zahl/24	Datum	Addiert eine Anzahl von Stunden zu einem Datum

## Arithmetische Operatoren für Datumswerte

```
SELECT last_name, (SYSDATE-hire_date)/7 AS WEEKS
FROM   employees
WHERE  department_id = 90;
```

	LAST_NAME	WEEKS
1	King	478.871917989417989417989417989418
2	Kochhar	360.729060846560846560846560846561
3	De Haan	605.300489417989417989417989417989

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Das Beispiel auf der Folie zeigt für alle Mitarbeiter aus Abteilung 90 den Nachnamen und die Beschäftigungsdauer in Wochen an. Es subtrahiert das Datum, an dem der Mitarbeiter eingestellt wurde, vom aktuellen Datum (`SYSDATE`) und dividiert das Ergebnis durch 7. Auf diese Weise wird berechnet, seit wie vielen Wochen der jeweilige Mitarbeiter angestellt ist.



# Lektionsagenda

- Single-Row-SQL-Funktionen
- Zeichenfunktionen
- Funktionen verschachteln
- Numerische Funktionen
- Mit Datumswerten arbeiten
- **Datumsfunktionen**

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

# Funktionen zum Bearbeiten von Datumswerten

Funktion	Ergebnis
MONTHS_BETWEEN	Anzahl der Monate zwischen zwei Datumswerten
ADD_MONTHS	Kalendermonate zu einem Datum addieren
NEXT_DAY	Wochentag des angegebenen Datums
LAST_DAY	Letzter Tag des Monats
ROUND	Datumswert runden
TRUNC	Datumswert abschneiden

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Datumsfunktionen bearbeiten Oracle-Datumswerte. Alle Datumsfunktionen geben einen Wert vom Datentyp `DATE` zurück. Ausnahme: Die Funktion `MONTHS_BETWEEN` gibt einen numerischen Wert zurück.

- `MONTHS_BETWEEN(date1, date2)`: Berechnet die Anzahl der Monate zwischen `date1` und `date2`. Das Ergebnis kann positiv oder negativ sein. Ist `date1` ein späteres Datum als `date2`, dann ist das Ergebnis positiv. Wenn `date1` vor `date2` liegt, ist das Ergebnis negativ. Der nicht ganzzahlige Teil des Ergebnisses steht für den anteiligen Monat.
- `ADD_MONTHS(date, n)`: Addiert `n` Kalendermonate zu `date`. Der Wert von `n` muss eine Ganzzahl sein. Diese kann negativ sein.
- `NEXT_DAY(date, 'char')`: Sucht das erste Auftreten des angegebenen Wochentages (`'char'`) nach `date` und gibt dessen Datum zurück. Der Wert von `char` kann eine Zahl oder eine Zeichenfolge sein, die für einen Tag steht.
- `LAST_DAY(date)`: Sucht das Datum des letzten Tages des Monats, das `date` enthält.

Die vorstehende Liste enthält nur einige der verfügbaren Datumsfunktionen. Sie können Datumswerte auch wie folgt mit den numerischen Funktionen `ROUND` und `TRUNC` bearbeiten:

- `ROUND(date[, 'fmt'])`: Gibt `date` gerundet auf die durch die Formatmaske `fmt` angegebene Einheit zurück. Wird die Formatmaske `fmt` nicht angegeben, dann wird `date` auf den nächstgelegenen Tag auf- oder abgerundet.
- `TRUNC(date[, 'fmt'])`: Gibt `date` zurück und schneidet die Uhrzeitkomponente des Tages auf die durch die Formatmaske `fmt` angegebene Einheit ab. Wird die Formatmaske `fmt` nicht angegeben, dann wird `date` auf den nächstgelegenen Tag abgeschnitten.

Formatmasken werden in der Lektion "Konvertierungsfunktionen und bedingte Ausdrücke" ausführlich behandelt.

# Datumsfunktionen

Funktion	Ergebnis
MONTHS_BETWEEN ( '01-SEP-05' , '11-JAN-04' )	19.6774194
ADD_MONTHS ( '31-JAN-04' , 1 )	'29-FEB-04 '
NEXT_DAY ( '01-SEP-05' , 'FRIDAY' )	'08-SEP-05 '
LAST_DAY ( '01-FEB-05' )	'28-FEB-05 '

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf der Folie fügt die Funktion `ADD_MONTHS` dem bereitgestellten Datumswert "31-JAN-04" einen Monat hinzu und gibt "29-FEB-04" zurück. Die Funktion erkennt das Jahr 2004 als Schaltjahr und gibt daher den letzten Tag des Monats Februar zurück. Wenn Sie den Eingabewert in "31-JAN-05" ändern, gibt die Funktion "28-FEB-05" zurück.

Beispiel: Sie zeigen für alle Mitarbeiter, die seit weniger als 150 Monaten im Unternehmen beschäftigt sind, die Personalnummer, das Einstellungsdatum, die Beschäftigungsdauer in Monaten, das Ende der 6-monatigen Probezeit, den ersten Freitag nach dem Einstellungsdatum und den letzten Tag des Monats an, in dem die Einstellung erfolgte.

```
SELECT employee_id, hire_date, MONTHS_BETWEEN (SYSDATE, hire_date)
      TENURE, ADD_MONTHS (hire_date, 6) REVIEW, NEXT_DAY (hire_date,
      'FRIDAY'), LAST_DAY(hire_date)
FROM employees WHERE MONTHS_BETWEEN (SYSDATE, hire_date) < 150;
```

## Funktionen ROUND und TRUNC für Datumswerte

Funktion	Ergebnis
ROUND (SYSDATE, 'MONTH' )	01-AUG-03
ROUND (SYSDATE , 'YEAR' )	01-JAN-04
TRUNC (SYSDATE , 'MONTH' )	01-JUL-03
TRUNC (SYSDATE , 'YEAR' )	01-JAN-03

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Funktionen `ROUND` und `TRUNC` können für Zahlen- und Datumswerte verwendet werden. Diese Funktionen runden oder schneiden die Datumswerte dann entsprechend der angegebenen Formatmaske ab. Sie können Datumswerte daher auf das nächstgelegene Jahr oder den nächstgelegenen Monat runden. Bei der Formatmaske `MONTH` werden die Tage 1 bis 15 auf den ersten Tag des aktuellen Monats gerundet. Die Tage 16 bis 31 werden auf den ersten Tag des nächsten Monats gerundet. Bei der Formatmaske `YEAR` werden die Monate 1 bis 6 auf den 1. Januar des aktuellen Jahres gerundet. Die Monate 7 bis 12 werden auf den 1. Januar des nächsten Jahres gerundet.

### Beispiel

Sie möchten das Einstellungsdatum aller Mitarbeiter vergleichen, die im Jahr 2004 im Unternehmen begonnen haben. Dazu zeigen Sie die Personalnummer und das Einstellungsdatum sowie den Anfangsmonat mit den Funktionen `ROUND` und `TRUNC` an.

```
SELECT employee_id, hire_date,
       ROUND(hire_date, 'MONTH'), TRUNC(hire_date, 'MONTH')
FROM   employees
WHERE  hire_date LIKE '%04
```

## Quiz

Welche vier der folgenden Aussagen treffen auf Single-Row-Funktionen zu?

- a. Sie bearbeiten Datenelemente.
- b. Sie akzeptieren Argumente und geben einen Wert pro Argument zurück.
- c. Sie bearbeiten jede zurückgegebene Zeile.
- d. Sie geben ein Ergebnis pro Zeilengruppe zurück.
- e. Sie ändern nie den Datentyp.
- f. Sie können verschachtelt werden.
- g. Sie akzeptieren Spalten oder Ausdrücke als Argumente.

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

**Richtige Antworten a, c, f, g**

# Zusammenfassung

In dieser Lektion haben Sie Folgendes gelernt:

- Verschiedene Typen von Funktionen verwenden, die in SQL verfügbar sind
- Zeichenfunktionen, numerische Funktionen und Datumsfunktionen in `SELECT`-Anweisungen verwenden

**ORACLE**

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Übungen zu Lektion 4 – Überblick

Diese Übung behandelt folgende Themen:

- Abfrage erstellen, die das `SYSDATE` anzeigt
- Abfragen mithilfe von numerischen Funktionen, Zeichenfunktionen und Datumsfunktionen erstellen
- Dauer der Betriebszugehörigkeit eines Mitarbeiters in Jahren und Monaten berechnen

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Diese Übung enthält eine Reihe von Aufgaben, in denen die verschiedenen Funktionen verwendet werden, die für die Datentypen CHARACTER, NUMBER und DATE verfügbar sind.





# 5

## Konvertierungsfunktionen und bedingte Ausdrücke

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- Verschiedene Typen von Konvertierungsfunktionen beschreiben, die in SQL verfügbar sind
- Konvertierungsfunktionen `TO_CHAR`, `TO_NUMBER` und `TO_DATE` verwenden
- Bedingte Ausdrücke in `SELECT`-Anweisungen anwenden

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Diese Lektion behandelt im Wesentlichen Funktionen, die Daten von einem Datentyp in einen anderen konvertieren (etwa Zeichendaten in numerische Daten) und erläutert bedingte Ausdrücke in SQL-Anweisungen vom Typ `SELECT`.

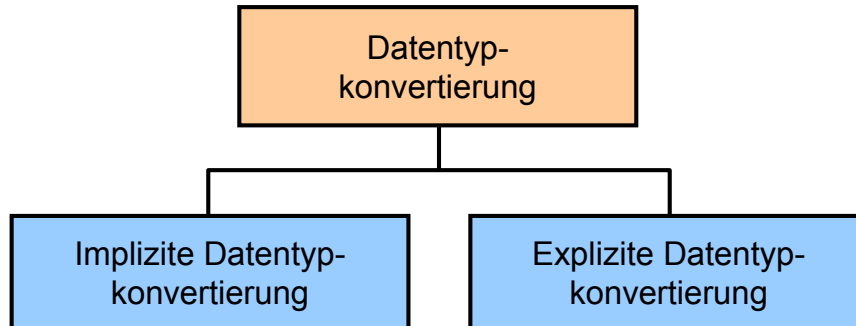
# Lektionsagenda

- Implizite und explizite Datentypkonvertierung
- Funktionen `TO_CHAR`, `TO_DATE` und `TO_NUMBER`
- Allgemeine Funktionen:
  - `NVL`
  - `NVL2`
  - `NULLIF`
  - `COALESCE`
- Bedingte Ausdrücke:
  - `CASE`
  - Searched `CASE`
  - `DECODE`

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

# Konvertierungsfunktionen



ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Zusätzlich zu den Oracle-Datentypen können die Tabellenspalten in einer Oracle-Datenbank als ANSI-(American National Standards Institute-), DB2- und SQL/DS-Datentypen definiert werden. Der Oracle-Server konvertiert diese Datentypen jedoch intern in Oracle-Datentypen.

In einigen Fällen empfängt der Oracle-Server Daten eines bestimmten Datentyps, während er Daten eines anderen Datentyps erwartet. Wenn dies geschieht, kann der Oracle-Server die Daten automatisch in den erwarteten Datentyp konvertieren. Diese Datentypkonvertierung kann *implizit* durch den Oracle-Server oder *explizit* durch den Benutzer erfolgen.

Implizite Datentypkonvertierungen werden gemäß den auf den folgenden Folien erläuterten Regeln durchgeführt.

Explizite Datentypkonvertierungen werden vom Benutzer mithilfe von Konvertierungsfunktionen vorgenommen. Konvertierungsfunktionen konvertieren einen Wert von einem Datentyp in einen anderen. Normalerweise entspricht die Form der Funktionsnamen der Konvention *data type TO data type*. Der erste Datentyp ist der Eingabedatentyp und der zweite Datentyp der Ausgabedatentyp.

**Hinweis:** Implizite Datentypkonvertierung ist zwar verfügbar, Sie sollten die Datentypkonvertierung jedoch explizit vornehmen, um die Zuverlässigkeit Ihrer SQL-Anweisungen sicherzustellen.

## Implizite Datentypkonvertierung

Der Oracle-Server kann folgende Datentypen in Ausdrücken automatisch konvertieren:

Von	Nach
VARCHAR2 oder CHAR	NUMBER
VARCHAR2 oder CHAR	DATE

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Der Oracle-Server kann die Datentypkonvertierung in einem Ausdruck automatisch durchführen. Beispiel: Im Ausdruck `hire_date > '01-JAN-90'` wird die Zeichenfolge `'01-JAN-90'` implizit in ein Datum konvertiert. Daher kann ein `VARCHAR2`- oder `CHAR`-Wert in einem Ausdruck implizit in den Datentyp `NUMBER` oder `DATE` konvertiert werden.

**Hinweis:** Konvertierungen von `CHAR` in `NUMBER` sind nur erfolgreich, wenn die Zeichenfolge eine gültige Zahl darstellt.

## Implizite Datentypkonvertierung

Zur Auswertung von Ausdrücken kann der Oracle-Server folgende Datentypen automatisch konvertieren:

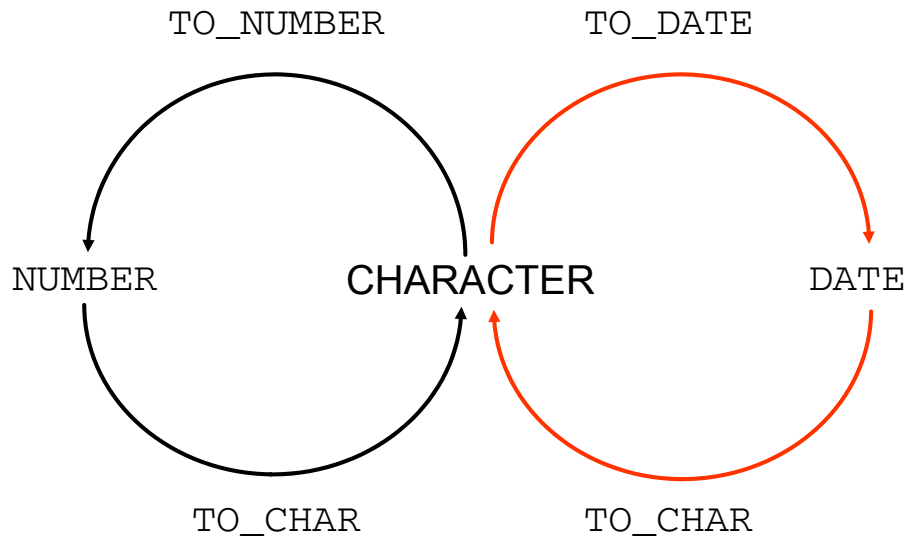
Von	Nach
NUMBER	VARCHAR2 oder CHAR
DATE	VARCHAR2 oder CHAR

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Allgemeinen verwendet der Oracle-Server die Regel für Ausdrücke, wenn eine Datentypkonvertierung erforderlich ist. Beispiel: Beim Ausdruck `job_id = 2` wird die Zahl 2 implizit in die Zeichenfolge "2" konvertiert, weil es sich bei `job_id` um eine Spalte vom Typ `VARCHAR(2)` handelt.

# Explizite Datentypkonvertierung



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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

Funktion	Zweck
<code>TO_CHAR(<i>number</i> <i>date</i> [, <i>fmt</i> [, <i>nlsparms</i>] ] )</code>	<p>Konvertiert einen Zahlen- oder Datumswert in eine Zeichenfolge vom Typ <code>VARCHAR2</code> entsprechend der Formatmaske <i>fmt</i>.</p> <p><b>Konvertierung von numerischen Werten:</b> Der Parameter <i>nlsparms</i> gibt die folgenden Zeichen an, die von Zahlenformatelementen zurückgegeben werden:</p> <ul style="list-style-type: none"><li>• Dezimalzeichen</li><li>• Gruppentrennzeichen</li><li>• Lokales Währungssymbol</li><li>• Internationales Währungssymbol</li></ul> <p>Wenn <i>nlsparms</i> nicht angegeben ist und auch keiner der anderen Parameter, verwendet diese Funktion die Standardparameterwerte für die Session.</p>

Funktion	Zweck
<code>TO_NUMBER(char[,fmt[,nlsparams]])</code>	<p>Konvertiert eine Zeichenfolge aus Ziffern in eine Zahl im Format, das durch die optionale Formatmaske <i>fmt</i> angegeben wird.</p> <p>Der Parameter <code>nlsparams</code> hat in dieser Funktion denselben Zweck wie in der Funktion <code>TO_CHAR</code> zur Konvertierung von Zahlen.</p>
<code>TO_DATE(char[,fmt[,nlsparams]])</code>	<p>Konvertiert eine Zeichenfolge, die ein Datum darstellt, gemäß der angegebenen <i>fmt</i> in einen Datumswert. Wird <i>fmt</i> nicht angegeben, ist das Format <code>DD-MON-YY</code>.</p> <p>Der Parameter <code>nlsparams</code> hat in dieser Funktion denselben Zweck wie in der Funktion <code>TO_CHAR</code> zur Konvertierung von Datumswerten.</p>

**Hinweis:** Die Liste der in dieser Lektion erwähnten Konvertierungsfunktionen ist nicht vollständig. Weitere Informationen finden Sie in der *Oracle Database SQL Language Reference* für Oracle Database 12c im Abschnitt "Conversion Functions".



# Lektionsagenda

- Implizite und explizite Datentypkonvertierung
- **Funktionen TO\_CHAR, TO\_DATE und TO\_NUMBER**
- Allgemeine Funktionen:
  - NVL
  - NVL2
  - NULLIF
  - COALESCE
- Bedingte Ausdrücke:
  - CASE
  - Searched CASE
  - DECODE

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

# Funktion TO\_CHAR in Verbindung mit Datumswerten

```
TO_CHAR(date [, 'format_model' ])
```

Die Formatmaske:

- muss in Hochkommas gesetzt werden
- unterscheidet zwischen Groß- und Kleinschreibung
- kann alle gültigen Elemente eines Datumsformats enthalten
- verfügt über ein Element *fm*, um füllende Leerzeichen zu löschen oder vorgestellte Nullen zu unterdrücken
- wird durch ein Komma vom Datumswert getrennt

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

TO\_CHAR konvertiert einen Wert vom Datentyp DATETIME in einen Wert vom Datentyp VARCHAR2 in dem durch die Formatmaske (*format\_model*) angegebenen Format. Eine Formatmaske ist ein Zeichenliteral, welches das Format von DATETIME beschreibt, das in einer Zeichenfolge gespeichert ist. Beispiel: Die Formatmaske DATETIME für die Zeichenfolge '11-Nov-2000' ist 'DD-Mon-YYYY'. Mithilfe der Funktion TO\_CHAR können Sie ein Datum aus diesem Standardformat in das von Ihnen angegebene Format konvertieren.

## Richtlinien

- Die Formatmaske muss in Hochkommas gesetzt werden. Die Groß-/Kleinschreibung wird beachtet.
- Die Formatmaske kann alle gültigen Elemente eines Datumsformats enthalten. Trennen Sie den Datumswert jedoch immer durch ein Komma von der Formatmaske.
- Die Namen der Tage und Monate in der Ausgabe werden automatisch mit Leerzeichen aufgefüllt.
- Um diese füllenden Leerzeichen zu löschen oder vorgestellte Nullen zu unterdrücken, verwenden Sie das Füllmoduselement *fm*.

```
SELECT employee_id, TO_CHAR(hire_date, 'MM/YY') Month_Hired
FROM   employees
WHERE  last_name = 'Higgins';
```

## Datumsformatmasken – Elemente

Element	Ergebnis
YYYY	Vollständiges Jahr als Zahl
YEAR	Jahr in Worten (in Englisch)
MM	Zweistelliger Zahlenwert für den Monat
MONTH	Vollständiger Name des Monats
MON	Abkürzung aus drei Buchstaben für den Monat
DY	Abkürzung aus drei Buchstaben für den
DAY	Vollständiger Name des Wochentags
DD	Tag des Monats als numerischer Wert

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Element	Beschreibung
SCC oder CC	Jahrhundert. Server verwendet als Präfix für vorchristliche Datumswerte ein Minuszeichen (-).
Jahre in Datumsangabe YYYY oder SYYYY	Jahr. Server verwendet als Präfix für vorchristliche Datumswerte ein Minuszeichen (-).
YYY oder YY oder Y	Bei Jahresangaben die letzten drei Ziffern, die letzten zwei Ziffern oder die letzte Ziffer
Y,YYY	Jahr mit Komma an dieser Stelle
IYYY, IYY, IY, I	Vier-, drei-, zwei- oder einstelliges Jahr nach ISO-Standard
SYEAR oder YEAR	Jahr in Worten. Server verwendet als Präfix für vorchristliche Datumswerte ein Minuszeichen (-).
BC oder AD	Gibt ein Jahr vor Christus oder nach Christus an
B.C. oder A.D.	Gibt ein Jahr vor Christus oder nach Christus an (wobei Punkte verwendet werden)
Q	Quartal des Jahres
MM	Monat: Zweistelliger Zahlenwert
MONTH	Name des Monats, bis zur Länge von neun Zeichen mit Leerzeichen aufgefüllt
MON	Name des Monats, Abkürzung aus drei Buchstaben
RM	Monatsname in römischen Zahlen
WW oder W	Woche des Jahres oder Monats
DDD oder DD oder D	Tag des Jahres, des Monats oder der Woche
DAY	Name des Tages, bis zur Länge von neun Zeichen mit Leerzeichen aufgefüllt
DY	Name des Tages, Abkürzung aus drei Buchstaben
J	Tag nach dem julianischen Kalender, Anzahl der Tage seit dem 31. Dezember 4713 v. Chr.
IW	Wochen im Jahr nach ISO-Standard (1 bis 53)

## Datumsformatmasken – Elemente

- Uhrzeitelemente formatieren den Uhrzeitteil des Datums:

HH24:MI:SS AM	15:45:32 PM
---------------	-------------

- Zeichenfolgen hinzufügen, indem diese in doppelte Anführungszeichen gesetzt werden:

DD "of" MONTH	12 of OCTOBER
---------------	---------------

- Zahlensuffixe geben Zahlen in Worten wieder:

ddspth	fourteenth
--------	------------

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Um Uhrzeitinformationen und Literale anzuzeigen sowie Zahlen in Worte zu ändern, verwenden Sie die Formate aus den folgenden Tabellen.

Element	Beschreibung
AM oder PM	AM/PM-Angaben
A.M. oder P.M.	AM/PM-Angaben mit Punkten
HH oder HH12	12-Stunden-Format
HH24	24-Stunden-Format
MI	Minute (0–59)
SS	Sekunde (0–59)
SSSSS	Sekunden nach Mitternacht (0–86399)

## Weitere Formate

Element	Beschreibung
/ . ,	Interpunktion wird im Ergebnis dargestellt.
“of the”	In Anführungszeichen stehende Zeichenfolge wird im Ergebnis dargestellt.

### Suffixe angeben, um die Anzeige von Zahlen zu beeinflussen

Element	Beschreibung
TH	Ordinalzahl (Beispiel: DDTH für 4TH)
SP	Zahl in Worten (Beispiel: DDSP für FOUR)
SPTH oder THSP	Ordinalzahlen in Worten (zum Beispiel DDSPTH für FOURTH)

## Funktion TO\_CHAR in Verbindung mit Datumswerten

```
SELECT last_name,  
       TO_CHAR(hire_date, 'fmDD Month YYYY')  
       AS HIREDATE  
FROM   employees;
```

	LAST_NAME	HIREDATE
1	King	17 June 2003
2	Kochhar	21 September 2005
3	De Haan	13 January 2001
4	Hunold	3 January 2006
5	Ernst	21 May 2007
6	Lorentz	7 February 2007
7	Mourgos	16 November 2007
8	Rajs	17 October 2003

...

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die SQL-Anweisung auf der Folie zeigt für alle Mitarbeiter den Nachnamen und das Einstellungsdatum an. Das Einstellungsdatum wird im Format "17 June 2003" angezeigt.

### Beispiel

Sie ändern das Beispiel auf der Folie, sodass das Datum im Format "Seventeenth of June 2003 12:00:00 AM" angezeigt wird.

```
SELECT last_name,  
       TO_CHAR(hire_date,  
               'fmDdsph "of" Month YYYY fmHH:MI:SS AM')  
       HIREDATE  
FROM   employees;
```

Die Schreibweise des Monats entspricht der angegebenen Formatmaske. Der erste Buchstabe wird großgeschrieben, und die restlichen Buchstaben des Wortes werden kleingeschrieben.

## Funktion TO\_CHAR in Verbindung mit Zahlenwerten

```
TO_CHAR(number[, 'format_model'])
```

Sie können unter anderem die folgenden Formatelemente mit der Funktion TO\_CHAR verwenden, um einen Zahlenwert als Zeichenwert anzuzeigen:

Element	Ergebnis
9	Stellt eine Zahl dar
0	Erzwingt die Anzeige einer Null
\$	Setzt ein führendes Dollarzeichen
L	Verwendet das führende lokale Währungssymbol
.	Gibt einen Dezimalpunkt aus
,	Gibt ein Komma als Tausendertrennzeichen aus

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wenn Sie mit Zahlenwerten arbeiten, etwa mit Zeichenfolgen, sollten Sie diese Zahlen mit der Funktion TO\_CHAR konvertieren. Diese Funktion übersetzt einen Wert vom Datentyp NUMBER in den Datentyp VARCHAR2. Diese Technik ist besonders bei Verkettungen hilfreich.



## Zahlenformatelemente

Bei der Konvertierung einer Zahl in den Zeichendatentyp können Sie folgende Formatelemente verwenden:

Element	Beschreibung	Beispiel	Ergebnis
9	Numerische Position (Die Anzahl der Neunen bestimmt die Anzeigebreite.)	999999	1234
0	Führende Nullen anzeigen	099999	001234
\$	Führendes Dollarzeichen	\$999999	\$1234
L	Führendes lokales Währungssymbol	L999999	FF1234
D	Gibt das Dezimalzeichen an der angegebenen Position zurück. Standardwert ist ein Punkt (.).	9999D99	1234.00
.	Dezimalpunkt an der angegebenen Position	999999.99	1234.00
G	Gibt das Gruppentrennzeichen an der angegebenen Position zurück. Sie können in einer Zahlenformatmaske mehrere Gruppentrennzeichen angeben.	9G999	1,234
,	Komma an der angegebenen Position	999,999	1,234
MI	Minuszeichen rechts (negative Werte)	999999MI	1234-
PR	Negative Zahlen in Klammern setzen	999999PR	<1234>
EEEE	Wissenschaftliche Darstellung (Format muss viermal den Buchstaben E angeben.)	99.999EEEE	1.234E+03
U	Gibt das Eurosymbol (oder ein anderes Symbol) bei der doppelten Währungsunterstützung an der angegebenen Position zurück	U9999	€1234
V	<i>n</i> Mal mit 10 multiplizieren ( <i>n</i> = Anzahl der Neunen nach dem V)	9999V99	123400
S	Gibt den negativen oder positiven Wert zurück	S9999	-1234 oder +1234
B	Wert Null nicht als 0, sondern als leeres Feld anzeigen	B9999,99	1234.00

## Funktion TO\_CHAR in Verbindung mit Zahlenwerten

```
SELECT TO_CHAR(salary, '$99,999.00') SALARY  
FROM   employees  
WHERE  last_name = 'Ernst';
```

	SALARY
1	\$6,000.00

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

- Wenn die Anzahl der Ziffern die in der Formatmaske festgelegte Anzahl an Ziffern überschreitet, zeigt der Oracle-Server anstelle einer ganzen Zahl eine Folge von Nummernzeichen (#) an.
- Der Oracle-Server rundet den gespeicherten Dezimalwert auf die in der Formatmaske angegebene Anzahl von Dezimalstellen.

## Funktionen TO\_NUMBER und TO\_DATE

- Mit der Funktion TO\_NUMBER eine Zeichenfolge in ein Zahlenformat konvertieren:

```
TO_NUMBER(char[, 'format_model'])
```

- Mit der Funktion TO\_DATE eine Zeichenfolge in ein Datumsformat konvertieren:

```
TO_DATE(char[, 'format_model'])
```

- Diese Funktionen enthalten den Modifizierer `fx`, der die exakte Übereinstimmung des Zeichenarguments mit der Datumsformatmaske einer TO\_DATE-Funktion angibt.

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Unter Umständen möchten Sie eine Zeichenfolge in einen Zahlen- oder Datumswert konvertieren. Dazu verwenden Sie die Funktion TO\_NUMBER oder TO\_DATE. Die gewählte Formatmaske besteht aus den bereits beschriebenen Formatelementen.

Der Modifizierer `fx` gibt die exakte Übereinstimmung des Zeichenarguments mit der Datumsformatmaske einer TO\_DATE-Funktion an:

- Interpunktion und in Anführungszeichen gesetzter Text im Zeichenargument (mit Ausnahme der Groß-/Kleinschreibung) müssen exakt mit den entsprechenden Teilen der Formatmaske übereinstimmen.
- Das Zeichenargument darf keine zusätzlichen Leerzeichen enthalten. Ohne `fx` ignoriert der Oracle-Server zusätzliche Leerzeichen.
- Numerische Daten im Zeichenargument müssen dieselbe Stellenanzahl wie das entsprechende Element in der Formatmaske haben. Ohne `fx` können für Zahlen im Zeichenargument führende Nullen weggelassen werden.

## Beispiel

Sie möchten für alle Mitarbeiter, die am 24. Mai 2007 eingestellt wurden, den Namen und das Einstellungsdatum anzeigen. Im folgenden Beispiel stehen zwei Leerzeichen zwischen dem Monat *May* und der Zahl 24. Da der Modifizierer `fx` verwendet wird, ist eine exakte Übereinstimmung erforderlich. Die Leerzeichen nach dem Wort *May* werden nicht erkannt:

```
SELECT last_name, hire_date
FROM   employees
WHERE  hire_date = TO_DATE('May  24, 2007', 'fxMonth DD, YYYY');
```

Folgender Fehler wird ausgegeben:

```
ORA-01858: a non-numeric character was found where a numeric was expected
01858. 00000 - "a non-numeric character was found where a numeric was expected"
*Cause:   The input data to be converted using a date format model was
          incorrect. The input data did not contain a number where a number was
          required by the format model.
*Action:  Fix the input data or the date format model to make sure the
          elements match in number and type. Then retry the operation.
```



Um die Ausgabe anzuzeigen, korrigieren Sie die Abfrage, indem Sie das zusätzliche Leerzeichen zwischen "May" und "24" löschen.

```
SELECT last_name, hire_date
FROM   employees
WHERE  hire_date = TO_DATE('May 24, 2007', 'fxMonth DD, YYYY');
```

## Funktionen TO\_CHAR und TO\_DATE in Verbindung mit dem Datumsformat RR

Mit dem Datumsformat `RR` kann nach Mitarbeitern gesucht werden, die vor 1990 eingestellt wurden. Die Ausführung des Befehls im Jahr 1999 führt dabei zu den gleichen Ergebnissen wie seine Ausführung heute:

```
SELECT last_name, TO_CHAR(hire_date, 'DD-Mon-YYYY')
FROM   employees
WHERE  hire_date < TO_DATE('01-Jan-90', 'DD-Mon-RR');
```

 LAST_NAME	 TO_CHAR(HIRE_DATE,'DD-MON-YYYY')
1 Popp	03-Feb-1989

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Um die Mitarbeiter zu suchen, die vor 1990 eingestellt wurden, können Sie das Format `RR` verwenden. Da die aktuelle Jahreszahl größer als 1999 ist, interpretiert das Format `RR` die Jahresinformation im Datum als Wert zwischen 1950 und 1999.

Der folgende Befehl hingegen gibt keine Zeilen zurück, weil das Format `YY` die Jahresinformationen im Datum als Werte aus dem aktuellen Jahrhundert (2090) interpretiert.

```
SELECT last_name, TO_CHAR(hire_date, 'DD-Mon-yyyy')
FROM   employees
WHERE  TO_DATE(hire_date, 'DD-Mon-yy') < '01-Jan-90';
```

Beachten Sie, dass bei der oben aufgeführten Abfrage keine Zeilen abgerufen werden.

# Lektionsagenda

- Implizite und explizite Datentypkonvertierung
- Funktionen TO\_CHAR, TO\_DATE, TO\_NUMBER
- **Allgemeine Funktionen:**
  - NVL
  - NVL2
  - NULLIF
  - COALESCE
- **Bedingte Ausdrücke:**
  - CASE
  - Searched CASE
  - DECODE

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Allgemeine Funktionen

Die folgenden Funktionen können für alle Datentypen eingesetzt werden und beziehen sich auf die Verwendung von Nullwerten:

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

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Diese Funktionen können für alle Datentypen eingesetzt werden und beziehen sich auf die Verwendung von Nullwerten in der Ausdrucksliste.

Funktion	Beschreibung
NVL	Konvertiert einen Nullwert in einen konkreten Wert
NVL2	Wenn <code>expr1</code> kein Nullwert ist, gibt NVL2 den Ausdruck <code>expr2</code> zurück. Wenn <code>expr1</code> ein Nullwert ist, gibt NVL2 den Ausdruck <code>expr3</code> zurück. Das Argument <code>expr1</code> kann einen beliebigen Datentyp aufweisen.
NULLIF	Vergleicht zwei Ausdrücke und gibt einen Nullwert zurück, wenn sie übereinstimmen, oder den ersten Ausdruck, wenn sie nicht übereinstimmen
COALESCE	Gibt den ersten Wert der Ausdrucksliste zurück, der kein Nullwert ist

**Hinweis:** Weitere Informationen zu den zahlreichen verfügbaren Funktionen finden Sie in der *Oracle Database SQL Language Reference* für Oracle Database 12c im Abschnitt "Functions".

## Funktion NVL

Konvertiert einen Nullwert in einen konkreten Wert:

- Folgende Datentypen können verwendet werden: DATE, CHARACTER und NUMBER.
- Datentypen müssen übereinstimmen:
  - `NVL(commission_pct, 0)`
  - `NVL(hire_date, '01-JAN-97')`
  - `NVL(job_id, 'No Job Yet')`

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Um einen Nullwert in einen konkreten Wert zu konvertieren, verwenden Sie die Funktion `NVL`.

### Syntax

`NVL (expr1, expr2)`

Für die Syntax gilt:

- `expr1` ist der Quellwert oder -ausdruck, der einen Nullwert enthalten kann.
- `expr2` ist der Zielwert, in den der Nullwert konvertiert wird.

Mit der Funktion `NVL` können Sie beliebige Datentypen konvertieren. Der Datentyp des Rückgabewertes entspricht jedoch immer dem Datentyp von `expr1`.

### NVL-Konvertierungen für verschiedene Datentypen

Datentyp	Konvertierungsbeispiel
NUMBER	<code>NVL(number_column, 9)</code>
DATE	<code>NVL(date_column, '01-JAN-95')</code>
CHAR oder VARCHAR2	<code>NVL(character_column, 'Unavailable')</code>



## Funktion NVL – Beispiel

```
SELECT last_name, salary, NVL(commission_pct, 0)
       (salary*12) + (salary*12*NVL(commission_pct, 0)) AN_SAL
FROM employees;
```

	LAST_NAME	SALARY	NVL(COMMISSION_PCT,0)	AN_SAL
1	King	24000	0	288000
2	Kochhar	17000	0	204000
3	De Haan	17000	0	204000
4	Hunold	9000	0	108000
5	Ernst	6000	0	72000
6	Lorentz	4200	0	50400
7	Mourgos	5800	0	69600
8	Rajs	3500	0	42000
9	Davies	3100	0	37200
10	Matos	2600	0	31200

...

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

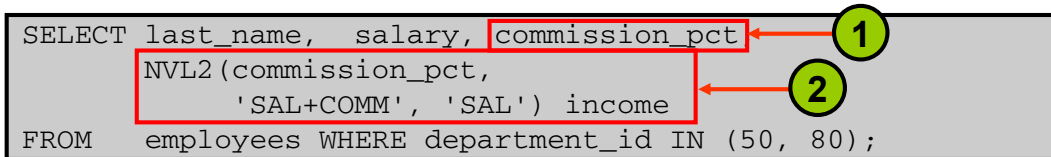
Um für alle Mitarbeiter die jährliche Vergütung zu berechnen, multiplizieren Sie das Monatsgehalt mit 12 und addieren dann die Provision zum Ergebnis:

```
SELECT last_name, salary, commission_pct,
       (salary*12) + (salary*12*commission_pct) AN_SAL
FROM employees;
```

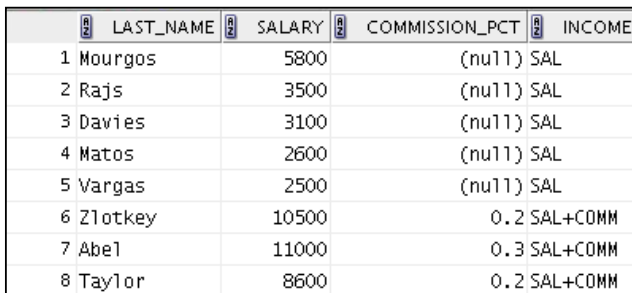
Die jährliche Vergütung wird nur für die Mitarbeiter berechnet, die provisionsberechtigt sind. Wenn es sich bei einem der Werte in einem Ausdruck um einen Nullwert handelt, ist auch das Ergebnis ein Nullwert. Um Werte für alle Mitarbeiter zu berechnen, konvertieren Sie den Nullwert in eine Zahl, bevor Sie den arithmetischen Operator anwenden. Im Beispiel auf der Folie wird die Funktion NVL verwendet, um Nullwerte in die Zahl Null zu konvertieren.

## Funktion NVL2 – Beispiel

```
SELECT last_name, salary, commission_pct  
      NVL2(commission_pct,  
            'SAL+COMM', 'SAL') income  
FROM   employees WHERE department_id IN (50, 80);
```



	LAST_NAME	SALARY	COMMISSION_PCT	INCOME
1	Mourgos	5800	(null)	SAL
2	Rajs	3500	(null)	SAL
3	Davies	3100	(null)	SAL
4	Matos	2600	(null)	SAL
5	Vargas	2500	(null)	SAL
6	Zlotkey	10500	0.2	SAL+COMM
7	Abel	11000	0.3	SAL+COMM
8	Taylor	8600	0.2	SAL+COMM



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Funktion NVL2 prüft den ersten Ausdruck. Entspricht der erste Ausdruck keinem Nullwert, gibt die Funktion NVL2 den zweiten Ausdruck zurück. Entspricht der erste Ausdruck hingegen einem Nullwert, wird der dritte Ausdruck zurückgegeben.

### Syntax

`NVL2(expr1, expr2, expr3)`

Für die Syntax gilt:

- *expr1* ist der Quellwert oder -ausdruck, der einen Nullwert enthalten kann.
- *expr2* ist der Rückgabewert, wenn *expr1* keinem Nullwert entspricht.
- *expr3* ist der Rückgabewert, wenn *expr1* einem Nullwert entspricht.

Im Beispiel auf der Folie wird die Spalte `COMMISSION_PCT` geprüft. Enthält die Spalte einen Wert, wird der Textliteralwert von `SAL+COMM` zurückgegeben. Enthält die Spalte `COMMISSION_PCT` einen Nullwert, wird der Textliteralwert von `SAL` zurückgegeben.

**Hinweis:** Das Argument *expr1* kann einen beliebigen Datentyp aufweisen. *expr2* und *expr3* sollten aber demselben Datentyp angehören.

## Funktion NULLIF – Beispiel

The diagram illustrates the NULLIF function with three numbered annotations:

- 1**: Points to the `LENGTH(first_name) "expr1",` part of the SQL query.
- 2**: Points to the `LENGTH(last_name) "expr2",` part of the SQL query.
- 3**: Points to the `NULLIF(LENGTH(first_name), LENGTH(last_name)) result` part of the SQL query.

The result table below shows the output of the query:

	FIRST_NAME	expr1	LAST_NAME	expr2	RESULT
1	Ellen	5	Abel	4	5
2	Curtis	6	Davies	6	(null)
3	Lex	3	De Haan	7	3
4	Bruce	5	Ernst	5	(null)
5	Pat	3	Fay	3	(null)
6	William	7	Gietz	5	7
7	Kimberely	9	Grant	5	9
8	Michael	7	Hartstein	9	7
9	Shelley	7	Higgins	7	(null)
...					

Annotations 1, 2, and 3 are also placed below the table, pointing to the columns `FIRST_NAME`, `expr1`, `LAST_NAME`, `expr2`, and `RESULT` respectively.

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Funktion NULLIF vergleicht zwei Ausdrücke.

### Syntax

NULLIF (*expr1*, *expr2*)

Für die Syntax gilt:

- NULLIF vergleicht *expr1* und *expr2*. Sind die Ausdrücke gleich, gibt die Funktion einen Nullwert zurück. Andernfalls gibt die Funktion *expr1* zurück. Sie können für *expr1* jedoch nicht das Literal NULL angeben.

Im Beispiel auf der Folie wird die Länge des Vornamens in der Tabelle EMPLOYEES mit der Länge des Nachnamens in der Tabelle EMPLOYEES verglichen. Wenn die Länge der Namen gleich ist, wird ein Nullwert angezeigt. Andernfalls wird die Länge des Vornamens angezeigt.

## Funktion COALESCE – Beispiel

- Der Vorteil der Funktion `COALESCE` gegenüber der Funktion `NVL` besteht darin, dass bei `COALESCE` mehrere alternative Werte angegeben werden können.
- Ist der erste Ausdruck kein Nullwert, gibt die Funktion `COALESCE` diesen Ausdruck zurück. Andernfalls werden die übrigen Ausdrücke durch `COALESCE` zusammengeführt.

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Funktion `COALESCE` gibt den ersten Ausdruck aus der Liste zurück, der keinen Nullwert enthält.

### Syntax

`COALESCE (expr1, expr2, ... exprn)`

Für die Syntax gilt:

- *expr1* ist der zurückgegebene Ausdruck, wenn er kein Nullwert ist.
- *expr2* ist der zurückgegebene Ausdruck, wenn der erste Ausdruck ein Nullwert und dieser Ausdruck kein Nullwert ist.
- *exprn* ist der zurückgegebene Ausdruck, wenn alle vorherigen Ausdrücke Nullwerte sind.

Alle Ausdrücke müssen denselben Datentyp aufweisen.

## Funktion COALESCE – Beispiel

```
SELECT last_name, salary, commission_pct,  
COALESCE((salary+(commission_pct*salary)), salary+2000) "New Salary"  
FROM employees;
```

	LAST_NAME	SALARY	COMMISSION_PCT	New Salary
1	King	24000	(null)	26000
2	Kochhar	17000	(null)	19000
3	De Haan	17000	(null)	19000
4	Hunold	9000	(null)	11000
5	Ernst	6000	(null)	8000
6	Lorentz	4200	(null)	6200
7	Mourgos	5800	(null)	7800
8	Rajs	3500	(null)	5500
9	Davies	3100	(null)	5100
10	Matos	2600	(null)	4600
11	Vargas	2500	(null)	4500
12	Zlotkey	10500	0.2	12600
13	Abel	11000	0.3	14300
14	Taylor	8600	0.2	10320
15	Grant	7000	0.15	8050
16	Whalen	4400	(null)	6400
17	Hartstein	13000	(null)	15000
18	Fay	6000	(null)	8000
19	Higgins	12008	(null)	14008
20	Gietz	8300	(null)	10300

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf der Folie sollen alle Mitarbeiter, die keine Provision erhalten, eine Gehalts-erhöhung in Höhe von \$ 2.000 erhalten. Für provisionsberechtigte Mitarbeiter soll die Abfrage das neue Gehalt berechnen, das dem vorhandenen Gehalt plus Provisionsbetrag entspricht.

**Hinweis:** Prüfen Sie die Ausgabe. Für Mitarbeiter, die keine Provision erhalten, zeigt die Spalte "New Salary" das um \$ 2.000 erhöhte Gehalt an. Für Mitarbeiter mit Provision wird das Gehalt mit der zum Gehalt addierten berechneten Provision angezeigt.

# Lektionsagenda

- Implizite und explizite Datentypkonvertierung
- Funktionen TO\_CHAR, TO\_DATE, TO\_NUMBER
- Allgemeine Funktionen:
  - NVL
  - NVL2
  - NULLIF
  - COALESCE
- Bedingte Ausdrücke:
  - CASE
  - Searched CASE
  - DECODE

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Bedingte Ausdrücke

- IF-THEN-ELSE-Logik in einer SQL-Anweisung bereitstellen
- Verfügbare Methoden:
  - CASE-Ausdrücke
  - Searched CASE-Ausdrücke
  - Funktion DECODE

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Zwei Methoden zum Implementieren der bedingten Verarbeitung (IF-THEN-ELSE-Logik) in einer SQL-Anweisung sind CASE-Ausdrücke und die Funktion DECODE.

**Hinweis:** CASE-Ausdrücke entsprechen dem ANSI SQL-Standard. Die Funktion DECODE gehört zur Oracle-spezifischen Syntax.

## CASE-Ausdrücke

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

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

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit CASE-Ausdrücken können Sie die IF-THEN-ELSE-Logik in SQL-Anweisungen verwenden, ohne Prozeduren aufrufen zu müssen.

In einem einfachen CASE-Ausdruck sucht Oracle nach dem ersten WHEN ... THEN-Paar, für das der Wert von *expr* dem Wert von *comparison\_expr* entspricht, und gibt *return\_expr* zurück. Wenn kein WHEN ... THEN-Paar diese Bedingung erfüllt und eine ELSE-Klausel existiert, gibt der Oracle-Server *else\_expr* zurück. Andernfalls gibt der Oracle-Server einen Nullwert zurück. Sie dürfen nicht für alle Werte von *return\_expr* und *else\_expr* das Literal NULL verwenden.

Die Ausdrücke *expr* und *comparison\_expr* müssen alle denselben Datentyp (CHAR, VARCHAR2, NCHAR oder NVARCHAR2, NUMBER, BINARY\_FLOAT oder BINARY\_DOUBLE) oder einen numerischen Datentyp aufweisen. Alle zurückgegebenen Werte (*return\_expr*) müssen denselben Datentyp aufweisen.



## CASE-Ausdrücke – Beispiel

```
SELECT last_name, job_id, salary,  
       CASE job_id WHEN 'IT_PROG' THEN 1.10*salary  
                  WHEN 'ST_CLERK' THEN 1.15*salary  
                  WHEN 'SA_REP' THEN 1.20*salary  
       ELSE salary END "REVISED_SALARY"  
FROM employees;
```

	LAST_NAME	JOB_ID	SALARY	REVISED_SALARY
1	King	AD_PRES	24000	24000
...				
4	Hunold	IT_PROG	9000	9900
5	Ernst	IT_PROG	6000	6600
6	Lorentz	IT_PROG	4200	4620
7	Mourgos	ST_MAN	5800	5800
8	Rajs	ST_CLERK	3500	4025
9	Davies	ST_CLERK	3100	3565
10	Matos	ST_CLERK	2600	2990
11	Vargas	ST_CLERK	2500	2875
...				
13	Abel	SA_REP	11000	13200
14	Taylor	SA_REP	8600	10320
15	Grant	SA_REP	7000	8400

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In der SQL-Anweisung auf der Folie wird der Wert von `JOB_ID` decodiert. Wenn `JOB_ID` den Wert `IT_PROG` enthält, beträgt die Gehaltserhöhung 10 %. Wenn `JOB_ID` den Wert `ST_CLERK` enthält, beträgt die Gehaltserhöhung 15 %. Wenn `JOB_ID` den Wert `SA_REP` enthält, beträgt die Gehaltserhöhung 20 %. Alle anderen Berufsgruppen erhalten keine Gehaltserhöhung. Dieselbe Anweisung können Sie mit der Funktion `DECODE` schreiben.

## Searched CASE-Ausdrücke

```
CASE
  WHEN condition1 THEN use_expression1
  WHEN condition2 THEN use_expression2
  WHEN condition3 THEN use_expression3
  ELSE default_use_expression
END
```

```
SELECT last_name,salary,
  (CASE WHEN salary<5000 THEN 'Low'
        WHEN salary<10000 THEN 'Medium'
        WHEN salary<20000 THEN 'Good'
        ELSE 'Excellent'
  END) qualified_salary
FROM employees;
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Ausdruck Searched CASE erfolgt die Suche von links nach rechts, bis die angegebene Bedingung zum ersten Mal gefunden wird. Anschließend wird der Rückgabebausdruck zurückgegeben. Wenn keine wahre Bedingung gefunden wird und eine ELSE-Klausel existiert, wird der Rückgabebausdruck in der Klausel ELSE zurückgegeben. Andernfalls wird NULL zurückgegeben. Der Ausdruck Searched CASE wertet die Bedingungen der einzelnen WHEN-Optionen unabhängig voneinander aus.

CASE-Ausdrücke und Searched CASE-Ausdrücke unterscheiden sich darin, dass Sie in einem Searched CASE-Ausdruck nach dem Schlüsselwort WHEN anstelle von *comparison\_expression* eine Bedingung oder ein Prädikat angeben.

In beiden Fällen müssen alle Werte von *return\_exprs* entweder denselben Datentyp (CHAR, VARCHAR2, NCHAR oder NVARCHAR2, NUMBER, BINARY\_FLOAT oder BINARY\_DOUBLE) oder einen numerischen Datentyp aufweisen.

Das Codebeispiel auf dieser Folie zeigt einen Searched CASE-Ausdruck.

## Funktion DECODE

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

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

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Funktion `DECODE` decodiert einen Ausdruck auf ähnliche Weise wie die in verschiedenen Sprachen verwendete `IF-THEN-ELSE`-Logik. Sie decodiert *expression*, nachdem der Wert mit jedem *search*-Wert verglichen wurde. Entspricht der Ausdruck dem Suchwert *search*, wird *result* zurückgegeben.

Wenn ein Suchwert keinem der Ergebniswerte entspricht und kein Standardwert angegeben wurde, wird ein Nullwert zurückgegeben.

## Funktion DECODE – Beispiel

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

	LAST_NAME	JOB_ID	SALARY	REVISED_SALARY
...				
4	Hunold	IT_PROG	9000	9900
5	Ernst	IT_PROG	6000	6600
6	Lorentz	IT_PROG	4200	4620
7	Mourgos	ST_MAN	5800	5800
8	Rajs	ST_CLERK	3500	4025
9	Davies	ST_CLERK	3100	3565
10	Matos	ST_CLERK	2600	2990
11	Vargas	ST_CLERK	2500	2875
12	Zlotkey	SA_MAN	10500	10500
...				
13	Abel	SA_REP	11000	13200
14	Taylor	SA_REP	8600	10320
15	Grant	SA_REP	7000	8400

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In der SQL-Anweisung auf der Folie wird der Wert von `JOB_ID` getestet. Wenn `JOB_ID` den Wert `IT_PROG` enthält, beträgt die Gehaltserhöhung 10 %. Wenn `JOB_ID` den Wert `ST_CLERK` enthält, beträgt die Gehaltserhöhung 15 %. Wenn `JOB_ID` den Wert `SA_REP` enthält, beträgt die Gehaltserhöhung 20 %. Alle anderen Berufsgruppen erhalten keine Gehaltserhöhung.

Die gleiche Anweisung kann in Pseudocode als `IF-THEN-ELSE`-Anweisung ausgedrückt werden:

```
IF job_id = 'IT_PROG'      THEN salary = salary*1.10  
IF job_id = 'ST_CLERK'    THEN salary = salary*1.15  
IF job_id = 'SA_REP'      THEN salary = salary*1.20  
ELSE salary = salary
```

## Funktion DECODE – Beispiel

Anwendbaren Steuersatz für jeden Mitarbeiter der Abteilung 80 anzeigen:

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

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Diese Folie zeigt ein weiteres Beispiel für die Funktion `DECODE`. Hier wird der Steuersatz für jeden Mitarbeiter der Abteilung 80 basierend auf dem Monatsgehalt ermittelt. Es gelten folgende Steuersätze:

<b>Gehaltsstufe (monatlich)</b>	<b>Steuersatz</b>
\$ 0,00 - 1.999,99	00 %
\$ 2.000,00 - 3.999,99	09 %
\$ 4.000,00 - 5.999,99	20 %
\$ 6.000,00 - 7.999,99	30 %
\$ 8.000,00 - 9.999,99	40 %
\$ 10.000,00 - 11.999,99	42 %
\$ 12.200,00 - 13.999,99	44 %
\$ 14.000,00 oder mehr	45 %

## Quiz

Die Funktion `TO_NUMBER` konvertiert Zeichenfolgen oder Datumswerte in eine Zahl in dem von der optionalen Formatmaske angegebenen Format.

- a. Richtig
- b. Falsch

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

**Richtige Antwort: b**

# Zusammenfassung

In dieser Lektion haben Sie Folgendes gelernt:

- Anzeige von Datumsformaten mithilfe von Funktionen ändern
- Spaltendatentypen mithilfe von Funktionen konvertieren
- NVL-Funktionen verwenden
- IF-THEN-ELSE-Logik und andere bedingte Ausdrücke in einer SELECT-Anweisung verwenden

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Beachten Sie folgende Punkte:

- Konvertierungsfunktionen können Zeichen-, Datums- und numerische Werte konvertieren: TO\_CHAR, TO\_DATE, TO\_NUMBER
- Es gibt verschiedene Funktionen, die sich auf Nullwerte beziehen. Hierzu gehören NVL, NVL2, NULLIF und COALESCE.
- Die IF-THEN-ELSE-Logik kann in SQL-Anweisungen mithilfe von CASE-Ausdrücken, Searched CASE-Ausdrücken oder der Funktion DECODE implementiert werden.

## Übungen zu Lektion 5 – Überblick

Diese Übung behandelt folgende Themen:

- Abfragen erstellen, die `TO_CHAR`, `TO_DATE` und andere `DATE`-Funktionen verwenden
- Abfragen mit bedingten Ausdrücken wie `CASE`, `Searched CASE` und `DECODE` erstellen

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Diese Übung enthält eine Reihe von Aufgaben zu den Funktionen `TO_CHAR` und `TO_DATE` sowie zu bedingten Ausdrücken wie `CASE`, `Searched CASE` und `DECODE`.

Beachten Sie, dass verschachtelte Funktionen von innen nach außen ausgewertet werden.



# 6

## **Mit Gruppenfunktionen Berichte aus aggregierten Daten erstellen**

**ORACLE®**

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- Verfügbare Gruppenfunktionen bestimmen
- Verwendung von Gruppenfunktionen beschreiben
- Daten mit der Klausel `GROUP BY` gruppieren
- Gruppierte Zeilen mit der Klausel `HAVING` ein- und ausschließen

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In dieser Lektion werden weitere Funktionen behandelt. Im Mittelpunkt steht die Abfrage von Aggregatinformationen (etwa Durchschnittswerte) für Gruppen von Zeilen. Sie lernen, wie Sie Zeilen einer Tabelle in kleineren Einheiten gruppieren und Suchkriterien für Gruppen von Zeilen angeben.

# Lektionsagenda

- Gruppenfunktionen:
  - Typen und Syntax
  - AVG, SUM, MIN, MAX, COUNT
  - Schlüsselwort `DISTINCT` in Gruppenfunktionen
  - NULL-Werte in Gruppenfunktionen
- Zeilen gruppieren:
  - Klausel `GROUP BY`
  - Klausel `HAVING`
- Gruppenfunktionen verschachteln

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

# Gruppenfunktionen

Gruppenfunktionen werden auf Zeilengruppen angewendet und geben ein Ergebnis pro Gruppe zurück.

EMPLOYEES

	DEPARTMENT_ID	SALARY
1	10	4400
2	20	13000
3	20	6000
4	110	12000
5	110	8300
6	90	24000
7	90	17000
8	90	17000
9	60	9000
10	60	6000
...		
18	80	11000
19	80	8600
20	(null)	7000

Höchstgehalt in der  
Tabelle EMPLOYEES

MAX(SALARY)
24000

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Gegensatz zu Single-Row-Funktionen werden Gruppenfunktionen auf Gruppen von Zeilen angewendet und geben ein Ergebnis pro Gruppe zurück. Bei diesen Gruppen kann es sich um die ganze Tabelle oder einzelne Gruppen der Tabelle handeln.

# Typen von Gruppenfunktionen

- AVG
- COUNT
- MAX
- MIN
- SUM
- LISTAGG
- STDDEV
- VARIANCE



ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Jede dieser Funktionen kann Argumente annehmen. Die folgende Tabelle beschreibt die Optionen, die Sie in der Syntax verwenden können:

Funktion	Beschreibung
AVG ( [DISTINCT   <u>ALL</u> ] <i>n</i> )	Durchschnittswert von <i>n</i> . Nullwerte werden ignoriert.
COUNT	Anzahl von Zeilen, für die die Auswertung von <i>expr</i> einen anderen Wert als einen Nullwert ergeben soll. (Alle mit * ausgewählten Zeilen zählen, einschließlich mehrfach vorhandener Zeilen und Zeilen mit Nullwerten.)
MAX ( [DISTINCT   ALL] <i>expr</i> )	Höchster Wert von <i>expr</i> . Nullwerte werden ignoriert.
MIN ( [DISTINCT   <u>ALL</u> ] <i>expr</i> )	Niedrigster Wert von <i>expr</i> . Nullwerte werden ignoriert.
STDDEV ( [DISTINCT   <u>ALL</u> ] <i>n</i> )	Standardabweichung von <i>n</i> . Nullwerte werden ignoriert.
SUM ( [DISTINCT   <u>ALL</u> ] <i>n</i> )	Summe der Werte von <i>n</i> . Nullwerte werden ignoriert.
LISTAGG	Sortiert Daten innerhalb jeder in der ORDER BY-Klausel angegebenen Gruppe und verkettet die Werte der Spalte mit Messergebnissen
VARIANCE ( [DISTINCT   <u>ALL</u> ] <i>n</i> )	Varianz von <i>n</i> . Nullwerte werden ignoriert.

## Gruppenfunktionen – Syntax

```
SELECT  group_function(column), ...  
FROM    table  
[WHERE  condition];
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Gruppenfunktion steht hinter dem Schlüsselwort `SELECT`. Sie können mehrere durch Kommas getrennte Gruppenfunktionen angeben.

Syntax:

`group_function([DISTINCT|ALL] expr)`

Richtlinien für Gruppenfunktionen:

- `DISTINCT` bewirkt, dass die Funktion keine doppelten Werte berücksichtigt. `ALL` bewirkt, dass alle Werte, einschließlich der doppelten Werte, berücksichtigt werden. Der Standardwert ist `ALL` und muss daher nicht angegeben werden.
- Mögliche Datentypen für Funktionen mit einem `expr`-Argument sind `CHAR`, `VARCHAR2`, `NUMBER` und `DATE`.
- Alle Gruppenfunktionen ignorieren Nullwerte. Um Nullwerte durch Werte zu ersetzen, verwenden Sie die Funktion `NVL`, `NVL2`, `COALESCE`, `CASE` oder `DECODE`.

## Funktionen AVG und SUM

AVG und SUM können für numerische Daten verwendet werden.

```
SELECT AVG(salary), MAX(salary),  
       MIN(salary), SUM(salary)  
FROM   employees  
WHERE  job_id LIKE '%REP%';
```

	AVG(SALARY)	MAX(SALARY)	MIN(SALARY)	SUM(SALARY)
1	8150	11000	6000	32600

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie können die Funktionen AVG, SUM, MIN und MAX für Spalten mit numerischen Daten verwenden. Das Beispiel auf der Folie zeigt den Durchschnitt, den höchsten Wert, den niedrigsten Wert und die Summe der Monatsgehälter aller Sales Representatives.

## Funktionen MIN und MAX

MIN und MAX können für numerische Datentypen sowie für Zeichen- und Datumsdatentypen verwendet werden.

```
SELECT MIN(hire_date), MAX(hire_date)
FROM employees;
```

	MIN(HIRE_DATE)	MAX(HIRE_DATE)
1	13-JAN-01	29-JAN-08

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie können die Funktionen MAX und MIN für numerische Datentypen sowie für Zeichen- und Datumsdatentypen verwenden. Im Beispiel auf der Folie werden der Mitarbeiter mit der kürzesten und der Mitarbeiter mit der längsten Beschäftigungsdauer angezeigt.

Das folgende Beispiel zeigt die Nachnamen der Mitarbeiter, die in der alphabetischen Liste aller Mitarbeiter an erster und an letzter Stelle stehen:

```
SELECT MIN(last_name), MAX(last_name)
FROM employees;
```

**Hinweis:** Die Funktionen AVG, SUM, VARIANCE und STDDEV können nur mit numerischen Datentypen verwendet werden. MAX und MIN können nicht mit den Datentypen LOB und LONG verwendet werden.



## Funktion COUNT

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

1

```
SELECT COUNT ( * )  
FROM employees  
WHERE department_id = 50;
```

	COUNT(*)
1	5

COUNT ( *expr* ) gibt die Anzahl der Zeilen mit Nicht-Nullwerten für *expr* zurück:

2

```
SELECT COUNT (commission_pct)  
FROM employees  
WHERE department_id = 50;
```

	COUNT(COMMISSION_PCT)
1	0

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Für die Funktion COUNT gibt es drei Formate:

- COUNT ( \* )
- COUNT ( *expr* )
- COUNT (DISTINCT *expr* )

COUNT ( \* ) gibt die Anzahl der Zeilen aus einer Tabelle zurück, die die Kriterien der Anweisung SELECT erfüllen, einschließlich mehrfach vorhandener Zeilen und Zeilen mit einem Nullwert in einer Spalte. Wenn in der Anweisung SELECT eine WHERE-Klausel angegeben wird, gibt COUNT ( \* ) die Anzahl der Zeilen zurück, die die Bedingung in der Klausel WHERE erfüllen.

Im Gegensatz dazu gibt COUNT ( *expr* ) die Anzahl der Nicht-Nullwerte in der durch *expr* angegebenen Spalte zurück.

COUNT (DISTINCT *expr* ) gibt die Anzahl der eindeutigen Nicht-Nullwerte in der durch *expr* angegebenen Spalte zurück.

### Beispiele

1. Das Beispiel auf der Folie zeigt die Anzahl der Mitarbeiter aus Abteilung 50.
2. Das Beispiel auf der Folie zeigt die Anzahl der provisionsberechtigten Mitarbeiter aus Abteilung 50.

## Schlüsselwort DISTINCT

- `COUNT(DISTINCT expr)` gibt die Anzahl der eindeutigen Nicht-Nullwerte von `expr` zurück.
- Anzahl der eindeutigen Abteilungsnummern aus der Tabelle `EMPLOYEES` anzeigen:

```
SELECT COUNT(DISTINCT department_id)
FROM employees;
```

	COUNT(DISTINCTDEPARTMENT_ID)
1	7

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Das Schlüsselwort `DISTINCT` sorgt dafür, dass doppelte Werte in einer Spalte nicht gezählt werden.

Das Beispiel auf der Folie zeigt die Anzahl der eindeutigen Abteilungsnummern aus der Tabelle `EMPLOYEES` an.

# Gruppenfunktionen und Nullwerte

Gruppenfunktionen ignorieren Nullwerte in der Spalte:

1 `SELECT AVG (commission_pct)  
FROM employees;`

AVG(COMMISSION_PCT)	
1	0.2125

Die Funktion NVL zwingt Gruppenfunktionen, Nullwerte zu berücksichtigen:

2 `SELECT AVG (NVL (commission_pct, 0))  
FROM employees;`

AVG(NVL(COMMISSION_PCT,0))	
1	0.0425

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Alle Gruppenfunktionen ignorieren Nullwerte in der Spalte.

Die Funktion NVL zwingt Gruppenfunktionen jedoch, Nullwerte einzubeziehen.

## Beispiele

1. Der Durchschnitt wird *ausschließlich* auf Basis der Zeilen in der Tabelle berechnet, für die die Spalte COMMISSION\_PCT einen gültigen Wert enthält. Zur Berechnung des durchschnittlichen Provisionsbetrags wird die Gesamtsumme der an alle Mitarbeiter gezahlten Provisionen durch die Anzahl der provisionsberechtigten Mitarbeiter (vier) dividiert.
2. Der Durchschnitt wird auf Basis *aller* Zeilen in der Tabelle berechnet, unabhängig davon, ob die Spalte COMMISSION\_PCT Nullwerte enthält. Zur Berechnung des durchschnittlichen Provisionsbetrags wird die Gesamtsumme der an alle Mitarbeiter gezahlten Provisionen durch die Gesamtzahl der Mitarbeiter im Unternehmen (20) dividiert.

# Lektionsagenda

- Gruppenfunktionen:
  - Typen und Syntax
  - AVG, SUM, MIN, MAX, COUNT
  - Schlüsselwort `DISTINCT` in Gruppenfunktionen
  - `NULL`-Werte in Gruppenfunktionen
- Zeilen gruppieren:
  - Klausel `GROUP BY`
  - Klausel `HAVING`
- Gruppenfunktionen verschachteln

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

# Datengruppen erstellen

EMPLOYEES

	DEPARTMENT_ID	SALARY
1	10	4400
2	20	13000
3	20	6000
4	50	2500
5	50	2600
6	50	3100
7	50	3500
8	50	5800
9	60	9000
10	60	6000
11	60	4200
12	80	11000
13	80	8600
...		
18	110	8300
19	110	12000
20	(null)	7000

4400

9500

3500

6400

10033

Durchschnittsgehalt in  
Tabelle EMPLOYEES für  
jede Abteilung

	DEPARTMENT_ID	AVG(SALARY)
1	(null)	7000
2	20	9500
3	90	19333.333333333333...
4	110	10150
5	50	3500
6	80	10033.333333333333...
7	10	4400
8	60	6400

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Bisher haben alle Gruppenfunktionen die Tabelle als eine große Gruppe von Informationen behandelt. Manchmal ist jedoch eine Aufteilung der Tabelle in kleinere Informationsgruppen erforderlich. Dies erfolgt mit der Klausel `GROUP BY`.

## Datengruppen erstellen – Syntax der Klausel GROUP BY

Mit der Klausel `GROUP BY` lassen sich die Zeilen einer Tabelle in kleinere Gruppen aufteilen.

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

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit der Klausel `GROUP BY` teilen Sie die Zeilen einer Tabelle in Gruppen auf. Anschließend können Sie mit den Gruppenfunktionen Aggregatinformationen für jede Gruppe zurückgeben. Für die Syntax gilt:

`group_by_expression`      Gibt die Spalten an, deren Werte die Basis für die Gruppierung der Zeilen darstellen

### Richtlinien

- Wenn Sie in einer `SELECT`-Klausel eine Gruppenfunktion angeben, können Sie nicht gleichzeitig eine einzelne Spalte wählen, *es sei denn*, die betreffende Spalte wird in der Klausel `GROUP BY` angegeben. Wenn Sie die Spaltenliste nicht in der Klausel `GROUP BY` angeben, erhalten Sie eine Fehlermeldung.
- Mit der Klausel `WHERE` können Sie Zeilen ausschließen, bevor Sie die übrigen Zeilen in Gruppen aufteilen.
- Sie können `column` durch einen Ausdruck in der Anweisung `SELECT` ersetzen.
- Geben Sie die *Spalten* in der Klausel `GROUP BY` an.
- In der Klausel `GROUP BY` können Sie keine Spaltenaliasnamen verwenden.

## Klausel GROUP BY

Alle Spalten aus der `SELECT`-Liste, die nicht in Gruppenfunktionen enthalten sind, müssen in der Klausel `GROUP BY` angegeben werden.

```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id ;
```

[illegible]

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Vergewissern Sie sich bei Verwendung der Klausel `GROUP BY`, dass alle Spalten aus der `SELECT`-Liste, die nicht in Gruppenfunktionen enthalten sind, in der Klausel `GROUP BY` angegeben werden. Das Beispiel auf der Folie zeigt die Abteilungsnummer und das Durchschnittsgehalt für jede Abteilung an. Die Anweisung `SELECT`, die eine `GROUP BY`-Klausel enthält, wird folgendermaßen ausgewertet:

- Die Klausel `SELECT` gibt die abzurufenden Spalten wie folgt an:
  - Abteilungsnummer in der Tabelle `EMPLOYEES`
  - Durchschnitt aller Gehälter für die in der Klausel `GROUP BY` angegebene Gruppe
- Die Klausel `FROM` gibt die Tabellen an, auf die die Datenbank zugreifen soll: die Tabelle `EMPLOYEES`.
- Die Klausel `WHERE` gibt die abzurufenden Zeilen an. Da in diesem Beispiel keine `WHERE`-Klausel vorhanden ist, werden standardmäßig alle Zeilen abgerufen.
- Die Klausel `GROUP BY` gibt an, wie die Zeilen gruppiert werden sollen. Die Zeilen werden nach Abteilungsnummer gruppiert, sodass die auf die Spalte "salary" angewendete Funktion `AVG` das Durchschnittsgehalt für jede Abteilung berechnet.

**Hinweis:** Wenn die Abfrageergebnisse in auf- oder absteigender Reihenfolge sortiert werden sollen, nehmen Sie die Klausel `ORDER BY` in die Abfrage auf.

## Klausel GROUP BY

Die in der Klausel GROUP BY angegebene Spalte muss nicht in der SELECT-Liste enthalten sein.

```
SELECT    AVG(salary)
FROM      employees
GROUP BY  department_id ;
```

[illegible]

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die in der Klausel `GROUP BY` angegebene Spalte muss nicht in der Klausel `SELECT` enthalten sein. Beispiel: Die Anweisung `SELECT` auf der Folie zeigt die Durchschnittsgehälter für jede Abteilung ohne die entsprechenden Abteilungsnummern an. Allerdings sind die Ergebnisse ohne Angabe der Abteilungsnummer nicht sehr aussagekräftig.

Sie können die Gruppenfunktion auch in der Klausel `ORDER BY` verwenden:

```
SELECT    department_id, AVG(salary)
FROM      employees
GROUP BY  department_id
ORDER BY  AVG(salary);
```



## Nach mehreren Spalten gruppieren

EMPLOYEES

	DEPARTMENT_ID	JOB_ID	SALARY
1	10	AD_ASST	4400
2	20	MK_MAN	13000
3	20	MK_REP	6000
4	50	ST_CLERK	2500
5	50	ST_CLERK	2600
6	50	ST_CLERK	3100
7	50	ST_CLERK	3500
8	50	ST_MAN	5800
9	60	IT_PROG	9000
10	60	IT_PROG	6000
11	60	IT_PROG	4200
12	80	SA_REP	11000
13	80	SA_REP	8600
14	80	SA_MAN	10500
...			
19	110	AC_MGR	12000
20	(null)	SA_REP	7000

Gehälter in der Tabelle EMPLOYEES  
nach Abteilung gruppiert für jede  
Tätigkeit addieren

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	110	AC_ACCOUNT	8300
2	110	AC_MGR	12008
3	10	AD_ASST	4400
4	90	AD PRES	24000
5	90	AD_VP	34000
6	60	IT_PROG	19200
7	20	MK_MAN	13000
8	20	MK_REP	6000
9	80	SA_MAN	10500
10	80	SA_REP	19600
11	(null)	SA_REP	7000
12	50	ST_CLERK	11700
13	50	ST_MAN	5800

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Manchmal ist es erforderlich, Ergebnisse für Gruppen innerhalb von Gruppen anzuzeigen. Die Folie zeigt einen Bericht, der die Summe der in den einzelnen Abteilungen für jede Tätigkeitsbezeichnung gezahlten Gehälter anzeigt.

Die Tabelle EMPLOYEES wird zuerst nach Abteilungsnummer und dann innerhalb dieser Gruppe nach Tätigkeitsbezeichnung gruppiert. Beispiel: Die vier Stock Clerks in Abteilung 50 werden zu einer Gruppe zusammengefasst, und für diese Gruppe wird ein einziges Ergebnis (Gesamtgehalt) berechnet.

Die folgende Anweisung SELECT gibt die auf der Folie gezeigten Ergebnisse zurück:

```
SELECT department_id, job_id, sum(salary)
FROM employees
GROUP BY department_id, job_id
ORDER BY job_id;
```

## Klausel GROUP BY über mehrere Spalten

```
SELECT department_id, job_id, SUM(salary)
FROM employees
WHERE department_id > 40
GROUP BY department_id, job_id
ORDER BY department_id;
```

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	50	ST_CLERK	11700
2	50	ST_MAN	5800
3	60	IT_PROG	19200
4	80	SA_MAN	10500
5	80	SA_REP	19600
6	90	AD_PRES	24000
7	90	AD_VP	34000
8	110	AC_ACCOUNT	8300
9	110	AC_MGR	12008

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie können Aggregatinformationen für Gruppen und Untergruppen zurückgeben, indem Sie mehrere Spalten in der Klausel `GROUP BY` angeben. Die Klausel `GROUP BY` gruppiert die Zeilen, garantiert jedoch nicht die Reihenfolge der Ergebnismenge. Um die Reihenfolge der Gruppen festzulegen, geben Sie die Klausel `ORDER BY` an.

Im Beispiel auf der Folie wird die Anweisung `SELECT`, die eine `GROUP BY`-Klausel enthält, wie folgt ausgewertet:

- Die Klausel `SELECT` gibt die abzurufenden Spalten an:
  - `DEPARTMENT_ID` in der Tabelle `EMPLOYEES`
  - `JOB_ID` in der Tabelle `EMPLOYEES`
  - Summe aller Gehälter für die in der Klausel `GROUP BY` angegebene Gruppe
- Die Klausel `FROM` gibt die Tabellen an, auf die die Datenbank zugreifen soll: die Tabelle `EMPLOYEES`.
- Die Klausel `WHERE` beschränkt die Ergebnismenge auf die Zeilen mit einer Abteilungsnummer (`DEPARTMENT_ID`) größer als 40.
- Die Klausel `GROUP BY` gibt an, wie die resultierenden Zeilen gruppiert werden sollen:
  - Zunächst werden die Zeilen nach Abteilungsnummer gruppiert.
  - Anschließend werden die Zeilen innerhalb der Abteilungsnummergruppen nach der Tätigkeits-ID (`JOB ID`) gruppiert.
- Die Klausel `ORDER BY` sortiert die Ergebnisse nach Abteilungsnummer.

**Hinweis:** Die Funktion `SUM` wird für alle Tätigkeits-IDs in der Ergebnismenge der einzelnen Abteilungsnummergruppen auf die Spalte "salary" angewendet. Die Zeile `SA_REP` wird nicht zurückgegeben. Die Abteilungsnummer für diese Zeile ist `NULL` und erfüllt somit die Bedingung `WHERE` nicht.

## Unzulässige Abfragen mit Gruppenfunktionen

Alle Spalten oder Ausdrücke in der `SELECT`-Liste, die keine Aggregatfunktion sind, müssen in der Klausel `GROUP BY` angegeben werden:

```
SELECT department_id, COUNT(last_name)
FROM employees;
```

ORA-00937: not a single-group group function  
00937. 00000 - "not a single-group group function"

Um die Nachnamen für die einzelnen  
Abteilungsnummern zu zählen, eine  
`GROUP BY`-Klausel hinzufügen

```
SELECT department_id, job_id, COUNT(last_name)
FROM employees
GROUP BY department_id;
```

ORA-00979: not a GROUP BY expression  
00979. 00000 - "not a GROUP BY expression"

Entweder `job_id` in Klausel `GROUP BY`  
einfügen oder Spalte `job_id` aus  
`SELECT`-Liste entfernen

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wenn Sie in einer `SELECT`-Anweisung einzelne Elemente (`DEPARTMENT_ID`) und Gruppenfunktionen (`COUNT`) mischen, müssen Sie die Elemente (in diesem Fall `DEPARTMENT_ID`) in einer `GROUP BY`-Klausel angeben. Wenn die Klausel `GROUP BY` fehlt, wird die Fehlermeldung "not a single-group group function" angezeigt, und ein Sternchen (\*) verweist auf die fehlerhafte Spalte. Den Fehler im ersten Beispiel auf der Folie beheben Sie, indem Sie die Klausel `GROUP BY` hinzufügen:

```
SELECT department_id, count(last_name)
FROM employees
GROUP BY department_id;
```

Alle Spalten oder Ausdrücke aus der `SELECT`-Liste, die keine Aggregatfunktion sind, müssen in der Klausel `GROUP BY` angegeben werden. Im zweiten Beispiel auf der Folie befindet sich `JOB_ID` nicht in der Klausel `GROUP BY` und wird von keiner Gruppenfunktion verwendet. Daher wird der Fehler "not a GROUP BY expression" ausgegeben. Den Fehler im zweiten Beispiel auf der Folie beheben Sie, indem Sie `JOB_ID` in der Klausel `GROUP BY` hinzufügen.

```
SELECT department_id, job_id, COUNT(last_name)
FROM employees
GROUP BY department_id, job_id;
```

## Unzulässige Abfragen mit Gruppenfunktionen

- Gruppen nicht mit der Klausel `WHERE` einschränken
- Gruppen mit der Klausel `HAVING` einschränken
- In der Klausel `WHERE` dürfen keine Gruppenfunktionen verwendet werden.

```
SELECT department_id, AVG(salary)
FROM employees
WHERE AVG(salary) > 8000
GROUP BY department_id;
```

```
ORA-00934: group function is not allowed here
00934. 00000 - "group function is not allowed here"
*Cause:
*Action:
Error at Line: 3 Column: 9
```

Gruppen dürfen  
nicht mit der  
Klausel `WHERE`  
eingeschränkt  
werden.

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Klausel `WHERE` ist zur Einschränkung von Gruppen unzulässig. Die Anweisung `SELECT` im Beispiel auf der Folie führt zu einem Fehler, weil die Klausel `WHERE` die Anzeige der Durchschnittsgehälter auf die Abteilungen einschränkt, deren Durchschnittsgehalt größer als \$ 8.000 ist. Sie können den Fehler im Beispiel jedoch korrigieren, indem Sie die Gruppen mit der Klausel `HAVING` einschränken:

```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id
HAVING AVG(salary) > 8000;
```

# Gruppenergebnisse einschränken

EMPLOYEES

	DEPARTMENT_ID	SALARY
1	10	4400
2	20	13000
3	20	6000
4	50	2500
5	50	2600
6	50	3100
7	50	3500
8	50	5800
9	60	9000
10	60	6000
11	60	4200
12	80	11000
13	80	8600
...		
18	110	8300
19	110	12000
20	(null)	7000

Höchstgehälter pro  
Abteilung, die  
\$ 10.000 überschreiten

	DEPARTMENT_ID	MAX(SALARY)
1	20	13000
2	90	24000
3	110	12000
4	80	11000

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Einschränkung von Gruppen mit der Klausel `HAVING` erfolgt auf dieselbe Weise wie die Einschränkung gewählter Zeilen mit der Klausel `WHERE`. Um die höchsten Gehälter der Abteilungen zu ermitteln, die \$ 10.000 überschreiten, gehen Sie wie folgt vor:

1. Ermitteln Sie für jede Abteilung das Durchschnittsgehalt, indem Sie eine Gruppierung nach Abteilungsnummern durchführen.
2. Schränken Sie die Gruppen auf diejenigen Abteilungen ein, deren Höchstgehalt \$ 10.000 überschreitet.

## Gruppenergebnisse mit der Klausel HAVING einschränken

Mit der Klausel `HAVING` schränkt der Oracle-Server Gruppen wie folgt ein:

1. Zeilen werden gruppiert.
2. Gruppenfunktion wird angewendet.
3. Gruppen, die die Klausel `HAVING` erfüllen, werden angezeigt.

```
SELECT    column, group_function
FROM      table
[WHERE    condition]
[GROUP BY group_by_expression]
[HAVING   group_condition]
[ORDER BY column];
```

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit der Klausel `HAVING` geben Sie an, welche Gruppen angezeigt werden sollen. Sie schränken die Gruppen auf Basis von Aggregatinformationen weiter ein.

In der Syntax schränkt *group\_condition* die zurückgegebenen Zeilengruppen auf die Gruppen ein, die die angegebene Bedingung erfüllen.

Der Oracle-Server führt die folgenden Schritte aus, wenn Sie die Klausel `HAVING` verwenden:

1. Die Zeilen werden gruppiert.
2. Die Gruppenfunktion wird auf die Gruppe angewendet.
3. Die Gruppen, die die Kriterien in der Klausel `HAVING` erfüllen, werden angezeigt.

Die Klausel `HAVING` kann vor der Klausel `GROUP BY` stehen. Es ist jedoch logischer, die Klausel `GROUP BY` zuerst anzugeben. Die Gruppen werden gebildet und die Gruppenfunktionen berechnet, bevor die Klausel `HAVING` auf die in der `SELECT`-Liste aufgeführten Gruppen angewendet wird.

**Hinweis:** Die Klausel `WHERE` schränkt Zeilen ein, während die Klausel `HAVING` Gruppen einschränkt.

## Klausel HAVING

```
SELECT department_id, MAX(salary)
FROM employees
GROUP BY department_id
HAVING MAX(salary)>10000 ;
```

	DEPARTMENT_ID	MAX(SALARY)
1	90	24000
2	20	13000
3	110	12008
4	80	11000

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf der Folie werden die Abteilungsnummern und Höchstgehälter für diejenigen Abteilungen angezeigt, deren Höchstgehalt \$ 10.000 überschreitet.

Sie können die Klausel `GROUP BY` auch ohne Gruppenfunktion in der `SELECT`-Liste verwenden. Wenn Sie Zeilen auf Basis des Ergebnisses einer Gruppenfunktion einschränken möchten, müssen Sie sowohl eine `GROUP BY`-Klausel als auch die Klausel `HAVING` angeben.

Das folgende Beispiel zeigt die Abteilungsnummern und Durchschnittsgehälter derjenigen Abteilungen an, deren Höchstgehalt \$ 10.000 überschreitet:

```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id
HAVING max(salary)>10000;
```

## Klausel HAVING

```
SELECT  job_id, SUM(salary) PAYROLL
FROM    employees
WHERE   job_id NOT LIKE '%REP%'
GROUP BY job_id
HAVING  SUM(salary) > 13000
ORDER BY SUM(salary);
```

	JOB_ID	PAYROLL
1	IT_PROG	19200
2	AD_PRES	24000
3	AD_VP	34000

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf der Folie werden Tätigkeits-ID (JOB\_ID) und monatliches Gesamtgehalt für alle Tätigkeiten angezeigt, deren Gesamtgehaltsliste \$ 13.000 überschreitet. Sales Representatives werden nicht berücksichtigt. Die ausgegebene Liste wird nach dem monatlichen Gesamtgehalt sortiert.



# Lektionsagenda

- Gruppenfunktionen:
  - Typen und Syntax
  - AVG, SUM, MIN, MAX, COUNT
  - Schlüsselwort `DISTINCT` in Gruppenfunktionen
  - `NULL`-Werte in Gruppenfunktionen
- Zeilen gruppieren:
  - Klausel `GROUP BY`
  - Klausel `HAVING`
- Gruppenfunktionen verschachteln

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Gruppenfunktionen verschachteln

Höchstes Durchschnittsgehalt anzeigen:

```
SELECT MAX(AVG(salary))
FROM employees
GROUP BY department_id;
```

[illegible]

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Gruppenfunktionen können in bis zu zwei Funktionen verschachtelt werden. Das Beispiel auf der Folie berechnet das Durchschnittsgehalt für jede `DEPARTMENT_ID` und zeigt dann das höchste Durchschnittsgehalt an.

Beim Verschachteln von Funktionen ist die Klausel `GROUP BY` obligatorisch.

## Quiz

Welche zwei Richtlinien gelten für Gruppenfunktionen und die Klausel `GROUP BY`?

- a. In der Klausel `GROUP BY` dürfen keine Spaltenaliasnamen verwendet werden.
- b. Die Spalte `GROUP BY` muss in der Klausel `SELECT` enthalten sein.
- c. Mit einer `WHERE`-Klausel können Sie Zeilen ausschließen, bevor Sie die übrigen Zeilen in Gruppen aufteilen.
- d. Die Klausel `GROUP BY` gruppiert Zeilen und stellt die Reihenfolge der Ergebnismenge sicher.
- e. Um Gruppenfunktionen in `SELECT`-Klauseln aufzunehmen, müssen Sie eine `GROUP BY`-Klausel verwenden.

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

**Richtige Antworten: a, c**

# Zusammenfassung

In dieser Lektion haben Sie Folgendes gelernt:

- Gruppenfunktionen COUNT, MAX, MIN, SUM, AVG, LISTAGG, STDDEV und VARIANCE verwenden
- Abfragen mit der Klausel GROUP BY erstellen
- Abfragen mit der Klausel HAVING erstellen

```
SELECT      column, group_function
FROM        table
[WHERE      condition]
[GROUP BY  group_by_expression]
[HAVING    group_condition]
[ORDER BY  column];
```

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In SQL stehen verschiedene Gruppenfunktionen wie AVG, COUNT, MAX, MIN, SUM, LISTAGG, STDDEV und VARIANCE zur Verfügung.

Sie können Untergruppen erstellen, indem Sie die Klausel GROUP BY verwenden. Darüber hinaus lassen sich Gruppen mit der Klausel HAVING einschränken.

Sie geben die Klauseln HAVING und GROUP BY in Anweisungen nach der Klausel WHERE an. Die Reihenfolge der Klauseln nach der Klausel WHERE ist dabei irrelevant. Beide Klauseln können zuerst angegeben werden. Wichtig ist lediglich, dass sie jeweils der Klausel WHERE folgen. Geben Sie die Klausel ORDER BY am Ende an.

Der Oracle-Server wertet die Klauseln in der folgenden Reihenfolge aus:

1. Wenn die Anweisung eine WHERE-Klausel enthält, ermittelt der Server die infrage kommenden Zeilen.
2. Der Server identifiziert die in der Klausel GROUP BY angegebenen Gruppen.
3. Die Klausel HAVING schränkt die Ergebnismengen weiter ein, indem Gruppen, die die Kriterien in der Klausel HAVING nicht erfüllen, ausgeschlossen werden.

**Hinweis:** Eine vollständige Liste der Gruppenfunktionen finden Sie in der *Oracle Database SQL Language Reference* für Oracle Database 12c.

## Übungen zu Lektion 6 – Überblick

Diese Übung behandelt folgende Themen:

- Abfragen mit Gruppenfunktionen erstellen
- Nach Zeilen gruppieren, um mehrere Ergebnisse zu erhalten
- Gruppen mit der Klausel `HAVING` einschränken

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In dieser Übung verwenden Sie Gruppenfunktionen und wählen Datengruppen.





# **Daten aus mehreren Tabellen mit Joins anzeigen**

**ORACLE®**

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- `SELECT`-Anweisungen erstellen, um über Equi Joins oder Non-Equi Joins auf Daten aus mehreren Tabellen zuzugreifen
- Tabellen über einen Self Join mit sich selbst verknüpfen
- Mit `OUTER`-Joins Daten anzeigen, die eine Join-Bedingung nicht erfüllen
- Kartesisches Produkt aller Zeilen aus zwei oder mehr Tabellen erzeugen

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In dieser Lektion lernen Sie, wie Sie Daten aus mehreren Tabellen abrufen. Um Informationen aus mehreren Tabellen anzuzeigen, verwenden Sie *Joins*. Mit *Joins* lassen sich Tabellen für die Anzeige von Informationen miteinander verknüpfen.

**Hinweis:** Weitere Informationen zu Joins finden Sie in der *Oracle Database SQL Language Reference* für Oracle Database 12c im Abschnitt "SQL Queries and Subqueries: Joins".



# Lektionsagenda

- Typen von JOINS und ihre Syntax
  - Natural Joins
  - Joins mit der Klausel USING erstellen
  - Joins mit der Klausel ON erstellen
  - Self Joins
  - Non-Equi Joins
  - OUTER-Joins:
    - LEFT OUTER-Joins
    - RIGHT OUTER-Joins
    - FULL OUTER-Joins
  - Kartesische Produkte
    - Cross Joins

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.


## Daten aus mehreren Tabellen abrufen

EMPLOYEES

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	JOB_ID
1	100	Steven	King	AD_PRES
2	101	Neena	Kochhar	AD_VP
3	102	Lex	De Haan	AD_VP
4	103	Alexander	Hunold	IT_PROG
5	104	Bruce	Ernst	IT_PROG
6	105	David	Austin	IT_PROG
7	106	Valli	Pataballa	IT_PROG
8	107	Diana	Lorentz	IT_PROG
9	108	Nancy	Greenberg	FI_MGR
10	109	Daniel	Faviet	FI_ACCOUNT

JOBS

	JOB_ID	JOB_TITLE
1	AD_PRES	President
2	AD_VP	Administration Vice President
3	AD_ASST	Administration Assistant
4	FI_MGR	Finance Manager
5	FI_ACCOUNT	Accountant
6	AC_MGR	Accounting Manager
7	AC_ACCOUNT	Public Accountant
8	SA_MAN	Sales Manager
9	SA_REP	Sales Representative



	EMPLOYEE_ID	JOB_ID	JOB_TITLE
1	206	AC_ACCOUNT	Public Accountant
2	205	AC_MGR	Accounting Manager
3	200	AD_ASST	Administration Assistant
4	100	AD_PRES	President
5	101	AD_VP	Administration Vice President
6	102	AD_VP	Administration Vice President
7	109	FI_ACCOUNT	Accountant

...

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In manchen Fällen benötigen Sie Daten aus mehreren Tabellen. Im Beispiel auf der Folie zeigt der Bericht Daten aus zwei getrennten Tabellen an:

- Personalnummern (EMPLOYEE\_ID) sind in der Tabelle EMPLOYEES gespeichert.
- Tätigkeits-IDs (JOB\_ID) sind in der Tabelle EMPLOYEES und in der Tabelle JOBS gespeichert.
- Die Tätigkeitsbezeichnungen sind in der Tabelle JOBS gespeichert.

Um den Bericht zu erstellen, verknüpfen Sie die Tabellen EMPLOYEES und JOBS und rufen Daten aus beiden Tabellen ab.

# Typen von Joins

Zu den Joins, die dem Standard SQL:1999 entsprechen, zählen:

- Natural Joins mit der Klausel `NATURAL JOIN`
- Joins mit der Klausel `USING`
- Joins mit der Klausel `ON`
- OUTER-Joins:
  - `LEFT OUTER JOIN`
  - `RIGHT OUTER JOIN`
  - `FULL OUTER JOIN`
- Cross Joins

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Um Tabellen zu verknüpfen, können Sie eine dem Standard SQL:1999 entsprechende Join-Syntax verwenden.

**Hinweis:**

- Vor Oracle9i hat sich die Oracle-spezifische Join-Syntax von den ANSI-(American National Standards Institute-)Standards unterschieden. Die SQL:1999-konforme Join-Syntax bietet jedoch keine Performancevorteile gegenüber der Oracle-Syntax aus früheren Versionen.
- Auf der folgenden Folie wird die SQL:1999-Join-Syntax beschrieben.

## Tabellen mithilfe der SQL:1999-Syntax verknüpfen

Mithilfe von Joins Daten aus mehreren Tabellen abfragen:

```
SELECT    table1.column, table2.column
FROM      table1
[NATURAL JOIN table2] |
[JOIN table2 USING (column_name)] |
[JOIN table2 ON (table1.column_name = table2.column_name)] |
[LEFT|RIGHT|FULL OUTER JOIN table2
ON (table1.column_name = table2.column_name)] |
[CROSS JOIN table2];
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Für die Syntax gilt:

- `table1.column` gibt die Tabelle und die Spalte an, aus der Daten abgerufen werden.
- `NATURAL JOIN` verknüpft zwei Tabellen basierend auf demselben Spaltennamen.
- `JOIN table2 USING column_name` führt einen Equi Join basierend auf dem Spaltennamen aus.
- `JOIN table2 ON table1.column_name = table2.column_name` führt einen Equi Join basierend auf der Bedingung in der Klausel `ON` aus.
- `LEFT/RIGHT/FULL OUTER` führt OUTER-Joins aus.
- `CROSS JOIN` gibt das kartesische Produkt der beiden Tabellen an.

Weitere Informationen finden Sie in der *Oracle Database SQL Language Reference* für Oracle Database 12c im Abschnitt "SELECT".

# Lektionsagenda

- Typen von JOINS und ihre Syntax
- Natural Joins
- Joins mit der Klausel USING erstellen
- Joins mit der Klausel ON erstellen
- Self Joins
- Non-Equi Joins
- OUTER-Joins:
  - LEFT OUTER-Joins
  - RIGHT OUTER-Joins
  - FULL OUTER-Joins
- Kartesische Produkte
  - Cross Joins

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Natural Joins erstellen

- Die Klausel `NATURAL JOIN` basiert auf allen Spalten, die in zwei Tabellen denselben Namen haben.
- Sie wählt diejenigen Zeilen aus den beiden Tabellen, die in allen gemeinsamen Spalten übereinstimmende Werte aufweisen.
- Wenn die Spalten mit denselben Namen unterschiedliche Datentypen haben, wird ein Fehler zurückgegeben.

```
SELECT * FROM table1 NATURAL JOIN table2;
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie können Tabellen auf der Basis von Spalten mit gleichem Namen und gleichem Datentyp automatisch verknüpfen. Hierzu verwenden Sie die Schlüsselwörter `NATURAL JOIN`.

**Hinweis:** Joins sind nur für Spalten möglich, die in beiden Tabellen denselben Namen und denselben Datentyp aufweisen. Wenn die Spalten denselben Namen haben, ihre Datentypen sich jedoch unterscheiden, verursacht die Syntax `NATURAL JOIN` einen Fehler.

## Datensätze mit Natural Joins abrufen

```
SELECT employee_id, first_name, job_id, job_title
from employees NATURAL JOIN jobs;
```

	EMPLOYEE_ID	FIRST_NAME	JOB_ID	JOB_TITLE
1	100	Steven	AD_PRES	President
2	101	Neena	AD_VP	Administration Vice President
3	102	Lex	AD_VP	Administration Vice President
4	103	Alexander	IT_PROG	Programmer
5	104	Bruce	IT_PROG	Programmer
6	105	David	IT_PROG	Programmer
7	106	Valli	IT_PROG	Programmer
8	107	Diana	IT_PROG	Programmer
9	108	Nancy	FI_MGR	Finance Manager
10	109	Daniel	FI_ACCOUNT	Accountant
11	110	John	FI_ACCOUNT	Accountant

...

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf der Folie werden die Tabellen `JOBS` und `EMPLOYEES` über einen Join basierend auf der Spalte `JOB_ID` verknüpft. Dies ist die einzige Spalte, die in beiden Tabellen denselben Namen trägt. Wären weitere übereinstimmende Spalten vorhanden, würde der Join auch diese Spalten berücksichtigen.

### Natural Joins mit einer `WHERE`-Klausel

Zusätzliche Einschränkungen für einen Natural Join können mit der Klausel `WHERE` implementiert werden. Im folgenden Beispiel werden nur die Zeilen für die Abteilungsnummern 20 und 50 ausgegeben:

```
SELECT department_id, department_name,
       location_id, city
FROM departments
NATURAL JOIN locations
WHERE department_id IN (20, 50);
```

## Joins mit der Klausel `USING` erstellen

- Bei mehreren Spalten mit gleichem Namen, aber unterschiedlichen Datentypen für den Equi Join die Spalten mit der Klausel `USING` angeben
- Klausel `USING` verwenden, um nur eine Spalte zu verknüpfen, wenn mehrere übereinstimmende Spalten existieren

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Natural Joins verknüpfen Tabellen anhand von allen Spalten mit gleichem Namen und Datentyp. Mit der Klausel `USING` können Sie vorgeben, dass nur bestimmte Spalten für einen Equi Join berücksichtigt werden sollen.



## Spaltennamen verknüpfen

EMPLOYEES

	EMPLOYEE_ID	DEPARTMENT_ID
1	200	10
2	201	20
3	202	20
4	205	110
5	206	110
6	100	90
7	101	90
8	102	90
9	103	60
10	104	60

...

Fremdschlüssel

DEPARTMENTS

	DEPARTMENT_ID	DEPARTMENT_NAME
1	10	Administration
2	20	Marketing
3	50	Shipping
4	60	IT
5	80	Sales
6	90	Executive
7	110	Accounting
8	190	Contracting

Primärschlüssel

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Um den Abteilungsnamen eines Mitarbeiters zu ermitteln, vergleichen Sie die Werte in der Spalte `DEPARTMENT_ID` der Tabelle `EMPLOYEES` mit den Werten der Spalte `DEPARTMENT_ID` in der Tabelle `DEPARTMENTS`. Die Beziehung zwischen den Tabellen `EMPLOYEES` und `DEPARTMENTS` ist ein *Equi Join*. Die Werte in der Spalte `DEPARTMENT_ID` müssen daher in beiden Tabellen gleich sein. Häufig sind bei dieser Art von Join Primärschlüssel- und Fremdschlüsselkomponenten beteiligt.

## Datensätze mit der Klausel USING abrufen

```
SELECT employee_id, last_name,  
       location_id, department_id  
FROM   employees JOIN departments  
USING (department_id);
```

	EMPLOYEE_ID	LAST_NAME	LOCATION_ID	DEPARTMENT_ID
1	200	Whalen	1700	10
2	201	Hartstein	1800	20
3	202	Fay	1800	20
4	144	Vargas	1500	50
5	143	Matos	1500	50
6	142	Davies	1500	50
7	141	Rajs	1500	50
8	124	Mourgos	1500	50
...				
18	206	Gietz	1700	110
19	205	Higgins	1700	110

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf der Folie werden die Spalten `DEPARTMENT_ID` in den Tabellen `EMPLOYEES` und `DEPARTMENTS` verknüpft und die `LOCATION_ID` der Abteilung des Mitarbeiters angezeigt.

## Mehrdeutige Spaltennamen eindeutig kennzeichnen

- Spaltennamen, die in mehreren Tabellen vorkommen, müssen durch ein Tabellenpräfix eindeutig gekennzeichnet werden.
- Tabellenpräfixe steigern die Geschwindigkeit beim Parsen von Anweisungen.
- Anstelle von vollständigen Tabellennamenpräfixen können Sie Tabellenaliasnamen verwenden.
- Tabellenaliasnamen verkürzen den Tabellennamen:
  - Kürzerer SQL-Code, weniger Speicherbedarf
- Spaltenaliasnamen dienen der Unterscheidung von Spalten, die identische Namen haben, aber in verschiedenen Tabellen vorkommen.

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wenn Sie zwei oder mehr Tabellen miteinander verknüpfen, müssen Sie die Spaltennamen durch den jeweiligen Tabellennamen kennzeichnen, um Mehrdeutigkeit zu vermeiden. Ohne das Tabellenpräfix könnte sich der Spaltenname `DEPARTMENT_ID` in der `SELECT`-Liste auf die entsprechende Spalte in der Tabelle `DEPARTMENTS` oder in der Tabelle `EMPLOYEES` beziehen. Daher müssen Sie vor Ausführung Ihrer Abfrage das Tabellenpräfix hinzufügen. Gibt es in den Tabellen keine identischen Spaltennamen, besteht keine Notwendigkeit, die Spalten durch Angabe des Tabellennamens eindeutig zu kennzeichnen. Tabellenpräfixe steigern jedoch die Geschwindigkeit beim Parsen von Anweisungen, weil sie dem Oracle-Server genau angeben, wo sich die Spalten befinden.

Allerdings kann die eindeutige Kennzeichnung von Spaltennamen durch Tabellennamen insbesondere bei langen Tabellennamen zeitaufwendig sein. Sie können stattdessen kürzere *Tabellenaliasnamen* verwenden. (Analog zu Aliasnamen für Spalten.) Tabellenaliasnamen verkürzen den SQL-Code und sind daher weniger speicherintensiv.

Der Tabellenname wird komplett angegeben. Danach folgt ein Leerzeichen und dann der Tabellenalias. Beispielsweise kann die Tabelle `EMPLOYEES` den Alias `e` und die Tabelle `DEPARTMENTS` den Alias `d` erhalten.

## Richtlinien

- Tabellenaliasnamen können bis zu 30 Zeichen lang sein, kürzere Aliasnamen sind jedoch besser.
- Tabellenaliasnamen, die in der Klausel `FROM` bestimmte Tabellennamen ersetzen, müssen auch in der gesamten `SELECT`-Anweisung anstelle des Tabellennamens verwendet werden.
- Wählen Sie möglichst aussagekräftige Tabellenaliasnamen.
- Tabellenaliasnamen gelten nur für die aktuelle `SELECT`-Anweisung.

## Tabellenaliasnamen mit der Klausel USING

- Spalten, die in einem NATURAL-Join oder einem Join mit einer USING-Klausel verwendet werden, dürfen nicht gekennzeichnet werden.
- Falls die gleiche Spalte auch an anderer Stelle in der SQL-Anweisung verwendet wird, dürfen Sie keinen Aliasnamen für sie angeben.

```
SELECT l.city, d.department_name
FROM   locations l JOIN departments d
USING (location_id)
WHERE d.location_id = 1400;
```

```
ORA-25154: column part of USING clause cannot have qualifier
25154. 00000 - "column part of USING clause cannot have qualifier"
*Cause:   Columns that are used for a named-join (either a NATURAL join
          or a join with a USING clause) cannot have an explicit qualifier.
*Action:  Remove the qualifier.
Error at Line: 4 Column: 6
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Beim Verknüpfen mit der Klausel `USING` können Sie keine Spalten kennzeichnen, die in der Klausel `USING` verwendet werden. Des Weiteren können Sie keinen Alias für Spalten angeben, die an anderer Stelle in der SQL-Anweisung verwendet werden. Beispiel: In der Abfrage auf der Folie sollten Sie für die Spalte `location_id` in der Klausel `WHERE` keinen Alias angeben, da die Spalte in der Klausel `USING` verwendet wird.

Für die in der Klausel `USING` referenzierten Spalten darf in der gesamten SQL-Anweisung kein Bezeichner (Tabellenname oder Alias) verwendet werden. Beispiel: Die folgende Anweisung ist gültig:

```
SELECT l.city, d.department_name
FROM   locations l JOIN departments d USING (location_id)
WHERE  location_id = 1400;
```

Den Spalten, die in beiden Tabellen vorkommen, jedoch nicht in der Klausel `USING` verwendet werden, muss ein Tabellenalias vorangestellt werden. Andernfalls wird die Fehlermeldung "column ambiguously defined" ausgegeben.

Bei der folgenden Anweisung ist `manager_id` sowohl in der Tabelle `employees` als auch in der Tabelle `departments` vorhanden. Wird nun für `manager_id` kein Tabellenalias als Präfix angegeben, wird der Fehler "column ambiguously defined" ausgegeben.

Die folgende Anweisung ist gültig:

```
SELECT first_name, e.department_name, d.manager_id
FROM   employees e JOIN departments d USING (department_id)
WHERE  department_id = 50;
```

## Joins mit der Klausel ON erstellen

- Die Join-Bedingung für Natural Joins ist im Prinzip ein Equi Join aller Spalten mit gleichem Namen.
- Mit der Klausel ON können Sie beliebige Bedingungen oder die zu verknüpfenden Spalten angeben.
- Die Join-Bedingung wird von anderen Suchkriterien getrennt.
- Die Klausel ON macht den Code verständlicher.

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit der Klausel ON geben Sie eine Join-Bedingung an. Auf diese Weise können Sie Join-Bedingungen getrennt von Such- oder Filterkriterien in der Klausel WHERE angeben.

## Datensätze mit der Klausel ON abrufen

```
SELECT e.employee_id, e.last_name, e.department_id,  
       d.department_id, d.location_id  
FROM   employees e JOIN departments d  
ON     (e.department_id = d.department_id);
```

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID_1	LOCATION_ID
1	200	Whalen	10	10	1700
2	201	Hartstein	20	20	1800
3	202	Fay	20	20	1800
4	124	Mourgos	50	50	1500
5	144	Vargas	50	50	1500
6	143	Matos	50	50	1500
7	142	Davies	50	50	1500
8	141	Rajs	50	50	1500
9	107	Lorentz	60	60	1400
10	104	Ernst	60	60	1400
11	103	Hunold	60	60	1400

...

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In diesem Beispiel werden die Spalten `DEPARTMENT_ID` in den Tabellen `EMPLOYEES` und `DEPARTMENTS` über die Klausel `ON` miteinander verknüpft. Alle Abteilungsnummern, die in der Tabelle `EMPLOYEES` und der Tabelle `DEPARTMENTS` übereinstimmen, werden zurückgegeben. Der Tabellenalias ist erforderlich, um die übereinstimmenden Spaltennamen zu kennzeichnen.

Sie können mit der Klausel `ON` auch Spalten mit unterschiedlichen Namen verknüpfen. Im Beispiel auf der Folie werden die verknüpften Spalten in Klammern gesetzt: `(e.department_id = d.department_id)`. Diese Klammern sind optional. Entsprechend wäre auch `ON e.department_id = d.department_id` korrekt.

**Hinweis:** Wenn Sie die Abfrage mit dem Symbol **Execute Statement** ausführen, verwendet SQL Developer das Suffix `"_1"`, um zwischen den beiden `department_id`-Spalten zu unterscheiden.

## 3-Way Joins erstellen

```
SELECT employee_id, city, department_name
FROM   employees e
JOIN   departments d
ON     d.department_id = e.department_id
JOIN   locations l
ON     d.location_id = l.location_id;
```

	EMPLOYEE_ID	CITY	DEPARTMENT_NAME
1	100	Seattle	Executive
2	101	Seattle	Executive
3	102	Seattle	Executive
4	103	Southlake	IT
5	104	Southlake	IT
6	107	Southlake	IT
7	124	South San Francisco	Shipping
8	141	South San Francisco	Shipping
9	142	South San Francisco	Shipping

...

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Ein 3-Way Join ist eine Verknüpfung von drei Tabellen. Der erste ausgeführte Join ist `EMPLOYEES JOIN DEPARTMENTS`. Die erste Join-Bedingung kann Spalten aus den Tabellen `EMPLOYEES` und `DEPARTMENTS` referenzieren, nicht aber Spalten aus der Tabelle `LOCATIONS`. Die zweite Join-Bedingung kann Spalten aus allen drei Tabellen referenzieren.

**Hinweis:** Das Codebeispiel auf der Folie kann auch mit der Klausel `USING` realisiert werden:

```
SELECT e.employee_id, l.city, d.department_name
FROM   employees e
JOIN   departments d
      USING (department_id)
JOIN   locations l
      USING (location_id);
```



## Zusätzliche Bedingungen in Joins anwenden

Mit der Klausel **AND** oder **WHERE** zusätzliche Bedingungen anwenden:

```
SELECT e.employee_id, e.last_name, e.department_id,  
       d.department_id, d.location_id  
FROM   employees e JOIN departments d  
ON     (e.department_id = d.department_id)  
AND    e.manager_id = 149 ;
```

oder

```
SELECT e.employee_id, e.last_name, e.department_id,  
       d.department_id, d.location_id  
FROM   employees e JOIN departments d  
ON     (e.department_id = d.department_id)  
WHERE  e.manager_id = 149 ;
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie können in einem Join zusätzliche Bedingungen anwenden.

Das dargestellte Beispiel führt einen Join für die Tabellen **EMPLOYEES** und **DEPARTMENTS** aus und zeigt darüber hinaus nur Mitarbeiter mit der Managernummer 149 an. Um der Klausel **ON** weitere Bedingungen hinzuzufügen, können Sie **AND**-Klauseln einfügen. Alternativ können Sie zusätzliche Bedingungen über eine **WHERE**-Klausel anwenden.

Beide Abfragen führen zum selben Ergebnis.

# Lektionsagenda

- Typen von JOINS und ihre Syntax
- Natural Joins
- Joins mit der Klausel USING erstellen
- Joins mit der Klausel ON erstellen
- **Self Joins**
- Non-Equi Joins
- OUTER-Joins:
  - LEFT OUTER-Joins
  - RIGHT OUTER-Joins
  - FULL OUTER-Joins
- Kartesische Produkte
  - Cross Joins

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Tabellen mit sich selbst verknüpfen

EMPLOYEES (WORKER)

EMPLOYEE_ID	LAST_NAME	MANAGER_ID
200	Whalen	101
201	Hartstein	100
202	Fay	201
205	Higgins	101
206	Gietz	205
100	King	(null)
101	Kochhar	100
102	De Haan	100
103	Hunold	102
104	Ernst	103

...

EMPLOYEES (MANAGER)

EMPLOYEE_ID	LAST_NAME
200	Whalen
201	Hartstein
202	Fay
205	Higgins
206	Gietz
100	King
101	Kochhar
102	De Haan
103	Hunold
104	Ernst

...

MANAGER\_ID in der Tabelle WORKER gleich  
EMPLOYEE\_ID in der Tabelle MANAGER

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Gelegentlich müssen Sie eine Tabelle mit sich selbst verknüpfen. Um für jeden Mitarbeiter den Namen des zuständigen Managers zu ermitteln, verknüpfen Sie die Tabelle `EMPLOYEES` mit sich selbst und führen damit einen Self Join durch. Beispiel: Um den Namen des Managers von "Ernst" abzurufen, gehen Sie wie folgt vor:

- Ernst durch Überprüfen der Namen in der Spalte `LAST_NAME` der Tabelle `EMPLOYEES` finden
- Managernummer für Ernst in der Spalte `MANAGER_ID` suchen: Die Nummer des Managers von Ernst ist 103.
- Namen des Managers mit der `EMPLOYEE_ID` 103 in der Spalte `LAST_NAME` suchen. Die Personalnummer für Hunold ist 103. Somit ist Hunold der Manager von Ernst.

In diesem Prozess durchsuchen Sie die Tabelle zweimal. Beim ersten Mal suchen Sie in der Spalte `LAST_NAME` den Wert Ernst und ermitteln in der Spalte `MANAGER_ID` den Wert 103. Im zweiten Durchgang suchen Sie in der Spalte `EMPLOYEE_ID` nach der Nummer 103 und ermitteln in der Spalte `LAST_NAME` den Wert Hunold.

## Self Joins mit der Klausel ON

```
SELECT worker.last_name emp, manager.last_name mgr
FROM   employees worker JOIN employees manager
ON     (worker.manager_id = manager.employee_id);
```

	EMP	MGR
1	Hunold	De Haan
2	Fay	Hartstein
3	Gietz	Higgins
4	Lorentz	Hunold
5	Ernst	Hunold
6	Zlotkey	King
7	Mourgos	King
8	Kochhar	King

...

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit der Klausel ON können Sie auch Spalten mit verschiedenen Namen innerhalb derselben Tabelle oder in verschiedenen Tabellen verknüpfen.

Das Beispiel auf der Folie ist ein Self Join der Tabelle `EMPLOYEES`, basierend auf den Spalten `EMPLOYEE_ID` und `MANAGER_ID`.

# Lektionsagenda

- Typen von JOINS und ihre Syntax
- Natural Joins
- Joins mit der Klausel USING erstellen
- Joins mit der Klausel ON erstellen
- Self Joins
- **Non-Equi Joins**
- OUTER-Joins:
  - LEFT OUTER-Joins
  - RIGHT OUTER-Joins
  - FULL OUTER-Joins
- Kartesische Produkte
  - Cross Joins

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

# Non-Equi Joins

EMPLOYEES

R	LAST_NAME	R	SALARY
1	Whalen		4400
2	Hartstein		13000
3	Fay		6000
4	Higgins		12000
5	Gietz		8300
6	King		24000
7	Kochhar		17000
8	De Haan		17000
9	Hunold		9000
10	Ernst		6000
...			
19	Taylor		8600
20	Grant		7000

JOB\_GRADES

R	GRADE_LEVEL	R	LOWEST_SAL	R	HIGHEST_SAL
1	A		1000		2999
2	B		3000		5999
	C		6000		9999
4	D		10000		14999
5	E		15000		24999
6	F		25000		40000

Die Tabelle JOB\_GRADES definiert den Wertebereich LOWEST\_SAL und HIGHEST\_SAL für jeden GRADE\_LEVEL. Daher kann jedem Mitarbeiter über die Spalte GRADE\_LEVEL eine Gehaltsstufe zugewiesen werden.

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Ein Non-Equi Join ist eine Join-Bedingung, die einen anderen Operator als den Gleich-Operator enthält.

Die Beziehung zwischen den Tabellen EMPLOYEES und JOB\_GRADES ist ein Beispiel für einen Non-Equi Join. Die Werte in der Spalte SALARY der Tabelle EMPLOYEES liegen im Wertebereich der Spalten LOWEST\_SAL und HIGHEST\_SAL aus der Tabelle JOB\_GRADES. Daher kann jeder Mitarbeiter auf Basis seines Gehalts eingestuft werden. Die Beziehung wird durch einen anderen Operator als den Gleich-Operator (=) gebildet.

## Datensätze mithilfe von Non-Equi Joins abrufen

```
SELECT e.last_name, e.salary, j.grade_level
FROM   employees e JOIN job_grades j
ON     e.salary
      BETWEEN j.lowest_sal AND j.highest_sal;
```

	LAST_NAME	SALARY	GRADE_LEVEL
1	Vargas	2500	A
2	Matos	2600	A
3	Davies	3100	B
4	Rajs	3500	B
5	Lorentz	4200	B
6	Whalen	4400	B
7	Mourgos	5800	B
8	Ernst	6000	C
9	Fay	6000	C
10	Grant	7000	C

...

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf der Folie wird ein Non-Equi Join erstellt, um die Gehaltsstufe eines Mitarbeiters auszuwerten. Das Gehalt muss *zwischen* einem niedrigsten und einem höchsten Gehalt der Gehaltsstufe in beliebiger Kombination liegen.

Jeder Mitarbeiter wird beim Ausführen der Abfrage genau einmal angezeigt. Kein Mitarbeiter wird in der Liste mehrfach aufgeführt. Dafür gibt es zwei Gründe:

- Keine der Zeilen in der Tabelle `JOB_GRADES` enthält Gehaltsstufen, die sich überschneiden. Das bedeutet, der Gehaltswert eines Mitarbeiters kann nur zwischen dem niedrigsten und dem höchsten Gehaltswert in einer der Zeilen der Gehaltsstufentabelle liegen.
- Alle Gehälter der Mitarbeiter liegen innerhalb der durch die Tabelle `JOB_GRADES` festgelegten Grenzwerte. Kein Mitarbeiter verdient also weniger als das niedrigste Gehalt aus der Spalte `LOWEST_SAL` oder mehr als das höchste Gehalt aus der Spalte `HIGHEST_SAL`.

**Hinweis:** Es können auch andere Bedingungen verwendet werden (z. B. `<=` und `>=`). `BETWEEN` ist jedoch die einfachste Möglichkeit. Geben Sie zuerst den Mindest- und dann den Höchstwert an, wenn Sie den Operator `BETWEEN` verwenden. Der Oracle-Server übersetzt den Operator `BETWEEN` in ein `AND`-Bedingungs paar. Daher bietet `BETWEEN` keine Performancevorteile und dient nur der logischen Einfachheit.

Im Beispiel auf der Folie wurden die Tabellenaliasnamen aus Performancegründen verwendet, nicht wegen möglicher Mehrdeutigkeiten.

# Lektionsagenda

- Typen von JOINS und ihre Syntax
- Natural Joins
- Joins mit der Klausel USING erstellen
- Joins mit der Klausel ON erstellen
- Self Joins
- Non-Equi Joins
- OUTER-Joins:
  - LEFT OUTER-Joins
  - RIGHT OUTER-Joins
  - FULL OUTER-Join
- Kartesische Produkte
  - Cross Joins

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.



## Mithilfe von OUTER-Joins Datensätze ohne direkte Übereinstimmung zurückgeben

DEPARTMENTS

R	2	DEPARTMENT_NAME	R	2	DEPARTMENT_ID
1		Administration			10
2		Marketing			20
3		Shipping			50
4		IT			60
5		Sales			80
6		Executive			90
7		Accounting			110
8		Contracting			190

In Abteilung 190 arbeiten keine Mitarbeiter.

Mitarbeiter Grant wurde keine Abteilungsnummer zugewiesen.

Equi Join mit EMPLOYEES

R	2	DEPARTMENT_ID	R	2	LAST_NAME
1		10			Whalen
2		20			Hartstein
3		20			Fay
4		110			Higgins
5		110			Gietz
6		90			King
7		90			Kochhar
8		90			De Haan
9		60			Hunold
10		60			Ernst

...

18		80			Abel
19		80			Taylor

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Zeilen, die die Join-Bedingung nicht erfüllen, werden im Abfrageergebnis nicht angezeigt.

Im Beispiel auf der Folie wird eine einfache Equi Join-Bedingung auf die Tabellen `EMPLOYEES` und `DEPARTMENTS` angewendet. Das Ergebnis wird auf der rechten Seite zurückgegeben. In der Ergebnismenge fehlen folgende Elemente:

- Abteilungsnummer (`DEPARTMENT_ID`) 190, da in der Tabelle `EMPLOYEES` keine Mitarbeiter mit dieser Abteilungsnummer erfasst sind
- Der Mitarbeiter mit dem Nachnamen Grant, da diesem Mitarbeiter keine Abteilungsnummer zugewiesen wurde

Um den Datensatz der Abteilung ohne Mitarbeiter oder die Mitarbeiter ohne Abteilung zurückzugeben, können Sie einen `OUTER-Join` verwenden.

## INNER- und OUTER-Joins – Vergleich

- In SQL:1999 werden Joins zwischen zwei Tabellen, die nur übereinstimmende Zeilen zurückgeben, als `INNER-Joins` bezeichnet.
- Joins zwischen zwei Tabellen, die die Ergebnisse des `INNER-Joins` sowie die Zeilen ohne Übereinstimmungen in der linken (oder rechten) Tabelle zurückgeben, werden als `Left (bzw. Right) OUTER-Joins` bezeichnet.
- Joins zwischen zwei Tabellen, die die Ergebnisse eines `INNER-Joins` sowie die Ergebnisse eines `Left Outer Joins` und `Right Outer Joins` zurückgeben, werden als `Full OUTER-Joins` bezeichnet.

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Das Ergebnis einer Verknüpfung von Tabellen mithilfe der Klauseln `NATURAL JOIN`, `USING` und `ON` ist ein `INNER-Join`. Nicht übereinstimmende Zeilen werden nicht in der Ausgabe angezeigt. Um die nicht übereinstimmenden Zeilen zurückzugeben, können Sie einen `OUTER-Join` verwenden. `OUTER-Joins` geben alle Zeilen zurück, die die Join-Bedingung erfüllen, und zusätzlich einige oder alle Zeilen aus einer Tabelle, für die es in der anderen Tabelle keine entsprechenden Zeilen gibt, die die Join-Bedingung erfüllen.

Es gibt drei Typen von `OUTER-Joins`:

- `LEFT OUTER`
- `RIGHT OUTER`
- `FULL OUTER`

## LEFT OUTER JOIN

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e LEFT OUTER JOIN departments d
ON     (e.department_id = d.department_id) ;
```

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Fay	20	Marketing
3	Hartstein	20	Marketing
4	Vargas	50	Shipping
5	Matos	50	Shipping

...

16	Kochhar	90	Executive
17	King	90	Executive
18	Gietz	110	Accounting
19	Higgins	110	Accounting
20	Grant	(null)	(null)

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Diese Abfrage ruft alle Zeilen aus der Tabelle `EMPLOYEES` ab (linke Tabelle), auch wenn keine Übereinstimmung in der Tabelle `DEPARTMENTS` gefunden wird.

## RIGHT OUTER JOIN

```
SELECT e.last_name, d.department_id, d.department_name
FROM   employees e RIGHT OUTER JOIN departments d
ON     (e.department_id = d.department_id) ;
```

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Hartstein	20	Marketing
3	Fay	20	Marketing
4	Davies	50	Shipping
5	Vargas	50	Shipping
6	Rajs	50	Shipping
7	Mourgos	50	Shipping
8	Matos	50	Shipping

...

18	Higgins	110	Accounting
19	Gietz	110	Accounting
20	(null)	190	Contracting

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Diese Abfrage ruft alle Zeilen aus der Tabelle `DEPARTMENTS` (rechte Tabelle) ab, auch wenn keine Übereinstimmung in der Tabelle `EMPLOYEES` gefunden wird.

## FULL OUTER JOIN

```
SELECT e.last_name, d.department_id, d.department_name
FROM   employees e FULL OUTER JOIN departments d
ON     (e.department_id = d.department_id) ;
```

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	King	90	Executive
2	Kochhar	90	Executive
3	De Haan	90	Executive
4	Hunold	60	IT

...

15	Grant	(null)	(null)
16	Whalen	10	Administration
17	Hartstein	20	Marketing
18	Fay	20	Marketing
19	Higgins	110	Accounting
20	Gietz	110	Accounting
21	(null)	190	Contracting

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Diese Abfrage ruft alle Zeilen aus der Tabelle `EMPLOYEES` ab, auch wenn keine Übereinstimmung in der Tabelle `DEPARTMENTS` gefunden wird. Darüber hinaus ruft sie alle Zeilen aus der Tabelle `DEPARTMENTS` ab, auch wenn es keine Übereinstimmung in der Tabelle `EMPLOYEES` gibt.

# Lektionsagenda

- Typen von JOINS und ihre Syntax
- Natural Joins
- Joins mit der Klausel USING erstellen
- Joins mit der Klausel ON erstellen
- Self Joins
- Non-Equi Joins
- OUTER-Joins:
  - LEFT OUTER-Joins
  - RIGHT OUTER-Joins
  - FULL OUTER-Joins
- Kartesische Produkte
  - Cross Joins

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Kartesische Produkte

- Ein kartesisches Produkt wird gebildet, wenn alle Zeilen einer Tabelle durch einen Join mit allen Zeilen einer zweiten Tabelle verknüpft sind.
- Ein kartesisches Produkt generiert eine Vielzahl von Zeilen, und das Ergebnis ist selten nützlich.

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Ein kartesisches Produkt generiert in der Regel eine Vielzahl von Zeilen, und das Ergebnis ist selten nützlich. Sie sollten daher stets eine gültige Join-Bedingung angeben, es sei denn, Sie möchten tatsächlich alle Zeilen aus allen Tabellen kombinieren.

Kartesische Produkte sind manchmal zu Testzwecken nützlich, wenn Sie eine große Zahl von Zeilen generieren möchten, um größere Datenmengen zu simulieren.

# Kartesische Produkte generieren

EMPLOYEES (20 Zeilen)

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	200 Whalen	10
2	201 Hartstein	20
3	202 Fay	20
4	205 Higgins	110
...		
19	176 Taylor	80
20	178 Grant	(null)

DEPARTMENTS (8 Zeilen)

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
1	10 Administration	1700
2	20 Marketing	1800
3	50 Shipping	1500
4	60 IT	1400
5	80 Sales	2500
6	90 Executive	1700
7	110 Accounting	1700
8	190 Contracting	1700

Kartesisches Produkt:  
20 x 8 = 160 Zeilen

EMPLOYEE_ID	DEPARTMENT_ID	LOCATION_ID
1	200	10
2	201	20
...		
21	200	10
22	201	20
...		
159	176	80
160	178	(null)

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Ein kartesisches Produkt wird gebildet, wenn eine Join-Bedingung fehlt. Das Beispiel auf der Folie zeigt die Nachnamen der Mitarbeiter und die Abteilungsnamen aus den Tabellen `EMPLOYEES` und `DEPARTMENTS` an. Da keine Join-Bedingung angegeben wurde, werden alle (20) Zeilen aus der Tabelle `EMPLOYEES` mit allen (8) Zeilen aus der Tabelle `DEPARTMENTS` verknüpft. Dadurch wird eine Ausgabe von 160 Zeilen erzeugt.



## Cross Joins erstellen

- Ein CROSS JOIN ist ein JOIN-Vorgang, der das kartesische Produkt aus zwei Tabellen zurückgibt.
- Um ein kartesisches Produkt zu erstellen, geben Sie in der Anweisung SELECT den CROSS JOIN an.

```
SELECT last_name, department_name  
FROM employees  
CROSS JOIN departments ;
```

	LAST_NAME	DEPARTMENT_NAME
1	Abel	Administration
2	Davies	Administration
3	De Haan	Administration
4	Ernst	Administration
5	Fay	Administration
...		
158	Vargas	Contracting
159	Whalen	Contracting
160	Zlotkey	Contracting

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Das Beispiel auf der Folie erzeugt ein kartesisches Produkt der Tabellen EMPLOYEES und DEPARTMENTS.

Wenn Sie ein kartesisches Produkt erstellen möchten, empfiehlt es sich, CROSS JOIN explizit in der Anweisung SELECT anzugeben. Auf diese Weise ist klar, dass ein kartesisches Produkt gebildet werden soll und die Ausgabe nicht das Ergebnis fehlender Joins ist.

## Quiz

Welche Art von Join verwenden Sie, um eine Tabelle mit sich selbst zu verknüpfen?

- a. Non-Equi Join
- b. Left OUTER-Join
- c. Right OUTER-Join
- d. Full OUTER-Join
- e. Self Join
- f. Natural Join
- g. Kartesisches Produkt

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

**Richtige Antwort: e**

# Zusammenfassung

In dieser Lektion haben Sie Folgendes gelernt:

- `SELECT`-Anweisungen erstellen, um über Equi Joins oder Non-Equi Joins auf Daten aus mehreren Tabellen zuzugreifen
- Tabellen über einen Self Join mit sich selbst verknüpfen
- Mit `OUTER`-Joins Daten anzeigen, die eine Join-Bedingung nicht erfüllen
- Kartesisches Produkt aller Zeilen aus zwei oder mehr Tabellen erzeugen

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Es gibt mehrere Möglichkeiten, Tabellen zu verknüpfen.

## Typen von Joins

- Equi Joins
- Non-Equi Joins
- `OUTER`-Joins
- Self Joins
- Cross Joins
- Natural Joins
- Full `OUTER`-Joins (zweiseitig)

## Kartesische Produkte

Ein kartesisches Produkt zeigt alle Zeilenkombinationen in der Ausgabe an. Sie erhalten ein kartesisches Produkt, wenn Sie die Klausel `WHERE` auslassen oder die Klausel `CROSS JOIN` angeben.

## Tabellenaliasnamen

- Tabellenaliasnamen beschleunigen den Datenbankzugriff.
- Sie tragen dazu bei, den SQL-Code zu verkürzen und dadurch Speicher einzusparen.
- Tabellenaliasnamen sind manchmal obligatorisch, um Mehrdeutigkeit von Spalten zu vermeiden.

## Übungen zu Lektion 7 – Überblick

Diese Übung behandelt folgende Themen:

- Tabellen mit Equi Joins verknüpfen
- Outer Joins und Self Joins ausführen
- Bedingungen hinzufügen

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In diesen Übungen extrahieren Sie Daten aus mehreren Tabellen mithilfe von SQL:1999-konformen Joins.

# 8

## Unterabfragen in Abfragen

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- Unterabfragen definieren
- Problemstellungen beschreiben, die mit Unterabfragen gelöst werden können
- Typen von Unterabfragen auflisten
- Single-Row-, Multiple-Row- und Multiple-Column-Unterabfragen erstellen

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In dieser Lektion lernen Sie erweiterte Features von `SELECT`-Anweisungen kennen. Sie können Unterabfragen in die Klausel `WHERE` einer anderen SQL-Anweisung einbetten, um Werte abzufragen, die auf einem unbekannten bedingten Wert basieren. Darüber hinaus behandelt diese Lektion Single-Row-, Multiple-Row- und Multiple-Column-Unterabfragen.

# Lektionsagenda

- Unterabfragen – Typen, Syntax und Richtlinien
- Single-Row-Unterabfragen:
  - Gruppenfunktionen in Unterabfragen
  - Unterabfragen in `HAVING`-Klauseln
- Multiple-Row-Unterabfragen
  - Operatoren `ALL` und `ANY`
- Multiple-Column-Unterabfragen
- Nullwerte in Unterabfragen

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

# Problemstellungen mit Unterabfragen lösen

Wer wurde nach Davies eingestellt?

Hauptabfrage:



Namen aller Mitarbeiter bestimmen, die nach Davies eingestellt wurden

Unterabfrage:



Wann wurde Davies eingestellt?

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie möchten eine Abfrage erstellen, mit der die Namen aller Mitarbeiter ermittelt werden, die nach Davies eingestellt wurden.

Um diese Aufgabe zu lösen, benötigen Sie *zwei* Abfragen: eine Abfrage, um das Einstellungsdatum von Davies zu ermitteln, und eine zweite Abfrage, die prüft, welche Mitarbeiter zu einem späteren Zeitpunkt eingestellt wurden.

Diese beiden Abfragen lassen sich kombinieren, indem Sie eine Abfrage *in* die andere Abfrage einbetten.

Die innere Abfrage (*Unterabfrage*) gibt einen Wert zurück, der von der äußeren Abfrage (*Hauptabfrage*) verwendet wird. Der Ausführungsplan der Abfrage hängt von der vom Optimizer gewählten Struktur der Unterabfrage ab.



## Unterabfragen – Syntax

- Die Unterabfrage (innere Abfrage) wird *vor* der Hauptabfrage (äußere Abfrage) ausgeführt.
- Das Ergebnis der Unterabfrage wird von der Hauptabfrage verwendet.

```
SELECT  select_list
FROM    table
WHERE   expr operator
        (SELECT      select_list
         FROM         table);
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Eine Unterabfrage ist eine `SELECT`-Anweisung, die in die Klausel einer anderen `SELECT`-Anweisung eingebettet ist. Mit Unterabfragen können Sie aus einfachen Anweisungen leistungsstarke Anweisungen erstellen. Unterabfragen sind sehr nützlich, wenn die Bedingung, mit der Sie Zeilen aus einer Tabelle abfragen möchten, von den Daten in der Tabelle selbst abhängt.

Unterabfragen lassen sich in verschiedene SQL-Klauseln einfügen, darunter:

- `WHERE`
- `HAVING`
- `FROM`

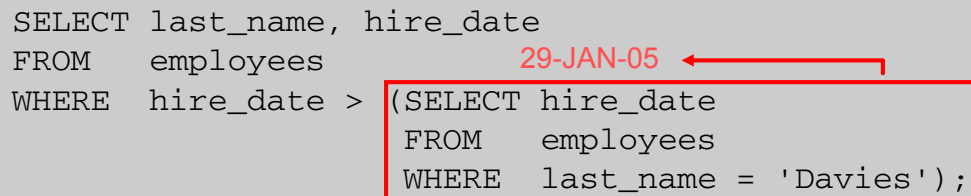
Für die Syntax gilt:

*operator* umfasst eine Vergleichsbedingung wie `>`, `=` oder `IN`

Unterabfragen werden häufig als verschachtelte `SELECT`-Anweisungen, `SELECT`-Unteranweisungen oder innere `SELECT`-Anweisungen bezeichnet. Unterabfragen werden generell zuerst ausgeführt. Ihre Ausgabe wird verwendet, um die Abfragebedingung für die Hauptabfrage (äußere Abfrage) zu vervollständigen.

## Unterabfragen – Beispiel

```
SELECT last_name, hire_date
FROM   employees
WHERE  hire_date > (SELECT hire_date
                    FROM   employees
                    WHERE  last_name = 'Davies');
```



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die auf der Folie gezeigte innere Abfrage ermittelt das Einstellungsdatum des Mitarbeiters Davies. Die äußere Abfrage verwendet das Ergebnis der inneren Abfrage, um alle Mitarbeiter anzuzeigen, die zu einem späteren Zeitpunkt eingestellt wurden als Davies.

## Unterabfragen – Regeln und Richtlinien

- Unterabfragen in Klammern einschließen
- Unterabfragen zur besseren Lesbarkeit auf der rechten Seite der Vergleichsbedingung angeben (Die Unterabfrage darf jedoch auf jeder Seite des Vergleichsoperators stehen.)
- Single-Row-Operatoren für Single-Row-Unterabfragen und Multiple-Row-Operatoren für Multiple-Row-Unterabfragen verwenden

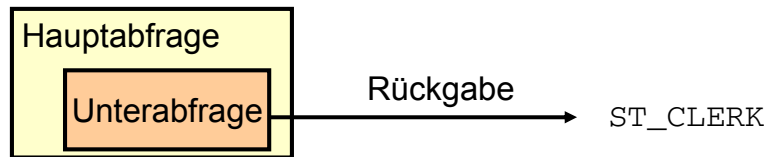
ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

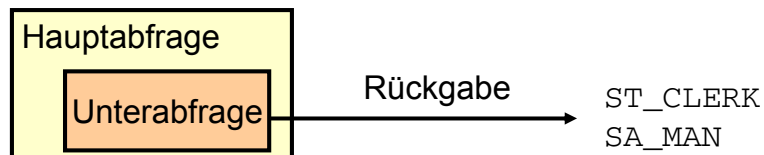
- Eine Unterabfrage muss in Klammern eingeschlossen werden.
- Geben Sie die Unterabfrage zur besseren Lesbarkeit auf der rechten Seite der Vergleichsbedingung an. Die Unterabfrage darf jedoch auf jeder Seite des Vergleichsoperators stehen.
- In Unterabfragen werden zwei Klassen von Vergleichsoperatoren verwendet: Single-Row-Operatoren und Multiple-Row-Operatoren.

# Typen von Unterabfragen

- Single-Row-Unterabfrage



- Multiple-Row-Unterabfrage



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

- **Single-Row-Unterabfragen:** Abfragen, deren innere `SELECT`-Anweisung nur eine Zeile zurückgibt
- **Multiple-Row-Unterabfragen:** Abfragen, deren innere `SELECT`-Anweisung mehr als eine Zeile zurückgibt

**Hinweis:** Es gibt außerdem Multiple-Column-Unterabfragen. Hierbei handelt es sich um Abfragen, deren innere `SELECT`-Anweisung mehrere Spalten zurückgibt. Diese Abfragen werden im Kurs *Oracle Database: SQL Workshop II* ausführlich behandelt.

# Lektionsagenda

- Unterabfragen – Typen, Syntax und Richtlinien
- Single-Row-Unterabfragen:
  - Gruppenfunktionen in Unterabfragen
  - Unterabfragen in HAVING-Klauseln
- Multiple-Row-Unterabfragen
  - Operatoren ALL und ANY
- Multiple-Column-Unterabfragen
- Nullwerte in Unterabfragen

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Single-Row-Unterabfragen

- Geben nur eine Zeile zurück
- Verwenden Single-Row-Vergleichsoperatoren

Operator	Bedeutung
=	Gleich
>	Größer als
>=	Größer/gleich
<	Kleiner als
<=	Kleiner/gleich
<>	Ungleich

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Eine Single-Row-Unterabfrage ist eine Unterabfrage, deren innere `SELECT`-Anweisung eine einzelne Zeile zurückgibt. Diese Art der Unterabfrage verwendet einen Single-Row-Operator. Die Folie enthält eine Liste der Single-Row-Operatoren.

### Beispiel

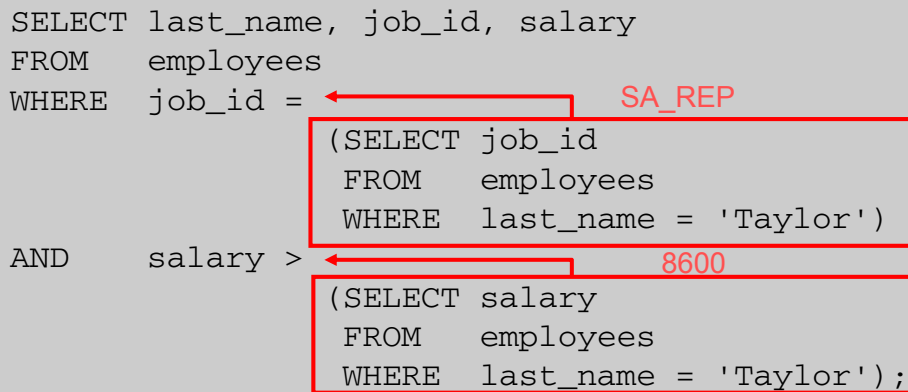
Sie möchten die Mitarbeiter anzeigen, deren Tätigkeits-ID der des Mitarbeiters mit der Nummer 141 entspricht:

```
SELECT last_name, job_id
FROM   employees
WHERE  job_id =
        (SELECT job_id
         FROM   employees
         WHERE  employee_id = 141);
```

	LAST_NAME	JOB_ID
1	Rajs	ST_CLERK
2	Davies	ST_CLERK
3	Matos	ST_CLERK
4	Vargas	ST_CLERK

## Single-Row-Unterabfragen ausführen

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  job_id = (SELECT job_id
                 FROM   employees
                 WHERE  last_name = 'Taylor')
AND    salary > (SELECT salary
                 FROM   employees
                 WHERE  last_name = 'Taylor');
```



	LAST_NAME	JOB_ID	SALARY
1	Abel	SA_REP	11000

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Eine `SELECT`-Anweisung kann als Abfrageblock betrachtet werden. Das Beispiel auf der Folie zeigt Mitarbeiter an, die dieselbe Tätigkeit wie "Taylor" ausführen, aber ein höheres Gehalt beziehen.

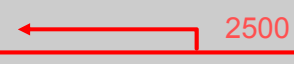
Das Beispiel besteht aus drei Abfrageblöcken: der äußeren Abfrage und zwei inneren Abfragen. Die inneren Abfrageblöcke werden zuerst ausgeführt und geben die Abfrageergebnisse `SA_REP` und `8600` zurück. Anschließend wird der äußere Abfrageblock verarbeitet. Dieser vervollständigt die Suchkriterien anhand der von den inneren Abfragen zurückgegebenen Werte.

Die beiden inneren Abfragen geben jeweils einen einzelnen Wert zurück (`SA_REP` und `8600`). Somit handelt es sich bei dieser SQL-Anweisung um eine Single-Row-Unterabfrage.

**Hinweis:** Äußere und innere Abfragen können Daten aus unterschiedlichen Tabellen abrufen.

## Gruppenfunktionen in Unterabfragen – Beispiel

```
SELECT last_name, job_id, salary
FROM employees
WHERE salary = (SELECT MIN(salary)
                FROM employees);
```



	LAST_NAME	JOB_ID	SALARY
1	Vargas	ST_CLERK	2500

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie können Daten aus einer Hauptabfrage anzeigen, indem Sie eine Gruppenfunktion in einer Unterabfrage verwenden, um eine einzelne Zeile zurückzugeben. Die Unterabfrage wird in Klammern gesetzt und nach der Vergleichsbedingung angegeben.

Das Beispiel auf der Folie zeigt Nachname, Tätigkeits-ID und Gehalt aller Mitarbeiter an, deren Gehalt dem Mindestgehalt entspricht. Die Gruppenfunktion `MIN` gibt einen einzelnen Wert (2500) an die äußere Abfrage zurück.



## Unterabfragen in HAVING-Klauseln

- Der Oracle-Server führt die Unterabfragen zuerst aus.
- Der Oracle-Server gibt die Ergebnisse an die Klausel HAVING der Hauptabfrage zurück.

```
SELECT  department_id, MIN(salary)
FROM    employees
GROUP BY department_id
HAVING  MIN(salary) > (SELECT MIN(salary)
                       FROM    employees
                       WHERE   department_id = 30);
```

	DEPARTMENT_ID	MIN(SALARY)
1	100	6900
2	(null)	7000
3	90	17000
4	20	6000
5	70	10000
6	110	8300
7	80	6100
8	40	6500
9	60	4200
10	10	4400

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie können Unterabfragen nicht nur in WHERE-Klauseln, sondern auch in HAVING-Klauseln verwenden. Der Oracle-Server führt die Unterabfrage aus und übergibt die Ergebnisse an die Klausel HAVING der Hauptabfrage.

Die SQL-Anweisung auf der Folie zeigt alle Abteilungen an, in denen das Mindestgehalt über dem Mindestgehalt von Abteilung 30 liegt.

### Beispiel

Sie möchten die Tätigkeits-ID mit dem niedrigsten Durchschnittsgehalt ermitteln.

```
SELECT  job_id, AVG(salary)
FROM    employees
GROUP BY job_id
HAVING  AVG(salary) = (SELECT  MIN(AVG(salary))
                       FROM    employees
                       GROUP BY job_id);
```

## Wo liegt der Fehler in dieser Anweisung?

```
SELECT employee_id, last_name
FROM   employees
WHERE  salary =
      (SELECT MIN(salary)
       FROM   employees
       GROUP BY department_id);
```

```
ORA-01427: single-row subquery returns more than one row
01427. 00000 - "single-row subquery returns more than one row"
*Cause:
*Action:
```

Single-Row-Operator in  
Multiple-Row-  
Unterabfrage

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Ein häufiger Fehler in Unterabfragen besteht darin, dass für eine Single-Row-Unterabfrage mehrere Zeilen zurückgegeben werden.

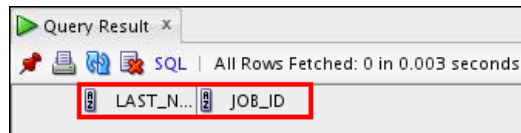
In der SQL-Anweisung auf der Folie enthält die Unterabfrage eine `GROUP BY`-Klausel. Diese impliziert, dass die Unterabfrage mehrere Zeilen zurückgibt: eine Zeile für jede ermittelte Gruppe. In diesem Fall lauten die Ergebnisse der Unterabfrage 4400, 6000, 2500, 4200, 7000, 17000 und 8300.

Die äußere Abfrage übernimmt diese Ergebnisse und verwendet sie in der Klausel `WHERE`. Die Klausel `WHERE` enthält einen Gleich-Operator (`=`), also einen Single-Row-Vergleichsoperator, der nur einen Wert erwartet. Der Gleich-Operator kann nur einen Wert aus der Unterabfrage annehmen und generiert daher den Fehler.

Um diesen Fehler zu korrigieren, ersetzen Sie den Operator `=` durch den Operator `IN`.

## Innere Abfrage gibt keine Zeilen zurück

```
SELECT last_name, job_id
FROM   employees
WHERE  job_id =
      (SELECT job_id
       FROM   employees
       WHERE  last_name = 'Haas');
```



Die Unterabfrage gibt keine Zeilen zurück, weil es keinen Mitarbeiter mit dem Namen "Haas" gibt.

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Ein weiteres häufiges Problem bei Unterabfragen tritt auf, wenn die innere Abfrage keine Zeilen zurückgibt.

In der SQL-Anweisung auf der Folie enthält die Unterabfrage eine `WHERE`-Klausel. Vermutlich soll nach einem Mitarbeiter mit dem Namen Haas gesucht werden. Die Anweisung ist korrekt. Bei der Ausführung werden aber keine Zeilen gewählt, da es keinen Mitarbeiter mit dem Namen Haas gibt. Die Unterabfrage gibt daher keine Zeilen zurück.

Die äußere Abfrage übernimmt das Ergebnis der Unterabfrage (`NULL`) und verwendet es in der Klausel `WHERE`. Die äußere Abfrage findet keinen Mitarbeiter mit einer Tätigkeits-ID gleich `NULL` und gibt daher keine Zeilen zurück. Auch wenn es eine Tätigkeits-ID mit dem Wert `NULL` gäbe, würden keine Zeilen zurückgegeben, da der Vergleich zweier Nullwerte einen Nullwert zum Ergebnis hat und die Bedingung `WHERE` damit nicht erfüllt ist.

# Lektionsagenda

- Unterabfragen – Typen, Syntax und Richtlinien
- Single-Row-Unterabfragen:
  - Gruppenfunktionen in Unterabfragen
  - Unterabfragen in `HAVING`-Klauseln
- Multiple-Row-Unterabfragen
  - Operatoren `IN`, `ALL` und `ANY`
- Multiple-Column-Unterabfragen
- Nullwerte in Unterabfragen

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Multiple-Row-Unterabfragen

- Geben mehrere Zeilen zurück
- Verwenden Multiple-Row-Vergleichsoperatoren

Operator	Bedeutung
IN	Ist gleich einem Element aus der Liste
ANY	Muss nach =, !=, >, <, <= oder >= stehen. Gibt TRUE zurück, wenn mindestens ein Element in der Ergebnismenge der Unterabfrage vorhanden ist, für das die Relation TRUE ist.
ALL	Muss nach =, !=, >, <, <= oder >= stehen. Gibt TRUE zurück, wenn die Relation für alle Elemente in der Ergebnismenge der Unterabfrage TRUE ist.

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Unterabfragen, die mehrere Zeilen zurückgeben, werden als Multiple-Row-Unterabfragen bezeichnet. Bei Multiple-Row-Unterabfragen verwenden Sie einen Multiple-Row-Operator anstelle eines Single-Row-Operators. Der Multiple-Row-Operator erwartet einen oder mehrere Werte:

```
SELECT last_name, salary, department_id
FROM employees
WHERE salary IN (SELECT MIN(salary)
                  FROM employees
                  GROUP BY department_id);
```

### Beispiel

Sie möchten die Mitarbeiter ermitteln, die das für die jeweilige Abteilung geltende Mindestgehalt verdienen.

Die innere Abfrage wird zuerst ausgeführt und erstellt ein Abfrageergebnis. Anschließend wird der Hauptabfrageblock verarbeitet. Dieser vervollständigt das Suchkriterium anhand der von der inneren Abfrage zurückgegebenen Werte. Die Hauptabfrage sieht für den Oracle-Server dann wie folgt aus:

```
SELECT last_name, salary, department_id
FROM employees
WHERE salary IN (2500, 4200, 4400, 6000, 7000, 8300,
                8600, 17000);
```

## Operator ANY in Multiple-Row-Unterabfragen – Beispiel

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary < ANY
      (SELECT salary
       FROM   employees
       WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

9000, 6000, 4200

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	144	Vargas	ST_CLERK	2500
2	143	Matos	ST_CLERK	2600
3	142	Davies	ST_CLERK	3100
4	141	Rajs	ST_CLERK	3500
5	200	Whalen	AD_ASST	4400

...

9	206	Gietz	AC_ACCOUNT	8300
10	176	Taylor	SA_REP	8600

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Der Operator **ANY** vergleicht (wie sein Synonym, der Operator **SOME**) einen Wert mit *jedem* von einer Unterabfrage zurückgegebenen Wert. Im Beispiel auf der Folie werden die Mitarbeiter angezeigt, die keine IT-Programmierer sind und die ein geringeres Gehalt beziehen als jeder der IT-Programmierer. Das höchste Gehalt, das ein Programmierer verdient, beträgt \$ 9.000.

- **<ANY** bedeutet niedriger als der Höchstwert.
- **>ANY** bedeutet höher als der Mindestwert.
- **=ANY** hat dieselbe Bedeutung wie **IN**.

## Operator ALL in Multiple-Row-Unterabfragen – Beispiel

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary < ALL
      (SELECT salary
       FROM   employees
       WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

9000, 6000, 4200

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	141	Rajs	ST_CLERK	3500
2	142	Davies	ST_CLERK	3100
3	143	Matos	ST_CLERK	2600
4	144	Vargas	ST_CLERK	2500

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Der Operator `ALL` vergleicht einen Wert mit *allen* von einer Unterabfrage zurückgegebenen Werten. Im Beispiel auf der Folie werden die Mitarbeiter angezeigt, deren Gehalt unter dem Gehalt aller Mitarbeiter mit der Tätigkeits-ID `IT_PROG` liegt und deren Tätigkeits-ID nicht `IT_PROG` ist.

`>ALL` bedeutet höher als der Höchstwert und `<ALL` bedeutet niedriger als der Mindestwert.

Der Operator `NOT` kann mit den Operatoren `IN`, `ANY` und `ALL` verwendet werden.

## Multiple-Column-Unterabfragen

- Multiple-Column-Unterabfragen geben mehrere Spalten an die äußere Abfrage zurück.
- Bei einem Multiple-Column-Vergleich können Spalten paarweise oder nicht paarweise verglichen werden.
- Multiple-Column-Unterabfragen können auch in der Klausel `FROM` einer `SELECT`-Anweisung verwendet werden.

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Multiple-Column-Abfragen geben mehrere Spalten an die äußere Abfrage zurück und können in der Klausel `FROM`, `WHERE` oder `HAVING` der äußeren Abfrage angegeben werden.

Wenn Sie zwei oder mehr Spalten vergleichen möchten, müssen Sie eine zusammengesetzte `WHERE`-Klausel mit logischen Operatoren erstellen. Mit Multiple-Column-Unterabfragen können Sie mehrfach vorhandene `WHERE`-Bedingungen in einer einzelnen `WHERE`-Klausel kombinieren.

Der Operator `IN` prüft auf einen Wert innerhalb einer Werteliste. Die Liste der Werte kann den von einer Unterabfrage zurückgegebenen Ergebnissen entsprechen.

Syntax:

```
SELECT column, column, ...
FROM table
WHERE (column, column, ...) IN
      (SELECT column, column, ...
       FROM table
       WHERE condition);
```



## Multiple-Column-Unterabfragen – Beispiel

Alle Mitarbeiter mit dem jeweils niedrigsten Gehalt der einzelnen Abteilungen anzeigen

```
SELECT first_name, department_id, salary
FROM employees
WHERE (salary, department_id) IN
      (SELECT min(salary), department_id
       FROM employees
       GROUP BY department_id)
ORDER BY department_id;
```

	FIRST_NAME	DEPARTMENT_ID	SALARY
1	Jennifer	10	4400
2	Pat	20	6000
3	Peter	50	2500
4	Diana	60	4200
5	Jonathon	80	8600
6	Neena	90	17000
7	Lex	90	17000
8	William	110	8300

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Das Beispiel auf der Folie zeigt eine Multiple-Column-Unterabfrage, da die Unterabfrage mehrere Spalten zurückgibt.

Die innere Abfrage wird zuerst ausgeführt und gibt für jede Abteilung das niedrigste Gehalt und die Abteilungsnummer zurück. Anschließend wird der Hauptabfrageblock verarbeitet. Dieser vervollständigt das Suchkriterium anhand der von der inneren Abfrage zurückgegebenen Werte.

# Lektionsagenda

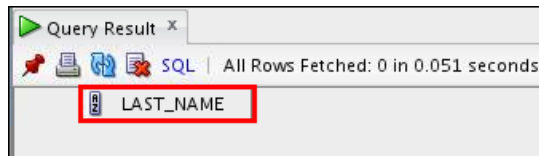
- Unterabfragen – Typen, Syntax und Richtlinien
- Single-Row-Unterabfragen:
  - Gruppenfunktionen in Unterabfragen
  - Unterabfragen in `HAVING`-Klauseln
- Multiple-Row-Unterabfragen
  - Operatoren `ALL` und `ANY`
- Multiple-Column-Unterabfragen
- Nullwerte in Unterabfragen

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Nullwerte in Unterabfragen

```
SELECT emp.last_name
FROM   employees emp
WHERE  emp.employee_id NOT IN
      (SELECT mgr.manager_id
       FROM   employees mgr);
```



Die Unterabfrage gibt keine Zeilen zurück, da einer der von der Unterabfrage zurückgegebenen Werte ein Nullwert ist.

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die SQL-Anweisung auf der Folie soll alle Mitarbeiter anzeigen, denen keine anderen Mitarbeiter unterstellt sind. Logischerweise müsste diese SQL-Anweisung 12 Zeilen zurückgeben. Stattdessen werden keine Zeilen zurückgegeben. Einer der von der inneren Abfrage zurückgegebenen Werte ist ein Nullwert. Deshalb gibt die gesamte Abfrage keine Zeilen zurück.

Der Grund ist, dass alle Bedingungen, die einen Nullwert vergleichen, einen Nullwert zurückgeben. Verwenden Sie daher nicht den Operator `NOT IN`, wenn die Ergebnismenge einer Unterabfrage voraussichtlich Nullwerte enthält. Der Operator `NOT IN` entspricht `<> ALL`.

Ein Nullwert als Teil der Ergebnismenge einer Unterabfrage stellt kein Problem dar, wenn Sie den Operator `IN` verwenden. Der Operator `IN` entspricht `=ANY`. Beispiel: Um die Mitarbeiter anzuzeigen, denen andere Mitarbeiter unterstellt sind, verwenden Sie die folgende SQL-Anweisung:

```
SELECT emp.last_name
FROM   employees emp
WHERE  emp.employee_id IN
      (SELECT mgr.manager_id
       FROM   employees mgr);
```

Alternativ können Sie eine `WHERE`-Klausel in die Unterabfrage einfügen, um alle Mitarbeiter ohne unterstellte Mitarbeiter anzuzeigen:

```
SELECT last_name FROM employees
WHERE  employee_id NOT IN
      (SELECT manager_id
       FROM   employees
       WHERE  manager_id IS NOT NULL);
```

## Quiz

Eine Unterabfrage entspricht der Ausführung von zwei aufeinanderfolgenden Abfragen, wobei die Ergebnisse der ersten Abfrage als Suchwerte in der zweiten Abfrage verwendet werden.

- a. Richtig
- b. Falsch

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

**Richtige Antwort: a**

# Zusammenfassung

In dieser Lektion haben Sie Folgendes gelernt:

- Unterabfragen definieren
- Problemstellungen bestimmen, die mit Unterabfragen gelöst werden können
- Single-Row-, Multiple-Row- und Multiple-Column-Unterabfragen erstellen

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In dieser Lektion wurde der Umgang mit Unterabfragen behandelt. Unterabfragen sind `SELECT`-Anweisungen, die in die Klausel einer anderen SQL-Anweisung eingebettet sind. Unterabfragen sind nützlich, wenn eine Abfrage auf einem Suchkriterium mit unbekannten Zwischenwerten basiert.

Unterabfragen verfügen über folgende Eigenschaften:

- Sie übergeben eine Datenzeile an eine Hauptanweisung, die einen Single-Row-Operator enthält, z. B. `=`, `<>`, `>`, `>=`, `<` oder `<=`.
- Sie übergeben mehrere Datenzeilen an eine Hauptanweisung, die einen Multiple-Row-Operator enthält, z. B. `IN`.
- Sie werden vom Oracle-Server zuerst verarbeitet. Das Ergebnis wird anschließend von der Klausel `WHERE` oder `HAVING` verwendet.
- Sie können Gruppenfunktionen enthalten.

## Übungen zu Lektion 8 – Überblick

Diese Übung behandelt folgende Themen:

- Unterabfragen erstellen, um Werte basierend auf unbekannten Kriterien abzufragen
- Unterabfragen erstellen, um Werte zu ermitteln, die in einer Datengruppe, nicht aber in einer anderen Datengruppe vorhanden sind

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In diesen Übungen erstellen Sie komplexe Abfragen mit verschachtelten `SELECT`-Anweisungen.

Aus praktischen Gründen erstellen Sie zuerst die innere Abfrage. Dabei prüfen Sie, ob die Abfrage korrekt ausgeführt wird und die erwarteten Ergebnisse liefert, bevor Sie die äußere Abfrage erstellen.

