



Hardware and Software
Engineered to Work Together

Oracle Database 12c: SQL Workshop II

Schulungsunterlagen – Band I

D80194DE11

Production 1.1 | Dezember 2014 | D88605

Learn more from Oracle University at 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

Maheshwari Krishnamurthy

Daniel Milne

Michael Almeida

Diganta Choudhury

Manish Pawar

Clair Bennett

Yanti Chang

Joel Goodman

Gerlinde Frenzen

Madhavi Siddireddy

Redaktion

Raj Kumar

Malavika Jinka

Herausgeber

Jobi Varghese

Pavithran Adka

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.

Inhalt

1 Einführung

- Ziele 1-2
- Lektionsagenda 1-3
- Kursziele 1-4
- Kursvoraussetzungen 1-5
- Kursagenda 1-6
- Lektionsagenda 1-7
- In diesem Kurs verwendete Tabellen 1-8
- In diesem Kurs verwendete Anhänge und Übungen 1-10
- Entwicklungsumgebungen 1-11
- Lektionsagenda 1-12
- Daten einschränken – Wiederholung 1-13
- Daten sortieren – Wiederholung 1-14
- SQL-Funktionen – Wiederholung 1-15
- Single-Row-Funktionen – Wiederholung 1-16
- Typen von Gruppenfunktionen – Wiederholung 1-17
- Unterabfragen – Wiederholung 1-18
- Tabellen mithilfe von DML-Anweisungen verwalten – Wiederholung 1-20
- Lektionsagenda 1-22
- Oracle Database – SQL-Dokumentation 1-23
- Zusätzliche Ressourcen 1-24
- Zusammenfassung 1-25
- Übungen zu Lektion 1 – Überblick 1-26

2 Data Dictionary Views – Einführung

- Ziele 2-2
- Lektionsagenda 2-3
- Data Dictionary 2-4
- Data Dictionary-Struktur 2-5
- Dictionary Views – Verwendung 2-7
- Views `USER_OBJECTS` und `ALL_OBJECTS` 2-8
- View `USER_OBJECTS` 2-9
- Lektionsagenda 2-10
- Tabelleninformationen 2-11
- Spalteninformationen 2-12

Constraint-Informationen 2-14
USER_CONSTRAINTS – Beispiel 2-15
USER_CONS_COLUMNS abfragen 2-16
Lektionsagenda 2-17
Kommentare zu Tabellen hinzufügen 2-18
Quiz 2-19
Zusammenfassung 2-20
Übungen zu Lektion 2 – Überblick 2-21

3 Sequences, Synonyme und Indizes erstellen

Ziele 3-2
Lektionsagenda 3-3
Datenbankobjekte 3-4
Tabellen anderer Benutzer referenzieren 3-5
Sequences 3-6
Anweisung CREATE SEQUENCE – Syntax 3-7
Sequences erstellen 3-9
Pseudospalten NEXTVAL und CURRVAL 3-10
Sequences 3-12
SQL-Spaltenstandards mithilfe von Sequences einstellen 3-13
Sequence-Werte im Cache speichern 3-14
Sequences ändern 3-15
Sequences ändern – Richtlinien 3-16
Sequence-Informationen 3-17
Lektionsagenda 3-18
Synonyme 3-19
Synonyme für Objekte erstellen 3-20
Synonyme erstellen und entfernen 3-21
Synonyminformationen 3-22
Lektionsagenda 3-23
Indizes 3-24
Indexerstellung – Verfahren 3-25
Indizes erstellen 3-26
CREATE INDEX mit Anweisung CREATE TABLE 3-27
Funktionsbasierte Indizes 3-29
Mehrere Indizes für dieselbe Gruppe von Spalten erstellen 3-30
Mehrere Indizes für dieselbe Gruppe von Spalten erstellen – Beispiel 3-31
Indexinformationen 3-32
USER_INDEXES – Beispiele 3-33
USER_IND_COLUMNS abfragen 3-34

Indizes entfernen 3-35
Quiz 3-36
Zusammenfassung 3-37
Übungen zu Lektion 3 – Überblick 3-38

4 Views erstellen

Ziele 4-2
Lektionsagenda 4-3
Datenbankobjekte 4-4
Views 4-5
Views – Vorteile 4-6
Einfache und komplexe Views 4-7
Lektionsagenda 4-8
Views erstellen 4-9
Daten aus Views abrufen 4-12
Views ändern 4-13
Komplexe Views erstellen 4-14
View-Informationen 4-15
Lektionsagenda 4-16
DML-Vorgänge an Views ausführen – Regeln 4-17
Klausel `WITH CHECK OPTION` 4-20
DML-Vorgänge verweigern 4-21
Lektionsagenda 4-23
Views entfernen 4-24
Quiz 4-25
Zusammenfassung 4-26
Übungen zu Lektion 4 – Überblick 4-27

5 Schemaobjekte verwalten

Ziele 5-2
Lektionsagenda 5-3
Constraints hinzufügen – Syntax 5-4
Constraints hinzufügen 5-5
Constraints löschen 5-6
Constraints mit Schlüsselwort `ONLINE` löschen 5-7
Klausel `ON DELETE` 5-8
Klausel `CASCADE CONSTRAINTS` 5-9
Tabellenspalten und Constraints umbenennen 5-11
Constraints deaktivieren 5-12
Constraints aktivieren 5-13

Constraints – Statusmöglichkeiten 5-14
Constraints verzögern 5-15
INITIALLY DEFERRED und INITIALLY IMMEDIATE – Vergleich 5-16
DROP TABLE ... PURGE 5-18
Lektionsagenda 5-19
Temporäre Tabellen 5-20
Temporäre Tabellen erstellen 5-21
Lektionsagenda 5-22
Externe Tabellen 5-23
Verzeichnisse für externe Tabellen erstellen 5-24
Externe Tabellen erstellen 5-26
Externe Tabellen mit ORACLE_LOADER erstellen 5-28
Externe Tabellen abfragen 5-30
Externe Tabellen mit ORACLE_DATAPUMP erstellen – Beispiel 5-31
Quiz 5-32
Zusammenfassung 5-33
Übungen zu Lektion 5 – Überblick 5-34

6 Daten mithilfe von Unterabfragen abrufen

Ziele 6-2
Lektionsagenda 6-3
Daten mit Unterabfragen als Quelle abrufen 6-4
Lektionsagenda 6-6
Multiple-Column-Unterabfragen 6-7
Spaltenvergleiche 6-8
Unterabfragen mit paarweisen Vergleichen 6-9
Unterabfragen mit nicht paarweise durchgeführten Vergleichen 6-10
Lektionsagenda 6-11
Skalare Unterabfrageausdrücke 6-12
Skalare Unterabfragen – Beispiele 6-13
Lektionsagenda 6-14
Korrelierte Unterabfragen 6-15
Korrelierte Unterabfragen – 1. Beispiel 6-18
Korrelierte Unterabfragen – 2. Beispiel 6-19
Lektionsagenda 6-20
Operator EXISTS 6-21
Alle Abteilungen ermitteln, die keine Mitarbeiter enthalten 6-23
Lektionsagenda 6-24
Klausel WITH 6-25
Klausel WITH – Beispiel 6-26
Rekursive Klausel WITH 6-27
Rekursive Klausel WITH – Beispiel 6-28

Quiz 6-29
Zusammenfassung 6-30
Übungen zu Lektion 6 – Überblick 6-31

7 Daten mit Unterabfragen bearbeiten

Ziele 7-2
Lektionsagenda 7-3
Daten mit Unterabfragen bearbeiten 7-4
Lektionsagenda 7-5
Werte einfügen und Unterabfrage als Ziel verwenden 7-6
Lektionsagenda 7-8
Schlüsselwort `WITH CHECK OPTION` in DML-Anweisungen 7-9
Lektionsagenda 7-11
Korrelierte `UPDATE`-Anweisungen 7-12
Korrelierte `UPDATE`-Anweisungen – Beispiel 7-13
Korrelierte `DELETE`-Anweisungen 7-15
Korrelierte `DELETE`-Anweisungen – Beispiel 7-16
Zusammenfassung 7-17
Übungen zu Lektion 7 – Überblick 7-18

8 Benutzerzugriff steuern

Ziele 8-2
Lektionsagenda 8-3
Benutzerzugriff steuern 8-4
Berechtigungen 8-5
Systemberechtigungen 8-6
Benutzer erstellen 8-7
Systemberechtigungen für Benutzer 8-8
Systemberechtigungen erteilen 8-10
Lektionsagenda 8-11
Was ist eine Rolle? 8-12
Rollen erstellen und Berechtigungen zuweisen 8-13
Kennwörter ändern 8-14
Lektionsagenda 8-15
Objektberechtigungen 8-16
Objektberechtigungen erteilen 8-18
Berechtigungen weitergeben 8-19
Erteilte Berechtigungen prüfen 8-20
Lektionsagenda 8-21
Objektberechtigungen entziehen 8-22

Quiz 8-24

Zusammenfassung 8-25

Übungen zu Lektion 8 – Überblick 8-26

9 Daten bearbeiten

Ziele 9-2

Lektionsagenda 9-3

Explizites Standardfeature – Überblick 9-4

Explizite Standardwerte 9-5

Lektionsagenda 9-6

INSERT-Anweisungen für mehrere Tabellen – Überblick 9-7

Typen von INSERT-Anweisungen für mehrere Tabellen 9-9

INSERT-Anweisungen für mehrere Tabellen 9-10

INSERT ALL ohne Bedingung 9-12

INSERT ALL mit Bedingung – Beispiel 9-13

INSERT ALL mit Bedingung 9-14

INSERT FIRST mit Bedingung – Beispiel 9-16

INSERT FIRST mit Bedingung 9-17

INSERT mit Pivoting 9-19

Lektionsagenda 9-22

MERGE-Anweisungen 9-23

MERGE-Anweisungen – Syntax 9-24

Zeilen zusammenführen – Beispiel 9-25

Lektionsagenda 9-28

FLASHBACK TABLE-Anweisungen 9-29

FLASHBACK TABLE-Anweisungen – Beispiel 9-31

Lektionsagenda 9-32

Datenänderungen überwachen 9-33

Flashback Query – Beispiel 9-34

Flashback Version Query – Beispiel 9-35

VERSIONS BETWEEN-Klauseln 9-36

Quiz 9-37

Zusammenfassung 9-39

Übungen zu Lektion 9 – Überblick 9-40

10 Daten in verschiedenen Zeitzonen verwalten

Ziele 10-2

Lektionsagenda 10-3

Zeitzone 10-4

Sessionparameter `TIME_ZONE` 10-5

CURRENT_DATE, CURRENT_TIMESTAMP und LOCALTIMESTAMP	10-6
Datum und Uhrzeit in einer Sessionzeitzone vergleichen	10-7
DBTIMEZONE und SESSIONTIMEZONE	10-9
TIMESTAMP-Datentypen	10-10
TIMESTAMP-Felder	10-11
DATE und TIMESTAMP – Unterschiede	10-12
TIMESTAMP-Datentypen vergleichen	10-13
Lektionsagenda	10-14
INTERVAL-Datentypen	10-15
INTERVAL-Felder	10-17
INTERVAL YEAR TO MONTH – Beispiel	10-18
Datentyp INTERVAL DAY TO SECOND – Beispiel	10-20
Lektionsagenda	10-21
EXTRACT	10-22
TZ_OFFSET	10-23
FROM_TZ	10-25
TO_TIMESTAMP	10-26
TO_YMINTERVAL	10-27
TO_DSINTERVAL	10-28
Sommerzeit (DST)	10-29
Quiz	10-31
Zusammenfassung	10-32
Übungen zu Lektion 10 – Überblick	10-33

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-15
Befehl SERVEROUTPUT C-16
SQL*Plus-Befehl SPOOL C-17
Befehl AUTOTRACE C-18
Zusammenfassung C-19

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

E Berichte durch Gruppieren zusammenhängender Daten generieren

Ziele E-2
Gruppenfunktionen – Wiederholung E-3
Klausel GROUP BY – Wiederholung E-4
Klausel HAVING – Wiederholung E-5
GROUP BY mit den Operatoren ROLLUP und CUBE E-6
Operator ROLLUP E-7

Operator `ROLLUP` – Beispiel E-8
Operator `CUBE` E-9
Operator `CUBE` – Beispiel E-10
Funktion `GROUPING` E-11
Funktion `GROUPING` – Beispiel E-12
`GROUPING SETS` E-13
`GROUPING SETS` – Beispiel E-15
Zusammengesetzte Spalten E-17
Zusammengesetzte Spalten – Beispiel E-19
Verkettete Gruppierungen E-21
Verkettete Gruppierungen – Beispiel E-22
Zusammenfassung E-23

F Hierarchische Datenabfragen

Ziele F-2
Beispieldaten aus der Tabelle `EMPLOYEES` F-3
Natürliche Baumstrukturen F-4
Hierarchische Abfragen F-5
Baumstruktur durchlaufen F-6
Baumstruktur durchlaufen – Von unten nach oben F-8
Baumstruktur durchlaufen – Von oben nach unten F-9
Rangfolge von Zeilen mit der Pseudospalte `LEVEL` festlegen F-10
Hierarchische Berichte mit `LEVEL` und `LPAD` formatieren F-11
Verzweigungen ausblenden (Pruning) F-13
Zusammenfassung F-14

G Fortgeschrittene Skripte erstellen

Ziele G-2
SQL-Skripte mit SQL generieren G-3
Einfache Skripte erstellen G-4
Umgebung steuern G-5
Gesamtbild G-6
Tabelleninhalt in eine Datei ausgeben G-7
Dynamische Prädikate generieren G-9
Zusammenfassung G-11

H Oracle Database – Architekturkomponenten

Ziele H-2
Architektur von Oracle Database – Überblick H-3
Oracle-Datenbankserver – Strukturen H-4

Bei der Datenbank anmelden H-5
Mit einer Oracle-Datenbank interagieren H-6
Oracle-Memoryarchitektur H-8
Prozessarchitektur H-10
Database Writer H-12
Log Writer H-13
Checkpoint H-14
System Monitor H-15
Process Monitor H-16
Speicherarchitektur von Oracle-Datenbanken H-17
Logische und physische Datenbankstrukturen H-19
SQL-Anweisungen verarbeiten H-21
Abfragen verarbeiten H-22
Shared Pool H-23
Datenbank-Puffercache H-25
Program Global Area (PGA) H-26
DML-Anweisungen verarbeiten H-27
Redo-Logpuffer H-29
Rollback-Segmente H-30
COMMIT-Verarbeitung H-31
Architektur von Oracle Database – Überblick H-33
Zusammenfassung H-34

I Unterstützung regulärer Ausdrücke

Ziele I-2
Was sind reguläre Ausdrücke? I-3
Reguläre Ausdrücke – Vorteile I-4
Funktionen und Bedingungen für reguläre Ausdrücke in SQL und PL/SQL I-5
Was sind Metazeichen? I-6
Metazeichen in regulären Ausdrücken I-7
Funktionen und Bedingungen für reguläre Ausdrücke – Syntax I-9
Einfache Suche mit der Bedingung `REGEXP_LIKE` durchführen I-10
Muster mit der Funktion `REGEXP_REPLACE` ersetzen I-11
Muster mit der Funktion `REGEXP_INSTR` suchen I-12
Teilzeichenfolgen mit der Funktion `REGEXP_SUBSTR` extrahieren I-13
Teilausdrücke I-14
Teilausdrücke in Verbindung mit regulären Ausdrücken I-15
Wieso ist der Zugriff auf den n. Teilausdruck wichtig? I-16
`REGEXP_SUBSTR` – Beispiel I-17
`REGEXP_COUNT`-Funktion I-18

Reguläre Ausdrücke und CHECK-Constraints – Beispiele I-19

Quiz I-20

Zusammenfassung I

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 erörtern
- Datenbankschema und Datenbanktabellen für diesen Kurs beschreiben
- Verfügbare Umgebungen für diesen Kurs bestimmen
- Grundkonzepte von SQL wiederholen

ORACLE

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

Lektionsagenda

- **Kursziele und Kursagenda**
- Datenbankschema, Anhänge und Übungen sowie Entwicklungsumgebungen für diesen Kurs
- Grundkonzepte von SQL – Wiederholung
- 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:

- Objekte mithilfe von Data Dictionary Views verwalten
- Schemaobjekte erstellen
- Schemaobjekte verwalten
- Multiple-Column-Unterabfragen erstellen
- Skalare und korrelierte Unterabfragen verwenden
- Benutzerzugriff auf bestimmte Datenbankobjekte steuern
- Neue Benutzer mit unterschiedlichen Ebenen von Zugriffsberechtigungen hinzufügen
- Große Datasets in der Oracle-Datenbank mithilfe von Unterabfragen bearbeiten
- Daten in verschiedenen Zeitzonen verwalten

ORACLE

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

Kursvoraussetzungen

Dieser Kurs setzt die Teilnahme am Kurs *Oracle Database: SQL Workshop I* voraus.

ORACLE

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

Der Kurs *Oracle Database: SQL Workshop I* vermittelt Ihnen eine Einführung in die Technologie von Oracle Database. In diesem Kurs lernen Sie die Grundkonzepte relationaler Datenbanken sowie die leistungsstarke Programmiersprache SQL kennen und eignen sich grundlegende SQL-Kenntnisse an, sodass Sie Abfragen für einzelne oder mehrere Tabellen durchführen, Daten in Tabellen bearbeiten, Datenbankobjekte erstellen und Metadaten abfragen können.

Kursagenda

- 1. Tag:
 - Einführung
 - Data Dictionary Views – Einführung
 - Sequences, Synonyme und Indizes erstellen
 - Views erstellen
 - Schemaobjekte verwalten
- 2. Tag:
 - Daten mithilfe von Unterabfragen abrufen
 - Daten mithilfe von Unterabfragen bearbeiten
 - Benutzerzugriff steuern
 - Daten bearbeiten
 - Daten in verschiedenen Zeitzonen verwalten

ORACLE

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

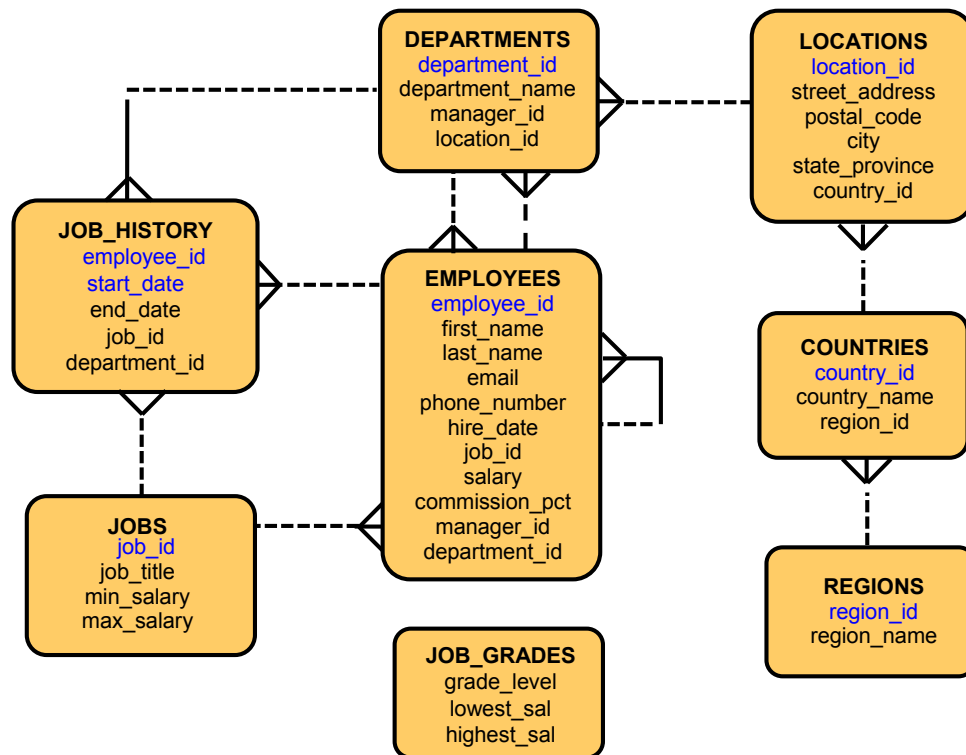
Lektionsagenda

- Kursziele und Kursagenda
- **Datenbankschema, Anhänge und Übungen sowie Entwicklungsumgebungen für diesen Kurs**
- Grundkonzepte von SQL – Wiederholung
- Dokumentation und zusätzliche Ressourcen zu Oracle Database 12c

ORACLE

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

In diesem Kurs verwendete Tabellen



ORACLE

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

In diesem Kurs arbeiten Sie mit Daten aus folgenden Tabellen:

Tabellenbeschreibungen

- Die Tabelle **EMPLOYEES** enthält Informationen zu allen Mitarbeitern, darunter Vor- und Nachname, Tätigkeits-ID, Gehalt, Einstellungsdatum, Abteilungsnummer und Manager-ID. Diese Tabelle ist abhängig von der Tabelle **DEPARTMENTS**.
- Die Tabelle **DEPARTMENTS** enthält Informationen wie Abteilungs-ID, Abteilungsname, Manager-ID und Standort-ID und stellt die Primärschlüsseltabelle für die Tabelle **EMPLOYEES** dar.
- Die Tabelle **LOCATIONS** enthält Informationen zum Abteilungsstandort, darunter Standort-ID, Straße, Ort, Provinz/Bundesstaat/Bundesland, Postleitzahl und Länder-ID. Sie stellt die Primärschlüsseltabelle für die Tabelle **DEPARTMENTS** dar und ist der Tabelle **COUNTRIES** untergeordnet.
- Die Tabelle **COUNTRIES** enthält die Ländernamen sowie die ID für das jeweilige Land und die betreffende Region. Diese Tabelle ist der Tabelle **REGIONS** untergeordnet und stellt die Primärschlüsseltabelle für die Tabelle **LOCATIONS** dar.
- Die Tabelle **REGIONS** enthält die Namen und IDs zu den Regionen der einzelnen Länder. Sie stellt die Primärschlüsseltabelle für die Tabelle **COUNTRIES** dar.

- Die Tabelle `JOB_GRADES` gibt einen Gehaltsbereich pro Gehaltsgruppe an. Zwischen den Gehaltsbereichen gibt es keine Überschneidungen.
- Die Tabelle `JOB_HISTORY` speichert die Tätigkeitshistorie der Mitarbeiter.
- Die Tabelle `JOBS` enthält die Tätigkeiten und Gehaltsbereiche.

In diesem Kurs verwendete Anhänge und Übungen

- Anhang A: Tabellenbeschreibungen
- Anhang B: SQL Developer
- Anhang C: SQL*Plus
- Anhang D: Häufig verwendete SQL-Befehle
- Anhang E: Berichte durch Gruppierung zusammenhängender Daten generieren
- Anhang F: Hierarchische Datenabfragen
- Anhang G: Fortgeschrittene Skripte erstellen
- Anhang H: Oracle Database – Architekturkomponenten
- Anhang I: Unterstützung regulärer Ausdrücke
- Übungen und Lösungen
- Zusätzliche Übungen und Lösungen

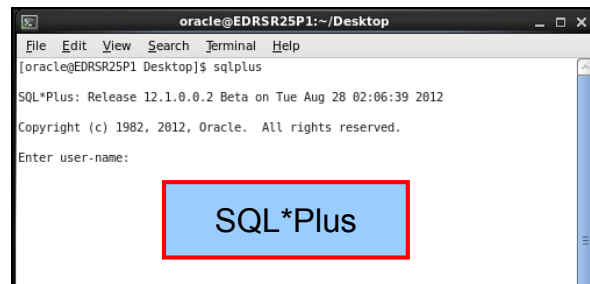
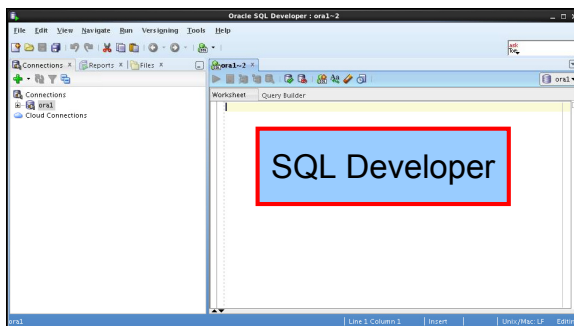
ORACLE

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

Entwicklungsumgebungen

In diesem Kurs sind zwei Entwicklungsumgebungen verfügbar:

- Das bevorzugte Tool ist Oracle SQL Developer.
- Daneben können Sie auch die SQL*Plus-Befehlszeilenschnittstelle verwenden.



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*Plus

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

Hinweis:

- Informationen zur Verwendung von SQL Developer finden Sie in Anhang B, "SQL Developer".
- Informationen zur Verwendung von SQL*Plus finden Sie in Anhang C, "SQL*Plus".

Lektionsagenda

- Kursziele und Kursagenda
- Datenbankschema, Anhänge und Übungen sowie Entwicklungsumgebungen für diesen Kurs
- Grundkonzepte von SQL – Wiederholung
- Dokumentation und zusätzliche Ressourcen zu Oracle Database 12c

ORACLE

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

Auf den folgenden Folien werden einige Grundkonzepte aus dem Kurs *Oracle Database: SQL Workshop I* kurz zusammengefasst.

Daten einschränken – Wiederholung

- Zurückzugebende Zeilen mit der Klausel `WHERE` einschränken
- Mithilfe von Vergleichsbedingungen Ausdrücke mit anderen Werten und Ausdrücken vergleichen

Operator	Bedeutung
<code>BETWEEN</code> <code>...AND...</code>	Zwischen zwei Werten (einschließlich der gegebenen Werte)
<code>IN (set)</code>	Entspricht einem Wert aus einer Werteliste
<code>LIKE</code>	Entspricht einem Zeichenmuster

- Ergebnisse von zwei Teilbedingungen mithilfe von logischen Bedingungen kombinieren und auf Basis dieser Bedingungen ein einzelnes Ergebnis erzeugen

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. `WHERE`-Klauseln enthalten Bedingungen, die erfüllt sein müssen. Sie werden unmittelbar nach der Klausel `FROM` angegeben.

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

- Spaltenname
- Vergleichsbedingung
- Spaltenname, Konstante oder Werteliste

Vergleichsbedingungen lassen sich in der Klausel `WHERE` im folgenden Format verwenden:

`... WHERE expr operator value`

Neben den auf der Folie genannten Operatoren können Sie weitere Vergleichsbedingungen verwenden, etwa `=`, `<`, `>`, `<>`, `<=` und `>=`.

In SQL sind drei logische Operatoren verfügbar:

- `AND`
- `OR`
- `NOT`

Daten sortieren – Wiederholung

- Abgerufene Zeilen mit der Klausel `ORDER BY` sortieren:
 - `ASC`: Aufsteigende Reihenfolge, Standard
 - `DESC`: Absteigende Reihenfolge
- Die Klausel `ORDER BY` steht am Ende der Anweisung `SELECT`:

```
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	Baer	PR_REP	70	07-JUN-02
4	Mavris	HR_REP	40	07-JUN-02
5	Higgins	AC_MGR	110	07-JUN-02
6	Faviet	FI_ACCOUNT	100	16-AUG-02
7	Greenberg	FI_MGR	100	17-AUG-02

...

ORACLE

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

Die Reihenfolge der Zeilen, die im Abfrageergebnis zurückgegeben werden, ist nicht festgelegt. Sie können die Zeilen mit der Klausel `ORDER BY` sortieren. Wenn Sie die Klausel `ORDER BY` verwenden, muss sie in der SQL-Anweisung als letzte Klausel angegeben werden. Sie können einen Ausdruck, einen Alias oder eine Spaltenposition als Sortierbedingung angeben.

Syntax

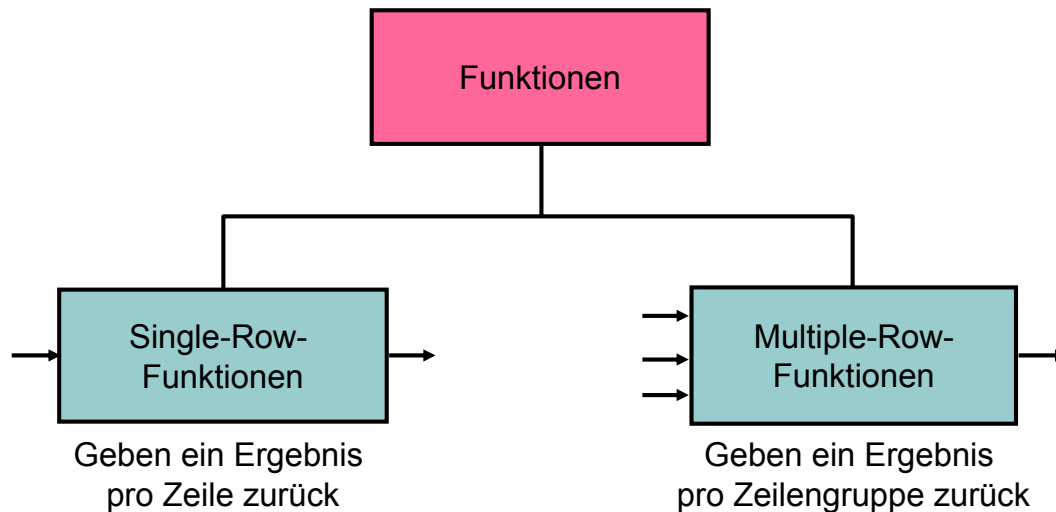
```
SELECT          expr
FROM            table
[WHERE          condition(s)]
[ORDER BY      {column, expr, numeric_position} [ASC|DESC]];
```

Für die Syntax gilt:

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

Wird die Klausel `ORDER BY` nicht angegeben, ist die Sortierreihenfolge nicht festgelegt. Der Oracle-Server ruft die Zeilen dann bei zwei Ausführungen derselben Abfrage möglicherweise in unterschiedlicher Reihenfolge ab. Um die Zeilen in einer bestimmten Reihenfolge anzuzeigen, verwenden Sie die Klausel `ORDER BY`.

SQL-Funktionen – Wiederholung



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

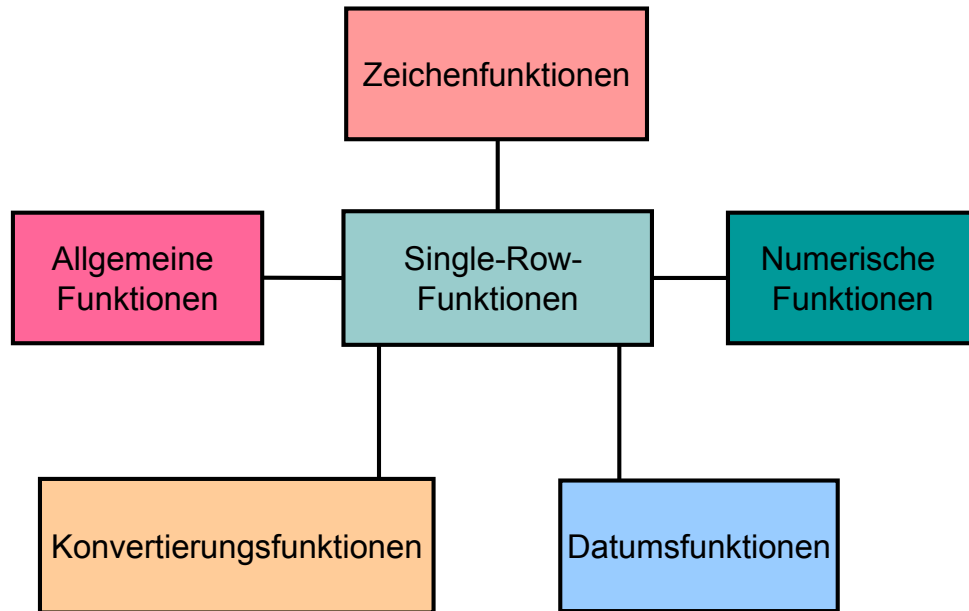
Single-Row-Funktionen

Diese Funktionen bearbeiten nur einzelne Zeilen und geben ein Ergebnis pro Zeile zurück. Es gibt verschiedene Typen von Single-Row-Funktionen, darunter Zeichen-, Datums- und Konvertierungsfunktionen sowie numerische und allgemeine Funktionen.

Multiple-Row-Funktionen

Diese Funktionen können Zeilengruppen bearbeiten und geben ein Ergebnis pro Zeilengruppe zurück. Sie werden auch als *Gruppenfunktionen* bezeichnet.

Single-Row-Funktionen – Wiederholung



ORACLE

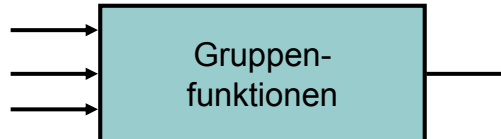
Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Es gibt verschiedene Typen von Single-Row-Funktionen:

- **Zeichenfunktionen:** Akzeptieren Zeichenwerte als Eingabe und geben Zeichenwerte und numerische Werte zurück
- **Numerische Funktionen:** Akzeptieren numerische Werte als Eingabe und geben numerische Werte zurück
- **Datumsfunktionen:** Bearbeiten Werte vom Datentyp `DATE` (Alle Datumsfunktionen geben einen Wert vom Datentyp `DATE` zurück. Eine Ausnahme bildet die Funktion `MONTHS_BETWEEN`, die einen Wert vom Datentyp `NUMBER` zurückgibt.)
- **Konvertierungsfunktionen:** Konvertieren einen Wert von einem Datentyp in einen anderen
- **Allgemeine Funktionen:**
 - `NVL`
 - `NVL2`
 - `NULLIF`
 - `COALESCE`
 - `CASE`
 - `DECODE`

Typen von Gruppenfunktionen – Wiederholung

- AVG
- COUNT
- MAX
- MIN
- STDDEV
- SUM
- 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 ({ * [DISTINCT <u>ALL</u>] <i>expr</i> })	Anzahl von Zeilen, bei denen die Auswertung von <i>expr</i> einen anderen Wert als einen Nullwert ergibt (Mit * werden alle gewählten Zeilen gezählt, einschließlich mehrfach vorhandener Zeilen und Zeilen mit Nullwerten.)
MAX ([DISTINCT <u>ALL</u>] <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.
VARIANCE ([DISTINCT <u>ALL</u>] <i>n</i>)	Varianz von <i>n</i> . Nullwerte werden ignoriert.

Unterabfragen – Wiederholung

- Eine Unterabfrage ist eine `SELECT`-Anweisung in einer Klausel, die innerhalb einer anderen `SELECT`-Anweisung verschachtelt ist.
- Syntax:

```
SELECT select_list
FROM   table
WHERE  expr operator
        (SELECT select_list
         FROM   table );
```

- Typen von Unterabfragen:

Single-Row-Unterabfrage	Multiple-Row-Unterabfrage
Gibt nur eine Zeile zurück	Gibt mehrere Zeilen zurück
Verwendet Single-Row-Vergleichsoperatoren	Verwendet Multiple-Row-Vergleichsoperatoren

ORACLE

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

Mithilfe von Unterabfragen können Sie einfache Anweisungen deutlich leistungstärker gestalten. Unterabfragen sind nützlich, wenn eine Abfrage auf einem Suchkriterium mit unbekannten Zwischenwerten basiert.

Unterabfragen lassen sich in verschiedene SQL-Klauseln einfügen, darunter:

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

Die Unterabfrage (innere Abfrage) wird einmal vor der Hauptabfrage (äußere Abfrage) ausgeführt. Das Ergebnis der Unterabfrage wird von der Hauptabfrage verwendet.

Single-Row-Unterabfragen verwenden Single-Row-Operatoren wie `=`, `>`, `<`, `>=`, `<=` und `<>`. Für Multiple-Row-Unterabfragen verwenden Sie einen Multiple-Row-Operator wie `IN`, `ANY` oder `ALL`.

Beispiel: Sie möchten Details von allen Mitarbeitern anzeigen, deren Gehalt dem Mindestgehalt entspricht.

```
SELECT last_name, salary, job_id
FROM   employees
WHERE  salary = (SELECT MIN(salary)
                FROM   employees );
```


Im Beispiel gibt die Gruppenfunktion `MIN` einen einzelnen Wert an die äußere Abfrage zurück.
Hinweis: In diesem Kurs lernen Sie den Umgang mit Multiple-Column-Unterabfragen. Bei Multiple-Column-Unterabfragen gibt die innere Anweisung `SELECT` mehrere Spalten zurück.

Tabellen mithilfe von DML-Anweisungen verwalten – Wiederholung

DML-(Data Manipulation Language-)Anweisungen werden ausgeführt, wenn Sie:

- neue Zeilen in Tabellen einfügen
- vorhandene Zeilen in Tabellen bearbeiten
- vorhandene Zeilen aus Tabellen löschen

Funktion	Beschreibung
INSERT	Fügt der Tabelle neue Zeilen hinzu
UPDATE	Ändert vorhandene Zeilen in der Tabelle
DELETE	Entfernt vorhandene Zeilen aus der Tabelle
MERGE	Aktualisiert, löscht oder fügt Tabellenzeilen ein

ORACLE

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

Um Datenbankdaten hinzuzufügen, zu aktualisieren oder zu löschen, führen Sie eine DML-Anweisung aus. Eine Zusammenstellung von DML-Anweisungen, die eine logische Arbeitseinheit bilden, wird als Transaktion bezeichnet. Mit der Anweisung `INSERT` fügen Sie einer Tabelle neue Zeilen hinzu. Mit der folgenden Syntax wird jeweils nur eine Zeile eingefügt.

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

Mit der Anweisung `INSERT` können Sie einer Tabelle Zeilen hinzufügen, indem Werte aus vorhandenen Tabellen abgeleitet werden. Anstelle der Klausel `VALUES` verwenden Sie eine Unterabfrage. Anzahl und Datentypen der Spalten in der Spaltenliste der Klausel `INSERT` müssen mit der Anzahl und den Datentypen der Werte in der Unterabfrage übereinstimmen.

Um die Werte bestimmter Zeilen zu ändern, geben Sie in der Anweisung `UPDATE` die Klausel `WHERE` an.

```
UPDATE table
SET column = value [, column = value, ...]
[WHERE condition];
```

Mit der Anweisung `DELETE` löschen Sie vorhandene Zeilen. Um bestimmte Zeilen zu löschen, geben Sie in der Anweisung `DELETE` die Klausel `WHERE` an.

```
DELETE [FROM] table  
[WHERE condition];
```

Die Anweisung `MERGE` wird in der Lektion "Daten bearbeiten" erläutert.

Lektionsagenda

- Kursziele und Kursagenda
- Datenbankschema, Anhänge und Übungen sowie Entwicklungsumgebungen für diesen Kurs
- Grundkonzepte von SQL – Wiederholung
- Dokumentation und zusätzliche Ressourcen zu Oracle Database 12c

ORACLE

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

Oracle Database – SQL-Dokumentation

- *Oracle Database New Features Guide*
- *Oracle Database Reference*
- *Oracle Database SQL Language Reference*
- *Oracle Database Concepts*
- *Oracle Database SQL Developer User's Guide Release 3.2*

ORACLE®

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

Die Dokumentationsbibliothek zu Oracle Database 12c Release 1 finden Sie unter <http://www.oracle.com/pls/db121/homepage>.

Zusätzliche Ressourcen

Weitere Informationen zur neuen Oracle Database 12c SQL:

- *Oracle Database 12c: New Features Self Studies*
- *Oracle By Example (OBE): Oracle Database 12c*
- *Oracle Learning Library:*
 - <http://www.oracle.com/goto/oll>
- Onlinehomepage von SQL Developer:
 - 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 erörtern
- Datenbankschema und Datenbanktabellen für diesen Kurs beschreiben
- Verfügbare Umgebungen für diesen Kurs bestimmen
- Grundkonzepte von SQL wiederholen

ORACLE

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

Übungen zu Lektion 1 – Überblick

Diese Übung behandelt folgende Themen:

- Online-Tutorial zu SQL Developer ausführen
- SQL Developer starten, eine neue Datenbankverbindung erstellen und durch die Tabellen navigieren
- SQL-Anweisungen mithilfe des SQL Worksheets ausführen
- Allgemeine SQL-Befehle ausführen

ORACLE

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

In diesen Übungen führen Sie SQL-Anweisungen mithilfe von SQL Developer aus.

Hinweis: Für alle schriftlichen Übungen wird SQL Developer als Entwicklungsumgebung verwendet. Sie können in diesem Kurs zwar auch die SQL*Plus-Umgebung verwenden, die Verwendung von SQL Developer wird jedoch empfohlen.

2

Data Dictionary Views – 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:

- Mit Data Dictionary Views nach Daten zu Ihren Objekten suchen
- Verschiedene Data Dictionary Views abfragen

ORACLE

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

In dieser Lektion werden Data Dictionary Views vorgestellt. Sie erfahren, wie Sie mit Dictionary Views Metadaten abrufen und Berichte zu Schemaobjekten erstellen.

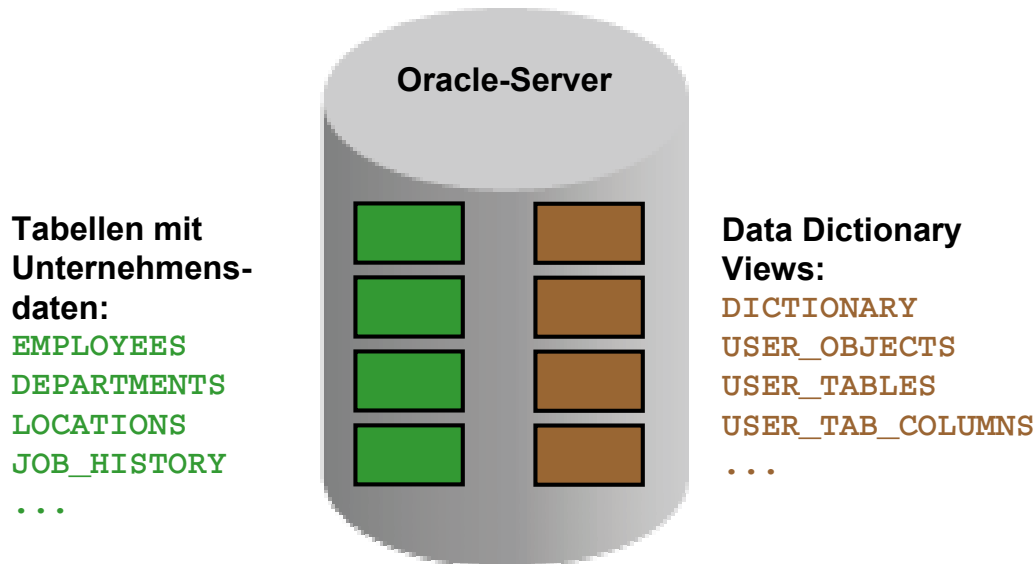
Lektionsagenda

- **Data Dictionary – Einführung**
- Dictionary Views nach folgenden Informationen abfragen:
 - Tabelleninformationen
 - Spalteninformationen
 - Constraint-Informationen
- Kommentare zu Tabellen hinzufügen und Dictionary Views nach Kommentarinformationen abfragen

ORACLE

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

Data Dictionary



ORACLE

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

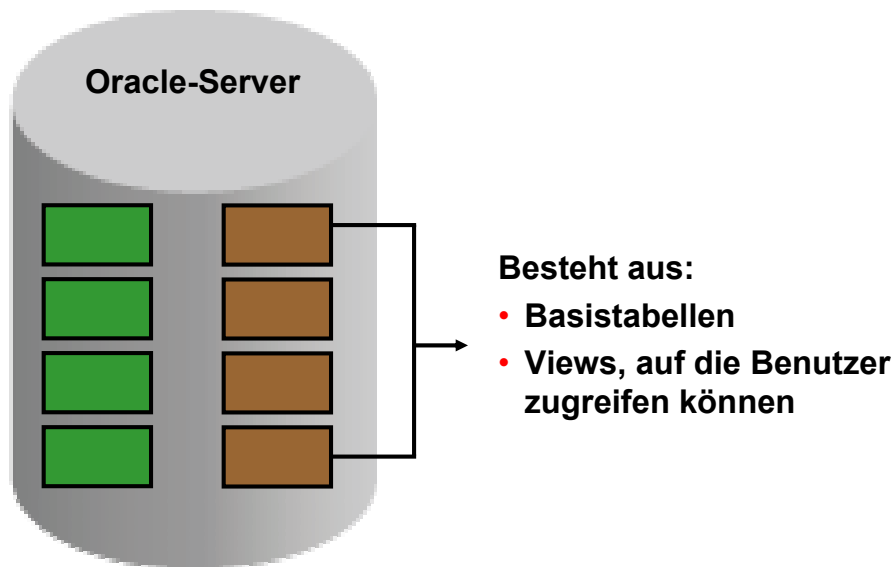
Benutzertabellen werden vom Benutzer erstellt und enthalten Unternehmensdaten (Beispiel: Tabelle `EMPLOYEES`). Daneben gibt es in der Oracle-Datenbank noch eine weitere Zusammenstellung von Tabellen und Views: das *Data Dictionary*. Diese Zusammenstellung wird vom Oracle-Server erstellt und verwaltet und enthält Informationen zur Datenbank. Das Data Dictionary ist genau wie andere Datenbankdaten in Tabellen und Views gegliedert. Das Data Dictionary ist nicht nur für jede Oracle-Datenbank von zentraler Bedeutung, sondern stellt gleichzeitig ein wichtiges Tool für alle Benutzer dar, vom Endbenutzer über den Anwendungsdesigner bis hin zum DBA.

Der Zugriff auf das Data Dictionary erfolgt mit SQL-Anweisungen. Da das Data Dictionary schreibgeschützt ist, können Sie nur Abfragen für die enthaltenen Tabellen und Views absetzen.

Mit Abfragen der auf den Dictionary-Tabellen basierenden Dictionary Views erhalten Sie beispielsweise folgende Informationen:

- Definitionen aller Schemaobjekte in der Datenbank (Tabellen, Views, Indizes, Synonyme, Sequences, Prozeduren, Funktionen, Packages, Trigger usw.)
- Standardwerte für Spalten
- Informationen zu Integritäts-Constraints
- Namen von Oracle-Benutzern
- Berechtigungen und Rollen der einzelnen Benutzer
- Andere allgemeine Datenbankinformationen

Data Dictionary-Struktur



ORACLE

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

In den zugrunde liegenden Basistabellen sind Informationen über die Datenbank gespeichert. Lese- und Schreibvorgänge an diesen Tabellen sollten nur vom Oracle-Server vorgenommen werden. Ein direkter Benutzerzugriff auf diese Tabellen erfolgt nur selten.

Die in den Basistabellen des Data Dictionarys gespeicherten Informationen werden in verschiedenen Views zusammengefasst und angezeigt. Diese Views schlüsseln die Daten aus den Basistabellen in nützliche Informationen auf (etwa Benutzer- oder Tabellennamen). Zur Vereinfachung werden Joins und `WHERE`-Klauseln angewendet. Den meisten Benutzern werden Zugriffsrechte auf die Views erteilt, nicht auf die Basistabellen.

Der Oracle-Benutzer `SYS` ist Eigentümer aller Basistabellen und aller Views, auf die Benutzer im Data Dictionary zugreifen können. Die Zeilen oder Schemaobjekte im Schema `SYS` sollten *unter keinen Umständen* von Oracle-Benutzern geändert werden (durch `UPDATE`-, `DELETE`- oder `INSERT`-Anweisungen), da dies die Datenintegrität beeinträchtigen kann.

Data Dictionary-Struktur

Benennungskonventionen für Views:

View-Präfix	Zweck
USER	View des Benutzers (Inhalt Ihres Schemas; Elemente, deren Eigentümer Sie sind)
ALL	Erweiterte View des Benutzers (Objekte, auf die Sie zugreifen können)
DBA	View des Datenbankadministrators (Inhalt der einzelnen Benutzerschemas)
V\$	Performancebezogene Daten

ORACLE

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

Das Data Dictionary besteht aus Gruppen von Views. In vielen Fällen enthält eine Gruppe drei Views mit ähnlichen Informationen, die sich durch ihr Präfix unterscheiden. Beispiel: `USER_OBJECTS`, `ALL_OBJECTS` und `DBA_OBJECTS`

Diese drei Views enthalten ähnliche Informationen über Objekte in der Datenbank. Sie unterscheiden sich nur in ihrem Gültigkeitsbereich. `USER_OBJECTS` enthält Informationen über Objekte, deren Eigentümer Sie sind oder die Sie erstellt haben. `ALL_OBJECTS` enthält Informationen über alle Objekte, auf die Sie Zugriff haben. `DBA_OBJECTS` enthält Informationen über alle Objekte, die allen Benutzern gehören. Views mit dem Präfix `ALL` oder `DBA` enthalten in der Regel die zusätzliche Spalte `OWNER`, die den Eigentümer des Objekts angibt.

Eine weitere Gruppe von Views ist mit dem Präfix `V$` versehen. Dabei handelt es sich um dynamische Views, die Informationen zur Performance enthalten. Dynamische Performance-tabellen sind keine echten Tabellen. Auf sie dürfen nur wenige Benutzer zugreifen. DBAs können jedoch Views zu den Tabellen abfragen und erstellen sowie anderen Benutzern Zugriff auf diese Views erteilen. Auf diese Views wird im Rahmen dieses Kurses nicht näher eingegangen.

Dictionary Views – Verwendung

Die View `DICTIONARY` enthält die Namen und Beschreibungen der Dictionary-Tabellen und Views.

```
DESCRIBE DICTIONARY
```

```
DESCRIBE dictionary
Name      Null Type
-----
TABLE_NAME  VARCHAR2(128)
COMMENTS    VARCHAR2(4000)
```

```
SELECT *
FROM   dictionary
WHERE  table_name = 'USER_OBJECTS';
```

	TABLE_NAME	COMMENTS
1	USER_OBJECTS	Objects owned by the user

ORACLE

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

Für den Einstieg in den Umgang mit Data Dictionary Views empfiehlt sich die View `DICTIONARY`. Sie enthält Namen und Kurzbeschreibungen aller Dictionary Views, auf die Sie zugreifen können.

Sie können Abfragen erstellen, um nach Informationen über einen bestimmten View-Namen zu suchen, oder die Spalte `COMMENTS` nach einem Wort oder einer Wortgruppe durchsuchen. Im Beispiel oben wird die View `DICTIONARY` beschrieben. Sie besteht aus zwei Spalten. Die Anweisung `SELECT` ruft Informationen über die Dictionary View `USER_OBJECTS` ab. Diese View enthält Informationen über alle Objekte, deren Eigentümer Sie sind.

Sie können Abfragen erstellen, um die Spalte `COMMENTS` nach einem Wort oder einer Wortgruppe zu durchsuchen. Beispiel: Die folgende Abfrage gibt die Namen aller Views zurück, auf die Sie zugreifen dürfen und die in der Spalte `COMMENTS` das Wort *columns* enthalten:

```
SELECT table_name
FROM   dictionary
WHERE  LOWER(comments) LIKE '%columns%';
```

Hinweis: Die Namen im Data Dictionary werden in Großbuchstaben angezeigt.

Views USER_OBJECTS und ALL_OBJECTS

USER_OBJECTS:

- USER_OBJECTS abfragen, um alle Objekte anzuzeigen, deren Eigentümer Sie sind
- Mit USER_OBJECTS können Sie eine Liste aller Objekt-namen und -typen in Ihrem Schema sowie folgende Informationen abrufen:
 - Erstellungsdatum
 - Datum der letzten Änderung
 - Status (gültig oder ungültig)

ALL_OBJECTS:

- ALL_OBJECTS abfragen, um alle Objekte anzuzeigen, auf die Sie zugreifen können

ORACLE®

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

Um die Namen und Typen aller Objekte in Ihrem Schema anzuzeigen, fragen Sie die View USER_OBJECTS ab. Diese View enthält verschiedene Spalten:

- OBJECT_NAME: Name des Objekts
- OBJECT_ID: Dictionary-Objektnummer des Objekts
- OBJECT_TYPE: Typ des Objekts (z. B. TABLE, VIEW, INDEX oder SEQUENCE)
- CREATED: Zeitstempel für die Erstellung des Objekts
- LAST_DDL_TIME: Zeitstempel für die letzte Änderung des Objekts aufgrund eines DDL-(Data Definition Language-)Befehls
- STATUS: Status des Objekts (VALID, INVALID oder N/A)
- GENERATED: Systemgenerierter Objektname? (Y|N)

Hinweis: Diese Spaltenliste ist nicht vollständig. Die vollständige Liste finden Sie in der *Oracle® Database Reference 12c Release 1* unter "USER_OBJECTS".

Sie können auch die View ALL_OBJECTS abfragen, um eine Liste aller Objekte anzuzeigen, auf die Sie Zugriff haben.

View USER_OBJECTS

```
SELECT object_name, object_type, created, status
FROM   user_objects
ORDER BY object_type;
```

	OBJECT_NAME	OBJECT_TYPE	CREATED	STATUS
1	JHIST_EMPLOYEE_IX	INDEX	23-AUG-12	VALID
2	EMP_DEPARTMENT_IX	INDEX	23-AUG-12	VALID
3	LOC_CITY_IX	INDEX	23-AUG-12	VALID
4	LOC_STATE_PROVINCE_IX	INDEX	23-AUG-12	VALID
5	LOC_COUNTRY_IX	INDEX	23-AUG-12	VALID
6	JHIST_DEPARTMENT_IX	INDEX	23-AUG-12	VALID
7	COUNTRY_C_ID_PK	INDEX	23-AUG-12	VALID
8	JHIST_EMP_ID_ST_DATE_PK	INDEX	23-AUG-12	VALID

...

ORACLE

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

Im Beispiel werden Name, Typ, Erstellungsdatum und Status aller Objekte dieses Benutzers angezeigt.

Die Spalte **OBJECT_TYPE** enthält den Wert **TABLE**, **VIEW**, **SEQUENCE**, **INDEX**, **PROCEDURE**, **FUNCTION**, **PACKAGE** oder **TRIGGER**.

Die Spalte **STATUS** enthält den Wert **VALID**, **INVALID** oder **N/A**. Während Tabellen immer gültig sind, können Views, Prozeduren, Funktionen, Packages und Trigger ungültig sein.

View CAT

Für eine vereinfachte Abfrage und Ausgabe fragen Sie die View **CAT** ab. Diese View enthält nur zwei Spalten: **TABLE_NAME** und **TABLE_TYPE**. Sie gibt die Namen all Ihrer Objekte vom Typ **INDEX**, **TABLE**, **CLUSTER**, **VIEW**, **SYNONYM**, **SEQUENCE** oder **UNDEFINED** an.

Hinweis: **CAT** ist ein Synonym für **USER_CATALOG**. Diese View listet Tabellen, Views, Synonyme und Sequences des Benutzers auf.

Lektionsagenda

- Data Dictionary – Einführung
- Dictionary Views nach folgenden Informationen abfragen:
 - Tabelleninformationen
 - Spalteninformationen
 - Constraint-Informationen
- Kommentare zu Tabellen hinzufügen und Dictionary Views nach Kommentarinformationen abfragen

ORACLE

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

Tabelleninformationen

USER_TABLES:

```
DESCRIBE user_tables
```

```
DESCRIBE user_tables
Name                Null    Type
-----
TABLE_NAME          NOT NULL VARCHAR2(128)
TABLESPACE_NAME      VARCHAR2(30)
CLUSTER_NAME         VARCHAR2(128)
IOT_NAME             VARCHAR2(128)
```

...

```
SELECT table_name
FROM   user_tables;
```

	TABLE_NAME
1	REGIONS
2	LOCATIONS
3	DEPARTMENTS
4	JOBS
5	EMPLOYEES
6	JOB_HISTORY

...

ORACLE

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

Mit der View `USER_TABLES` können Sie die Namen all Ihrer Tabellen abrufen. Die View `USER_TABLES` enthält Informationen über diese Tabellen. Neben dem Tabellennamen zeigt sie auch detaillierte Speicherinformationen an.

Die View `TABS` ist ein Synonym für die View `USER_TABLES`. Fragen Sie diese View ab, um eine Liste Ihrer Tabellen anzuzeigen:

```
SELECT table_name
FROM   tabs;
```

Hinweis: Eine vollständige Liste der Spalten in der View `USER_TABLES` finden Sie in der *Oracle® Database Reference 12c Release 1* unter "`USER_TABLES`".

Sie können auch die View `ALL_TABLES` abfragen, um eine Liste aller Tabellen anzuzeigen, auf die Sie Zugriff haben.

Spalteninformationen

USER_TAB_COLUMNS:

```
DESCRIBE user_tab_columns
```

Name	Null	Type
TABLE_NAME	NOT NULL	VARCHAR2(128)
COLUMN_NAME	NOT NULL	VARCHAR2(128)
DATA_TYPE		VARCHAR2(128)
DATA_TYPE_MOD		VARCHAR2(3)
DATA_TYPE_OWNER		VARCHAR2(128)
DATA_LENGTH	NOT NULL	NUMBER
DATA_PRECISION		NUMBER
DATA_SCALE		NUMBER
NULLABLE		VARCHAR2(1)

...

ORACLE

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

Um detaillierte Informationen über die Spalten in Ihren Tabellen zu erhalten, fragen Sie die View `USER_TAB_COLUMNS` ab. Während die View `USER_TABLES` Informationen zum Namen und Speicherort der Tabellen bereitstellt, liefert die View `USER_TAB_COLUMNS` detaillierte Spalteninformationen.

Diese View enthält beispielsweise folgende Informationen:

- Spaltennamen
- Datentypen der Spalten
- Länge der Datentypen
- Gesamtstellenzahl und Nachkommastellen in Spalten des Typs `NUMBER`
- Zulässigkeit von Nullwerten (Constraint `NOT NULL` für die Spalte vorhanden?)
- Standardwert

Hinweis: Eine vollständige Liste und Beschreibung der Spalten in der View `USER_TAB_COLUMNS` finden Sie in der *Oracle® Database Reference 12c Release 1* unter "`USER_TAB_COLUMNS`".

Spalteninformationen

```
SELECT column_name, data_type, data_length,  
       data_precision, data_scale, nullable  
FROM   user_tab_columns  
WHERE  table_name = 'EMPLOYEES';
```

	COLUMN_NAME	DATA_TYPE	DATA_LENGTH	DATA_PRECISION	DATA_SCALE	NULLABLE
1	EMPLOYEE_ID	NUMBER	22	6	0	N
2	FIRST_NAME	VARCHAR2	20	(null)	(null)	Y
3	LAST_NAME	VARCHAR2	25	(null)	(null)	N
4	EMAIL	VARCHAR2	25	(null)	(null)	N
5	PHONE_NUMBER	VARCHAR2	20	(null)	(null)	Y
6	HIRE_DATE	DATE	7	(null)	(null)	N
7	JOB_ID	VARCHAR2	10	(null)	(null)	N
8	SALARY	NUMBER	22	8	2	Y
9	COMMISSION_PCT	NUMBER	22	2	2	Y
10	MANAGER_ID	NUMBER	22	6	0	Y
11	DEPARTMENT_ID	NUMBER	22	4	0	Y

ORACLE

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

Über die Tabelle `USER_TAB_COLUMNS` können Sie Details zu Ihren Spalten abfragen, darunter Name, Datentyp, Länge des Datentyps, Constraints vom Typ `NULL` und Standardwert.

Im Beispiel oben werden die Spalten, Datentypen, Datenlängen und `NULL`-Constraints für die Tabelle `EMPLOYEES` angezeigt. Ähnliche Informationen gibt auch der Befehl `DESCRIBE` aus.

Mit der Dictionary View `USER_UNUSED_COL_TABS` zeigen Sie Informationen zu Spalten an, die als nicht verwendet (`UNUSED`) markiert wurden.

Hinweis: Die Namen der Objekte im Data Dictionary werden in Großbuchstaben angezeigt.

Constraint-Informationen

- USER_CONSTRAINTS beschreibt die Constraint-Definitionen für Ihre Tabellen.
- USER_CONS_COLUMNS beschreibt Spalten, deren Eigentümer Sie sind und die in Constraints angegeben sind.

```
DESCRIBE user_constraints
```

```
DESCRIBE user_constraints
Name          Null    Type
-----
OWNER         VARCHAR2(128)
CONSTRAINT_NAME NOT NULL VARCHAR2(128)
CONSTRAINT_TYPE VARCHAR2(1)
TABLE_NAME    NOT NULL VARCHAR2(128)
SEARCH_CONDITION LONG()
R_OWNER       VARCHAR2(128)
R_CONSTRAINT_NAME VARCHAR2(128)
DELETE_RULE   VARCHAR2(9)
STATUS        VARCHAR2(8)
```

...

ORACLE

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

Sie können zahlreiche Informationen zu Ihren Constraints abrufen, darunter den Namen und Typ der Constraints, den Namen der Tabelle, für die das Constraint gilt, die Bedingung für Constraints vom Typ CHECK, Informationen zu Constraints vom Typ FOREIGN KEY, die Löschregel für Constraints vom Typ FOREIGN KEY und den Status.

Hinweis: Eine vollständige Liste und Beschreibung der Spalten in der View USER_CONSTRAINTS finden Sie in der *Oracle® Database Reference 12c Release 1* unter "USER_CONSTRAINTS".

USER_CONSTRAINTS – Beispiel

```
SELECT constraint_name, constraint_type,
       search_condition, r_constraint_name,
       delete_rule, status
FROM   user_constraints
WHERE  table_name = 'EMPLOYEES';
```

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_CONSTRAINT_NAME	DELETE_RULE	STATUS
1	EMP_MANAGER_FK	R	(null)	EMP_EMP_ID_PK	NO ACTION	ENABLED
2	EMP_JOB_FK	R	(null)	JOB_ID_PK	NO ACTION	ENABLED
3	EMP_DEPT_FK	R	(null)	DEPT_ID_PK	NO ACTION	ENABLED
4	EMP_EMP_ID_PK	P	(null)	(null)	(null)	ENABLED
5	EMP_EMAIL_UK	U	(null)	(null)	(null)	ENABLED
6	EMP_SALARY_MIN	C	salary > 0	(null)	(null)	ENABLED
7	EMP_JOB_NN	C	"JOB_ID" IS NOT NULL	(null)	(null)	ENABLED
8	EMP_HIRE_DATE_NN	C	"HIRE_DATE" IS NOT NULL	(null)	(null)	ENABLED
9	EMP_EMAIL_NN	C	"EMAIL" IS NOT NULL	(null)	(null)	ENABLED
10	EMP_LAST_NAME_NN	C	"LAST_NAME" IS NOT NULL	(null)	(null)	ENABLED

ORACLE

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

Im Beispiel auf der Folie wird die View `USER_CONSTRAINTS` abgefragt, um für die Tabelle `EMPLOYEES` die Namen und Typen der Constraints, die CHECK-Bedingungen, den Namen des vom Fremdschlüssel referenzierten UNIQUE-Constraints, die Löschregel für einen Fremdschlüssel und den Status der Constraints zu ermitteln.

`CONSTRAINT_TYPE` kann folgende Werte aufweisen:

- C (CHECK-Constraint für eine Tabelle oder NOT NULL)
- P (Primärschlüssel)
- U (eindeutiger Schlüssel)
- R (referenzielle Integrität)
- V (mit Option CHECK für eine View)
- O (mit Option READ-ONLY für eine View)

`DELETE_RULE` kann folgende Werte aufweisen:

- **CASCADE:** Wenn Sie den übergeordneten Datensatz löschen, werden auch die untergeordneten Datensätze gelöscht.
- **SET NULL:** Wenn Sie den übergeordneten Datensatz löschen, wird der entsprechende untergeordnete Datensatz in NULL geändert.
- **NO ACTION:** Ein übergeordneter Datensatz kann nur gelöscht werden, wenn kein untergeordneter Datensatz vorhanden ist.

`STATUS` kann folgende Werte aufweisen:

- **ENABLED:** Das Constraint ist aktiviert.
- **DISABLED:** Das Constraint ist deaktiviert.

USER_CONS_COLUMNS abfragen

```
DESCRIBE user_cons_columns
```

```
DESCRIBE user_cons_columns
Name          Null    Type
-----
OWNER          NOT NULL VARCHAR2(128)
CONSTRAINT_NAME NOT NULL VARCHAR2(128)
TABLE_NAME     NOT NULL VARCHAR2(128)
COLUMN_NAME    VARCHAR2(4000)
POSITION      NUMBER
```

```
SELECT constraint_name, column_name
FROM   user_cons_columns
WHERE  table_name = 'EMPLOYEES';
```

	CONSTRAINT_NAME	COLUMN_NAME
1	EMP_LAST_NAME_NN	LAST_NAME
2	EMP_EMAIL_NN	EMAIL
3	EMP_HIRE_DATE_NN	HIRE_DATE
4	EMP_JOB_NN	JOB_ID
5	EMP_SALARY_MIN	SALARY
6	EMP_EMAIL_UK	EMAIL
7	EMP_EMP_ID_PK	EMPLOYEE_ID
8	EMP_DEPT_FK	DEPARTMENT_ID
9	EMP_JOB_FK	JOB_ID
10	EMP_MANAGER_FK	MANAGER_ID

ORACLE

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

Um die Namen der Spalten zu ermitteln, für die ein Constraint gilt, fragen Sie die Dictionary View `USER_CONS_COLUMNS` ab. Diese View enthält den Namen des Constraint-Eigentümers, den Namen des Constraints, die Tabelle, für die das Constraint gilt, die Namen der Spalten mit dem Constraint sowie die ursprüngliche Position der Spalte oder des Attributs in der Definition des Objekts.

Hinweis: Ein Constraint kann für mehrere Spalten gelten.

Sie können auch einen Join erstellen, der `USER_CONSTRAINTS` und `USER_CONS_COLUMNS` verknüpft, um eine angepasste Ausgabe aus beiden Tabellen zu erzeugen.

Lektionsagenda

- Data Dictionary – Einführung
- Dictionary Views nach folgenden Informationen abfragen:
 - Tabelleninformationen
 - Spalteninformationen
 - Constraint-Informationen
- Kommentare zu Tabellen hinzufügen und Dictionary Views nach Kommentarinformationen abfragen

ORACLE

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

Kommentare zu Tabellen hinzufügen

- Mit der Anweisung `COMMENT` Kommentare zu einer Tabelle oder Spalte hinzufügen:

```
COMMENT ON TABLE employees  
IS 'Employee Information';
```

```
COMMENT ON COLUMN employees.first_name  
IS 'First name of the employee';
```

- Kommentare mit folgenden Data Dictionary Views anzeigen:
 - `ALL_COL_COMMENTS`
 - `USER_COL_COMMENTS`
 - `ALL_TAB_COMMENTS`
 - `USER_TAB_COMMENTS`

ORACLE

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

Mit der Anweisung `COMMENT` können Sie Kommentare mit bis zu 4.000 Byte über Spalten, Tabellen, Views oder Snapshots hinzufügen. Der Kommentar wird im Data Dictionary gespeichert und kann in folgenden Data Dictionary Views in der Spalte `COMMENTS` angezeigt werden:

- `ALL_COL_COMMENTS`
- `USER_COL_COMMENTS`
- `ALL_TAB_COMMENTS`
- `USER_TAB_COMMENTS`

Syntax

```
COMMENT ON {TABLE table | COLUMN table.column}  
IS 'text';
```

Für die Syntax gilt:

table Steht für den Namen der Tabelle
column Steht für den Namen der Spalte in einer Tabelle
text Steht für den Kommentartext

Um einen Kommentar aus der Datenbank zu löschen, legen Sie ihn auf eine leere Zeichenfolge (`' '`) fest:

```
COMMENT ON TABLE employees IS '';
```

Quiz

Welche Informationen sind in den auf Dictionary-Tabellen basierenden Dictionary Views enthalten?

- a. Definitionen aller Schemaobjekte in der Datenbank
- b. Standardwerte für die Spalten
- c. Informationen zu Integritäts-Constraints
- d. Berechtigungen und Rollen der einzelnen Benutzer
- e. Alle oben genannten Angaben

ORACLE

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

Richtige Antwort: e

Zusammenfassung

In dieser Lektion haben Sie gelernt, Informationen zu Ihren Objekten mithilfe der folgenden Dictionary Views abzurufen:

- `DICTIONARY`
- `USER_OBJECTS`
- `USER_TABLES`
- `USER_TAB_COLUMNS`
- `USER_CONSTRAINTS`
- `USER_CONS_COLUMNS`

ORACLE

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

In dieser Lektion haben Sie einige der für Sie verfügbaren Dictionary Views kennengelernt, mit denen Sie Informationen über Ihre Tabellen, Constraints, Views, Sequences und Synonyme abrufen können.

Übungen zu Lektion 2 – Überblick

Diese Übung behandelt folgende Themen:

- Dictionary Views nach Tabellen- und Spalteninformationen abfragen
- Dictionary Views nach Constraint-Informationen abfragen
- Kommentare zu Tabellen hinzufügen und Dictionary Views nach Kommentarinformationen abfragen

ORACLE

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

In dieser Übung fragen Sie Dictionary Views ab, um Informationen über Objekte in Ihrem Schema zu erhalten.

3

Sequences, Synonyme und Indizes 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:

- Sequences erstellen, verwalten und verwenden
- Private und öffentliche Synonyme erstellen
- Indizes erstellen und verwalten
- Verschiedene Data Dictionary Views nach Informationen zu Sequences, Synonymen und Indizes abfragen

ORACLE

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

In dieser Lektion lernen Sie Sequences, Synonyme und Indizes kennen und erhalten grundlegende Informationen zur Erstellung und Verwendung dieser Objekte.

Lektionsagenda

- Sequences – Überblick
 - Sequences erstellen, verwenden und ändern
 - Sequence-Werte cachen
 - Pseudospalten `NEXTVAL` und `CURRVAL`
 - SQL-Spaltenstandards mithilfe von Sequences einstellen
- Synonyme – Überblick
 - Synonyme erstellen und löschen
- Indizes – Überblick
 - Indizes erstellen
 - `CREATE TABLE`-Anweisungen
 - Funktionsbasierte Indizes erstellen
 - Mehrere Indizes für dieselbe Spaltengruppe erstellen
 - Indizes entfernen

ORACLE

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

Datenbankobjekte

Objekt	Beschreibung
Tabelle	Grundlegende Speichereinheit, die aus Zeilen besteht
View	Stellt Teilmengen von Daten aus einzelnen oder mehreren Tabellen logisch dar
Sequence	Generiert numerische Werte
Index	Verbessert die Performance von Datenabfragen
Synonym	Gibt Objekten alternative Namen

ORACLE

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

Neben Tabellen gibt es in einer Datenbank verschiedene andere Objekte.

Mit Views können Sie Daten aus den Tabellen darstellen und verbergen.

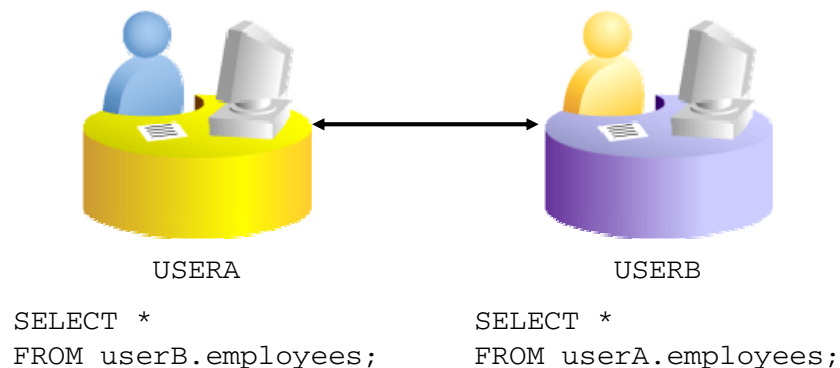
Viele Anwendungen erfordern eindeutige Zahlen als Primärschlüsselwerte. Um dieser Anforderung Rechnung zu tragen, können Sie entweder Code in die Anwendung einfügen oder eindeutige Nummern mithilfe einer Sequence generieren.

Ein Index kann unter Umständen dazu beitragen, die Datenabfrage zu beschleunigen. Sie können Indizes auch verwenden, um für eine Spalte oder eine Gruppe von Spalten Eindeutigkeit zu erzwingen.

Mit Synonymen können Sie Objekten alternative Namen geben.

Tabellen anderer Benutzer referenzieren

- Tabellen anderer Benutzer befinden sich nicht im Schema des Benutzers.
- Der Name des Eigentümers muss als Präfix für diese Tabellen verwendet werden.



ORACLE

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

Ein Schema ist eine Zusammenstellung von logischen Strukturen aus Daten oder *Schemaobjekten*. Jedes Schema gehört einem Datenbankbenutzer und trägt den Namen dieses Benutzers. Pro Benutzer gibt es ein Schema.

Schemaobjekte, zu denen Tabellen, Views, Synonyme, Sequences, Stored Procedures, Indizes, Cluster und Datenbanklinks zählen, können mit SQL erstellt und bearbeitet werden.

Gehört eine Tabelle nicht dem Benutzer, muss ihr der Name des Eigentümers vorangestellt werden. Beispiel: In den beiden Schemas `USERA` und `USERB` gibt es jeweils eine Tabelle `EMPLOYEES`. Möchte `USERA` auf die Tabelle `EMPLOYEES` im Schema von `USERB` zugreifen, muss `USERA` dem Tabellennamen den Schemanamen als Präfix voranstellen:

```
SELECT *  
FROM   userb.employees;
```

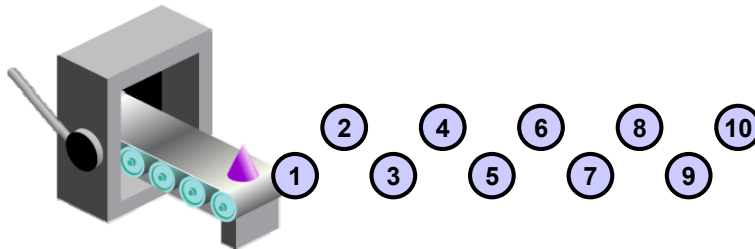
Wenn `USERB` auf die Tabelle `EMPLOYEES` im Schema von `USERA` zugreifen möchte, muss `USERB` dem Tabellennamen den Schemanamen als Präfix voranstellen:

```
SELECT *  
FROM   usera.employees;
```

Sequences

Eine Sequence:

- kann automatisch eindeutige Nummern generieren
- ist ein gemeinsam verwendbares Objekt
- kann verwendet werden, um einen Primärschlüsselwert zu erstellen
- ersetzt Anwendungscode
- beschleunigt bei Speicherung im Cache den Zugriff auf Sequence-Werte



ORACLE

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

Eine Sequence ist ein vom Benutzer erstelltes Datenbankobjekt, das von mehreren Benutzern gemeinsam verwendet werden kann, um Ganzzahlen zu generieren.

Sequences werden definiert, um eindeutige Werte zu generieren oder um dieselben Nummern erneut zu verwenden.

Sequences werden üblicherweise verwendet, um einen Primärschlüsselwert zu erstellen, der für jede Zeile eindeutig sein muss. Eine Sequence wird von einer internen Oracle-Routine generiert und schrittweise erhöht (oder verringert). Dieses Objekt kann Zeit sparen, da auf diese Weise weniger Anwendungscode für die Erstellung einer Routine zur Sequence-Generierung benötigt wird.

Sequence-Nummern werden unabhängig von Tabellen gespeichert und generiert. Daher kann dieselbe Sequence für mehrere Tabellen verwendet werden.

Anweisung CREATE SEQUENCE – Syntax

Sequence definieren, um automatisch aufeinanderfolgende Nummern zu generieren:

```
CREATE SEQUENCE [ schema. ] sequence
  [ { START WITH|INCREMENT BY } integer
    | { MAXVALUE integer | NOMAXVALUE }
    | { MINVALUE integer | NOMINVALUE }
    | { CYCLE | NOCYCLE }
    | { CACHE integer | NOCACHE }
    | { ORDER | NOORDER }
  ];
```

ORACLE®

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

Mit der Anweisung `CREATE SEQUENCE` generieren Sie automatisch aufeinanderfolgende Nummern.

Für die Syntax gilt:

<i>Sequence</i>	Steht für den Namen des Sequence-Generators
<code>START WITH <i>n</i></code>	Gibt die erste Sequence-Nummer an, die generiert werden soll (Fehlt diese Klausel, beginnt die Sequence mit 1.)
<code>INCREMENT BY <i>n</i></code>	Gibt das Intervall zwischen Sequence-Nummern an, wobei <i>n</i> eine Ganzzahl ist (Fehlt diese Klausel, wird 1 als Intervall in der Sequence verwendet.)
<code>MAXVALUE <i>n</i></code>	Gibt den höchsten Wert an, den die Sequence generieren kann
<code>NOMAXVALUE</code>	Gibt den Höchstwert 10^{27} für aufsteigende Sequences und -1 für absteigende Sequences an (Standardoption)
<code>MINVALUE <i>n</i></code>	Gibt den kleinsten Wert der Sequence an
<code>NOMINVALUE</code>	Gibt den Mindestwert 1 für aufsteigende Sequences und $-(10^{26})$ für absteigende Sequences an (Standardoption)

ORDER	Geben Sie <code>ORDER</code> an, um sicherzustellen, dass Sequence-Nummern in der Reihenfolge ihrer Anforderung generiert werden. Diese Klausel ist nützlich, wenn Sie die Sequence-Nummern als Zeitstempel verwenden.
NOORDER	Geben Sie <code>NOORDER</code> an, wenn die Sequence-Nummern nicht unbedingt in der Reihenfolge ihrer Anforderung generiert werden müssen. Dies ist die Standardeinstellung.
CYCLE NOCYCLE	Gibt an, ob die Sequence weiterhin Werte generieren soll, nachdem ihr Höchst- oder Mindestwert erreicht wurde (<code>NOCYCLE</code> ist die Standardoption.)
CACHE <i>n</i> NOCACHE	Gibt an, wie viele Werte der Oracle-Server reserviert und im Cache speichert (Standardmäßig werden 20 Werte im Cache gespeichert.)

Sequences erstellen

- Erstellen Sie die Sequence `DEPT_DEPTID_SEQ` für den Primärschlüssel der Tabelle `DEPARTMENTS`.
- Verwenden Sie dabei nicht die Option `CYCLE`.

```
CREATE SEQUENCE dept_deptid_seq  
    START WITH 280  
    INCREMENT BY 10  
    MAXVALUE 9999  
    NOCACHE  
    NOCYCLE;
```

```
sequence DEPT_DEPTID_SEQ created.
```

ORACLE®

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

Im Beispiel auf der Folie wird die Sequence `DEPT_DEPTID_SEQ` erstellt, die für die Spalte `DEPARTMENT_ID` in der Tabelle `DEPARTMENTS` verwendet werden soll. Die Sequence beginnt bei 280, lässt kein Caching zu und ist nicht zyklisch.

Verwenden Sie die Option `CYCLE` nicht, wenn die Sequence Primärschlüsselwerte generieren soll, es sei denn, Sie verfügen über einen zuverlässigen Mechanismus, der alte Zeilen schneller entfernt als die Sequence den Zyklus durchläuft.

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

Hinweis: Sequences sind nicht an Tabellen gebunden. Generell sollten Sie Sequences nach dem beabsichtigten Verwendungszweck benennen. Sequences können jedoch unabhängig von ihrem Namen überall verwendet werden.

Pseudospalten NEXTVAL und CURRVAL

- NEXTVAL gibt den nächsten verfügbaren Sequence-Wert zurück. Bei jeder Referenzierung wird ein eindeutiger Wert zurückgegeben, selbst bei verschiedenen Benutzern.
- CURRVAL ruft den aktuellen Sequence-Wert ab.
- CURRVAL enthält erst dann einen Wert, wenn NEXTVAL für die Sequence ausgegeben wurde.

ORACLE

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

Nachdem Sie eine Sequence erstellt haben, generiert sie aufeinanderfolgende Nummern zur Verwendung in den Tabellen. Um die Sequence-Werte zu referenzieren, verwenden Sie die Pseudospalten NEXTVAL und CURRVAL.

Mit der Pseudospalte NEXTVAL extrahieren Sie aufeinanderfolgende Sequence-Nummern aus einer bestimmten Sequence. Sie müssen NEXTVAL zusammen mit dem Sequence-Namen angeben. Wenn Sie *sequence*.NEXTVAL referenzieren, wird eine neue Sequence-Nummer generiert und die aktuelle Sequence-Nummer in CURRVAL übertragen.

Mithilfe der Pseudospalte CURRVAL referenzieren Sie eine vom aktuellen Benutzer soeben generierte Sequence-Nummer. Bevor CURRVAL referenziert werden kann, muss jedoch in der Session des aktuellen Benutzers mit NEXTVAL eine Sequence-Nummer generiert werden. Sie müssen CURRVAL zusammen mit dem Sequence-Namen angeben. Bei Referenzierung von *sequence*.CURRVAL wird der Wert angezeigt, der zuletzt an den Prozess des Benutzers zurückgegeben wurde.

Regeln zur Verwendung von NEXTVAL und CURRVAL

Sie können NEXTVAL und CURRVAL in folgenden Kontexten einsetzen:

- SELECT-Liste einer SELECT-Anweisung, die nicht Bestandteil einer Unterabfrage ist
- SELECT-Liste einer Unterabfrage in einer INSERT-Anweisung
- Klausel VALUES einer INSERT-Anweisung
- Klausel SET einer UPDATE-Anweisung

In folgenden Kontexten können Sie NEXTVAL und CURRVAL nicht einsetzen:

- SELECT-Liste einer View
- Anweisung SELECT mit dem Schlüsselwort DISTINCT
- Anweisung SELECT mit Klauseln vom Typ GROUP BY, HAVING oder ORDER BY
- Unterabfrage in einer Anweisung vom Typ SELECT, DELETE oder UPDATE

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

Sequences – Beispiele

- Neue Abteilung "Support" am Standort 2500 einfügen:

```
INSERT INTO departments (department_id,  
                        department_name, location_id)  
VALUES (dept_deptid_seq.NEXTVAL,  
      'Support', 2500);
```

1 rows inserted

- Aktuellen Wert für die Sequence DEPT_DEPTID_SEQ anzeigen:

```
SELECT dept_deptid_seq.CURRVAL  
FROM dual;
```

ORACLE

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

Im Beispiel auf der Folie wird eine neue Abteilung in die Tabelle `DEPARTMENTS` eingefügt. Mithilfe der Sequence `DEPT_DEPTID_SEQ` wird eine neue Abteilungsnummer generiert.

Den aktuellen Wert der Sequence zeigen Sie mit `sequence_name.CURRVAL` an, wie im zweiten Beispiel auf der Folie dargestellt.

Angenommen, Sie möchten jetzt Mitarbeiter für die neue Abteilung einstellen. Die Anweisung `INSERT`, die für alle neuen Mitarbeiter ausgeführt werden muss, kann folgenden Code enthalten:

```
INSERT INTO employees (employee_id, department_id, ...)  
VALUES (employees_seq.NEXTVAL, dept_deptid_seq.CURRVAL, ...);
```

Hinweis: Im vorhergehenden Beispiel wird davon ausgegangen, dass bereits eine Sequence `EMPLOYEE_SEQ` erstellt wurde, um neue Personalnummern zu generieren.

SQL-Spaltenstandards mithilfe von Sequences einstellen

- Die SQL-Syntax für Spaltenstandards lässt `<sequence>.nextval`, `<sequence>.currval` als Ausdruck für numerische Spalten zu, wobei `<sequence>` einer Oracle Database-Sequence entspricht.
- Der Ausdruck `DEFAULT` kann die Sequence-Pseudospalten `CURRVAL` und `NEXTVAL` enthalten. Voraussetzung dabei ist, dass die Sequence vorhanden ist und Sie über entsprechende Zugriffsberechtigungen verfügen.

```
CREATE SEQUENCE s1 START WITH 1;  
CREATE TABLE emp (a1 NUMBER DEFAULT s1.NEXTVAL NOT  
NULL, a2 VARCHAR2(10));  
INSERT INTO emp (a2) VALUES ('john');  
INSERT INTO emp (a2) VALUES ('mark');  
SELECT * FROM emp;
```

```
sequence s1 created.  
table EMP created.  
1 rows inserted.  
1 rows inserted.  
A1 A2  
--  
1 john  
2 mark
```

ORACLE

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

Die SQL-Syntax für Spaltenstandards wurde erweitert und unterstützt nun den Ausdruck `<sequence>.nextval`, `<sequence>.currval` für numerische Spalten, wobei `<sequence>` einer Oracle Database-Sequence entspricht.

Der Ausdruck `DEFAULT` kann die Sequence-Pseudospalten `CURRVAL` und `NEXTVAL` enthalten. Voraussetzung dabei ist, dass die Sequence vorhanden ist und Sie über entsprechende Zugriffsberechtigungen verfügen. Benutzer, die `INSERT`-Vorgänge an einer Tabelle ausführen, benötigen Zugriffsberechtigungen für die Sequence. Wird die Sequence gelöscht, resultieren nachfolgende DML-Vorgänge vom Typ `INSERT`, bei denen *expr* für Standardwerte verwendet wird, in einem Kompilierungsfehler.

Im Beispiel auf der Folie wird die Sequence `s1` erstellt, die mit dem Wert 1 beginnt.

Sequence-Werte im Cache speichern

- Das Caching von Sequence-Werten ermöglicht den schnelleren Zugriff auf diese Werte.
- Lücken zwischen den Sequence-Werten können in folgenden Fällen auftreten:
 - Bei einem Rollback
 - Bei Systemabstürzen
 - Wenn eine Sequence in einer anderen Tabelle verwendet wird

ORACLE

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

Sie können Sequences im Cache speichern, um einen schnelleren Zugriff auf diese Sequence-Werte zu ermöglichen. Der Cache wird gefüllt, wenn Sie die Sequence zum ersten Mal referenzieren. Bei jeder Anforderung des nächsten Sequence-Wertes wird der Wert aus der Sequence im Cache abgerufen. Nachdem der letzte Sequence-Wert abgerufen wurde, liest die nächste Anforderung neue Sequence-Werte in den Cache ein.

Sequence-Lücken

Sequence-Generatoren erzeugen aufeinanderfolgende Nummern normalerweise ohne Lücken. Dieser Vorgang läuft jedoch unabhängig von einem Commit oder Rollback ab. Beim Zurückrollen einer Anweisung, die eine Sequence enthält, geht der Wert verloren.

Auch ein Systemausfall kann zu Lücken in einer Sequence führen. Wenn die Sequence Werte im Cache speichert, gehen diese Werte bei einem Systemabsturz verloren.

Da Sequences nicht direkt an eine bestimmte Tabelle gebunden sind, kann dieselbe Sequence für mehrere Tabellen verwendet werden. In diesem Fall kann jedoch jede Tabelle Lücken in den aufeinanderfolgenden Nummern enthalten.

Sequences ändern

Inkrementwert, Höchstwert, Mindestwert, Option CYCLE oder Option CACHE ändern:

```
ALTER SEQUENCE dept_deptid_seq  
        INCREMENT BY 20  
        MAXVALUE 999999  
        NOCACHE  
        NOCYCLE;
```

```
sequence DEPT_DEPTID_SEQ altered.
```

ORACLE

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

Wenn der Grenzwert `MAXVALUE` für Ihre Sequence erreicht ist, werden keine weiteren Werte aus der Sequence zugewiesen. Sie erhalten dann die Fehlermeldung, dass die Sequence den `MAXVALUE` überschreitet. Um die Sequence weiterhin benutzen zu können, ändern Sie sie mit der Anweisung `ALTER SEQUENCE`.

Syntax

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

In der Syntax steht *sequence* für den Namen des Sequence-Generators.

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

Sequences ändern – Richtlinien

- Sie müssen der Eigentümer sein oder über die Berechtigung `ALTER` für die Sequence verfügen.
- Änderungen wirken sich nur auf künftige Sequence-Nummern aus.
- Die Sequence muss gelöscht und neu erstellt werden, wenn die Sequence bei einer anderen Zahl neu beginnen soll.
- Bestimmte Validierungsmaßnahmen werden durchgeführt.
- Sequences werden mit der Anweisung `DROP` entfernt:

```
DROP SEQUENCE dept_deptid_seq;
```

```
sequence DEPT_DEPTID_SEQ dropped.
```

ORACLE

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

- Um eine Sequence ändern zu können, müssen Sie der Eigentümer sein oder über die Berechtigung `ALTER` für die Sequence verfügen. Um Sequences zu entfernen, müssen Sie der Eigentümer sein oder über die Berechtigung `DROP ANY SEQUENCE` verfügen.
- Die Anweisung `ALTER SEQUENCE` wirkt sich nur auf künftige Sequence-Nummern aus.
- Die Option `START WITH` kann nicht mit `ALTER SEQUENCE` geändert werden. Die Sequence muss gelöscht und neu erstellt werden, wenn die Sequence bei einer anderen Zahl neu beginnen soll.
- Bestimmte Validierungsmaßnahmen werden durchgeführt. Dadurch wird zum Beispiel verhindert, dass ein neuer `MAXVALUE` kleiner als die aktuelle Sequence-Nummer ist.

```
ALTER SEQUENCE dept_deptid_seq  
    INCREMENT BY 20  
    MAXVALUE 90  
    NOCACHE  
    NOCYCLE;
```

- Fehler:

```
SQL Error: ORA-04009: MAXVALUE cannot be made to be less than the current value  
04009. 00000 - "MAXVALUE cannot be made to be less than the current value"
```

*Cause: the current value exceeds the given `MAXVALUE`

*Action: make sure that the new `MAXVALUE` is larger than the current value

Sequence-Informationen

- Die View `USER_SEQUENCES` beschreibt alle Sequences, deren Eigentümer Sie sind.

```
DESCRIBE user_sequences
```

NAME	NULL	TYPE
SEQUENCE_NAME	NOT NULL	VARCHAR2(128)
MIN_VALUE		NUMBER
MAX_VALUE		NUMBER
INCREMENT_BY	NOT NULL	NUMBER
CYCLE_FLAG		VARCHAR2(1)
ORDER_FLAG		VARCHAR2(1)
CACHE_SIZE	NOT NULL	NUMBER
LAST_NUMBER	NOT NULL	NUMBER
PARTITION_COUNT		NUMBER
SESSION_FLAG		VARCHAR2(1)
KEEP_VALUE		VARCHAR2(1)

- Sequence-Werte mithilfe der Data Dictionary-Tabelle `USER_SEQUENCES` prüfen

```
SELECT  sequence_name, min_value, max_value,
        increment_by, last_number
FROM    user_sequences;
```

ORACLE

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

Die View `USER_SEQUENCES` beschreibt alle Sequences, deren Eigentümer Sie sind. Beim Erstellen der Sequence geben Sie Kriterien an, die in der View `USER_SEQUENCES` gespeichert werden. Diese View enthält folgende Spalten:

- `SEQUENCE_NAME`: Name der Sequence
- `MIN_VALUE`: Mindestwert der Sequence
- `MAX_VALUE`: Höchstwert der Sequence
- `INCREMENT_BY`: Wert für die schrittweise Erhöhung der Sequence
- `CYCLE_FLAG`: Wird die Sequence zyklisch fortgeführt, wenn der Grenzwert erreicht wird?
- `ORDER_FLAG`: Werden Sequence-Nummern der Reihe nach generiert?
- `CACHE_SIZE`: Anzahl der Sequence-Nummern, die im Cache gespeichert werden
- `LAST_NUMBER`: Letzte auf den Datenträger geschriebene Sequence-Nummer. Wenn eine Sequence die Werte cacht, ist die auf den Datenträger geschriebene Nummer die letzte im Sequence-Cache abgelegte Nummer. Diese Nummer ist wahrscheinlich höher als die zuletzt verwendete Sequence-Nummer. Wurde `NOCACHE` angegeben, zeigt die Spalte `LAST_NUMBER` die nächste verfügbare Sequence-Nummer an.

Sequences werden nach ihrer Erstellung im Data Dictionary dokumentiert. Da eine Sequence ein Datenbankobjekt ist, können Sie sie in der Tabelle `USER_OBJECTS` im Data Dictionary identifizieren.

Sie können die Einstellungen der Sequence auch prüfen, indem Sie die Sequence mithilfe der Data Dictionary View `USER_SEQUENCES` wählen.

Lektionsagenda

- Sequences –Überblick
 - Sequences erstellen, verwenden und ändern
 - Sequence-Werte cachern
 - Pseudospalten NEXTVAL und CURRVAL
 - SQL-Spaltenstandards mithilfe von Sequences einstellen
- Synonyme – Überblick
 - Synonyme erstellen und löschen
- Indizes – Überblick
 - Indizes erstellen
 - CREATE TABLE-Anweisungen
 - Funktionsbasierte Indizes erstellen
 - Mehrere Indizes für dieselbe Spaltengruppe erstellen
 - Indizes entfernen

ORACLE

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

Synonyme

Ein Synonym:

- ist ein Datenbankobjekt
- kann als alternativer Name für eine Tabelle oder ein anderes Datenbankobjekt erstellt werden
- belegt außer der Definition im Data Dictionary keinen Speicherplatz
- ist nützlich, um Identität und Verzeichnis eines zugrunde liegenden Schemaobjekts zu verbergen

ORACLE

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

Synonyme sind Datenbankobjekte, die den Aufruf einer Tabelle über einen anderen Namen ermöglichen.

Sie können Synonyme als alternativen Namen für eine Tabelle oder ein anderes Datenbankobjekt erstellen. Beispiel: Synonyme können für Tabellen oder Views, Sequences, PL/SQL-Programmeinheiten, benutzerdefinierte Objekte und andere Synonyme erstellt werden.

Da ein Synonym nichts anderes als ein Alias ist, belegt es abgesehen von seiner Definition im Data Dictionary keinen Speicherplatz.

Synonyme können SQL-Anweisungen für Datenbankbenutzer erleichtern und eignen sich auch dafür, Identität und Verzeichnis eines zugrunde liegenden Schemaobjekts zu verbergen.

Synonyme für Objekte erstellen

Sie können den Zugriff auf Objekte mithilfe von Synonymen (anderen Namen für Objekte) vereinfachen. Mit Synonymen können Sie:

- einfachere Referenzen auf Tabellen anderer Benutzer erstellen
- lange Objektnamen kürzen

```
CREATE [PUBLIC] SYNONYM synonym
FOR      object;
```

ORACLE

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

Um eine Tabelle eines anderen Benutzers zu referenzieren, setzen Sie als Präfix vor den Tabellennamen den Namen des Benutzers, der die Tabelle erstellt hat, gefolgt von einem Punkt. Wenn Sie ein Synonym erstellen, müssen Sie den Objektnamen nicht mehr mit dem Schema kennzeichnen und können den alternativen Namen für eine Tabelle, View, Sequence, Prozedur oder andere Objekte angeben. Dieses Verfahren ist besonders bei langen Objektnamen hilfreich, zum Beispiel bei Views.

Für die Syntax gilt:

<code>PUBLIC</code>	Erstellt ein Synonym, das allen Benutzern zugänglich ist
<code><i>synonym</i></code>	Steht für den Namen des zu erstellenden Synonyms
<code><i>object</i></code>	Bezeichnet das Objekt, für das das Synonym erstellt wird

Richtlinien

- Das Objekt darf nicht in einem Package enthalten sein.
- Der Name eines privaten Synonyms muss sich von allen anderen Objekten des Benutzers unterscheiden.
- Um ein öffentliches Synonym zu erstellen, benötigen Sie die Systemberechtigung `CREATE PUBLIC SYNONYM`.

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

Synonyme erstellen und entfernen

- Kurznamen für die View DEPT_SUM_VU erstellen:

```
CREATE SYNONYM d_sum
FOR dept_sum_vu;
synonym D_SUM created.
```

- Synonym löschen:

```
DROP SYNONYM d_sum;
synonym D_SUM dropped.
```

ORACLE

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

Synonyme erstellen

Im Beispiel auf der Folie wird ein Synonym für die View DEPT_SUM_VU erstellt, damit sie schneller referenziert werden kann.

Der DBA kann ein öffentliches Synonym erstellen, das allen Benutzern zugänglich ist. Im folgenden Beispiel wird das öffentliche Synonym DEPT für die Tabelle DEPARTMENTS der Benutzerin Alice erstellt:

```
CREATE PUBLIC SYNONYM dept
FOR alice.departments;
```

Synonyme entfernen

Um ein Synonym zu löschen, verwenden Sie die Anweisung DROP SYNONYM. Öffentliche Synonyme können nur von DBAs gelöscht werden.

```
DROP PUBLIC SYNONYM dept;
```

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

Synonyminformationen

```
DESCRIBE user_synonyms
```

```
DESCRIBE user_synonyms
Name          Null    Type
-----
SYNONYM_NAME  NOT NULL VARCHAR2(128)
TABLE_OWNER   VARCHAR2(128)
TABLE_NAME    NOT NULL VARCHAR2(128)
DB_LINK       VARCHAR2(128)
```

```
SELECT *
FROM   user_synonyms;
```

	SYNONYM_NAME	TABLE_OWNER	TABLE_NAME	DB_LINK
1	D_SUM	ORA21	DEPT_SUM_VU	(null)

ORACLE

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

Die Dictionary View `USER_SYNONYMS` beschreibt private Synonyme (Synonyme, deren Eigentümer Sie sind).

Sie können diese View abfragen, um Ihre Synonyme zu ermitteln. Mit einer Abfrage von `ALL_SYNONYMS` ermitteln Sie die Namen aller Synonyme, die für Sie verfügbar sind, sowie die Objekte, für die diese Synonyme gelten.

Diese View enthält folgende Spalten:

- `SYNONYM_NAME`: Name des Synonyms
- `TABLE_OWNER`: Eigentümer des Objekts, das durch das Synonym referenziert wird
- `TABLE_NAME`: Name der Tabelle oder View, die durch das Synonym referenziert wird
- `DB_LINK`: Name der Datenbanklinkreferenz (sofern vorhanden)

Lektionsagenda

- Sequences –Überblick
 - Sequences erstellen, verwenden und ändern
 - Sequence-Werte cachern
 - Pseudospalten NEXTVAL und CURRVAL
 - SQL-Spaltenstandards mithilfe von Sequences einstellen
- Synonyme – Überblick
 - Synonyme erstellen und löschen
- Indizes – Überblick
 - Indizes erstellen
 - CREATE TABLE-Anweisungen
 - Funktionsbasierte Indizes erstellen
 - Mehrere Indizes für dieselbe Spaltengruppe erstellen
 - Indizes entfernen

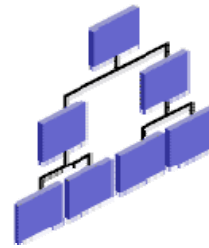
ORACLE®

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

Indizes

Ein Index:

- ist ein Schemaobjekt
- kann vom Oracle-Server verwendet werden, um den Abruf von Zeilen mithilfe eines Zeigers zu beschleunigen
- kann I/O-(Input/Output-)Vorgänge mithilfe einer beschleunigten Pfadzugriffsmethode reduzieren, die Daten schneller findet
- ist abhängig von der indizierten Tabelle
- wird vom Oracle-Server automatisch verwendet und verwaltet



ORACLE

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

Ein Oracle-Serverindex ist ein Schemaobjekt, das den Abruf von Zeilen mithilfe eines Zeigers beschleunigen und die Performance bestimmter Abfragen verbessern kann. Indizes können explizit oder automatisch erstellt werden. Wenn kein Index in der Spalte vorhanden ist, wird ein Full Table Scan ausgeführt.

Ein Index ermöglicht den direkten und schnellen Zugriff auf Zeilen in einer Tabelle. Er reduziert die I/O-Vorgänge, da er einen indizierten Pfad verwendet, um Daten schnell zu finden. Indizes werden vom Oracle-Server automatisch verwendet und verwaltet. Nach der Erstellung des Index sind keine weiteren Benutzeraktivitäten erforderlich.

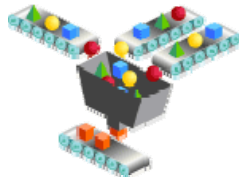
Indizes sind weder logisch noch physisch von den Daten in den mit ihnen verknüpften Objekten abhängig. Das bedeutet, sie können jederzeit erstellt oder gelöscht werden, ohne dass sich dies auf die Basistabellen oder andere Indizes auswirkt.

Hinweis: Beim Löschen einer Tabelle werden die zugehörigen Indizes ebenfalls gelöscht.

Weitere Informationen finden Sie in *Oracle Database Concepts 12c Release 1* im Abschnitt "Schema Objects: Indexes".

Indexerstellung – Verfahren

- **Automatisch:** Ein eindeutiger Index wird automatisch erstellt, wenn Sie das Constraint `PRIMARY KEY` oder `UNIQUE` in einer Tabellendefinition angeben.



- **Manuell:** Sie können eindeutige oder nicht eindeutige Indizes für Spalten erstellen, um den Zeilenzugriff zu beschleunigen.



ORACLE

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

Sie können zwei Typen von Indizes erstellen.

- **Eindeutiger Index:** Der Oracle-Server erstellt diesen Index automatisch, wenn Sie eine Tabellenspalte mit dem Constraint `PRIMARY KEY` oder `UNIQUE` versehen. Der Indexname entspricht dem Namen des Constraints.
- **Nicht eindeutiger Index:** Dieser Index kann vom Benutzer erstellt werden. Beispiel: Sie erstellen für einen Join in einer Abfrage den Spaltenindex `FOREIGN KEY`, um den Datenabruf zu beschleunigen.

Hinweis: Eindeutige Indizes lassen sich manuell erstellen. Es ist jedoch ratsam, ein Constraint vom Typ `UNIQUE` zu erstellen, das implizit einen eindeutigen Index erzeugt.

Indizes erstellen

- Index für eine oder mehrere Spalten erstellen:

```
CREATE [UNIQUE] INDEX index
ON table (column[, column]...);
```

- Abfragezugriff auf die Spalte `LAST_NAME` in der Tabelle `EMPLOYEES` verbessern:

```
CREATE INDEX emp_last_name_idx
ON employees(last_name);
index EMP_LAST_NAME_IDX created
```

ORACLE®

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

Sie können Indizes mithilfe der Anweisung `CREATE INDEX` für einzelne oder mehrere Spalten erstellen.

Für die Syntax gilt:

- `index` Steht für den Namen des Index
- `table` Steht für den Namen der Tabelle
- `Column` Steht für den Namen der Tabellenspalte, die indiziert werden soll

Mit `UNIQUE` geben Sie an, dass der Wert der Spalten, auf denen der Index basiert, eindeutig sein muss. Mit `BITMAP` geben Sie an, dass der Index mit einem Bitmap für jeden eindeutigen Schlüssel erstellt werden soll, sodass nicht jede Zeile separat indiziert werden muss. Bitmapindizes speichern die einem Schlüsselwert zugeordneten `rowids` als Bitmap.

Weitere Informationen finden Sie in der *Oracle Database SQL Language Reference* für Oracle Database 12c im Abschnitt "`CREATE INDEX`".

CREATE INDEX mit Anweisung CREATE TABLE

```
CREATE TABLE NEW_EMP  
(employee_id NUMBER(6)  
    PRIMARY KEY USING INDEX  
    (CREATE INDEX emp_id_idx ON  
    NEW_EMP(employee_id)),  
first_name VARCHAR2(20),  
last_name VARCHAR2(25));
```

```
table NEW_EMP created.
```

```
SELECT INDEX_NAME, TABLE_NAME  
FROM    USER_INDEXES  
WHERE   TABLE_NAME = 'NEW_EMP';
```

INDEX_NAME	TABLE_NAME
1 EMP_ID_IDX	NEW_EMP

ORACLE

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

Im Beispiel auf der Folie wird die Klausel `CREATE INDEX` mit der Anweisung `CREATE TABLE` verwendet und damit explizit ein `PRIMARY KEY`-Index erstellt. Bei der Primärschlüsselerstellung können Sie den Indizes Namen geben, die sich vom Namen des Constraints `PRIMARY KEY` unterscheiden.

Informationen zu den Indizes können Sie aus der Data Dictionary View `USER_INDEXES` abfragen.

Das folgende Beispiel zeigt das Datenbankverhalten, wenn der Index nicht explizit benannt ist:

```
CREATE TABLE EMP_UNNAMED_INDEX  
(employee_id NUMBER(6) PRIMARY KEY ,  
first_name VARCHAR2(20),  
last_name VARCHAR2(25));
```

```
SELECT INDEX_NAME, TABLE_NAME  
FROM    USER_INDEXES  
WHERE   TABLE_NAME = 'EMP_UNNAMED_INDEX';
```

Der Oracle-Server gibt dem Index, der für die Spalte `PRIMARY KEY` erstellt wird, einen allgemeinen Namen.

Sie können auch einen vorhandenen Index für die Spalte `PRIMARY KEY` verwenden. Beispiel: Sie gehen davon aus, dass große Datenmengen geladen werden, und möchten den Vorgang beschleunigen. Sie können die Constraints deaktivieren, während Sie den Ladevorgang durchführen, und sie dann erneut aktivieren. Wenn Sie in diesem Fall einen eindeutigen Index für den Primärschlüssel verwenden, werden die Daten beim Laden weiterhin geprüft. Sie können also zuerst einen nicht eindeutigen Index für die Spalte erstellen, die als Primärschlüssel dienen soll, und dann die Spalte `PRIMARY KEY` erstellen und angeben, dass sie den vorhandenen Index verwenden soll. Das folgende Beispiel veranschaulicht diesen Prozess:

1. Schritt: Tabelle erstellen:

```
CREATE TABLE NEW_EMP2
(
  employee_id NUMBER(6),
  first_name  VARCHAR2(20),
  last_name   VARCHAR2(25)
);
```

2. Schritt: Index erstellen:

```
CREATE INDEX emp_id_idx2 ON
  new_emp2(employee_id);
```

3. Schritt: `PRIMARY KEY` erstellen:

```
ALTER TABLE new_emp2 ADD PRIMARY KEY (employee_id) USING INDEX
  emp_id_idx2;
```

Funktionsbasierte Indizes

- Funktionsbasierte Indizes basieren auf Ausdrücken.
- Der Indexausdruck setzt sich aus Tabellenspalten, Konstanten, SQL-Funktionen und benutzerdefinierten Funktionen zusammen.

```
CREATE INDEX upper_dept_name_idx  
ON dept2 (UPPER(department_name));
```

```
index UPPER_DEPT_NAME_IDX created.
```

```
SELECT *  
FROM   dept2  
WHERE  UPPER(department_name) = 'SALES';
```

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	80	Sales	145	2500

ORACLE

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

Mit den Schlüsselwörtern `UPPER(column_name)` und `LOWER(column_name)` definierte funktionsbasierte Indizes bewirken, dass bei Suchvorgängen die Groß-/Kleinschreibung nicht beachtet wird. Beispiel: Der Index:

```
CREATE INDEX upper_last_name_idx ON emp2 (UPPER(last_name));
```

erleichtert die Verarbeitung von Abfragen wie:

```
SELECT * FROM emp2 WHERE UPPER(last_name) = 'KING';
```

Der Oracle-Server verwendet den Index nur, wenn diese besondere Funktion in einer Abfrage angegeben ist. Beispiel: Die folgende Anweisung kann zwar den Index verwenden, doch ohne die Klausel `WHERE` würde der Oracle-Server eventuell einen Full Table Scan durchführen:

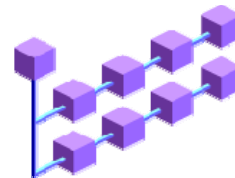
```
SELECT *  
FROM   employees  
WHERE  UPPER (last_name) IS NOT NULL  
ORDER BY UPPER (last_name);
```

Hinweis: Zur Erstellung eines funktionsbasierten Index benötigen Sie die Systemberechtigung `QUERY REWRITE`. Der Initialisierungsparameter `QUERY_REWRITE_ENABLED` muss auf `TRUE` eingestellt werden, damit ein funktionsbasierter Index verwendet werden kann.

Der Oracle-Server behandelt Indizes mit Spalten, die mit `DESC` gekennzeichnet sind, als funktionsbasierte Indizes. Die entsprechenden Spalten werden in absteigender Reihenfolge sortiert.

Mehrere Indizes für dieselbe Gruppe von Spalten erstellen

- Sie können mehrere Indizes für dieselbe Spaltengruppe erstellen.
- Zu einer Spaltengruppe können mehrere Indizes erstellt werden, wenn:
 - die Indizes jeweils einen unterschiedlichen Typ aufweisen
 - die Indizes eine unterschiedliche Partitionierung verwenden
 - die Indizes unterschiedliche Eindeutigkeitseigenschaften aufweisen



ORACLE

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

Sie können mehrere Indizes zu einer Gruppe von Spalten erstellen, sofern es sich dabei um unterschiedliche Typen von Indizes handelt, die Indizes jeweils eine unterschiedliche Partitionierung verwenden oder unterschiedliche Eindeutigkeitseigenschaften aufweisen. Beispiel: Sie können für dieselbe Gruppe von Spalten einen B*-Baumindex und einen Bitmapindex erstellen.

In ähnlicher Weise können Sie sowohl einen eindeutigen als auch einen nicht eindeutigen Index für dieselbe Gruppe von Spalten erstellen.

Bei mehreren Indizes für dieselbe Gruppe von Spalten kann immer nur jeweils einer davon sichtbar sein.

Mehrere Indizes für dieselbe Gruppe von Spalten erstellen – Beispiel

```
CREATE INDEX emp_id_name_ix1  
ON employees(employee_id, first_name);
```

```
index EMP_ID_NAME_IX1 created.
```

```
ALTER INDEX emp_id_name_ix1 INVISIBLE;
```

```
index EMP_ID_NAME_IX1 altered.
```

```
CREATE BITMAP INDEX emp_id_name_ix2  
ON employees(employee_id, first_name);
```

```
bitmap index EMP_ID_NAME_IX2 created.
```

ORACLE

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

Das Codebeispiel zeigt die Erstellung des B*-Baumindex `emp_id_name_ix1` für die Spalten `employee_id` und `first_name` aus der Tabelle `employees` im Schema `HR`. Nachdem der Index erstellt wurde, wird er als unsichtbar markiert. Anschließend wird ein Bitmapindex für die Spalten `employee_id` und `first_name` aus der Tabelle `employees` im Schema `HR` erstellt. Der Bitmapindex `emp_id_name_ix2` ist standardmäßig sichtbar.

Indexinformationen

- USER_INDEXES enthält Informationen zu Ihren Indizes.
- USER_IND_COLUMNS beschreibt Spalten von Indizes, deren Eigentümer Sie sind, sowie Spalten von Indizes zu Ihren Tabellen.

```
DESCRIBE user_indexes
```

DESCRIBE user_indexes		
Name	Null	Type
INDEX_NAME	NOT NULL	VARCHAR2(128)
INDEX_TYPE		VARCHAR2(27)
TABLE_OWNER	NOT NULL	VARCHAR2(128)
TABLE_NAME	NOT NULL	VARCHAR2(128)
TABLE_TYPE		VARCHAR2(11)
UNIQUENESS		VARCHAR2(9)

...

ORACLE

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

Die Namen Ihrer Indizes, den Namen der Tabelle, für die der Index erstellt wurde, und Angaben darüber, ob der Index eindeutig ist, ermitteln Sie durch Abfrage der View USER_INDEXES.

Hinweis: Eine vollständige Liste und Beschreibung der Spalten in der View USER_INDEXES finden Sie in der *Oracle® Database Reference 12c Release 1* unter "USER_INDEXES".

USER_INDEXES – Beispiele

a `SELECT index_name, table_name, uniqueness
FROM user_indexes
WHERE table_name = 'EMPLOYEES';`

	INDEX_NAME	TABLE_NAME	UNIQUENESS
1	EMP_NAME_IX	EMPLOYEES	NONUNIQUE
2	EMP_MANAGER_IX	EMPLOYEES	NONUNIQUE
3	EMP_JOB_IX	EMPLOYEES	NONUNIQUE
4	EMP_DEPARTMENT_IX	EMPLOYEES	NONUNIQUE
5	EMP_EMP_ID_PK	EMPLOYEES	UNIQUE
6	EMP_EMAIL_UK	EMPLOYEES	UNIQUE

...

b `SELECT index_name, table_name
FROM user_indexes
WHERE table_name = 'EMP_LIB';`

	INDEX_NAME	TABLE_NAME
1	SYS_C0010979	EMP_LIB

ORACLE

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

Im Beispiel **a** auf der Folie wird die View `USER_INDEXES` abgefragt, um den Namen des Index, den Namen der Tabelle, für die der Index erstellt wurde, und Angaben darüber zu ermitteln, ob der Index eindeutig ist.

Im Beispiel **b** auf der Folie gibt der Oracle-Server dem Index, der für die Spalte `PRIMARY KEY` erstellt wird, einen generischen Namen. Die Tabelle `EMP_LIB` wird mit folgendem Code erstellt:

```
CREATE TABLE emp_lib  
(book_id NUMBER(6) PRIMARY KEY,  
title VARCHAR2(25),  
category VARCHAR2(20));
```

USER_IND_COLUMNS abfragen

```
DESCRIBE user_ind_columns
```

```
DESCRIBE user_ind_columns
Name          Null Type
-----
INDEX_NAME     VARCHAR2(128)
TABLE_NAME     VARCHAR2(128)
COLUMN_NAME    VARCHAR2(4000)
COLUMN_POSITION NUMBER
COLUMN_LENGTH  NUMBER
CHAR_LENGTH    NUMBER
DESCEND        VARCHAR2(4)
```

```
SELECT index_name, column_name, table_name
FROM   user_ind_columns
WHERE  index_name = 'LNAME_IDX';
```

	INDEX_NAME	COLUMN_NAME	TABLE_NAME
1	LNAME_IDX	LAST_NAME	EMP_TEST

ORACLE

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

Die Dictionary View `USER_IND_COLUMNS` liefert Informationen wie den Namen des Index, den Namen der indizierten Tabelle sowie den Namen und die Position einer Spalte im Index.

Im Beispiel auf der Folie werden die Tabelle `emp_test` und der Index `LNAME_IDX` mit folgendem Code erstellt:

```
CREATE TABLE emp_test AS SELECT * FROM employees;
CREATE INDEX lname_idx ON emp_test(last_name);
```


Indizes entfernen

- Index mit dem Befehl `DROP INDEX` aus dem Data Dictionary löschen:

```
DROP INDEX index;
```

- Index `emp_last_name_idx` aus dem Data Dictionary entfernen:

```
DROP INDEX emp_last_name_idx;
```

```
index EMP_LAST_NAME_IDX dropped.
```

- Um einen Index zu löschen, müssen Sie der Eigentümer sein oder über die Berechtigung `DROP ANY INDEX` verfügen.

ORACLE

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

Sie können Indizes nicht ändern. Indizes können nur gelöscht und dann neu erstellt werden.

Sie entfernen eine Indexdefinition mit der Anweisung `DROP INDEX` aus dem Data Dictionary. Um einen Index zu löschen, müssen Sie der Eigentümer sein oder über die Berechtigung `DROP ANY INDEX` verfügen.

In der Syntax steht *index* für den Namen des Index.

Sie können Indizes mit dem Schlüsselwort `ONLINE` löschen.

```
DROP INDEX emp_idx ONLINE;
```

ONLINE: Geben Sie `ONLINE` an, um festzulegen, dass DML-Vorgänge für die Tabelle zulässig sind, während der Index gelöscht wird.

Hinweis: Wenn Sie eine Tabelle löschen, werden die zugehörigen Indizes und Constraints automatisch ebenfalls gelöscht. Views bleiben jedoch erhalten.

Quiz

Indizes müssen manuell erstellt werden und ermöglichen den schnelleren Zugriff auf Zeilen in einer Tabelle.

- a. Richtig
- b. Falsch

ORACLE

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

Richtige Antwort: b

Hinweis: Indizes dienen zur Verbesserung der Abfrageperformance. Allerdings werden nicht alle Indizes manuell erstellt. Der Oracle-Server erstellt Indizes automatisch, wenn Sie eine Tabellenspalte mit dem Constraint `PRIMARY KEY` oder `UNIQUE` versehen.

Zusammenfassung

In dieser Lektion haben Sie Folgendes gelernt:

- Sequence-Nummern mithilfe eines Sequence-Generators automatisch generieren
- Synonyme als alternative Namen für Objekte verwenden
- Indizes zur Beschleunigung von Abfragen erstellen
- Objektinformationen über folgende Dictionary Views ermitteln:
 - USER_VIEWS
 - USER_SEQUENCES
 - USER_SYNONYMS

ORACLE

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

In dieser Lektion haben Sie die Datenbankobjekte Sequences, Indizes und Synonyme kennengelernt.

Übungen zu Lektion 3 – Überblick

Diese Übung behandelt folgende Themen:

- Sequences erstellen
- Sequences verwenden
- Dictionary Views nach Sequence-Informationen abfragen
- Synonyme erstellen
- Dictionary Views nach Synonyminformationen abfragen
- Indizes erstellen
- Dictionary Views nach Indexinformationen abfragen

ORACLE

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

Die Übungen zu dieser Lektion enthalten eine Reihe von Aufgaben, in denen Sie eine Sequence, einen Index und ein Synonym erstellen und verwenden. Außerdem lernen Sie, die Data Dictionary Views nach Informationen zu Sequences, Synonymen und Indizes abzufragen.

4

Views 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:

- Einfache und komplexe Views erstellen
- Daten aus Views abrufen
- Dictionary Views nach View-Informationen abfragen

ORACLE

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

In dieser Lektion werden Views sowie die Grundlagen zu ihrer Erstellung und Verwendung vorgestellt.

Lektionsagenda

- Views – Überblick
- View-Daten erstellen, ändern und abrufen
- DML-(Data Manipulation Language-)Vorgänge an Views
- Views löschen

ORACLE®

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

Datenbankobjekte

Objekt	Beschreibung
Tabelle	Grundlegende Speichereinheit, die aus Zeilen besteht
View	Stellt Teilmengen von Daten aus einzelnen oder mehreren Tabellen logisch dar
Sequence	Generiert numerische Werte
Index	Verbessert die Performance von Datenabfragen
Synonym	Gibt Objekten alternative Namen

ORACLE

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

Neben Tabellen gibt es in einer Datenbank verschiedene andere Objekte.

Mit Views können Sie Daten aus den Tabellen darstellen und verbergen.

Viele Anwendungen erfordern eindeutige Zahlen als Primärschlüsselwerte. Um dieser Anforderung Rechnung zu tragen, können Sie entweder Code in die Anwendung einfügen oder eindeutige Nummern mithilfe einer Sequence generieren.

Ein Index kann unter Umständen dazu beitragen, die Datenabfrage zu beschleunigen. Sie können Indizes auch verwenden, um für eine Spalte oder eine Gruppe von Spalten Eindeutigkeit zu erzwingen.

Mit Synonymen können Sie Objekten alternative Namen geben.

Views

Tabelle EMPLOYEES

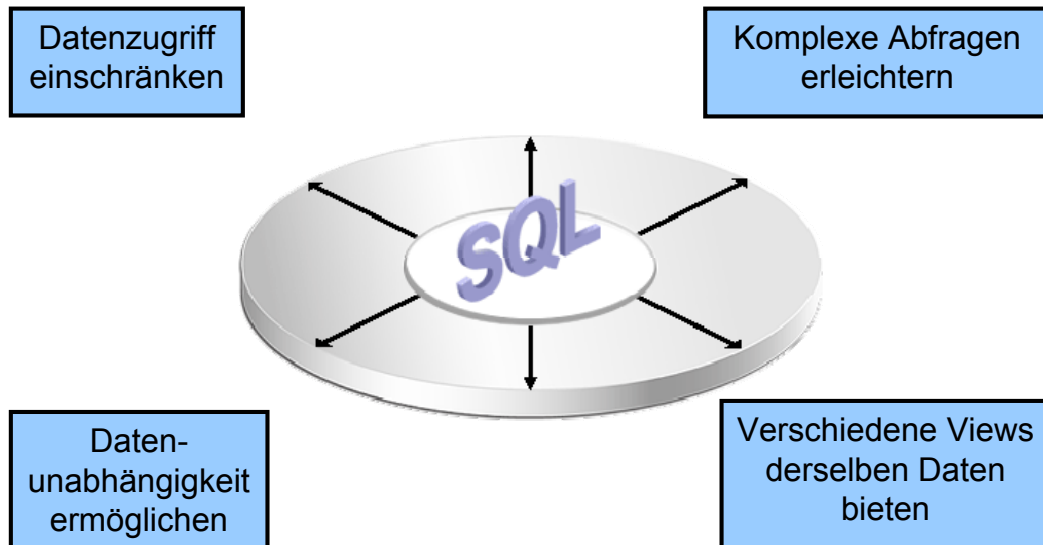
EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY
100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	17000
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	9000
104	Bruce	Ernst	BERNST	590.423.4566	17-SEP-87	IT_PROG	6000
107	John	King	JKING	515.123.4567	17-FEB-96	IT_PROG	4200
110	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_ACCOUNT	6900
111	William	Gietz	WGIEZT	515.123.8181	07-JUN-94	AC_ACCOUNT	5800
112	Walter	Taylor	WTAYLOR	515.123.8181	07-JUN-94	AC_ACCOUNT	3500
113	Luis	Chen	LCHEN	515.123.8181	07-JUN-94	AC_ACCOUNT	3100
114	Jena	Martens	JMARTENS	515.123.8181	07-JUN-94	AC_ACCOUNT	2600
115	Part	Simon	PSIMON	515.123.8181	07-JUN-94	AC_ACCOUNT	2500
116	Part	Simon	PSIMON	515.123.8181	07-JUN-94	AC_ACCOUNT	10500
117	Part	Simon	PSIMON	515.123.8181	07-JUN-94	AC_ACCOUNT	11000
118	Part	Simon	PSIMON	515.123.8181	07-JUN-94	AC_ACCOUNT	8600
119	Part	Simon	PSIMON	515.123.8181	07-JUN-94	AC_ACCOUNT	7000
120	Part	Simon	PSIMON	515.123.8181	07-JUN-94	AC_ACCOUNT	4400
121	Part	Simon	PSIMON	515.123.8181	07-JUN-94	AC_ACCOUNT	13000
122	Part	Simon	PSIMON	515.123.8181	07-JUN-94	AC_ACCOUNT	6000
123	Part	Simon	PSIMON	515.123.8181	07-JUN-94	AC_ACCOUNT	12000
124	Part	Simon	PSIMON	515.123.8181	07-JUN-94	AC_ACCOUNT	8300

ORACLE

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

Sie können logische Teilmengen oder Kombinationen von Daten darstellen, indem Sie Views für Tabellen erstellen. Eine View ist ein Schemaobjekt, eine gespeicherte `SELECT`-Anweisung auf Basis einer Tabelle oder einer anderen View. Eine View enthält selbst keine Daten, sondern ist vergleichbar mit einem Fenster, durch das Tabellendaten betrachtet und geändert werden können. Die Tabellen, auf denen eine View basiert, werden *Basistabellen* genannt. Views werden im Data Dictionary als `SELECT`-Anweisungen gespeichert.

Views – Vorteile



ORACLE

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

- Views schränken den Zugriff auf Daten ein, da sie ausgewählte Spalten der Tabelle anzeigen.
- Benutzer können mit Views einfache Abfragen ausführen, um die Ergebnisse komplexer Abfragen abzurufen. Beispiel: Mithilfe von Views können Benutzer Informationen aus mehreren Tabellen abrufen, ohne zu wissen, wie eine Join-Anweisung erstellt wird.
- Views ermöglichen Datenunabhängigkeit für Ad-hoc-Benutzer und Anwendungsprogramme. Mithilfe von Views können Daten aus mehreren Tabellen abgerufen werden.
- Views ermöglichen Benutzergruppen den Zugriff auf Daten gemäß ihren jeweiligen Kriterien.

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

Einfache und komplexe Views

Feature	Einfache Views	Komplexe Views
Anzahl der Tabellen	Eine	Eine oder mehrere
Enthalten Funktionen	Nein	Ja
Enthalten Datengruppen	Nein	Ja
DML-Vorgänge über eine View	Ja	Nicht immer

ORACLE

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

Es gibt zwei Arten von Views: einfache und komplexe. Der grundlegende Unterschied hängt mit den DML-Vorgängen (INSERT, UPDATE und DELETE) zusammen.

- Eine einfache View:
 - leitet Daten aus nur einer Tabelle ab
 - enthält keine Funktionen oder Datengruppen
 - führt DML-Vorgänge in der Regel über die View aus
- Eine komplexe View:
 - leitet Daten aus mehreren Tabellen ab
 - enthält Funktionen oder Datengruppen
 - lässt DML-Vorgänge über die View nicht immer zu

Lektionsagenda

- Views – Überblick
- View-Daten erstellen, ändern und abrufen
- DML-(Data Manipulation Language-)Vorgänge an Views
- Views löschen

ORACLE®

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

Views erstellen

- Unterabfragen in die Anweisung `CREATE VIEW` einbetten:

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

- Die Unterabfrage kann komplexe `SELECT`-Syntax enthalten.

ORACLE

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

Sie können eine View erstellen, indem Sie eine Unterabfrage in die Anweisung `CREATE VIEW` einbetten.

Für die Syntax gilt:

`OR REPLACE`

Erstellt die View neu, falls sie bereits vorhanden ist. Mit dieser Klausel können Sie die Definition einer vorhandenen View ändern, ohne bereits erteilte Objektberechtigungen löschen, neu erstellen und neu erteilen zu müssen.

`FORCE`

Erstellt die View unabhängig davon, ob die Basistabellen vorhanden sind

`NOFORCE`

Erstellt die View nur, wenn die Basistabellen vorhanden sind (Standard)

view

Steht für den Namen der View

alias

Gibt Namen für die von der Abfrage der View gewählten Ausdrücke an (Die Anzahl der Aliasnamen muss der Anzahl der gewählten Ausdrücke entsprechen.)

subquery

Steht für eine vollständige `SELECT`-Anweisung (In der `SELECT`-Liste können Sie Aliasnamen für die Spalten verwenden.)

`WITH CHECK OPTION`

Gibt an, dass nur Zeilen eingefügt oder aktualisiert werden können, auf die über die View zugegriffen werden kann

Constraint

Ist der dem `CHECK OPTION` zugewiesene Name

`WITH READ ONLY`

Stellt sicher, dass keine DML-Vorgänge an dieser View vorgenommen werden können

Hinweis: Um DDL-(Data Definition Language-)Anweisungen auszuführen, klicken Sie in SQL Developer auf das Symbol **Run Script** oder drücken F5. Die Rückmeldungen werden in der Registerkarte **Script Output** angezeigt.

Views erstellen

- View EMPVU80 mit Details zu den Mitarbeitern aus Abteilung 80 erstellen:

```
CREATE VIEW empvu80
AS SELECT employee_id, last_name, salary
FROM employees
WHERE department_id = 80;
view EMPVU80 created.
```

- Struktur der View mithilfe des SQL*Plus-Befehls DESCRIBE beschreiben:

```
DESCRIBE empvu80;
```

ORACLE

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

Im Beispiel auf der Folie wird eine View erstellt, die für jeden Mitarbeiter aus Abteilung 80 Personalnummer, Nachname und Gehalt enthält.

Die Struktur der View zeigen Sie mit dem Befehl DESCRIBE an.

```
DESCRIBE empvu80
Name          Null      Type
-----
EMPLOYEE_ID   NOT NULL  NUMBER(6)
LAST_NAME     NOT NULL  VARCHAR2(25)
SALARY                NUMBER(8,2)
```

Richtlinien

- Die Unterabfrage, die eine View definiert, kann eine komplexe SELECT-Syntax mit Joins, Gruppen und Unterabfragen enthalten.
- Wenn Sie keinen Constraint-Namen für eine mit WITH CHECK OPTION erstellte View angeben, weist das System einen Standardnamen im Format SYS_Cn zu.
- Mit der Option OR REPLACE können Sie die Definition einer View ändern, ohne dabei die View löschen und neu erstellen bzw. die bereits erteilten Objektberechtigungen neu vergeben zu müssen.

Views erstellen

- View mithilfe von Spaltenaliasnamen in der Unterabfrage erstellen:

```
CREATE VIEW   salvu50
AS SELECT    employee_id ID_NUMBER, last_name NAME,
            salary*12 ANN_SALARY
FROM         employees
WHERE        department_id = 50;
```

view SALVU50 created.

- Spalten dieser View über die angegebenen Aliasnamen wählen

ORACLE

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

Sie können die Spaltennamen steuern, indem Sie in der Unterabfrage Aliasnamen für die Spalten angeben.

Im Beispiel auf der Folie wird eine View erstellt, die für jeden Mitarbeiter aus Abteilung 50 die Personalnummer (`EMPLOYEE_ID`) mit dem Alias `ID_NUMBER`, den Nachnamen (`LAST_NAME`) mit dem Alias `NAME` und das Jahresgehalt (`SALARY`) mit dem Alias `ANN_SALARY` enthält.

Alternativ können Sie Aliasnamen nach der Anweisung `CREATE` und vor der Unterabfrage `SELECT` verwenden. Die Anzahl der angegebenen Aliasnamen muss der Anzahl der in der Unterabfrage gewählten Ausdrücke entsprechen.

```
CREATE OR REPLACE VIEW salvu50 (ID_NUMBER, NAME, ANN_SALARY)
AS SELECT    employee_id, last_name, salary*12
FROM         employees
WHERE        department_id = 50;
```

Daten aus Views abrufen

```
SELECT *  
FROM salvu50;
```

	ID_NUMBER	NAME	ANN_SALARY
1	120	Weiss	96000
2	121	Fripp	98400
3	122	Kaufling	94800
4	123	Vollman	78000
5	124	Mourgos	69600
6	125	Nayer	38400
7	126	Mikkilineni	32400

...

ORACLE

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

Daten aus einer View rufen Sie genauso ab wie Daten aus einer Tabelle. Dabei können Sie wahlweise den Inhalt der gesamten View oder nur bestimmte Zeilen und Spalten anzeigen.

Views ändern

- View EMPVU80 mit der Klausel CREATE OR REPLACE VIEW ändern und für jeden Spaltennamen einen Alias hinzufügen:

```
CREATE OR REPLACE VIEW empvu80
(id_number, name, sal, department_id)
AS SELECT  employee_id, first_name || ' '
           || last_name, salary, department_id
FROM      employees
WHERE     department_id = 80;
view EMPVU80 created.
```

- Spaltenaliasnamen werden in der Klausel CREATE OR REPLACE VIEW in derselben Reihenfolge aufgeführt wie die Spalten in der Unterabfrage.

ORACLE

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

Mit der Option REPLACE kann eine View auch dann erstellt werden, wenn bereits eine View mit dem gleichen Namen vorhanden ist. Die ältere Version der View wird dabei für deren Eigentümer ersetzt. Auf diese Weise ändern Sie die View, ohne sie löschen, neu anlegen und Objektberechtigungen neu erteilen zu müssen.

Hinweis: Achten Sie darauf, Spaltenaliasnamen in der Klausel CREATE OR REPLACE VIEW in derselben Reihenfolge aufzuführen wie die Spalten in der Unterabfrage.

Komplexe Views erstellen

Komplexe View mit Gruppenfunktionen zur Anzeige von Werten aus zwei Tabellen erstellen:

```
CREATE OR REPLACE VIEW dept_sum_vu
(name, minsal, maxsal, avgsal)
AS SELECT  d.department_name, MIN(e.salary),
           MAX(e.salary), AVG(e.salary)
FROM      employees e JOIN departments d
USING     (department_id)
GROUP BY  d.department_name;
```

```
view DEPT_SUM_VU created.
```

ORACLE

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

Im Beispiel auf der Folie wird eine komplexe View mit Abteilungsnamen sowie Mindest-, Höchst- und Durchschnittsgehältern für die einzelnen Abteilungen erstellt. Beachten Sie, dass alternative Namen für die View angegeben werden. Dies ist erforderlich, wenn eine Spalte der View aus einer Funktion oder einem Ausdruck abgeleitet ist.

Die Struktur der View zeigen Sie mit dem Befehl `DESCRIBE` an. Der Inhalt der View wird über eine `SELECT`-Anweisung angezeigt.

```
SELECT  *
FROM    dept_sum_vu;
```

View-Informationen

1

```
DESCRIBE user_views
```

Name	Null	Type
VIEW_NAME	NOT NULL	VARCHAR2(30)
TEXT_LENGTH		NUMBER
TEXT		LONG()

...

2

```
SELECT view_name FROM user_views;
```

VIEW_NAME
1 EMP_DETAILS_VIEW
2 SALVU50
3 EMPVU80
4 DEPT_SUM_VU

3

```
SELECT text FROM user_views  
WHERE view_name = 'EMP_DETAILS_VIEW';
```

TEXT
1 SELECT e.employee_id, e.job_id, e.manager_id, e.department_id, d.location_id, l.co

...

```
AND c.region_id = r.region_id AND j.job_id = e.job_id WITH READ ONLY
```

ORACLE

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

Nachdem Sie eine View erstellt haben, können Sie die Data Dictionary View `USER_VIEWS` abfragen, um Namen und Definition der View anzuzeigen. Der Text der zugrunde liegenden Anweisung `SELECT` für diese View wird in einer Spalte vom Typ `LONG` gespeichert. Die Spalte `TEXT_LENGTH` entspricht der Anzahl der Zeichen in der Anweisung `SELECT`. Bei der Auswahl aus einer Spalte vom Typ `LONG` werden standardmäßig nur die ersten 80 Zeichen des Spaltenwertes angezeigt. Um in SQL*Plus mehr als 80 Zeichen anzuzeigen, verwenden Sie den Befehl `SET LONG`:

```
SET LONG 1000
```

In den Beispielen auf der Folie werden folgende Aktionen ausgeführt:

1. Die Spalten von `USER_VIEWS` werden angezeigt. Diese Liste ist nicht vollständig.
2. Die Namen Ihrer Views werden abgerufen.
3. Die `SELECT`-Anweisung für `EMP_DETAILS_VIEW` wird aus dem Dictionary angezeigt.

Mit Views auf Daten zugreifen

Wenn Sie mithilfe einer View auf Daten zugreifen, führt der Oracle-Server folgende Vorgänge aus:

- Der Server ruft die View-Definition aus der Data Dictionary-Tabelle `USER_VIEWS` ab.
- Er prüft die Zugriffsberechtigungen für die Basistabelle der View.
- Er konvertiert die View-Abfrage in einen äquivalenten Vorgang für die zugrunde liegenden Basistabellen. Die Daten werden also aus den Basistabellen abgerufen oder dort aktualisiert.

Lektionsagenda

- Views – Überblick
- View-Daten erstellen, ändern und abrufen
- **DML-(Data Manipulation Language-)Vorgänge an Views**
- Views löschen

ORACLE®

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

DML-Vorgänge an Views ausführen – Regeln

- DML-Vorgänge können in der Regel an allen einfachen Views ausgeführt werden.
- Sie können keine Zeilen entfernen, wenn die View eines der folgenden Elemente enthält:
 - Gruppenfunktionen
 - Klausel `GROUP BY`
 - Schlüsselwort `DISTINCT`
 - Pseudospalten-Schlüsselwort `ROWNUM`



ORACLE

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

- Über Views lassen sich DML-Vorgänge für Daten ausführen, wenn diese Vorgänge bestimmte Regeln befolgen.
- Sie können Zeilen aus einer View entfernen, wenn sie keines der folgenden Elemente enthält:
 - Gruppenfunktionen
 - Klausel `GROUP BY`
 - Schlüsselwort `DISTINCT`
 - Pseudospalten-Schlüsselwort `ROWNUM`

DML-Vorgänge an Views ausführen – Regeln

Daten in einer View können nicht geändert werden, wenn die View eines der folgenden Elemente enthält:

- Gruppenfunktionen
- Klausel `GROUP BY`
- Schlüsselwort `DISTINCT`
- Pseudospalten-Schlüsselwort `ROWNUM`
- Ausdrücke

ORACLE

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

Sie können Daten nur dann über eine View ändern, wenn sie keine der auf der vorigen Folie angegebenen Elemente enthält.

DML-Vorgänge an Views ausführen – Regeln

Daten können nicht über eine View hinzugefügt werden, wenn die View eines der folgenden Elemente enthält:

- Gruppenfunktionen
- Klausel `GROUP BY`
- Schlüsselwort `DISTINCT`
- Pseudospalten-Schlüsselwort `ROWNUM`
- Durch Ausdrücke definierte Spalten
- `NOT NULL`-Spalten ohne Standardwert in den Basistabellen, die nicht von der View gewählt wurden

ORACLE

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

Sie können Daten nur dann über eine View hinzufügen, wenn die View keine der auf der Folie genannten Elemente enthält. Sie können einer View keine Daten hinzufügen, wenn sie Spalten vom Typ `NOT NULL` ohne Standardwerte in der Basistabelle enthält. Alle benötigten Werte müssen in der View vorhanden sein. Beachten Sie, dass Sie Werte *über* die View direkt der zugrunde liegenden Tabelle hinzufügen.

Weitere Informationen finden Sie in der *Oracle Database SQL Language Reference* für Oracle Database 12c im Abschnitt "`CREATE VIEW`".

Klausel WITH CHECK OPTION

- Mit der Klausel `WITH CHECK OPTION` sicherstellen, dass an einer View ausgeführte DML-Vorgänge in der Domain der View bleiben:

```
CREATE OR REPLACE VIEW empvu20
AS SELECT      *
   FROM        employees
   WHERE       department_id = 20
   WITH CHECK OPTION CONSTRAINT empvu20_ck ;
```

view EMPVU20 created.

- Versuche, mit `INSERT` eine Zeile mit einer anderen `department_id` als 20 einzufügen oder mit `UPDATE` die Abteilungsnummer für eine Zeile in der View zu aktualisieren, scheitern, da gegen die Klausel `WITH CHECK OPTION` verstoßen wird.

ORACLE

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

Mithilfe von Views können Sie Prüfungen zur referenziellen Integrität durchführen. Außerdem lassen sich Constraints auf Datenbankebene erzwingen. Die View kann zum Schutz der Datenintegrität verwendet werden, ihr Einsatz ist jedoch sehr begrenzt.

Die Klausel `WITH CHECK OPTION` gibt an, dass über die View ausgeführte `INSERT`- und `UPDATE`-Vorgänge keine Zeilen erstellen können, die die View nicht wählen kann. Somit können Integritäts-Constraints und Datenvalidierungen für Daten erzwungen werden, die eingefügt oder aktualisiert werden sollen. Beim Versuch, DML-Vorgänge an Zeilen auszuführen, die von der View nicht gewählt wurden, wird ein Fehler angezeigt. Außerdem wird der Constraint-Name angezeigt, sofern angegeben.

```
UPDATE empvu20
SET    department_id = 10
WHERE  employee_id = 201;
```

Fehler:

```
SQL Error: ORA-01402: view WITH CHECK OPTION where-clause violation
01402. 00000 - "view WITH CHECK OPTION where-clause violation"
*Cause:
*Action:
```

Hinweis: Die Zeilen werden nicht aktualisiert, da die View diesen Mitarbeiter nicht mehr anzeigen kann, wenn sich die Abteilungsnummer in 10 ändert. Aufgrund der Klausel `WITH CHECK OPTION` kann die View nur Mitarbeiter aus Abteilung 20 anzeigen. Es ist nicht möglich, die Abteilungsnummer für diese Mitarbeiter über die View zu ändern.

DML-Vorgänge verweigern

- Sie können verhindern, dass DML-Vorgänge ausgeführt werden, indem Sie die View-Definition um die Option `WITH READ ONLY` erweitern.
- Jeder Versuch, einen DML-Vorgang an einer Zeile in der View auszuführen, führt dann zu einem Oracle-Server-Fehler.



ORACLE

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

Sie können verhindern, dass DML-Vorgänge für eine View ausgeführt werden, indem Sie die View mit der Option `WITH READ ONLY` erstellen. Im Beispiel auf der nächsten Folie wird die View `EMPVU10` so geändert, dass keine DML-Vorgänge daran ausgeführt werden können.

DML-Vorgänge verweigern

```
CREATE OR REPLACE VIEW empvu10
  (employee_number, employee_name, job_title)
AS SELECT      employee_id, last_name, job_id
  FROM        employees
 WHERE      department_id = 10
 WITH READ ONLY ;
```

view EMPVU10 created.

ORACLE

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

Jeder Versuch, eine Zeile aus einer View mit einem READ-ONLY-Constraint zu löschen, führt zu einem Fehler:

```
DELETE FROM empvu10
 WHERE employee_number = 200;
```

Entsprechend führt jeder Versuch, Zeilen in einer View mit READ-ONLY-Constraint einzufügen oder zu ändern, zur gleichen Fehlermeldung.

```
Error report:
SQL Error: ORA-42399: cannot perform a DML operation on a read-only view
```

Lektionsagenda

- Views – Überblick
- View-Daten erstellen, ändern und abrufen
- DML-(Data Manipulation Language-)Vorgänge an Views
- **Views löschen**

ORACLE®

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

Views entfernen

Views können ohne Datenverlust gelöscht werden, da sie auf den zugrunde liegenden Tabellen in der Datenbank basieren.

```
DROP VIEW view;
```

```
DROP VIEW empvu80;
```

```
view EMPVU80 dropped.
```

ORACLE

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

Um eine View zu entfernen, verwenden Sie die Anweisung `DROP VIEW`. Mit dieser Anweisung wird die View-Definition aus der Datenbank entfernt. Das Löschen von Views hat jedoch keine Auswirkungen auf die Tabellen, auf denen die Views basieren. Andererseits werden Views und andere Anwendungen ungültig, die auf den gelöschten Views basieren. Nur der Ersteller oder ein Benutzer mit der Berechtigung `DROP ANY VIEW` kann eine View entfernen.

In der Syntax steht *view* für den Namen der View.

Quiz

Daten können nicht über eine View hinzugefügt werden, wenn die View die Klausel `GROUP BY` enthält.

- 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:

- Views erstellen, verwenden und entfernen
- Dictionary Views nach View-Informationen abfragen

ORACLE

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

In dieser Lektion haben Sie Views kennengelernt.

Übungen zu Lektion 4 – Überblick

Diese Übungen behandeln folgende Themen:

- Einfache View erstellen
- Komplexe View erstellen
- View mit einem CHECK-Constraint erstellen
- Datenänderungen in einer View versuchen
- Dictionary Views nach View-Informationen abfragen
- Views entfernen

ORACLE

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

Im Rahmen der Übung werden verschiedene Szenarios zur Erstellung, Verwendung, Abfrage und Entfernung von Data Dictionary Views behandelt.

5

Schemaobjekte verwalten

ORACLE®

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

Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- Constraints verwalten
- Temporäre Tabellen erstellen und verwenden
- Externe Tabellen erstellen und verwenden

ORACLE

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

In dieser Lektion werden Constraints erläutert. Darüber hinaus lernen Sie, wie Sie vorhandene Objekte ändern, mit externen Tabellen arbeiten und beim Erstellen von `PRIMARY KEY`-Constraints Indizes benennen.

Lektionsagenda

- Constraints verwalten:
 - Constraints hinzufügen und löschen
 - Constraints aktivieren und deaktivieren
 - Constraints verzögern
- Temporäre Tabellen erstellen und verwenden
- Externe Tabellen erstellen und verwenden

ORACLE

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

Constraints hinzufügen – Syntax

Mit der Anweisung `ALTER TABLE` können Sie:

- Constraints hinzufügen und löschen, jedoch nicht deren Struktur ändern
- Constraints aktivieren und deaktivieren
- NOT NULL-Constraints mithilfe der Klausel `MODIFY` hinzufügen

```
ALTER TABLE <table_name>  
ADD [CONSTRAINT <constraint_name>]  
type (<column_name>);
```

ORACLE

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

Mit der Anweisung `ALTER TABLE` und der Klausel `ADD` können Sie Constraints für vorhandene Tabellen hinzufügen.

Für die Syntax gilt:

<i>table</i>	Name der Tabelle
<i>constraint</i>	Name des Constraints
<i>type</i>	Constraint-Typ
<i>column</i>	Name der Spalte, für die das Constraint gilt

Die angegebene Syntax für den Constraint-Namen ist optional, wird jedoch empfohlen. Wenn Sie die Constraints nicht benennen, wird ein Name vom System generiert.

Richtlinien

- Sie können Constraints hinzufügen, löschen, aktivieren und deaktivieren, können jedoch nicht ihre Struktur ändern.
- Mit der Klausel `MODIFY` der Anweisung `ALTER TABLE` können Sie einer vorhandenen Spalte ein NOT NULL-Constraint hinzufügen.

Hinweis: NOT NULL-Spalten können nur für leere Tabellen definiert werden oder wenn die Spalte in jeder Zeile einen Wert enthält.

Constraints hinzufügen

Der Tabelle EMP2 ein FOREIGN KEY-Constraint hinzufügen, das vorgibt, dass der Vorgesetzte in der Tabelle EMP2 bereits als gültiger Angestellter vorhanden sein muss

```
ALTER TABLE emp2  
MODIFY employee_id PRIMARY KEY;
```

```
table EMP2 altered.
```

```
ALTER TABLE emp2  
ADD CONSTRAINT emp_mgr_fk  
FOREIGN KEY(manager_id)  
REFERENCES emp2(employee_id);
```

```
table EMP2 altered.
```

ORACLE

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

Im ersten Beispiel auf der Folie wird die Tabelle EMP2 geändert, indem ein PRIMARY KEY-Constraint für die Spalte EMPLOYEE_ID hinzugefügt wird. Da kein Name angegeben wurde, generiert der Oracle-Server automatisch einen Namen für das Constraint. Im zweiten Beispiel auf der Folie wird ein FOREIGN KEY-Constraint für die Tabelle EMP2 erstellt. Das Constraint stellt sicher, dass ein Vorgesetzter in der Tabelle EMP2 bereits als gültiger Angestellter vorhanden ist.

Constraints löschen

- Mit der Klausel `DROP CONSTRAINT` können Sie ein Integritäts-Constraint aus einer Datenbank löschen.
- Manager-Constraint aus Tabelle `EMP2` entfernen:

```
ALTER TABLE emp2
DROP CONSTRAINT emp_mgr_fk;
```

```
table EMP2 altered.
```

- PRIMARY KEY-Constraint aus Tabelle `DEPT2` entfernen und zugehörigen FOREIGN KEY-Constraint aus Spalte `EMP2.DEPARTMENT_ID` löschen:

```
ALTER TABLE emp2
DROP PRIMARY KEY CASCADE;
```

```
table EMP2 altered.
```

ORACLE

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

Mit der Klausel `DROP CONSTRAINT` können Sie ein Integritäts-Constraint aus einer Datenbank löschen.

Um ein Constraint zu löschen, identifizieren Sie den Constraint-Namen in den Data Dictionary Views `USER_CONSTRAINTS` und `USER_CONS_COLUMNS` und verwenden dann die Anweisung `ALTER TABLE` mit der Klausel `DROP`. Die Option `CASCADE` der Klausel `DROP` bewirkt, dass abhängige Constraints ebenfalls gelöscht werden.

Syntax

```
ALTER TABLE    table
DROP    PRIMARY KEY | UNIQUE (column) |
        CONSTRAINT constraint [CASCADE];
```

Für die Syntax gilt:

<i>table</i>	Name der Tabelle
<i>column</i>	Name der Spalte, für die das Constraint gilt
<i>constraint</i>	Name des Constraints

Wenn Sie ein Integritäts-Constraint löschen, wird das Constraint nicht mehr vom Oracle-Server durchgesetzt und ist im Data Dictionary nicht mehr verfügbar.

Constraints mit Schlüsselwort `ONLINE` löschen

Mit dem Schlüsselwort `ONLINE` angeben, dass DML-Vorgänge für die Tabelle zulässig sind, während das Constraint gelöscht wird.

```
ALTER TABLE myemp2  
DROP CONSTRAINT emp_name_pk ONLINE;
```

ORACLE

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

Sie können Constraints auch in Verbindung mit dem Schlüsselwort `ONLINE` löschen.

```
ALTER TABLE myemp2  
DROP CONSTRAINT emp_id_pk ONLINE;
```

Verwenden Sie die Anweisung `ALTER TABLE` mit der Klausel `DROP`. Die Option `ONLINE` der Klausel `DROP` gibt an, dass DML-Vorgänge für die Tabelle zulässig sind, während das Constraint gelöscht wird.

Klausel ON DELETE

- Mit der Klausel `ON DELETE CASCADE` untergeordnete Zeilen löschen, wenn ein übergeordneter Schlüssel gelöscht wird:

```
ALTER TABLE dept2 ADD CONSTRAINT dept_lc_fk  
FOREIGN KEY (location_id)  
REFERENCES locations(location_id) ON DELETE CASCADE;
```

```
table DEPT2 altered.
```

- Mit der Klausel `ON DELETE SET NULL` die Werte der untergeordneten Zeilen auf `NULL` einstellen, wenn ein übergeordneter Schlüssel gelöscht wird:

```
ALTER TABLE emp2 ADD CONSTRAINT emp_dt_fk  
FOREIGN KEY (Department_id)  
REFERENCES departments(department_id) ON DELETE SET NULL;
```

```
table EMP2 altered.
```

ORACLE

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

ON DELETE

Mit der Klausel `ON DELETE` können Sie festlegen, wie Oracle Database die referenzielle Integrität handhabt, wenn Sie den Wert eines referenzierten Primärschlüssels oder eindeutigen Schlüssels entfernen.

ON DELETE CASCADE

Mit der Aktion `ON DELETE CASCADE` können Sie übergeordnete Schlüsseldaten, die von der untergeordneten Tabelle referenziert werden, löschen, jedoch nicht aktualisieren. Wenn Daten im übergeordneten Schlüssel gelöscht werden, werden gleichzeitig alle Zeilen in der untergeordneten Tabelle gelöscht, die von den gelöschten Werten des übergeordneten Schlüssels abhängen. Um diese referenzielle Aktion anzugeben, nehmen Sie die Option `ON DELETE CASCADE` in die Definition des Constraints `FOREIGN KEY` auf.

ON DELETE SET NULL

Mit der Aktion `ON DELETE SET NULL` werden alle Zeilen, die von den gelöschten Werten des übergeordneten Schlüssels abhängen, in `NULL` umgewandelt.

Ohne diese Klausel ist das Löschen von referenzierten Schlüsselwerten in der übergeordneten Tabelle nicht zulässig, wenn in der untergeordneten Tabelle abhängige Zeilen vorhanden sind.

Klausel CASCADE CONSTRAINTS

- Die Klausel CASCADE CONSTRAINTS:
 - Wird zusammen mit der Klausel DROP COLUMN verwendet
 - Löscht alle referenziellen Integritäts-Constraints mit Bezug auf die Schlüssel PRIMARY und UNIQUE, die in den gelöschten Spalten definiert wurden
 - Löscht alle für mehrere Spalten geltenden Constraints, die in den gelöschten Spalten definiert wurden

ORACLE

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

Die folgende Anweisung veranschaulicht die Verwendung der Klausel CASCADE CONSTRAINTS. Die Tabelle TEST1 wird in diesem Beispiel wie folgt erstellt:

```
CREATE TABLE test1 (  
    col1_pk NUMBER PRIMARY KEY,  
    col2_fk NUMBER,  
    col1 NUMBER,  
    col2 NUMBER,  
    CONSTRAINT fk_constraint FOREIGN KEY (col2_fk) REFERENCES  
        test1,  
    CONSTRAINT ck1 CHECK (col1_pk > 0 and col1 > 0),  
    CONSTRAINT ck2 CHECK (col2_fk > 0));
```

Für die folgenden Anweisungen wird eine Fehlermeldung zurückgegeben:

```
ALTER TABLE test1 DROP (col1_pk);    – col1_pk ist ein übergeordneter Schlüssel.  
ALTER TABLE test1 DROP (col1);    – col1 wird von dem für mehrere Spalten geltenden  
                                   Constraint ck1 referenziert.
```

Klausel CASCADE CONSTRAINTS

Beispiel:

```
ALTER TABLE emp2
DROP COLUMN employee_id CASCADE CONSTRAINTS;
```

```
table EMP2 altered.
```

```
ALTER TABLE test1
DROP (col1_pk, col2_fk, col1);
```

```
table TEST1 altered.
```

ORACLE

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

Durch Absetzen der folgenden Anweisung werden die Spalte `EMPLOYEE_ID`, das Constraint `PRIMARY KEY` sowie sämtliche `FOREIGN KEY`-Constraints gelöscht, die das `PRIMARY KEY`-Constraint für die Tabelle `EMP2` referenzieren:

```
ALTER TABLE emp2 DROP COLUMN employee_id CASCADE CONSTRAINTS;
```

`CASCADE CONSTRAINTS` ist nicht erforderlich, wenn auch alle Spalten gelöscht werden, die von den Constraints für die gelöschten Spalten referenziert werden. Beispiel: Wenn keine anderen referenziellen Constraints anderer Tabellen die Spalte `COL1_PK` referenzieren, kann die folgende Anweisung ohne die Klausel `CASCADE CONSTRAINTS` für die auf der vorherigen Seite erstellte Tabelle `TEST1` abgesetzt werden:

```
ALTER TABLE test1 DROP (col1_pk, col2_fk, col1);
```

- Durch Aktivieren des `PRIMARY KEY`-Constraints, das über die Option `CASCADE` deaktiviert wurde, werden keine Fremdschlüssel (`FOREIGN KEY`) aktiviert, die vom Primärschlüssel (`PRIMARY KEY`) abhängig sind.
- Um ein `UNIQUE`- oder `PRIMARY KEY`-Constraint zu aktivieren, müssen Sie über die erforderlichen Berechtigungen zur Erstellung eines Index für die Tabelle verfügen.

Tabellenspalten und Constraints umbenennen

- Mit der Klausel **RENAME TABLE** der Anweisung **ALTER TABLE** Tabellen umbenennen:

```
ALTER TABLE marketing RENAME to new_marketing;
```

a

- Mit der Klausel **RENAME COLUMN** der Anweisung **ALTER TABLE** Tabellenspalten umbenennen:

```
ALTER TABLE new_marketing RENAME COLUMN team_id  
TO id;
```

b

- Mit der Klausel **RENAME CONSTRAINT** der Anweisung **ALTER TABLE** für eine Tabelle vorhandene Constraints umbenennen:

```
ALTER TABLE new_marketing RENAME CONSTRAINT mktg_pk  
TO new_mktg_pk;
```

c

ORACLE

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

Mit der Klausel **RENAME TABLE** können Sie eine vorhandene Tabelle in ein beliebiges Schema (außer SYS) umbenennen. Sie müssen entweder der Eigentümer der Datenbank oder der Tabelle sein.

Der neue Name der Tabellenspalte darf keinen Konflikt mit dem Namen einer vorhandenen Spalte verursachen. In Verbindung mit der Klausel **RENAME COLUMN** sind keine anderen Klauseln zulässig.

In den Beispielen auf der Folie wird die Tabelle `marketing` mit dem für die Spalte `id` definierten Primärschlüssel (**PRIMARY KEY**) `mktg_pk` verwendet.

```
CREATE TABLE marketing (team_id NUMBER(10),  
                        target VARCHAR2(50),  
                        CONSTRAINT mktg_pk PRIMARY KEY(team_id));
```

Beispiel **a** zeigt die Umbenennung der Tabelle `marketing` in `new_marketing`. Beispiel **b** zeigt die Umbenennung der Spalte `id` in der Tabelle `new_marketing` in `mktg_id`, und Beispiel **c** zeigt die Umbenennung von `mktg_pk` in `new_mktg_pk`.

Der neue Name für ein in einer Tabelle vorhandenes Constraint darf keinen Konflikt mit anderen vorhandenen Constraint-Namen verursachen. Mit der Klausel **RENAME CONSTRAINT** können Sie vom System generierte Constraint-Namen umbenennen.

Constraints deaktivieren

- Um ein Integritäts-Constraint zu deaktivieren, führen Sie die Klausel `DISABLE` der Anweisung `ALTER TABLE` aus.
- Mit der Option `CASCADE` werden zusammen mit dem Primärschlüssel automatisch auch alle abhängigen `FOREIGN KEY`-Constraints deaktiviert.

```
ALTER TABLE emp2  
DISABLE CONSTRAINTS emp_dt_pk;
```

```
table EMP2 altered.
```

```
ALTER TABLE dept2  
DISABLE primary key CASCADE;
```

```
table DEPT2 altered.
```

ORACLE

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

Mit der Anweisung `ALTER TABLE` und der Klausel `DISABLE` können Sie Constraints deaktivieren, ohne sie zu löschen oder neu zu erstellen. Mit der Option `CASCADE` können Sie auch den Primärschlüssel oder den eindeutigen Schlüssel deaktivieren.

Syntax

```
ALTER TABLE table  
DISABLE CONSTRAINT constraint [CASCADE];
```

Für die Syntax gilt:

<i>table</i>	Name der Tabelle
<i>constraint</i>	Name des Constraints

Richtlinien

- Sie können die Klausel `DISABLE` sowohl in der Anweisung `CREATE TABLE` als auch in der Anweisung `ALTER TABLE` verwenden.
- Die Klausel `CASCADE` deaktiviert abhängige Integritäts-Constraints.
- Durch Deaktivieren eines `UNIQUE`- oder `PRIMARY KEY`-Constraints wird der eindeutige Index entfernt.

Constraints aktivieren

- In der Tabellendefinition deaktiviertes Integritäts-Constraint mit der Klausel `ENABLE` aktivieren

```
ALTER TABLE      emp2
ENABLE CONSTRAINT emp_dt_fk;
```

```
table EMP2 altered.
```

- Bei Aktivierung eines `UNIQUE KEY`- oder `PRIMARY KEY`-Constraints wird automatisch ein `UNIQUE`-Index erstellt.

ORACLE

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

Mit der Anweisung `ALTER TABLE` und der Klausel `ENABLE` können Sie Constraints aktivieren, ohne sie zu löschen oder neu zu erstellen.

Syntax

```
ALTER    TABLE      table
ENABLE  CONSTRAINT constraint;
```

Für die Syntax gilt:

table Name der Tabelle
constraint Name des Constraints

Richtlinien

- Ein aktiviertes Constraint gilt für alle Daten in der Tabelle. Alle Daten in der Tabelle müssen die Vorgaben des Constraints erfüllen.
- Wenn Sie ein `UNIQUE KEY`- oder `PRIMARY KEY`-Constraint aktivieren, wird automatisch ein `UNIQUE`- oder `PRIMARY KEY`-Index erstellt. Ist bereits ein Index vorhanden, kann er von diesen Schlüsseln verwendet werden.
- Sie können die Klausel `ENABLE` sowohl in der Anweisung `CREATE TABLE` als auch in der Anweisung `ALTER TABLE` verwenden.

Constraints – Statusmöglichkeiten

Für Integritäts-Constraints, die für eine Tabelle definiert werden, gibt es die folgenden Statusmöglichkeiten:

- `ENABLE VALIDATE`
- `ENABLE NOVALIDATE`
- `DISABLE VALIDATE`
- `DISABLE NOVALIDATE`

```
ALTER TABLE dept2  
ENABLE NOVALIDATE PRIMARY KEY;
```

```
table DEPT2 altered.
```

ORACLE

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

Mit der Anweisung `CREATE TABLE` oder `ALTER TABLE` können Sie Integritäts-Constraints auf Tabellenebene aktivieren oder deaktivieren. Sie können Constraints auch auf `VALIDATE` oder `NOVALIDATE` einstellen, und zwar in jeder Kombination mit `ENABLE` oder `DISABLE`. Hierbei gilt:

- `ENABLE` stellt sicher, dass alle eingehenden Daten die Vorgaben des Constraints erfüllen.
- `DISABLE` lässt alle eingehenden Daten unabhängig von ihrer Übereinstimmung mit dem Constraint zu.
- `VALIDATE` stellt sicher, dass die vorhandenen Daten die Vorgaben des Constraints erfüllen.
- `NOVALIDATE` bedeutet, dass einige der vorhandenen Daten die Vorgaben des Constraints möglicherweise nicht erfüllen.

`ENABLE VALIDATE` ist identisch mit `ENABLE`. Das Constraint wird geprüft und wird garantiert von allen Zeilen eingehalten. `ENABLE NOVALIDATE` bedeutet, dass das Constraint geprüft wird, aber nicht von allen Zeilen eingehalten werden muss. Damit wird ermöglicht, dass vorhandene Zeilen das Constraint verletzen können, alle neuen oder geänderten Zeilen jedoch gültig sind. In einer `ALTER TABLE`-Anweisung wird mit `ENABLE NOVALIDATE` die Prüfung von Constraints in deaktivierten Constraints wieder aufgenommen, ohne dass zuerst alle Daten in der Tabelle geprüft werden. `DISABLE NOVALIDATE` ist identisch mit `DISABLE`. Das Constraint wird nicht geprüft und wird nicht zwingend erfüllt. Mit `DISABLE VALIDATE` deaktiviert das Constraint, löscht den Index im Constraint und untersagt jegliche Änderungen in den Spalten, die dem Constraint unterliegen.

Constraints verzögern

Constraints können über folgende Attribute verfügen:

- DEFERRABLE oder NOT DEFERRABLE
- INITIALLY DEFERRED oder INITIALLY IMMEDIATE

```
ALTER TABLE dept2  
ADD CONSTRAINT dept2_id_pk  
PRIMARY KEY (department_id)  
DEFERRABLE INITIALLY DEFERRED;
```

Constraints beim Erstellen verzögern

```
SET CONSTRAINTS dept2_id_pk IMMEDIATE;
```

Bestimmtes Constraint-Attribut ändern

```
ALTER SESSION  
SET CONSTRAINTS= IMMEDIATE;
```

Alle Constraints für eine Session ändern

ORACLE

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

Sie können die Prüfung von Constraints auf Gültigkeit an das Ende der Transaktion verschieben. Constraints sind verzögert, wenn das System erst nach dem Absetzen einer COMMIT-Anweisung prüft, ob die Constraint-Bedingung erfüllt ist. Bei Verletzung eines verzögerten Constraints gibt die Datenbank einen Fehler zurück. Die Transaktion wird nicht festgeschrieben, sondern zurückgerollt. Unmittelbare (nicht verzögerte) Constraints werden am Ende jeder Anweisung geprüft. Bei Verletzungen wird die Anweisung sofort zurückgerollt. Durch Constraints verursachte Aktionen (z. B. DELETE CASCADE) werden stets als Teil der sie verursachenden Anweisung gesehen. Dies gilt sowohl für verzögerte als auch für unmittelbare Constraints. Geben Sie mit der Anweisung SET CONSTRAINTS für eine bestimmte Transaktion an, ob ein verzögerbares Constraint nach jeder DML-(Data Manipulations Language-)Anweisung oder beim Festschreiben der Transaktion geprüft werden soll. Um verzögerbare Constraints zu erstellen, müssen Sie einen nicht eindeutigen Index für dieses Constraint erstellen.

Sie können Constraints entweder als verzögerbar (DEFERRABLE) oder nicht verzögerbar (NOT DEFERRABLE, Standard) definieren und entweder als anfänglich verzögert (INITIALLY DEFERRED) oder anfänglich unmittelbar (INITIALLY IMMEDIATE, Standard) festlegen. Diese Attribute können sich für jedes Constraint unterscheiden.

Verwendungsszenario: Entsprechend der Firmenpolitik soll die Abteilungsnummer 40 in 45 geändert werden. Das Ändern der Spalte DEPARTMENT_ID wirkt sich auf die dieser Abteilung zugeordneten Mitarbeiter aus. Daher legen Sie für den Primärschlüssel (PRIMARY KEY) und die Fremdschlüssel (FOREIGN KEY) DEFERRABLE und INITIALLY DEFERRED fest. Aktualisieren Sie sowohl die Abteilungs- als auch die Mitarbeiterinformationen. Beim Festschreiben werden alle Zeilen validiert.

INITIALLY DEFERRED und INITIALLY IMMEDIATE – Vergleich

INITIALLY DEFERRED	Wartet mit der Constraint-Prüfung bis zum Ende der Transaktion
INITIALLY IMMEDIATE	Prüft das Constraint am Ende der Anweisungsausführung

```
CREATE TABLE emp_new_sal (salary NUMBER
    CONSTRAINT sal_ck
    CHECK (salary > 100)
    DEFERRABLE INITIALLY IMMEDIATE,
    bonus NUMBER
    CONSTRAINT bonus_ck
    CHECK (bonus > 0 )
    DEFERRABLE INITIALLY DEFERRED );
```

```
table EMP_NEW_SAL created.
```

ORACLE

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

Ein als verzögerbar definiertes Constraint kann entweder als `INITIALLY DEFERRED` oder als `INITIALLY IMMEDIATE` festgelegt werden. Standardmäßig gilt die Klausel `INITIALLY IMMEDIATE`.

Die Folie enthält folgende Beispiele:

- Das Constraint `sal_ck` wird als `DEFERRABLE INITIALLY IMMEDIATE` erstellt.
- Das Constraint `bonus_ck` wird als `DEFERRABLE INITIALLY DEFERRED` erstellt.

Nachdem Sie die Tabelle `emp_new_sal` wie auf der Folie dargestellt erstellt haben, versuchen Sie, Werte in die Tabelle einzugeben. Sehen Sie sich die Ergebnisse an. Wenn die Bedingungen des Constraints `sal_ck` und des Constraints `bonus_ck` erfüllt sind, werden die Zeilen ohne Fehler eingefügt.

1. Beispiel: Fügen Sie eine Zeile ein, die `sal_ck` verletzt. In der Anweisung `CREATE TABLE` ist `sal_ck` als anfänglich unmittelbares Constraint festgelegt. Das Constraint wird also sofort nach der Anweisung `INSERT` geprüft, und ein Fehler wird angezeigt.

```
INSERT INTO emp_new_sal VALUES(90,5);
```

```
SQL Error: ORA-02290: check constraint (ORA21.SAL_CK) violated
02290. 00000 - "check constraint (%s.%s) violated"
```

2. Beispiel: Fügen Sie eine Zeile ein, die `bonus_ck` verletzt. In der Anweisung `CREATE TABLE` ist `bonus_ck` als verzögerbar sowie unmittelbar verzögert festgelegt. Daher wird das Constraint erst geprüft, wenn Sie die Transaktion über `COMMIT` festschreiben oder das Constraint in den Status `IMMEDIATE` zurücksetzen.


```
INSERT INTO emp_new_sal VALUES(110, -1);
```

```
1 rows inserted
```

Die Zeile wird erfolgreich eingefügt. Beim Festschreiben der Transaktion wird ein Fehler angezeigt.

```
COMMIT;
```

```
SQL Error: ORA-02091: transaction rolled back
ORA-02290: check constraint (ORA21.BONUS_CK) violated
02091. 00000 - "transaction rolled back"
```

Das Festschreiben war aufgrund der Constraint-Verletzung nicht erfolgreich. Daher wird die Transaktion an dieser Stelle von der Datenbank zurückgesetzt.

3. Beispiel: Legen Sie für alle Constraints, die verzögert werden können, den Status `DEFERRED` fest. Falls erforderlich, können Sie den Status `DEFERRED` auch für ein einzelnes Constraint festlegen.

```
SET CONSTRAINTS ALL DEFERRED;
```

```
constraints ALL succeeded.
```

Wenn Sie nun versuchen, eine Zeile einzufügen, die das Constraint `sal_ck` verletzt, wird die Anweisung erfolgreich ausgeführt.

```
INSERT INTO emp_new_sal VALUES(90,5);
```

```
1 rows inserted
```

Beim Festschreiben der Transaktion wird jedoch ein Fehler angezeigt. Die Transaktion ist nicht erfolgreich und wird zurückgesetzt. Dies ist darauf zurückzuführen, dass beide Constraints beim Festschreiben über `COMMIT` geprüft werden.

```
COMMIT;
```

```
SQL Error: ORA-02091: transaction rolled back
ORA-02290: check constraint (ORA21.SAL_CK) violated
02091. 00000 - "transaction rolled back"
```

4. Beispiel: Legen Sie für beide Constraints, die im vorherigen Beispiel als `DEFERRED` angegeben wurden, den Status `IMMEDIATE` fest.

```
SET CONSTRAINTS ALL IMMEDIATE;
```

```
constraints ALL succeeded.
```

Es wird ein Fehler angezeigt, wenn Sie versuchen, eine Zeile einzufügen, die entweder `sal_ck` oder `bonus_ck` verletzt.

```
INSERT INTO emp_new_sal VALUES(110, -1);
```

```
SQL Error: ORA-02290: check constraint (ORA21.BONUS_CK) violated
02290. 00000 - "check constraint (%s.%s) violated"
```

Hinweis: Wenn Sie bei Erstellung einer Tabelle keine Verzögerbarkeit für das Constraint angeben, wird das Constraint sofort am Ende der Anweisungen geprüft. Beispiel: In der Anweisung `CREATE TABLE` für die Tabelle `newemp_details` ist keine Verzögerbarkeit für das Constraint `newemp_det_pk` angegeben. In diesem Fall wird das Constraint sofort geprüft.

```
CREATE TABLE newemp_details(emp_id NUMBER, emp_name
VARCHAR2(20),
CONSTRAINT newemp_det_pk PRIMARY KEY(emp_id));
```

Beim Versuch, das nicht verzögerbare Constraint `newemp_det_pk` zu verzögern, wird der folgende Fehler angezeigt:

```
SET CONSTRAINT newemp_det_pk DEFERRED;
```

```
SQL Error: ORA-02447: cannot defer a constraint that is not deferrable
```

DROP TABLE ... PURGE

```
DROP TABLE emp_new_sal PURGE;
```

```
table DEPT80 dropped.
```

ORACLE®

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

Die Oracle-Datenbank enthält ein Feature zum Löschen von Tabellen. Die Datenbank gibt den Speicherplatz von gelöschten Tabellen nicht sofort frei, sondern benennt die Tabelle um und verschiebt sie in den Papierkorb. Falls die Tabelle also unbeabsichtigterweise gelöscht wurde, kann sie von dort mit der Anweisung `FLASHBACK TABLE` wiederhergestellt werden. Wenn der einer Tabelle zugewiesene Speicherplatz sofort beim Absetzen der Anweisung `DROP TABLE` freigegeben werden soll, fügen Sie die Klausel `PURGE` ein, wie in der Anweisung auf der Folie dargestellt.

Geben Sie `PURGE` nur dann an, wenn Sie in einem Schritt die Tabelle löschen und den ihr zugewiesenen Speicherplatz freigeben möchten. Mit `PURGE` werden Tabelle und abhängige Objekte nicht im Papierkorb abgelegt.

Mit der Klausel wird die Tabelle zunächst gelöscht und dann dauerhaft aus dem Papierkorb entfernt. Die Klausel spart also einen Schritt im Prozess. Sie bietet darüber hinaus erweiterte Sicherheit, falls Sie verhindern möchten, dass sensible Daten im Papierkorb gespeichert werden.

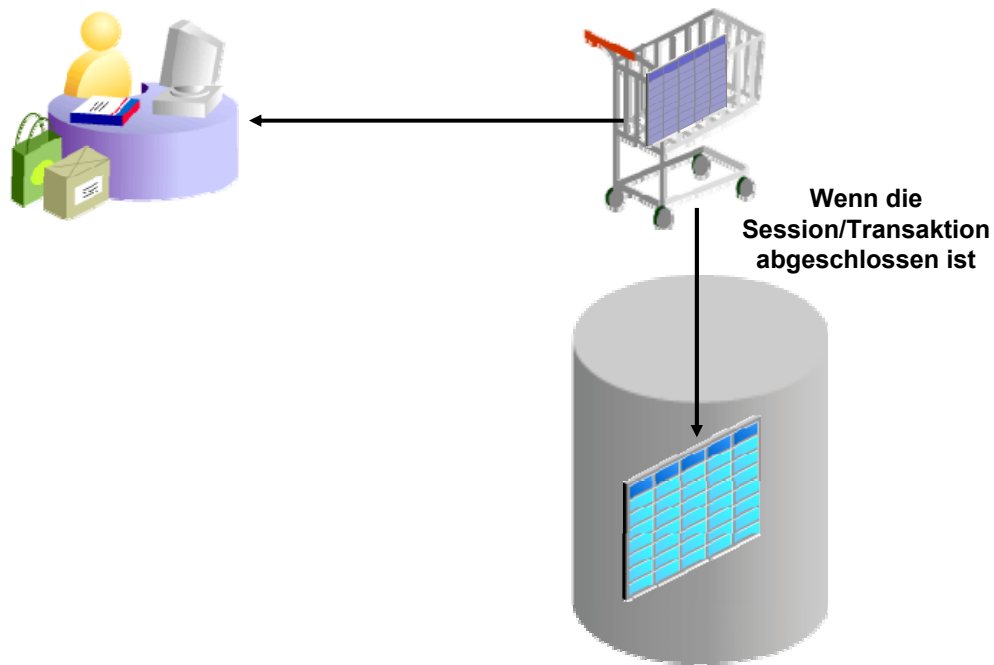
Lektionsagenda

- Constraints verwalten:
 - Constraints hinzufügen und löschen
 - Constraints aktivieren und deaktivieren
 - Constraints verzögern
- Temporäre Tabellen erstellen und verwenden
- Externe Tabellen erstellen und verwenden

ORACLE®

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

Temporäre Tabellen



ORACLE

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

Eine temporäre Tabelle enthält Daten, die nur für die Dauer einer Transaktion oder einer Session existieren. Daten in temporären Tabellen sind private Daten der Session. Das bedeutet, dass jede Session nur die eigenen Daten anzeigen und ändern kann.

Temporäre Tabellen sind nützlich in Anwendungen, in denen eine Ergebnismenge im Puffer gespeichert werden muss. Ein Warenkorb einer Onlineanwendung kann beispielsweise eine temporäre Tabelle sein. Jeder Artikel wird in der temporären Tabelle durch eine Zeile dargestellt. Während Ihres Einkaufs in einem Onlinegeschäft können Sie Artikel in den Warenkorb einfügen oder aus dem Warenkorb entfernen. Diese Warenkorbdaten sind während der Dauer der Session privat. Nachdem Sie Ihren Einkauf beendet und die Zahlung durchgeführt haben, werden die Zeilen für den gewählten Warenkorb in eine permanente Tabelle verschoben. Die Daten in der temporären Tabelle werden am Ende der Session automatisch gelöscht.

Da temporäre Tabellen statisch definiert werden, können Sie Indizes für diese Tabellen erstellen. Für temporäre Tabellen erstellte Indizes sind ebenfalls temporär. Die Daten im Index haben den gleichen Session- oder Transaktionsgeltungsbereich wie die Daten in der temporären Tabelle. Sie können für temporäre Tabellen auch Views oder Trigger erstellen.

Temporäre Tabellen erstellen

```
CREATE GLOBAL TEMPORARY TABLE cart(n NUMBER,d DATE)
ON COMMIT DELETE ROWS;
```

1

```
CREATE GLOBAL TEMPORARY TABLE today_sales
ON COMMIT PRESERVE ROWS AS
  SELECT * FROM orders
    WHERE order_date = SYSDATE;
```

2

ORACLE

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

Mit dem folgenden Befehl können Sie eine temporäre Tabelle erstellen:

```
CREATE GLOBAL TEMPORARY TABLE tablename
ON COMMIT [PRESERVE | DELETE] ROWS
```

Die folgenden Einstellungen für die Klausel `ON COMMIT` legen fest, ob die Daten in der temporären Tabelle transaktionsspezifisch (Standard) oder sessionspezifisch sind.

1. **DELETE ROWS:** Im ersten Beispiel auf der Folie wird mit der Einstellung `DELETE ROWS` eine transaktionsspezifische temporäre Tabelle erstellt. Sobald im Rahmen der Transaktion die ersten Daten in die Tabelle eingefügt werden, wird eine Session an die temporäre Tabelle gebunden. Die Bindung wird am Ende der Transaktion aufgehoben. Nach jedem Commit leert die Datenbank die Tabelle, das heißt, löscht alle Zeilen.
2. **PRESERVE ROWS:** Im zweiten Beispiel auf der Folie wird mit der Einstellung `PRESERVE ROWS` eine sessionspezifische temporäre Tabelle erstellt. Bei jeder Vertriebsmitarbeitersession können eigene Verkaufsdaten für den Tag (`today_sales`) in der Tabelle gespeichert werden. Wenn ein Vertriebsmitarbeiter zum ersten Mal Daten in die Tabelle `today_sales` einfügt, wird seine Session an die Tabelle `today_sales` gebunden. Diese Bindung wird am Ende der Session oder durch Absetzen eines `TRUNCATE`-Befehls für die Tabelle in der Session aufgehoben. Wenn Sie die Session beenden, leert die Datenbank die Tabelle.

Für temporäre Tabellen erstellen Sie in Oracle-Datenbanken eine statische Tabellendefinition. Temporäre Tabellen werden wie permanente Tabellen im Data Dictionary definiert. Jedoch weisen temporäre Tabellen und die zugehörigen Indizes Segmente nicht automatisch zu, wenn sie erstellt werden, sondern wenn die ersten Daten eingefügt werden. Bis Daten in eine Session geladen werden, erscheint die Tabelle leer.

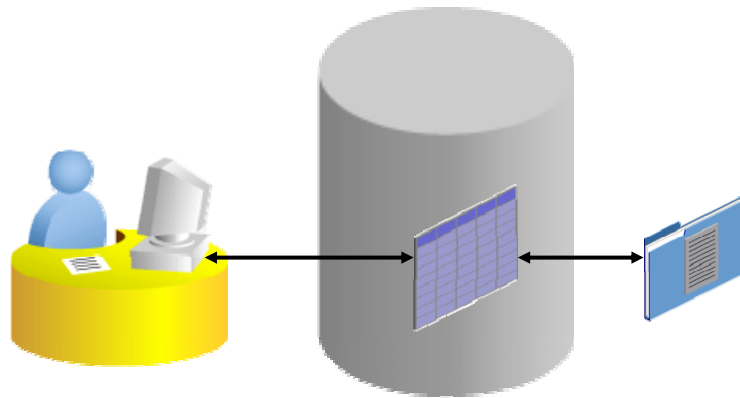
Lektionsagenda

- Constraints verwalten:
 - Constraints hinzufügen und löschen
 - Constraints aktivieren und deaktivieren
 - Constraints verzögern
- Temporäre Tabellen erstellen und verwenden
- Externe Tabellen erstellen und verwenden

ORACLE®

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

Externe Tabellen



ORACLE

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

Externe Tabellen sind schreibgeschützte Tabellen, deren Metadaten in der Datenbank, deren Daten jedoch außerhalb der Datenbank gespeichert sind. Diese externe Tabellendefinition kann man sich als eine View vorstellen, in der SQL-Abfragen für externe Daten ausgeführt werden, ohne dass die externen Daten zuerst in die Datenbank geladen werden müssen. Die Daten der externen Tabelle können direkt und parallel abgefragt und verknüpft werden. Sie müssen nicht in die Datenbank geladen werden. Sie können die Daten einer externen Tabelle mit SQL, PL/SQL und Java abfragen.

Der Hauptunterschied zwischen externen und normalen Tabellen besteht darin, dass extern organisierte Tabellen schreibgeschützt sind. DML-(Data Manipulation Language-)Vorgänge sind nicht möglich, und es können keine Indizes für diese Tabellen erstellt werden. Externe Tabellen können jedoch mit dem Befehl `CREATE TABLE AS SELECT` erstellt und Daten entladen werden.

Der Oracle-Server bietet zwei Hauptzugriffstreiber für den Zugriff auf externe Tabellen. Mit dem Loader-Zugriffstreiber (`ORACLE_LOADER`) werden Daten aus externen Dateien gelesen, deren Format vom SQL*Loader-Utility interpretiert werden kann. Externe Tabellen werden nicht im vollen Umfang durch die Funktionalität von SQL*Loader unterstützt. Mit dem Zugriffstreiber `ORACLE_DATAPUMP` können Sie Daten mithilfe eines plattformunabhängigen Formats sowohl importieren als auch exportieren. Der Zugriffstreiber `ORACLE_DATAPUMP` schreibt Zeilen aus einer `SELECT`-Anweisung, die als Teil der Anweisung `CREATE TABLE ... ORGANIZATION EXTERNAL ... AS SELECT` in eine externe Tabelle geladen werden sollen. Mit `SELECT` können Sie dann Daten aus dieser Datendatei auslesen. Sie können auch eine externe Tabellendefinition für ein anderes System erstellen und diese Datendatei verwenden. Auf diese Weise lassen sich Daten zwischen Oracle-Datenbanken verschieben.

Verzeichnisse für externe Tabellen erstellen

DIRECTORY-Objekt erstellen, das dem Verzeichnis im Dateisystem entspricht, in dem sich die externe Datenquelle befindet

```
CREATE OR REPLACE DIRECTORY emp_dir
AS '/.../emp_dir';

GRANT READ ON DIRECTORY emp_dir TO ora_21;
```

ORACLE

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

Ein Directory-Objekt erstellen Sie mit der Anweisung `CREATE DIRECTORY`. Ein Directory-Objekt gibt einen Alias für ein Verzeichnis im Dateisystem des Servers an, in dem sich eine externe Datenquelle befindet. Mit den Verzeichnisnamen können Sie externe Datenquellen referenzieren und müssen nicht den Pfadnamen des Betriebssystems hart codieren. Dies macht die Dateiverwaltung flexibler.

Um Verzeichnisse zu erstellen, benötigen Sie die Systemberechtigung `CREATE ANY DIRECTORY`. Für erstellte Verzeichnisse erhalten Sie automatisch die Objektberechtigungen `READ` und `WRITE` und können diese Berechtigungen auch anderen Benutzern und Rollen zuweisen. Auch der DBA kann diese Berechtigungen anderen Benutzern und Rollen zuweisen.

Ein Benutzer benötigt `READ`-Berechtigungen für alle in externen Tabellen liegenden Verzeichnisse, auf die zugegriffen werden soll, und `WRITE`-Berechtigungen für die Verzeichnisse der verwendeten Log-, Fehler- und Discard-Dateien.

Darüber hinaus ist eine `WRITE`-Berechtigung erforderlich, wenn Daten mithilfe des externen Tabellen-Frameworks entladen werden.

Oracle stellt zudem den Typ `ORACLE_DATAPUMP` bereit, mit dem Sie Daten entladen (das heißt Daten aus einer Tabelle der Datenbank lesen und sie in eine externe Tabelle einfügen) und sie dann wieder in eine Oracle-Datenbank laden können. Dieser einmalige Vorgang kann durchgeführt werden, wenn die Tabelle erstellt wird. Nachdem die Tabelle erstellt und zum ersten Mal gefüllt wurde, können Sie keine Zeilen mehr aktualisieren, einfügen oder löschen.

Syntax

```
CREATE [OR REPLACE] DIRECTORY AS 'path_name' ;
```

Für die Syntax gilt:

<code>OR REPLACE</code>	Mit <code>OR REPLACE</code> wird das Verzeichnisobjekt erneut erstellt, wenn es bereits vorhanden ist. Sie können die Definition eines vorhandenen Verzeichnisses ändern, ohne zuvor erteilte Datenbankobjektberechtigungen löschen, erneut erstellen oder erneut erteilen zu müssen. Benutzer können mit den zuvor erteilten Berechtigungen weiterhin auf ein neu definiertes Verzeichnis zugreifen. Die Berechtigungen müssen nicht neu vergeben werden.
<code>directory</code>	Der Name des zu erstellenden Verzeichnisobjekts. Die Maximallänge für den Verzeichnisnamen beträgt 30 Byte. Directory-Objekte können nicht durch einen Schemanamen spezifiziert werden.
<code>'path_name'</code>	Der vollständige Pfadname des Betriebssystemverzeichnisses, auf das zugegriffen werden soll. Beim Pfadnamen wird zwischen Groß- und Kleinschreibung unterschieden.

Externe Tabellen erstellen

```
CREATE TABLE <table_name>
  ( <col_name> <datatype>, ... )
  ORGANIZATION EXTERNAL
    (TYPE <access_driver_type>
     DEFAULT DIRECTORY <directory_name>
     ACCESS PARAMETERS
       (... ) )
     LOCATION ('<location_specifier>')
  REJECT LIMIT [0 | <number> | UNLIMITED];
```

ORACLE

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

Externe Tabellen werden mit der Klausel `ORGANIZATION EXTERNAL` der Anweisung `CREATE TABLE` erstellt. Dabei wird eigentlich keine Tabelle erstellt, sondern Metadaten im Data Dictionary, mit denen Sie auf externe Daten zugreifen können. Mit der Klausel `ORGANIZATION` können Sie angeben, in welcher Reihenfolge die Datenzeilen der Tabelle gespeichert werden. Die Option `EXTERNAL` in der Klausel `ORGANIZATION` gibt an, dass es sich um eine schreibgeschützte Tabelle außerhalb der Datenbank handelt. Die externen Dateien müssen bereits außerhalb der Datenbank vorhanden sein.

`TYPE <access_driver_type>` gibt den Zugriffstreiber für die externe Tabelle an. Der Zugriffstreiber ist die API (Application Programming Interface), die die externen Daten für die Datenbank interpretiert. Wenn Sie `TYPE` nicht angeben, verwendet Oracle den Standardzugriffstreiber `ORACLE_LOADER`. Die andere Option ist `ORACLE_DATAPUMP`.

Mit der Klausel `DEFAULT DIRECTORY` geben Sie ein oder mehrere Directory-Objekte der Oracle-Datenbank an, die Verzeichnissen im Dateisystem entsprechen, in denen sich die externen Datenquellen befinden könnten.

Mit der optionalen Klausel `ACCESS PARAMETERS` können Sie den Parametern des spezifischen Zugriffstreibers für die betreffende externe Tabelle Werte zuweisen.

Geben Sie mit der Klausel `LOCATION` einen externen Positionsanzeiger für jede externe Datenquelle an. Normalerweise ist `<location_specifier>` eine Datei. Dies ist jedoch nicht zwingend notwendig.

Mit der Klausel `REJECT LIMIT` können Sie festlegen, wie viele Konvertierungsfehler während der Abfrage externer Daten auftreten dürfen, bevor ein Oracle-Fehler zurückgegeben und die Abfrage abgebrochen wird. Der Standardwert ist 0.

Die Syntax zur Verwendung des Zugriffstreibers `ORACLE_DATAPUMP` lautet wie folgt:

```
CREATE TABLE extract_emps
  ORGANIZATION EXTERNAL (TYPE ORACLE_DATAPUMP
                        DEFAULT DIRECTORY ...
                        ACCESS PARAMETERS (... )
                        LOCATION (... )
                        PARALLEL 4
                        REJECT LIMIT UNLIMITED
AS
SELECT * FROM ...;
```

Externe Tabellen mit ORACLE_LOADER erstellen

```
CREATE TABLE oldemp (fname char(25), lname  
CHAR(25))  
ORGANIZATION EXTERNAL  
(TYPE ORACLE_LOADER  
DEFAULT DIRECTORY emp_dir  
ACCESS PARAMETERS  
(RECORDS DELIMITED BY NEWLINE  
FIELDS(fname POSITION ( 1:20) CHAR,  
lname POSITION (22:41) CHAR))  
LOCATION ('emp.dat'));
```

```
table OLDEMP created.
```

ORACLE

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

Eine Flat File enthält Datensätze in folgendem Format:

```
10,jones,11-Dec-1934  
20,smith,12-Jun-1972
```

Die Datensätze werden durch Zeilenvorschubzeichen getrennt. Der Name der Datei lautet
/emp_dir/emp.dat.

Um diese Datei zu konvertieren und als Datenquelle für eine externe Tabelle zu verwenden, deren Metadaten in der Datenbank gespeichert sind, gehen Sie wie folgt vor:

1. Erstellen Sie ein Verzeichnisobjekt emp_dir wie folgt:

```
CREATE DIRECTORY emp_dir AS '/emp_dir' ;
```
2. Führen Sie den auf der Folie angegebenen Befehl CREATE TABLE aus.

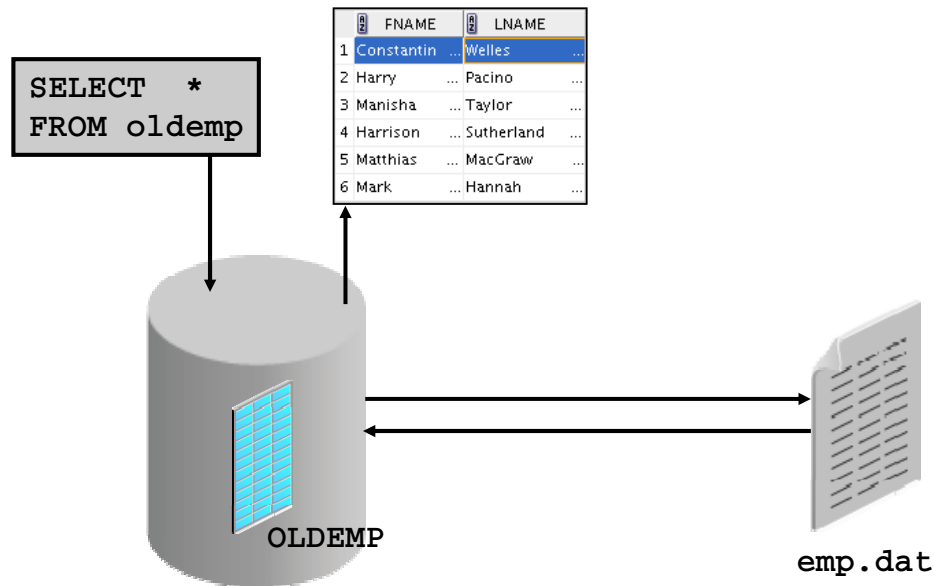
Das Beispiel auf der Folie zeigt die Tabellenspezifikation zum Erstellen einer externen Tabelle für die Datei:

```
/emp_dir/emp.dat
```

In diesem Beispiel wird die Angabe `TYPE` nur festgelegt, um ihre Verwendung zu demonstrieren. Erfolgt keine Angabe, ist `ORACLE_LOADER` der Standardzugriffstreiber. Die Option `ACCESS PARAMETERS` enthält Werte für die Parameter des jeweiligen Zugriffstreibers und wird vom Zugriffstreiber, nicht vom Oracle-Server interpretiert.

Wenn der Befehl `CREATE TABLE` erfolgreich ausgeführt wurde, kann die externe Tabelle `OLDEMP` wie eine relationale Tabelle beschrieben und abgefragt werden.

Externe Tabellen abfragen



ORACLE

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

Externe Tabellen beschreiben keine in der Datenbank gespeicherten Daten. Sie beschreiben auch nicht, auf welche Weise Daten in der externen Quelle gespeichert werden. Vielmehr beschreiben sie, wie die externe Tabellenebene die Daten für den Server darstellen muss. Der Zugriffstreiber und die externe Tabellenebene müssen die notwendigen Transformationen der Daten in der Datendatei durchführen, damit sie der externen Tabellendefinition entsprechen.

Wenn der Datenbankserver auf Daten in einer externen Quelle zugreift, ruft er den entsprechenden Zugriffstreiber auf, um die Daten in einer vom Datenbankserver erwarteten Form aus einer externen Quelle abzurufen.

Die Beschreibung der Daten in der Datenquelle ist von der Definition der externen Tabelle getrennt. Die Quelldatei kann mehr oder weniger Felder enthalten, als die Tabelle Spalten aufweist. Auch die Datentypen von Feldern in der Datenquelle können sich von den Tabellenspalten unterscheiden. Der Zugriffstreiber stellt sicher, dass die Daten aus der Datenquelle so verarbeitet werden, dass sie der Definition der externen Tabelle entsprechen.

Externe Tabellen mit ORACLE_DATAPUMP erstellen – Beispiel

```
CREATE TABLE emp_ext
(employee_id, first_name, last_name)
ORGANIZATION EXTERNAL
(
  TYPE ORACLE_DATAPUMP
  DEFAULT DIRECTORY emp_dir
  LOCATION
    ('emp1.exp', 'emp2.exp')
)
PARALLEL
AS
SELECT employee_id, first_name, last_name
FROM   employees;
```

table EMP_EXT created.

ORACLE

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

Mit dem Zugriffstreiber `ORACLE_DATAPUMP` führen Sie die Entlade- und Ladevorgänge im Zusammenhang mit externen Tabellen durch.

Hinweis: Im Kontext von externen Tabellen beinhaltet der Ladevorgang, dass Daten aus einer externen Tabelle gelesen und in eine Tabelle in der Datenbank geladen werden. Mit dem Entladen von Daten ist der Vorgang gemeint, bei dem die Daten aus einer Tabelle gelesen und in eine externe Tabelle eingefügt werden.

Das Beispiel auf der Folie veranschaulicht die Tabellenspezifikation zum Erstellen einer externen Tabelle mithilfe des Zugriffstreibers `ORACLE_DATAPUMP`. Zwei Dateien werden daraufhin mit den Daten befüllt: `emp1.exp` und `emp2.exp`.

Um Daten, die aus der Tabelle `EMPLOYEES` gelesen werden, in eine externe Tabelle zu füllen, gehen Sie wie folgt vor:

1. Erstellen Sie ein Verzeichnisobjekt `emp_dir` wie folgt:

```
CREATE DIRECTORY emp_dir AS '/emp_dir' ;
```

2. Führen Sie den auf der Folie angegebenen Befehl `CREATE TABLE` aus.

Hinweis: Es wird das gleiche Verzeichnis `emp_dir` wie im vorherigen Beispiel mit `ORACLE_LOADER` erstellt.

Um die externe Tabelle abzufragen, führen Sie die folgende Anweisung aus:

```
SELECT * FROM emp_ext;
```

Quiz

Ein `FOREIGN KEY`-Constraint setzt die folgende Aktion durch:
Wenn Daten im übergeordneten Schlüssel gelöscht werden,
werden gleichzeitig alle Zeilen in der untergeordneten Tabelle
gelöscht, die von den gelöschten Werten des übergeordneten
Schlüssels abhängen.

- 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:

- Constraints verwalten
- Temporäre Tabellen erstellen und verwenden
- Externe Tabellen erstellen und verwenden

ORACLE

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

In dieser Lektion haben Sie gelernt, wie Sie folgende Aufgaben zur Verwaltung von Schema-objekten ausführen:

- Tabellen mit dem Befehl `ALTER TABLE` ändern, um Spalten oder Constraints hinzuzufügen oder zu ändern
- Mit der Klausel `ORGANIZATION EXTERNAL` der Anweisung `CREATE TABLE` eine externe Tabelle erstellen. Eine externe Tabelle ist eine schreibgeschützte Tabelle, deren Metadaten in der Datenbank, deren Daten jedoch außerhalb der Datenbank gespeichert sind.
- Mit externen Tabellen Daten abfragen, ohne sie zuerst in die Datenbank zu laden

Übungen zu Lektion 5 – Überblick

Diese Übungen behandeln folgende Themen:

- Constraints hinzufügen und löschen
- Constraints verzögern
- Externe Tabellen erstellen

ORACLE

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

In dieser Übung verwenden Sie den Befehl `ALTER TABLE`, um Constraints hinzuzufügen, zu löschen oder zu verzögern. Sie erstellen externe Tabellen.

6

Daten mithilfe von Unterabfragen abrufen

ORACLE®

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

Ziele

Nach Ablauf dieses Lektion haben Sie folgende Ziele erreicht:

- Multiple-Column-Unterabfragen erstellen
- Skalare Unterabfragen in SQL
- Aufgabenstellungen mithilfe von korrelierten Unterabfragen lösen
- Operatoren `EXISTS` und `NOT EXISTS`
- Klausel `WITH`

ORACLE

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

In dieser Lektion lernen Sie, wie Sie in der Klausel `FROM` der Anweisung `SELECT` Multiple-Column-Unterabfragen und Unterabfragen erstellen. Sie lernen außerdem, wie Sie mit skalaren Unterabfragen, korrelierten Unterabfragen und der Klausel `WITH` Aufgabenstellungen lösen können.

Lektionsagenda

- **Daten mit Unterabfragen als Quelle abrufen**
- Multiple-Column-Unterabfragen erstellen
- Skalare Unterabfragen in SQL
- Aufgabenstellungen mithilfe von korrelierten Unterabfragen lösen
- Operatoren EXISTS und NOT EXISTS
- Klausel WITH

ORACLE®

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

Daten mit Unterabfragen als Quelle abrufen

```
SELECT department_name, city
FROM departments
NATURAL JOIN (SELECT l.location_id, l.city, l.country_id
              FROM locations l
              JOIN countries c
              ON (l.country_id = c.country_id)
              JOIN regions
              USING (region_id)
              WHERE region_name = 'Europe');
```

	DEPARTMENT_NAME	CITY
1	Human Resources	London
2	Sales	Oxford
3	Public Relations	Munich

ORACLE

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

Unterabfragen werden in der Klausel `FROM` von `SELECT`-Anweisungen auf ähnliche Weise verwendet wie Views. Unterabfragen in der Klausel `FROM` von `SELECT`-Anweisungen werden auch als *Inline-Views* bezeichnet. Sie definieren eine Datenquelle genau und ausschließlich für die betreffende `SELECT`-Anweisung. Wie bei einer Datenbank-View kann die `SELECT`-Anweisung in der Unterabfrage entsprechend Ihren Erfordernissen einfach oder komplex sein.

Wenn Sie eine Datenbank-View erstellen, wird die zugehörige `SELECT`-Anweisung im Data Dictionary gespeichert. Wenn Sie nicht über die notwendigen Berechtigungen zur Erstellung von Datenbank-Views verfügen oder die Eignung einer `SELECT`-Anweisung für eine View testen möchten, können Sie eine Inline-View verwenden.

In Inline-Views steht Ihnen der gesamte Code, den Sie zur Unterstützung der Abfrage benötigen, an einer Stelle zur Verfügung, sodass Sie die komplexe Erstellung einer separaten Datenbank-View umgehen können. Das Beispiel auf der Folie zeigt, wie Sie mithilfe einer Inline-View den Abteilungsnamen und die Stadt in Europa abfragen. Eine Unterabfrage in der Klausel `FROM` ruft den Ort, die Stadt und das Land ab, indem drei verschiedene Tabellen verknüpft werden. Die Ausgabe der inneren Abfrage gilt als Tabelle für die äußere Abfrage. Die innere Abfrage entspricht der Abfrage einer Datenbank-View, hat jedoch keinen physischen Namen.

Sie können dieselbe Ausgabe wie im Beispiel auf der Folie anzeigen, indem Sie die beiden folgenden Schritte ausführen:

1. Erstellen Sie eine Datenbank-View:

```
CREATE OR REPLACE VIEW european_cities
AS
SELECT l.location_id, l.city, l.country_id
FROM   locations l
JOIN    countries c
ON(l.country_id = c.country_id)
JOIN regions USING(region_id)
WHERE region_name = 'Europe';
```

2. Verbinden Sie die View EUROPEAN_CITIES mit der Tabelle DEPARTMENTS:

```
SELECT department_name, city
FROM   departments
NATURAL JOIN european_cities;
```

Hinweis: In der Lektion "*Views erstellen*" haben Sie gelernt, wie man Datenbank-Views erstellt.

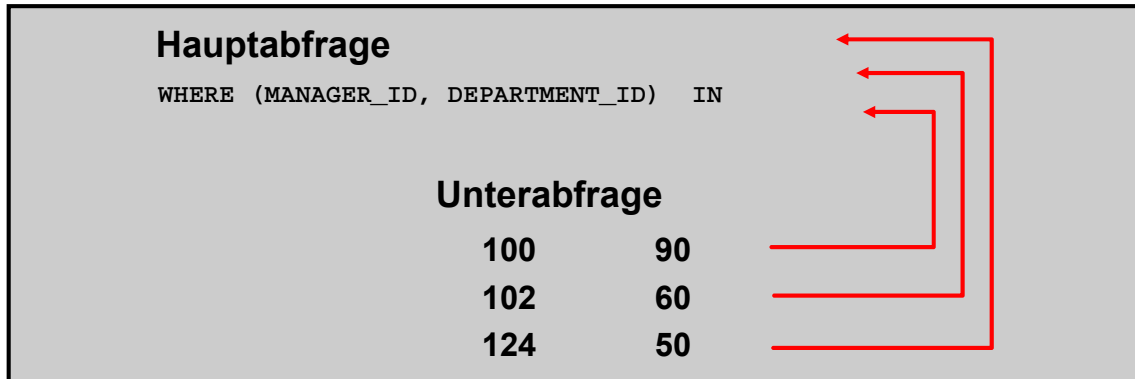
Lektionsagenda

- Daten mithilfe von Unterabfragen als Quelle abrufen
- **Multiple-Column-Unterabfragen erstellen**
- Skalare Unterabfragen in SQL
- Aufgabenstellungen mithilfe von korrelierten Unterabfragen lösen
- Operatoren EXISTS und NOT EXISTS
- Klausel WITH

ORACLE®

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

Multiple-Column-Unterabfragen



Jede Zeile der Hauptabfrage wird mit Werten aus einer Multiple-Row- und Multiple-Column-Unterabfrage verglichen.

ORACLE

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

Bisher haben Sie Single-Row- und Multiple-Row-Unterabfragen erstellt. Bei diesen Abfragen wird von der inneren `SELECT`-Anweisung nur eine Spalte zurückgegeben, die verwendet wird, um in der übergeordneten `SELECT`-Anweisung den Ausdruck auszuwerten. 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 `WHERE`-Klausel kombinieren.

Syntax

```
SELECT      column, column, ...
FROM table
WHERE (column, column, ...) IN
      (SELECT column, column, ...
        FROM table
        WHERE condition);
```

Die Grafik auf der Folie zeigt, dass die Werte der Spalten `MANAGER_ID` und `DEPARTMENT_ID` aus der Hauptabfrage mit den von der Unterabfrage abgerufenen Werten `MANAGER_ID` und `DEPARTMENT_ID` verglichen werden. Da die Anzahl der verglichenen Spalten größer als 1 ist, gilt das Beispiel als Multiple-Column-Unterabfrage.

Hinweis: Bevor Sie die Beispiele auf den nächsten Folien ausführen, müssen Sie die Tabelle `empl_demo` erstellen und mit Daten füllen. Verwenden Sie hierzu die Datei `lab_06_insert_empdata.sql`.

Spaltenvergleiche

Multiple-Column-Vergleiche mit Unterabfragen können Spalten wie folgt verglichen werden:

- Paarweise
- Nicht paarweise

ORACLE

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

Paarweise und nicht paarweise Vergleiche

Bei Multiple-Column-Vergleichen, die Unterabfragen enthalten, können Spalten paarweise oder nicht paarweise verglichen werden. Für die Aufgabenstellung "Details der Mitarbeiter anzeigen, die in der gleichen Abteilung arbeiten und demselben Manager unterstehen wie Daniel" erhalten Sie das korrekte Ergebnis mit folgender Anweisung:

```
SELECT first_name, last_name, manager_id, department_id
FROM empl_demo
WHERE manager_id IN (SELECT manager_id
                     FROM empl_demo
                     WHERE first_name = 'Daniel')
AND department_id IN (SELECT department_id
                     FROM empl_demo
                     WHERE first_name = 'Daniel');
```

Die Tabelle `EMPL_DEMO` enthält nur einen "Daniel" (Daniel Faviat, der dem Mitarbeiter 108 untersteht und in Abteilung 100 arbeitet). Wenn die Unterabfragen jedoch mehrere Zeilen zurückgeben, ist das Ergebnis unter Umständen nicht korrekt. Beispiel: Wenn Sie die gleiche Abfrage erneut ausführen, jedoch "Daniel" durch "John" ersetzen, erhalten Sie ein falsches Ergebnis, weil es wichtig ist, in welcher Kombination `department_id` und `manager_id` vorkommen. Das korrekte Ergebnis für diese Abfrage erhalten Sie nur mit einem paarweisen Vergleich.

Unterabfragen mit paarweisen Vergleichen

Details der Mitarbeiter anzeigen, die demselben Manager unterstehen und in der gleichen Abteilung arbeiten wie die Mitarbeiter mit `EMPLOYEE_ID` 199 oder 174

```
SELECT employee_id, manager_id, department_id
FROM employees
WHERE (manager_id, department_id) IN
      (SELECT manager_id, department_id
       FROM employees
       WHERE employee_id IN (174, 199))
AND employee_id NOT IN (174,199);
```

	EMPLOYEE_ID	MANAGER_ID	DEPARTMENT_ID
1	141	124	50
2	142	124	50
3	143	124	50
4	144	124	50

...

ORACLE

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

Das Beispiel zeigt einen paarweisen Vergleich der Spalten. Hierbei werden für jede Zeile der Tabelle `EMPLOYEES` die Werte aus den Spalten `MANAGER_ID` und `DEPARTMENT_ID` mit den Werten aus den Spalten `MANAGER_ID` und `DEPARTMENT_ID` der Mitarbeiter mit der `EMPLOYEE_ID` 199 oder 174 verglichen.

Zuerst wird die Unterabfrage ausgeführt, die für die Mitarbeiter mit der `EMPLOYEE_ID` 199 oder 174 die Werte aus den Spalten `MANAGER_ID` und `DEPARTMENT_ID` abrufen. Diese Werte werden mit den Spalten `MANAGER_ID` und `DEPARTMENT_ID` der einzelnen Zeilen der Tabelle `EMPLOYEES` verglichen. Wenn die Werte übereinstimmen, wird die Zeile angezeigt. In der Ausgabe werden die Datensätze der Mitarbeiter mit `EMPLOYEE_ID` 199 oder 174 nicht angezeigt. Unterhalb der Abfrage wird auf der Folie die Ausgabe angezeigt.

Unterabfragen mit nicht paarweise durchgeführten Vergleichen

Details der Mitarbeiter anzeigen, die demselben Manager unterstehen wie die Mitarbeiter mit der `EMPLOYEE_ID` 174 oder 141 und in der gleichen Abteilung arbeiten wie die Mitarbeiter mit der `EMPLOYEE_ID` 174 oder 141

```
SELECT employee_id, manager_id, department_id
FROM employees
WHERE manager_id IN
    (SELECT manager_id
     FROM employees
     WHERE employee_id IN (174,141))
AND department_id IN
    (SELECT department_id
     FROM employees
     WHERE employee_id IN (174,141))
AND employee_id NOT IN(174,141);
```

	EMPLOYEE_ID	MANAGER_ID	DEPARTMENT_ID
1	142	124	50
2	143	124	50

...

ORACLE

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

Das Beispiel zeigt einen nicht paarweise durchgeführten Vergleich der Spalten. Aberufen werden `EMPLOYEE_ID`, `MANAGER_ID` und `DEPARTMENT_ID` aller Mitarbeiter, deren `MANAGER_ID` mit einer Manager-ID der Mitarbeiter 174 oder 141 und deren `DEPARTMENT_ID` mit einer Abteilungs-ID der Mitarbeiter 174 oder 141 übereinstimmt.

Zuerst wird die Unterabfrage ausgeführt, die für die Mitarbeiter mit der `EMPLOYEE_ID` 174 oder 141 die Werte aus der Spalte `MANAGER_ID` abrufen. Die zweite Unterabfrage ruft entsprechend für die Mitarbeiter mit der `EMPLOYEE_ID` 174 oder 141 die Werte aus der Spalte `DEPARTMENT_ID` ab. Die abgerufenen Werte der Spalten `MANAGER_ID` und `DEPARTMENT_ID` werden dann zeilenweise mit den Spalten `MANAGER_ID` und `DEPARTMENT_ID` der Tabelle `EMPLOYEES` verglichen. Die Datensätze werden angezeigt, bei denen die Spalte `MANAGER_ID` in der Tabelle `EMPLOYEES` mit einem von der inneren Unterabfrage abgerufenen Wert von `MANAGER_ID` übereinstimmt und gleichzeitig die Spalte `DEPARTMENT_ID` in der Tabelle `EMPLOYEES` mit einem von der zweiten Unterabfrage aus der Spalte `DEPARTMENT_ID` abgerufenen Wert übereinstimmt.

Lektionsagenda

- Daten mithilfe von Unterabfragen als Quelle abrufen
- Multiple-Column-Unterabfragen erstellen
- **Skalare Unterabfragen in SQL**
- Aufgabenstellungen mithilfe von korrelierten Unterabfragen lösen
- Operatoren EXISTS und NOT EXISTS
- Klausel WITH

ORACLE®

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

Skalare Unterabfrageausdrücke

- Ein skalarer Unterabfrageausdruck ist eine Unterabfrage, die genau einen Spaltenwert aus einer Zeile zurückgibt.
- Skalare Unterabfragen können wie folgt verwendet werden:
 - Im Bedingungs- und Ausdrucksteil von `DECODE` und `CASE`
 - In allen Klauseln von `SELECT`, mit Ausnahme von `GROUP BY`
 - In den Klauseln `SET` und `WHERE` einer `UPDATE`-Anweisung

ORACLE

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

Eine Unterabfrage, die genau einen Spaltenwert aus einer Zeile zurückgibt, wird auch als skalare Unterabfrage bezeichnet. Multiple-Column-Unterabfragen, in denen zwei oder mehr Spalten mit einer zusammengesetzten `WHERE`-Klausel und logischen Operatoren verglichen werden sollen, gelten nicht als skalare Unterabfragen.

Der Wert des skalaren Unterabfrageausdrucks ist der Wert des Elements der `SELECT`-Liste der Unterabfrage. Wenn die Unterabfrage 0 Zeilen zurückgibt, ist der Wert des skalaren Unterabfrageausdrucks `NULL`. Gibt die Unterabfrage mehrere Zeilen zurück, meldet der Oracle-Server einen Fehler. Skalare Unterabfragen in `SELECT`-Anweisungen werden vom Oracle-Server schon immer unterstützt. Sie können skalare Unterabfragen wie folgt verwenden:

- Im Bedingungs- und Ausdrucksteil von `DECODE` und `CASE`
- In allen Klauseln von `SELECT`, mit Ausnahme von `GROUP BY`
- In den Klauseln `SET` und `WHERE` einer `UPDATE`-Anweisung

An folgenden Stellen sind skalare Unterabfragen jedoch nicht als Ausdrücke zulässig:

- In der Klausel `RETURNING` von DML-(Data Manipulation Language-)Anweisungen
- Als Basis eines funktionsbasierten Index
- In `GROUP BY`-Klauseln und `CHECK`-Constraints
- In `CONNECT BY`-Klauseln
- In Anweisungen, die nicht mit Abfragen verbunden sind, wie `CREATE PROFILE`

Skalare Unterabfragen – Beispiele

- Skalare Unterabfragen in CASE-Ausdrücken:

```
SELECT employee_id, last_name,  
       (CASE  
         WHEN department_id = 20  
           (SELECT department_id  
            FROM departments  
            WHERE location_id = 1800)  
         THEN 'Canada' ELSE 'USA' END) location  
FROM   employees;
```

- Skalare Unterabfragen in der SELECT-Anweisung:

```
select department_id, department_name,  
       (select count(*)  
        from employees e  
        where e.department_id = d.department_id) as emp_count  
from   departments d;
```

ORACLE

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

Das erste Beispiel auf der Folie zeigt, dass skalare Unterabfragen in CASE-Ausdrücken verwendet werden können. Die innere Abfrage gibt den Wert 20 zurück. Dies ist die Nummer der Abteilung mit der Standortnummer 1800. Der Ausdruck CASE in der äußeren Abfrage verwendet das Ergebnis der inneren Abfrage. Je nachdem, ob die Abteilungsnummer des von der äußeren Abfrage abgerufenen Datensatzes "20" lautet, werden die Personalnummer, der Nachname und der Wert "Canada" oder "USA" angezeigt.

Das zweite Beispiel auf der Folie zeigt, dass skalare Unterabfragen in SELECT-Anweisungen verwendet werden können.

Lektionsagenda

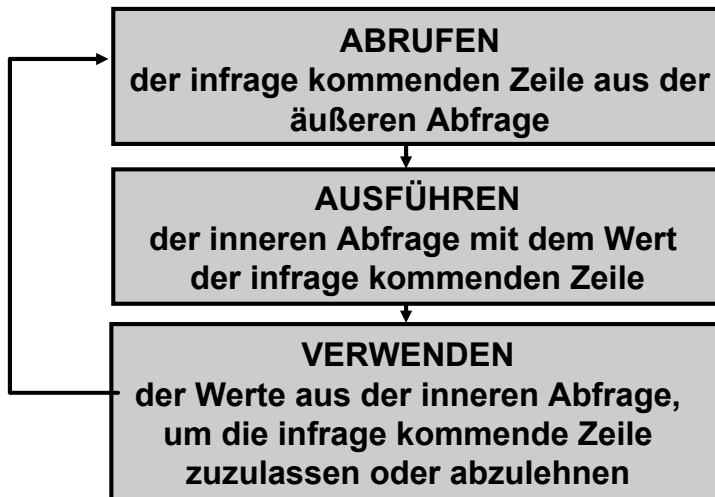
- Daten mithilfe von Unterabfragen als Quelle abrufen
- Multiple-Column-Unterabfragen erstellen
- Skalare Unterabfragen in SQL
- **Aufgabenstellungen mithilfe von korrelierten Unterabfragen lösen**
- Operatoren EXISTS und NOT EXISTS
- Klausel WITH

ORACLE®

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

Korrelierte Unterabfragen

Korrelierte Unterabfragen werden zur zeilenweisen Verarbeitung verwendet. Jede Unterabfrage wird für jede Zeile aus der äußeren Abfrage einmal ausgeführt.



ORACLE

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

Der Oracle-Server führt eine korrelierte Unterabfrage aus, wenn die Unterabfrage eine Spalte aus einer Tabelle referenziert, auf die in der übergeordneten Anweisung verwiesen wird. Eine korrelierte Unterabfrage wird für jede Zeile, die von der übergeordneten Anweisung verarbeitet wird, einmal ausgewertet. Die übergeordnete Anweisung kann eine `SELECT`-, `UPDATE`- oder `DELETE`-Anweisung sein.

Verschachtelte Unterabfragen und korrelierte Unterabfragen – Vergleich

Bei einer normalen verschachtelten Unterabfrage wird die innere `SELECT`-Anweisung zuerst und nur einmal ausgeführt. Sie gibt Werte zurück, die von der Hauptabfrage verwendet werden. Eine korrelierte Unterabfrage wird hingegen für jede infrage kommende Zeile, die von der äußeren Abfrage abgerufen wird, einmal ausgeführt. Die innere Abfrage wird also von der äußeren Abfrage gesteuert.

Verschachtelte Unterabfragen ausführen

- Die innere Abfrage wird zuerst ausgeführt und gibt einen Wert zurück.
- Die äußere Abfrage wird einmal ausgeführt und verwendet dabei den Wert aus der inneren Abfrage.

Korrelierte Unterabfragen ausführen

- Eine infrage kommende Zeile (von der äußeren Abfrage abgerufen) wird abgerufen.
- Die innere Abfrage wird mit dem Wert der infrage kommenden Zeile ausgeführt.
- Die aus der inneren Abfrage resultierenden Werte werden verwendet, um die infrage kommende Zeile zuzulassen oder abzulehnen.
- Diese Schritte werden wiederholt, bis keine infrage kommenden Zeilen mehr vorhanden sind.

Korrelierte Unterabfragen

Die Unterabfrage referenziert eine Spalte aus einer Tabelle in der übergeordneten Abfrage.

```
SELECT column1, column2, ...
FROM   table1 Outer_table
WHERE  column1 operator
              (SELECT column1, column2
                FROM   table2
                WHERE  expr1 =
                      Outer_table.expr2);
```

ORACLE®

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

Eine korrelierte Unterabfrage ist eine Möglichkeit, jede Zeile einer Tabelle zu lesen und die Werte der einzelnen Zeilen mit zugehörigen Daten zu vergleichen. Sie wird immer dann verwendet, wenn die Unterabfrage für jede für die Hauptabfrage infrage kommende Zeile ein anderes Ergebnis oder eine andere Ergebnismenge zurückgeben muss. Korrelierte Unterabfragen werden also zur Beantwortung einer mehrteiligen Frage verwendet, deren Antwort vom Wert der einzelnen von der übergeordneten Anweisung verarbeiteten Zeilen abhängt.

Der Oracle-Server führt eine korrelierte Unterabfrage aus, wenn die Unterabfrage eine Spalte aus einer Tabelle in der übergeordneten Abfrage referenziert.

Hinweis: In korrelierten Unterabfragen können die Operatoren `ANY` und `ALL` verwendet werden.

Korrelierte Unterabfragen – 1. Beispiel

Alle Mitarbeiter suchen, deren Gehalt über dem in der Abteilung üblichen Durchschnittsgehalt liegt

```
SELECT last_name, salary, department_id
FROM   employees outer_table
WHERE  salary >
      (SELECT AVG(salary)
       FROM   employees inner_table
       WHERE  inner_table.department_id =
              outer_table.department_id);
```

	LAST_NAME	SALARY	DEPARTMENT_ID
1	King	24000	90
2	Hunold	9000	60
3	Ernst	6000	60
4	Greenberg	12008	100
5	Faviet	9000	100
6	Raphaely	11000	30

....

Bei jeder Verarbeitung einer Zeile von der äußeren Abfrage wird die innere Abfrage ausgewertet.

ORACLE

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

Im Beispiel auf der Folie wird ermittelt, welche Mitarbeiter ein höheres Gehalt als das in ihrer Abteilung übliche Durchschnittsgehalt beziehen. In diesem Fall berechnet die korrelierte Unterabfrage das Durchschnittsgehalt für jede Abteilung.

Da die Tabelle `EMPLOYEES` in der Klausel `FROM` sowohl in der äußeren als auch in der inneren Abfrage verwendet wird, erhält `EMPLOYEES` zur besseren Lesbarkeit in der äußeren `SELECT`-Anweisung einen Alias. Durch den Alias wird die Lesbarkeit der gesamten Anweisung `SELECT` verbessert. Ohne den Alias würde die Abfrage nicht ordnungsgemäß funktionieren, da die innere Anweisung die innere Tabellenspalte nicht von der äußeren Tabellenspalte unterscheiden könnte.

Die korrelierte Unterabfrage führt für jede Zeile der Tabelle `EMPLOYEES` folgende Schritte aus:

1. Die `department_id` der Zeile wird bestimmt.
2. Anhand der `department_id` wird die übergeordnete Abfrage ausgewertet.
3. Ist das Gehalt in dieser Zeile höher als das Durchschnittsgehalt der Abteilungen dieser Zeile, wird die Zeile zurückgegeben.

Die Unterabfrage wird für jede Zeile in der Tabelle `EMPLOYEES` einmal ausgewertet.

Korrelierte Unterabfragen – 2. Beispiel

Details des Mitarbeiters mit dem höchsten Gehalt pro Abteilung anzeigen

```
SELECT department_id, employee_id, salary
FROM EMPLOYEES e
WHERE 1 =
    (SELECT COUNT(DISTINCT salary)
     FROM EMPLOYEES
     WHERE e.department_id = department_id
     AND e.salary <= salary)
```

	DEPARTMENT_ID	EMPLOYEE_ID	SALARY
1	90	100	24000
2	60	103	9000
3	100	108	12008
4	30	114	11000

....

ORACLE

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

Das Beispiel auf der Folie zeigt für jede Abteilung die Details des Mitarbeiters mit dem höchsten Gehalt an. Der Oracle-Server wertet korrelierte Unterabfragen wie folgt aus:

1. Er wählt eine Zeile aus der in der äußeren Abfrage angegebenen Tabelle. Dies ist die aktuell infrage kommende Zeile.
2. Er speichert den Wert aus der Spalte, die in der Unterabfrage dieser infrage kommenden Zeile referenziert ist. (Im Beispiel auf der Folie ist `e.salary` die in der Unterabfrage referenzierte Spalte.)
3. Er führt die Unterabfrage mit der Bedingung aus, die den Wert der infrage kommenden Zeile aus der äußeren Abfrage referenziert. (Im Beispiel auf der Folie wird die Gruppenfunktion `COUNT(DISTINCT salary)` auf Basis des im 2. Schritt ermittelten Wertes der Spalte `E.SALARY` ausgewertet.)
4. Er wertet die Klausel `WHERE` der äußeren Abfrage auf Basis der Ergebnisse der im 3. Schritt ausgeführten Unterabfrage aus. Dadurch wird bestimmt, ob die infrage kommende Zeile für die Ausgabe gewählt wird. (Im Beispiel wird die von der Unterabfrage ermittelte Anzahl der Jobwechsel eines Mitarbeiters mit dem Wert 2 in der Klausel `WHERE` der äußeren Abfrage verglichen. Ist die Bedingung erfüllt, wird der Mitarbeiterdatensatz angezeigt.)
5. Die Prozedur wird für die nächste infrage kommende Zeile der Tabelle wiederholt, bis alle Tabellenzeilen verarbeitet wurden.

Die Korrelation wird dadurch hergestellt, dass ein Element der äußeren Abfrage in der Unterabfrage verwendet wird. In diesem Beispiel wird `EMPLOYEE_ID` aus der Tabelle in der Unterabfrage mit `EMPLOYEE_ID` aus der Tabelle in der äußeren Abfrage verglichen.

Lektionsagenda

- Daten mithilfe von Unterabfragen als Quelle abrufen
- Multiple-Column-Unterabfragen erstellen
- Skalare Unterabfragen in SQL
- Aufgabenstellungen mithilfe von korrelierten Unterabfragen lösen
- Operatoren EXISTS und NOT EXISTS
- Klausel WITH

ORACLE®

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

Operator EXISTS

- Der Operator `EXISTS` prüft, ob in der Ergebnismenge der Unterabfrage Zeilen enthalten sind.
- Falls ja:
 - Die Suche in der inneren Abfrage wird nicht fortgesetzt.
 - Die Bedingung wird als `TRUE` gekennzeichnet.
- Falls nein:
 - Die Bedingung wird als `FALSE` gekennzeichnet.
 - Die Suche wird in der inneren Abfrage fortgesetzt.

ORACLE

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

In verschachtelten `SELECT`-Anweisungen sind alle logischen Operatoren gültig. Zusätzlich können Sie den Operator `EXISTS` verwenden. Dieser Operator wird häufig verwendet, um in korrelierten Unterabfragen zu prüfen, ob ein von der äußeren Abfrage abgerufener Wert in der Ergebnismenge der von der inneren Abfrage abgerufenen Werte enthalten ist. Wenn die Unterabfrage mindestens eine Zeile zurückgibt, gibt der Operator `TRUE` zurück. Ist der Wert nicht vorhanden, gibt der Operator `FALSE` zurück. Analog dazu prüft der Operator `NOT EXISTS`, ob ein von der äußeren Abfrage abgerufener Wert nicht Teil der von der inneren Abfrage abgerufenen Ergebnismenge ist.

Operator EXISTS – Beispiel

```
SELECT employee_id, last_name, job_id, department_id
FROM   employees outer
WHERE  EXISTS ( SELECT NULL
                  FROM   employees
                  WHERE  manager_id =
                        outer.employee_id);
```

	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	108	Greenberg	FI_MGR	100
6	114	Raphaely	PU_MAN	30
7	120	Weiss	ST_MAN	50
8	121	Fripp	ST_MAN	50

...

ORACLE

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

Der Operator `EXISTS` stellt sicher, dass der Suchvorgang in der inneren Abfrage nicht fortgesetzt wird, wenn mindestens eine Übereinstimmung für die Manager- und Personalnummer gefunden wird. Hierzu wird folgende Bedingung verwendet:

```
WHERE manager_id = outer.employee_id
```

Die innere `SELECT`-Anweisung muss keinen bestimmten Wert zurückgeben, sodass eine Konstante gewählt werden kann.

Alle Abteilungen ermitteln, die keine Mitarbeiter enthalten

```
SELECT department_id, department_name
FROM departments d
WHERE NOT EXISTS (SELECT NULL
                  FROM employees
                  WHERE department_id = d.department_id);
```

DEPARTMENT_ID	DEPARTMENT_NAME
1	120 Treasury
2	130 Corporate Tax
3	140 Control And Credit
4	150 Shareholder Services
5	160 Benefits
6	170 Manufacturing
7	180 Construction
8	190 Contracting
9	200 Operations
10	210 IT Support

...

All Rows Fetched: 16

ORACLE

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

Operator NOT EXISTS

Alternative Lösung

Als Alternative zum Operator NOT EXISTS kann ein NOT IN-Konstrukt verwendet werden.

Beispiel:

```
SELECT department_id, department_name
FROM departments
WHERE department_id NOT IN (SELECT department_id
                           FROM employees);
```

Jedoch wird NOT IN als FALSE ausgewertet, wenn ein Element der Ergebnismenge ein NULL-Wert ist. In diesem Fall gibt die Abfrage keine Zeilen zurück, selbst wenn die Tabelle DEPARTMENTS Zeilen enthält, die die Bedingung WHERE erfüllen.

Lektionsagenda

- Daten mithilfe von Unterabfragen als Quelle abrufen
- Multiple-Column-Unterabfragen erstellen
- Skalare Unterabfragen in SQL
- Aufgabenstellungen mithilfe von korrelierten Unterabfragen lösen
- Operatoren EXISTS und NOT EXISTS
- **Klausel WITH**

ORACLE®

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

Klausel WITH

- Mit der Klausel WITH können Abfrageblöcke in SELECT-Anweisungen wiederverwendet werden, wenn sie in komplexen Abfragen mehrmals vorkommen.
- Die Klausel WITH ruft die Ergebnisse eines Abfrageblocks ab und speichert sie im temporären Tablespace des Benutzers.
- Die Klausel WITH kann zur Performanceverbesserung beitragen.

ORACLE®

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

Mit der Klausel WITH (formale Bezeichnung `subquery_factoring_clause`) können Sie einen Abfrageblock zunächst definieren und dann in SELECT-Anweisungen von komplexen Abfragen mehrfach wiederverwenden. Dies ist besonders hilfreich, wenn eine Abfrage zahlreiche Verweise auf den gleichen Abfrageblock sowie Joins und Aggregationen umfasst.

Mit der Klausel WITH können Sie die gleiche Abfrage wiederverwenden, wenn die Auswertung des Abfrageblocks aufwendig ist und er in einer komplexen Abfrage mehrmals vorkommt. Der Oracle-Server ruft die Ergebnisse eines Abfrageblocks mit der Klausel WITH ab und speichert sie im temporären Tablespace des Benutzers. Dadurch kann die Performance verbessert werden.

Vorteile der Klausel WITH

- Verbessert die Lesbarkeit der Abfrage
- Wertet eine Klausel nur einmal aus, selbst wenn sie mehrmals in der Abfrage vorkommt
- Verbessert in den meisten Fällen die Performance großer Abfragen

Klausel WITH – Beispiel

```
WITH CNT_DEPT AS
(
  SELECT department_id,
     COUNT(1) NUM_EMP
  FROM EMPLOYEES
  GROUP BY department_id
)
SELECT employee_id,
   SALARY/NUM_EMP
FROM EMPLOYEES E
JOIN CNT_DEPT C
ON (e.department_id = c.department_id);
```

EMPLOYEE_ID	SALARY/NUM_EMP
1	100
2	103
3	108
4	110
5	114
6	137
7	139

...

ORACLE

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

Der SQL-Code auf der Folie ist ein Beispiel für eine Situation, in der Sie mit der Klausel `WITH` die Performance verbessern und einfacheren SQL-Code schreiben können. Die Abfrage erstellt den Abfragenamen `CNT_DEPT` und verwendet ihn im Body der Hauptabfrage. Hier führen Sie eine mathematische Operation aus, indem Sie das Gehalt des Mitarbeiters durch die Gesamtanzahl der Mitarbeiter in jeder Abteilung teilen. Die Klausel `WITH` wird intern entweder als Inline-View oder als temporäre Tabelle aufgelöst. Der Optimizer wählt die entsprechende Auflösung in Abhängigkeit von Kosten und Nutzen, die mit dem temporären Speichern der Ergebnisse der Klausel `WITH` verbunden sind.

Hinweise zur Verwendung der Klausel WITH

- Sie wird nur mit `SELECT`-Anweisungen verwendet.
- Abfragenamen sind für alle nachfolgend definierten Abfrageblöcke des Elements `WITH` (einschließlich der Unterabfrageblöcke) sowie für den Hauptabfrageblock (einschließlich der Unterabfrageblöcke) sichtbar.
- Ist der Abfragenamen mit einem vorhandenen Tabellennamen identisch, sucht der Parser von innen nach außen, wobei der Name des Abfrageblocks Vorrang vor dem Tabellennamen hat.
- Die Klausel `WITH` kann mehrere Abfragen enthalten. Die einzelnen Abfragen werden durch Kommas getrennt.

Rekursive Klausel WITH

Die rekursive Klausel WITH:

- ermöglicht die Formulierung rekursiver Abfragen
- erstellt eine Abfrage mit einem Namen, dem rekursiven WITH-Elementnamen
- enthält zwei Typen von Abfragen im Abfrageblock: eine Ankerabfrage und eine rekursive Abfrage
- ist ANSI-kompatibel



ORACLE

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

Die Klausel WITH wurde um die Möglichkeit der Formulierung von rekursiven Abfragen erweitert. Die rekursive WITH-Klausel definiert eine rekursive Abfrage mit einem Namen, dem rekursiven WITH-Elementnamen. Die Definition des rekursiven WITH-Elements muss mindestens zwei Abfrageblöcke enthalten: eine Ankerabfrage und eine rekursive Abfrage. Es können mehrere Ankerabfragen, jedoch nur eine rekursive Abfrage vorhanden sein. Die Ankerabfrage muss vor der rekursiven Abfrage angezeigt werden und darf keine Referenz auf *query_name* enthalten. Die Ankerabfrage kann aus einem oder mehreren Abfrageblöcken zusammengesetzt sein, die durch Mengenoperatoren verbunden sind, beispielsweise durch UNION ALL, UNION, INTERSECT oder MINUS. Die rekursive Abfrage muss auf die Ankerabfrage folgen und muss *query_name* genau einmal referenzieren. Die rekursive Abfrage und die Ankerabfrage müssen mit dem Mengenoperator UNION ALL verbunden sein.

Die rekursive WITH-Klausel entspricht dem ANSI-(American National Standards Institute-)Standard.

Mithilfe der rekursiven WITH-Klausel können Sie hierarchische Daten wie Organisationsdiagramme abfragen.

Rekursive Klausel WITH – Beispiel

Tabelle FLIGHTS

	SOURCE	DESTIN	FLIGHT_TIME
1	San Jose	Los Angeles	1.3
2	New York	Boston	1.1
3	Los Angeles	New York	5.8

1

```
WITH Reachable_From (Source, Destin, TotalFlightTime) AS
(
    SELECT Source, Destin, Flight_time
    FROM Flights
    UNION ALL
    SELECT incoming.Source, outgoing.Destin,
           incoming.TotalFlightTime+outgoing.Flight_time
    FROM Reachable_From incoming, Flights outgoing
    WHERE incoming.Destin = outgoing.Source
)
SELECT Source, Destin, TotalFlightTime
FROM Reachable_From;
```

2

3

	SOURCE	DESTIN	TOTALFLIGHTTIME
1	San Jose	Los Angeles	1.3
2	New York	Boston	1.1
3	Los Angeles	New York	5.8
4	Los Angeles	Boston	6.9
5	San Jose	New York	7.1
6	San Jose	Boston	8.2

ORACLE

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

Das 1. Beispiel auf der Folie zeigt Datensätze aus der Tabelle FLIGHTS an, die Flüge zwischen zwei Städten beschreibt.

Im 2. Beispiel fragen Sie die Tabelle FLIGHTS ab, um die Gesamtflugdauer zwischen einem Ausgangs- und einem Zielort anzuzeigen. Die Klausel WITH in der Abfrage mit dem Namen Reachable From enthält eine Abfrage vom Typ UNION ALL mit zwei Verzweigungen. Die erste Verzweigung ist die Anker-Verzweigung, die alle Zeilen aus der Tabelle FLIGHTS wählt. Die zweite Verzweigung ist die rekursive Verzweigung. Sie verknüpft den Inhalt von Reachable From mit der Tabelle FLIGHTS, um andere erreichbare Städte zu suchen, und fügt diese zum Inhalt von Reachable From hinzu. Der Vorgang ist beendet, wenn die rekursive Verzweigung keine weiteren Zeilen mehr findet.

Das 3. Beispiel zeigt das Ergebnis der Abfrage an, die alle Komponenten aus dem WITH-Klauselelement Reachable From wählt.

Weitere Informationen finden Sie in folgenden Dokumentationen:

- Oracle Database SQL Language Reference 12c Release 1.0
- Oracle Database Data Warehousing Guide 12c Release 1.0

Quiz

Bei einer korrelierten Abfrage steuert die innere `SELECT`-Anweisung die äußere `SELECT`-Anweisung.

- 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:

- Multiple-Column-Unterabfragen erstellen
- Skalare Unterabfragen in SQL verwenden
- Aufgabenstellungen mithilfe von korrelierten Unterabfragen lösen
- Operatoren `EXISTS` und `NOT EXISTS`
- Klausel `WITH`

ORACLE

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

Mit Multiple-Column-Unterabfragen können Sie mehrere `WHERE`-Bedingungen in einer `WHERE`-Klausel kombinieren. Spaltenvergleichen in Multiple-Column-Unterabfragen können paarweise oder nicht paarweise durchgeführt werden.

Mit Unterabfragen können Sie Tabellen definieren, die von einer enthaltenen Abfrage gesteuert werden.

Skalare Unterabfragen können wie folgt verwendet werden:

- Im Bedingungs- und Ausdrucksteil von `DECODE` und `CASE`
- In allen Klauseln von `SELECT`, mit Ausnahme von `GROUP BY`
- In den Klauseln `SET` und `WHERE` einer `UPDATE`-Anweisung

Der Oracle-Server führt eine korrelierte Unterabfrage aus, wenn die Unterabfrage eine Spalte aus einer Tabelle referenziert, auf die in der übergeordneten Anweisung verwiesen wird. Eine korrelierte Unterabfrage wird für jede Zeile, die von der übergeordneten Anweisung verarbeitet wird, einmal ausgewertet. Die übergeordnete Anweisung kann eine `SELECT`-Anweisung sein. Mit der Klausel `WITH` können Sie die gleiche Abfrage wiederverwenden, wenn die Auswertung des Abfrageblocks aufwendig ist und er in einer komplexen Abfrage mehrmals vorkommt.

Übungen zu Lektion 6 – Überblick

Diese Übung behandelt folgende Themen:

- Multiple-Column-Unterabfragen erstellen
- Korrelierte Unterabfragen erstellen
- Operator `EXISTS`
- Skalare Unterabfragen
- Klausel `WITH`

ORACLE

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

In dieser Übung erstellen Sie Multiple-Column-Unterabfragen sowie korrelierte und skalare Unterabfragen. Außerdem lösen Sie Aufgabenstellungen mithilfe der Klausel `WITH`.



Daten mit Unterabfragen bearbeiten

ORACLE®

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

Ziele

Nach Ablauf dieses Lektion haben Sie folgende Ziele erreicht:

- Daten mit Unterabfragen bearbeiten
- Werte einfügen und Unterabfrage als Ziel verwenden
- Schlüsselwort `WITH CHECK OPTION` in DML-Anweisungen
- Zeilen mit korrelierten Unterabfragen aktualisieren und löschen

ORACLE

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

In dieser Lektion lernen Sie, wie Sie Daten in einer Oracle-Datenbank mit Unterabfragen bearbeiten. Außerdem wird beschrieben, wie Sie Probleme mit korrelierten Unterabfragen lösen.

Lektionsagenda

- **Daten mit Unterabfragen bearbeiten**
- Werte einfügen, indem Sie eine Unterabfrage als Ziel verwenden
- Schlüsselwort `WITH CHECK OPTION` in DML-Anweisungen
- Zeilen mit korrelierten Unterabfragen aktualisieren und löschen

ORACLE

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

Daten mit Unterabfragen bearbeiten

Unterabfragen in DML-Anweisungen (Data Manipulation Language) können eingesetzt werden, um:

- Daten mit einer Inline-View abzurufen
- Daten aus einer Tabelle in eine andere Tabelle zu kopieren
- Daten in einer Tabelle auf der Basis von Werten einer anderen Tabelle zu aktualisieren
- Zeilen aus einer Tabelle auf der Basis von Zeilen einer anderen Tabelle zu löschen

ORACLE

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

Mit Unterabfragen können Sie Daten aus einer Tabelle abrufen und als `INSERT` in eine andere Tabelle einfügen. Auf diese Weise können Sie mit einer einzelnen `SELECT`-Anweisung große Datenmengen leicht von einer Tabelle in eine andere Tabelle kopieren. Analog können Sie mit Unterabfragen umfangreiche Aktualisierungs- und Löschvorgänge durchführen, indem Sie sie in der Klausel `WHERE` von `UPDATE`- und `DELETE`-Anweisungen angeben. Unterabfragen können Sie auch in der Klausel `FROM` einer `SELECT`-Anweisung verwenden. Dies wird als Inline-View bezeichnet.

Hinweis: Wie Sie Zeilen auf der Basis einer anderen Tabelle aktualisieren und löschen, haben Sie im Kurs *Oracle Database: SQL Workshop I* gelernt.

Lektionsagenda

- Daten mit Unterabfragen bearbeiten
- Werte einfügen, indem Sie eine Unterabfrage als Ziel verwenden
- Schlüsselwort `WITH CHECK OPTION` in DML-Anweisungen
- Zeilen mit korrelierten Unterabfragen aktualisieren und löschen

ORACLE®

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

Werte einfügen und Unterabfrage als Ziel verwenden

```
INSERT INTO (SELECT l.location_id, l.city, l.country_id
              FROM   loc l
              JOIN   countries c
              ON(l.country_id = c.country_id)
              JOIN   regions USING(region_id)
              WHERE  region_name = 'Europe')
VALUES (3300, 'Cardiff', 'UK');
```

```
1 rows inserted.
```

ORACLE

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

Anstelle des Tabellennamens können Sie eine Unterabfrage in der Klausel `INTO` der `INSERT`-Anweisung verwenden. Die `SELECT`-Liste dieser Unterabfrage muss über dieselbe Anzahl von Spalten verfügen wie die Spaltenliste der Klausel `VALUES`. Alle für die Spalten der Basistabelle geltenden Regeln müssen befolgt werden, damit die `INSERT`-Anweisung ordnungsgemäß funktionieren kann. Sie dürfen beispielsweise keine doppelten Standort-IDs eingeben und keine Werte für erforderliche `NOT NULL`-Spalten weglassen.

Auf diese Weise vermeiden Sie, dass Sie eine View erstellen müssen, nur um einen `INSERT`-Vorgang auszuführen.

Im Beispiel auf der Folie wird anstelle der Tabelle `LOC` eine Unterabfrage verwendet, um einen Datensatz für eine neue europäische Stadt zu erstellen.

Hinweis: Sie können den `INSERT`-Vorgang für die View `EUROPEAN_CITIES` auch wie folgt ausführen:

```
INSERT INTO european_cities
VALUES (3300, 'Cardiff', 'UK');
```

Im Beispiel auf der Folie wird die Tabelle **LOC** mit folgender Anweisung erstellt:

```
CREATE TABLE loc AS SELECT * FROM locations;
```


Werte einfügen und Unterabfrage als Ziel verwenden

Ergebnisse prüfen

```
SELECT location_id, city, country_id
FROM   loc;
```

20	2900 Geneva	CH
21	3000 Bern	CH
22	3100 Utrecht	NL
23	3200 Mexico City	MX
24	3300 Cardiff	UK

ORACLE

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

Das Beispiel auf der Folie zeigt, dass durch die Einfügung mit der Inline-View ein neuer Datensatz in der Basistabelle `LOC` erstellt wurde.

Das folgende Beispiel zeigt das Ergebnis der Unterabfrage, mit der die Tabelle für die `INSERT`-Anweisung ermittelt wurde:

```
SELECT l.location_id, l.city, l.country_id
FROM   loc l
JOIN   countries c
ON(l.country_id = c.country_id)
JOIN   regions USING(region_id)
WHERE  region_name = 'Europe';
```

Lektionsagenda

- Daten mit Unterabfragen bearbeiten
- Werte einfügen, indem Sie eine Unterabfrage als Ziel verwenden
- **Schlüsselwort** WITH CHECK OPTION in DML-Anweisungen
- Zeilen mit korrelierten Unterabfragen aktualisieren und löschen

ORACLE®

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

Schlüsselwort WITH CHECK OPTION in DML-Anweisungen

Mit dem Schlüsselwort `WITH CHECK OPTION` wird verhindert, dass nicht in der Unterabfrage enthaltene Zeilen geändert werden.

```
INSERT INTO ( SELECT location_id, city, country_id
              FROM   loc
              WHERE  country_id IN
                    (SELECT country_id
                     FROM countries
                     NATURAL JOIN regions
                     WHERE region_name = 'Europe')
              WITH CHECK OPTION )
VALUES (3600, 'Washington', 'US');
```

Error report:

**SQL Error: ORA-01402: view WITH CHECK OPTION where-clause violation
01402. 00000 - "view WITH CHECK OPTION where-clause violation"**

***Cause:**

***Action:**

ORACLE

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

Geben Sie das Schlüsselwort `WITH CHECK OPTION` an, um die folgende Bedingung festzulegen: Wenn in einer `INSERT`-, `UPDATE`- oder `DELETE`-Anweisung statt einer Tabelle die Unterabfrage verwendet wird, werden Änderungen, die nicht in der Unterabfrage enthaltene Zeilen generieren, in dieser Tabelle nicht zugelassen.

Das Beispiel auf der Folie zeigt, wie Sie eine Inline-View in Verbindung mit `WITH CHECK OPTION` verwenden. Die `INSERT`-Anweisung verhindert, dass in der Tabelle `LOC` Datensätze für eine Stadt erstellt werden, die nicht in Europa liegt.

Das folgende Beispiel wird aufgrund der Änderungen in der Liste `VALUES` erfolgreich ausgeführt:

```
INSERT INTO (SELECT location_id, city, country_id
              FROM   loc
              WHERE  country_id IN
                    (SELECT country_id
                     FROM countries
                     NATURAL JOIN regions
                     WHERE region_name = 'Europe')
              WITH CHECK OPTION)
VALUES (3500, 'Berlin', 'DE');
```

Die Verwendung einer Inline-View in Verbindung mit `WITH CHECK OPTION` stellt eine einfache Methode dar, um Änderungen für die Tabelle zu verhindern.

Sie können die Erstellung einer nicht europäischen Stadt auch mit einer Datenbank-View verhindern. Führen Sie hierzu folgende Schritte aus:

1. Erstellen Sie eine Datenbank-View:

```
CREATE OR REPLACE VIEW european_cities
AS
SELECT location_id, city, country_id
FROM   locations
WHERE  country_id in
      (SELECT country_id
       FROM countries
       NATURAL JOIN regions
       WHERE region_name = 'Europe')
WITH CHECK OPTION;
```

2. Prüfen Sie die Ergebnisse, indem Sie Daten einfügen:

```
INSERT INTO european_cities
VALUES (3400, 'New York', 'US');
```

Der zweite Schritt erzeugt denselben Fehler, wie auf der Folie angezeigt.

Lektionsagenda

- Daten mit Unterabfragen bearbeiten
- Werte einfügen, indem Sie eine Unterabfrage als Ziel verwenden
- Schlüsselwort `WITH CHECK OPTION` in DML-Anweisungen
- Zeilen mit korrelierten Unterabfragen aktualisieren und löschen

ORACLE®

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

Korrelierte UPDATE-Anweisungen

Mit einer korrelierten Unterabfrage Zeilen in einer Tabelle auf der Basis von Zeilen aus einer anderen Tabelle aktualisieren

```
UPDATE table1 alias1
SET    column = (SELECT expression
                     FROM  table2 alias2
                     WHERE  alias1.column =
                           alias2.column);
```

ORACLE

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

Bei der UPDATE-Anweisung können Sie mithilfe einer korrelierten Unterabfrage Zeilen in einer Tabelle auf der Basis von Zeilen einer anderen Tabelle aktualisieren.

Korrelierte UPDATE-Anweisungen

- Tabelle EMPL6 durch Hinzufügen einer Spalte zum Speichern des Abteilungsnamens denormalisieren
- Tabelle mit einer korrelierten UPDATE-Anweisung füllen

```
ALTER TABLE empl6  
ADD(department_name VARCHAR2(25));
```

```
UPDATE empl6 e  
SET    department_name =  
        (SELECT department_name  
         FROM   departments d  
         WHERE  e.department_id = d.department_id);
```

ORACLE

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

Im Beispiel auf der Folie wird die Tabelle EMPL6 durch Hinzufügen einer Spalte zum Speichern des Abteilungsnamens denormalisiert und anschließend mit einer korrelierten UPDATE-Anweisung gefüllt.

Weiter unten finden Sie ein weiteres Beispiel für eine korrelierte UPDATE-Anweisung.

Aufgabenstellung

Die Tabelle REWARDS enthält eine Liste der Mitarbeiter, die die Erwartungen an ihre Leistung übertroffen haben. Aktualisieren Sie mit einer korrelierten Unterabfrage Zeilen in der Tabelle EMPL6, und verwenden Sie hierzu Zeilen der Tabelle REWARDS:

```
UPDATE empl6  
SET    salary = (SELECT empl6.salary + rewards.pay_raise  
                 FROM   rewards  
                 WHERE  employee_id =  
                        empl6.employee_id  
                 AND    payraise_date =  
                        (SELECT MAX(payraise_date)  
                         FROM   rewards  
                         WHERE  employee_id = empl6.employee_id))  
WHERE  empl6.employee_id  
IN      (SELECT employee_id FROM rewards);
```

In diesem Beispiel wird die Tabelle `REWARDS` verwendet. Die Tabelle `REWARDS` enthält folgende Spalten: `EMPLOYEE_ID`, `PAY_RAISE` und `PAYRAISE_DATE`. Bei jeder Gehaltserhöhung für einen Mitarbeiter wird in die Tabelle `REWARDS` ein Datensatz mit der Personalnummer sowie dem Betrag und dem Datum der Gehaltserhöhung eingefügt. Die Tabelle `REWARDS` kann mehrere Datensätze für einen Mitarbeiter enthalten. Die Spalte `PAYRAISE_DATE` gibt an, wann ein Mitarbeiter seine letzte Gehaltserhöhung erhalten hat.

Im Beispiel wird die Spalte `SALARY` der Tabelle `EMPL6` mit der neuesten Gehaltserhöhung aktualisiert, die der Mitarbeiter erhalten hat. Hierzu wird die entsprechende Gehaltserhöhung aus der Tabelle `REWARDS` zum aktuellen Gehalt des Angestellten addiert.

Korrelierte DELETE-Anweisungen

Mit einer korrelierten Unterabfrage Zeilen in einer Tabelle auf der Basis von Zeilen einer anderen Tabelle löschen

```
DELETE FROM table1 alias1
WHERE column operator
      (SELECT expression
       FROM   table2 alias2
       WHERE  alias1.column = alias2.column);
```

ORACLE®

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

In DELETE-Anweisungen können Sie mithilfe einer korrelierten Unterabfrage nur die Zeilen löschen, die auch in einer anderen Tabelle vorhanden sind. Sie möchten nur die letzten vier Datensätze der Tätigkeitshistorie in der Tabelle `JOB_HISTORY` verwalten. Wenn ein Mitarbeiter zur fünften Tätigkeit wechselt, löschen Sie die älteste Zeile in der Tabelle `JOB_HISTORY`, indem Sie in dieser Tabelle den entsprechenden Wert in der Spalte `MIN(START_DATE)` für den Mitarbeiter suchen. Das folgende Codebeispiel zeigt, wie dieser Vorgang mit einer korrelierten DELETE-Anweisung durchgeführt werden kann:

```
DELETE FROM job_history JH
WHERE employee_id =
      (SELECT employee_id
       FROM employees E
       WHERE JH.employee_id = E.employee_id
       AND START_DATE =
            (SELECT MIN(start_date)
             FROM job_history JH
             WHERE JH.employee_id = E.employee_id)
       AND 5 > (SELECT COUNT(*)
                FROM job_history JH
                WHERE JH.employee_id = E.employee_id
                GROUP BY EMPLOYEE_ID
                HAVING COUNT(*) >= 4));
```

Korrelierte DELETE-Anweisungen

Mit einer korrelierten Unterabfrage nur die Zeilen in der Tabelle EMPL6 löschen, die auch in der Tabelle EMP_HISTORY vorhanden sind

```
DELETE FROM empl6 E
WHERE employee_id =
      (SELECT employee_id
       FROM   emp_history
       WHERE  employee_id = E.employee_id);
```

ORACLE

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

Beispiel

In diesem Beispiel werden zwei Tabellen verwendet. Sie lauten:

- EMPL6: Diese Tabelle enthält Details zu allen aktuellen Mitarbeitern.
- EMP_HISTORY: Diese Tabelle enthält Details zu früheren Mitarbeitern.

EMP_HISTORY enthält Daten früherer Mitarbeiter. Somit wäre es falsch, wenn die Daten eines Mitarbeiters sowohl in der Tabelle EMPL6 als auch in der Tabelle EMP_HISTORY enthalten wären. Sie können solche fehlerhaften Datensätze mit der auf der Folie gezeigten korrelierten Unterabfrage löschen.

Zusammenfassung

In dieser Lektion haben Sie Folgendes gelernt:

- Daten mit Unterabfragen bearbeiten
- Werte einfügen und Unterabfrage als Ziel verwenden
- Schlüsselwort `WITH CHECK OPTION` in DML-Anweisungen verwenden
- Korrelierte Unterabfragen in `UPDATE`- und `DELETE`-Anweisungen verwenden

ORACLE

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

In dieser Lektion haben Sie gelernt, wie Sie Daten in einer Oracle-Datenbank mit Unterabfragen bearbeiten. Außerdem haben Sie gelernt, wie Sie das Schlüsselwort `WITH CHECK OPTION` in DML-Anweisungen und korrelierte Unterabfragen in `UPDATE`- und `DELETE`-Anweisungen verwenden.

Übungen zu Lektion 7 – Überblick

Diese Übung behandelt folgende Themen:

- Daten mit Unterabfragen bearbeiten
- Werte einfügen, indem Sie eine Unterabfrage als Ziel verwenden
- Schlüsselwort `WITH CHECK OPTION` in DML-Anweisungen
- Zeilen mit korrelierten Unterabfragen aktualisieren und löschen

ORACLE

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

In dieser Übung lernen Sie die Konzepte zum Bearbeiten von Daten mithilfe von Unterabfragen, `WITH CHECK OPTION` und korrelierten Unterabfragen zum Aktualisieren und Löschen von Zeilen kennen.

8

Benutzerzugriff steuern

ORACLE®

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

Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- Systemberechtigungen von Objektberechtigungen unterscheiden
- Berechtigungen für Tabellen erteilen
- Rollen zuweisen
- Zwischen Berechtigungen und Rollen unterscheiden

ORACLE

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

In dieser Lektion wird beschrieben, wie Sie den Datenbankzugriff auf bestimmte Objekte steuern und neue Benutzer mit unterschiedlichen Zugriffsberechtigungen hinzufügen.

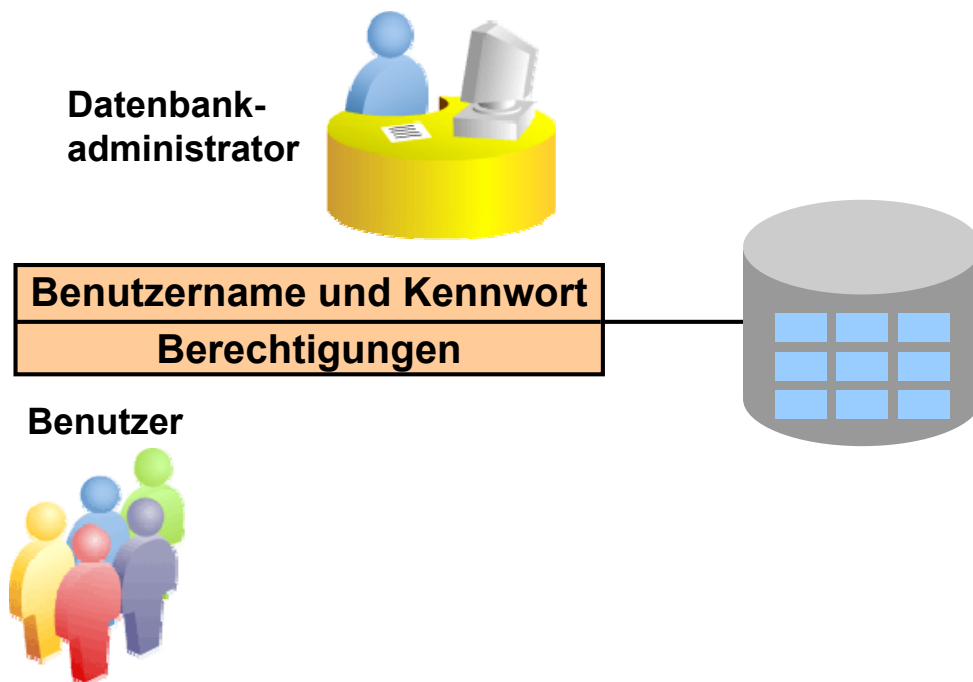
Lektionsagenda

- Systemberechtigungen
- Rollen erstellen
- Objektberechtigungen
- Objektberechtigungen entziehen

ORACLE®

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

Benutzerzugriff steuern



ORACLE

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

In Umgebungen mit mehreren Benutzern müssen Zugriff und Verwendung der Datenbank aus Sicherheitsgründen gesteuert werden. Mit Oracle Server sind folgende Sicherheitsmaßnahmen möglich:

- Datenbankzugriff steuern
- Zugriff auf bestimmte Objekte in der Datenbank gewähren
- Erteilte und erhaltene Berechtigungen im Oracle Data Dictionary bestätigen

Die Datenbanksicherheit kann in zwei Kategorien unterteilt werden: Systemsicherheit und Datensicherheit. Die Systemsicherheit deckt den Zugriff und die Verwendung der Datenbank auf Systemebene ab. Sie umfasst beispielsweise Benutzername und Kennwort, den Benutzern zugewiesenen Speicherplatz und die für Benutzer zulässigen Systemvorgänge. Die Datenbanksicherheit deckt den Zugriff und die Verwendung von Datenbankobjekten sowie die von Benutzern für die Objekte ausführbaren Aktionen ab.

Berechtigungen

- Datenbanksicherheit:
 - Systemsicherheit
 - Datensicherheit
- Systemberechtigungen: Bestimmte Aktionen in der Datenbank ausführen
- Objektberechtigungen: Inhalt der Datenbankobjekte bearbeiten
- Schemas: Zusammenstellung von Objekten wie Tabellen, Views und Sequences

ORACLE

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

Unter einer Berechtigung versteht man das Recht, bestimmte SQL-Anweisungen ausführen zu dürfen. Der Datenbankadministrator (DBA) ist ein Benutzer auf höchster Ebene, der Benutzer erstellen und Benutzern Zugriffsrechte auf die Datenbank und ihre Objekte erteilen kann. Benutzer benötigen *Systemberechtigungen*, um auf die Datenbank zugreifen zu können, und *Objektberechtigungen*, um den Inhalt von Objekten in der Datenbank bearbeiten zu können. Benutzern kann auch die Berechtigung erteilt werden, ihrerseits weitere Berechtigungen an andere Benutzer oder *Rollen* (benannte Gruppen von zusammengehörigen Berechtigungen) zu vergeben.

Schemas

Ein *Schema* ist eine Zusammenstellung von Objekten wie Tabellen, Views und Sequences. Eigentümer des Schemas ist ein Datenbankbenutzer, dessen Name gleichzeitig der Name des Schemas ist.

Systemberechtigungen berechtigen Benutzer, bestimmte Aktionen oder Aktionen für Schemaobjekte eines bestimmten Typs auszuführen. Objektberechtigungen berechtigen Benutzer, bestimmte Aktionen für ein spezifisches Schemaobjekt auszuführen.

Weitere Informationen finden Sie im Referenzhandbuch *Oracle Database 2 Day DBA* für Oracle Database 12c.

Systemberechtigungen

- Über 200 Berechtigungen verfügbar
- Der Datenbankadministrator verfügt über Systemberechtigungen auf höchster Ebene für Aufgaben wie:
 - Neue Benutzer erstellen
 - Benutzer entfernen
 - Tabellen entfernen
 - Tabellen sichern

ORACLE

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

Es stehen über 200 verschiedene Systemberechtigungen für Benutzer und Rollen zur Verfügung. Systemberechtigungen werden in der Regel vom Datenbankadministrator (DBA) erteilt.

Die Tabelle `SYSTEM_PRIVILEGE_MAP` enthält alle verfügbaren Systemberechtigungen, basierend auf der Releaseversion. Die Tabelle wird auch verwendet, um den jeweiligen Namen die Nummern von Berechtigungstypen zuzuordnen.

Typische DBA-Berechtigungen

Systemberechtigung	Autorisierte Vorgänge
CREATE USER	Andere Oracle-Benutzer erstellen
DROP USER	Andere Benutzer löschen
DROP ANY TABLE	Tabellen in jedem beliebigen Schema löschen
BACKUP ANY TABLE	Tabellen in jedem beliebigen Schema mit dem Export-Utility sichern
SELECT ANY TABLE	Tabellen, Views oder Materialized Views in jedem beliebigen Schema abfragen
CREATE ANY TABLE	Tabellen in jedem beliebigen Schema erstellen

Benutzer erstellen

Der DBA erstellt Benutzer mit der Anweisung `CREATE USER`.

```
CREATE USER user  
IDENTIFIED BY password;
```

```
CREATE USER demo  
IDENTIFIED BY demo;
```

ORACLE

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

Der DBA erstellt Benutzer mit der Anweisung `CREATE USER`. Der erstellte Benutzer hat zunächst keine Berechtigungen. Sie können nun vom DBA zugewiesen werden. Über Berechtigungen wird festgelegt, welche Vorgänge der Benutzer auf Datenbankebene ausführen kann.

Die Folie zeigt die verkürzte Syntax zur Erstellung von Benutzern.

Für die Syntax gilt:

<i>user</i>	Der Name des zu erstellenden Benutzers
<i>password</i>	Das Kennwort, mit dem sich der Benutzer anmelden muss

Weitere Informationen finden Sie in der *Oracle Database SQL Language Reference* für Oracle Database 12c.

Hinweis: Seit Oracle Database 11g wird bei Kennwörtern zwischen Groß- und Kleinschreibung unterschieden.

Systemberechtigungen für Benutzer

- Nach der Erstellung des Benutzers weist der DBA spezifische Systemberechtigungen zu.

```
GRANT privilege [, privilege...]  
TO user [, user/ role, PUBLIC...];
```

- Anwendungsentwickler können beispielsweise folgende Systemberechtigungen erhalten:
 - CREATE SESSION
 - CREATE TABLE
 - CREATE SEQUENCE
 - CREATE VIEW
 - CREATE PROCEDURE

ORACLE

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

Typische Benutzerberechtigungen

Nach der Erstellung des Benutzers kann der DBA spezifische Berechtigungen zuweisen.

Systemberechtigung	Autorisierte Vorgänge
CREATE SESSION	Bei der Datenbank anmelden
CREATE TABLE	Tabellen im Schema des Benutzers erstellen
CREATE SEQUENCE	Sequences im Schema des Benutzers erstellen
CREATE VIEW	Views im Schema des Benutzers erstellen
CREATE PROCEDURE	Stored Procedures, Funktionen oder Packages im Schema des Benutzers erstellen

Für die Syntax gilt:

privilege

Die zu erteilende Systemberechtigung.

user | *role* | PUBLIC

Der Name des Benutzers, der Name der Rolle oder PUBLIC
(alle Benutzer)

Hinweis: Die aktuellen Systemberechtigungen sind in der Dictionary View `SESSION_PRIVS` aufgeführt. Ein Data Dictionary ist eine Zusammenstellung von Tabellen und Views, die vom Oracle-Server erstellt und verwaltet werden. Sie enthalten Informationen zur Datenbank.

Systemberechtigungen erteilen

Der DBA kann Benutzern spezifische Systemberechtigungen erteilen.

```
GRANT  create session, create table,  
        create sequence, create view  
TO      demo;
```

```
GRANT succeeded.
```

ORACLE

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

DBAs weisen Benutzern mit der Anweisung `GRANT` Systemberechtigungen zu. Der Benutzer kann die ihm erteilten Berechtigungen sofort verwenden.

Im Beispiel auf der Folie wurden dem Benutzer `demo` Berechtigungen zum Erstellen von Sessions, Tabellen, Sequences und Views erteilt.

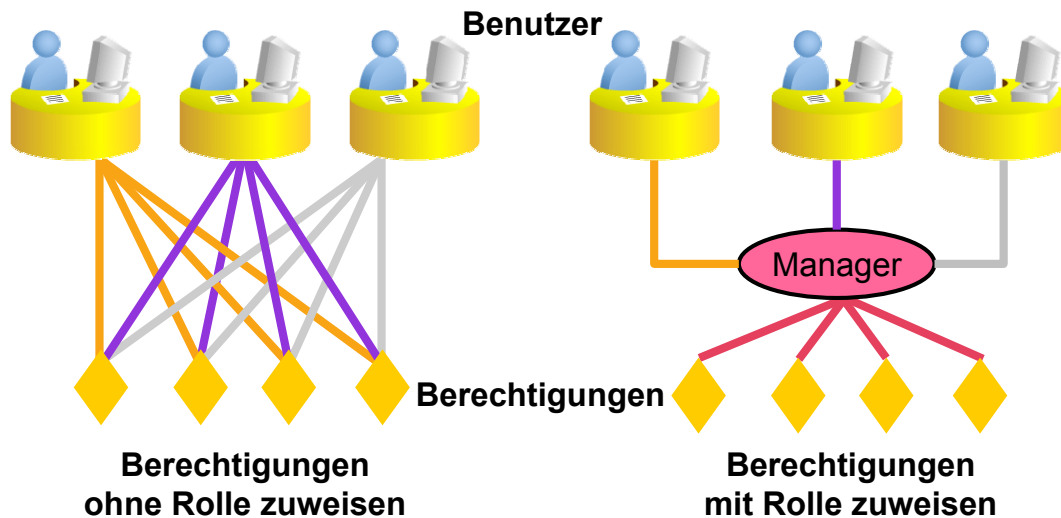
Lektionsagenda

- Systemberechtigungen
- **Rollen erstellen**
- Objektberechtigungen
- Objektberechtigungen entziehen

ORACLE®

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

Was ist eine Rolle?



ORACLE

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

Eine Rolle ist eine benannte Gruppe zusammengehöriger Berechtigungen, die Benutzern erteilt werden kann. Auf diese Weise können Berechtigungen leichter entzogen und verwaltet werden.

Ein Benutzer kann Zugriff auf mehrere Rollen haben. Umgekehrt können auch mehrere Benutzern dieselbe Rolle erhalten. Rollen werden normalerweise für eine Datenbankanwendung erstellt.

Rollen erstellen und zuweisen

Zunächst wird die Rolle vom DBA erstellt. Anschließend erhält sie Berechtigungen und wird Benutzern zugewiesen.

Syntax

```
CREATE    ROLE role;
```

Für die Syntax gilt:

role Der Name der zu erstellenden Rolle

Nachdem der DBA eine Rolle erstellt hat, kann er ihr mit der Anweisung `GRANT` Berechtigungen und Benutzer zuweisen. Eine Rolle ist kein Schemaobjekt. Daher kann jeder Benutzer Berechtigungen hinzufügen.

Rollen erstellen und Berechtigungen zuweisen

- Rollen erstellen:

```
CREATE ROLE manager;
```

- Rollen Berechtigungen erteilen:

```
GRANT create table, create view  
TO manager;
```

- Benutzern Rollen zuweisen:

```
GRANT manager TO alice;
```

ORACLE

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

Rollen erstellen

Im Beispiel auf der Folie wird die Rolle `manager` erstellt. Dem Manager wird dann die Berechtigung erteilt, Tabellen und Views zu erstellen. Schließlich wird dem Benutzer `alice` die Rolle des Managers zugewiesen. `alice` kann nun Tabellen und Views erstellen.

Benutzer, denen mehrere Rollen zugewiesen werden, erhalten alle mit den Rollen verbundenen Berechtigungen.

Kennwörter ändern

- Der DBA erstellt Ihren Benutzeraccount und initialisiert Ihr Kennwort.
- Sie können Ihr Kennwort mit der Anweisung `ALTER USER` ändern.

```
ALTER USER demo  
IDENTIFIED BY employ;
```

ORACLE

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

Der DBA erstellt für alle Benutzer einen Account und initialisiert das entsprechende Kennwort. Sie können Ihr Kennwort mit der Anweisung `ALTER USER` ändern.

Im Beispiel auf der Folie ändert der Benutzer `demo` sein Kennwort mit der Anweisung `ALTER USER`.

Syntax

```
ALTER USER user IDENTIFIED BY password;
```

Für die Syntax gilt:

<i>user</i>	Der Name des Benutzers
<i>password</i>	Das neue Kennwort

Sie können mit dieser Anweisung nicht nur Ihr Kennwort, sondern auch viele andere Optionen ändern. Um weitere Optionen ändern zu können, müssen Sie über die Berechtigung `ALTER USER` verfügen.

Weitere Informationen finden Sie in der *Oracle Database SQL Language Reference* für Oracle Database 12c.

Hinweis: SQL*Plus stellt einen Kennwortbefehl (`PASSW`) bereit, mit dem Sie das Kennwort eines angemeldeten Benutzers ändern können. Dieser Befehl steht in SQL Developer nicht zur Verfügung.

Lektionsagenda

- Systemberechtigungen
- Rollen erstellen
- **Objektberechtigungen**
- Objektberechtigungen entziehen

ORACLE®

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

Objektberechtigungen

Objekt- berechtigung	Tabelle	View	Sequence
ALTER	✓		✓
DELETE	✓	✓	
INDEX	✓		
INSERT	✓	✓	
REFERENCES	✓		
SELECT	✓	✓	✓
UPDATE	✓	✓	

ORACLE

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

Eine *Objektberechtigung* autorisiert Benutzer, eine bestimmte Aktion für eine spezifische Tabelle, View, Sequence oder Prozedur auszuführen. Für jedes Objekt können spezifische Berechtigungen gelten. Die Tabelle auf der Folie zeigt Berechtigungen für verschiedene Objekte. Für Sequences sind nur die Berechtigungen `SELECT` und `ALTER` relevant. Die Berechtigungen `UPDATE`, `REFERENCES` und `INSERT` können auf eine bestimmte Untermenge aktualisierbarer Spalten beschränkt werden.

Sie können eine `SELECT`-Berechtigung einschränken, indem Sie eine View mit einer Untermenge von Spalten erstellen und die Berechtigung `SELECT` nur dieser View erteilen. Eine für ein Synonym erteilte Berechtigung wird in eine Berechtigung für die Basistabelle konvertiert, auf die das Synonym verweist.

Hinweis: Mit der Berechtigung `REFERENCES` stellen Sie sicher, dass andere Benutzer `FOREIGN KEY`-Constraints erstellen können, die auf Ihre Tabelle verweisen.

Objektberechtigungen

- Objektberechtigungen variieren von Objekt zu Objekt.
- Objekteigentümer verfügen über alle Berechtigungen.
- Eigentümer können spezifische Berechtigungen für ihr Objekt vergeben.

```
GRANT      object_priv [(columns)]  
ON         object  
TO         {user|role|PUBLIC}  
[WITH GRANT OPTION];
```

ORACLE

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

Objektberechtigungen erteilen

Für die verschiedenen Arten von Schemaobjekten stehen unterschiedliche Objektberechtigungen zur Verfügung. Benutzer erwerben automatisch alle Objektberechtigungen für Schemaobjekte, die in ihrem Schema enthalten sind. Sie können anderen Benutzern oder Rollen beliebige Objektberechtigungen für die Schemaobjekte erteilen, deren Eigentümer sie sind. Berechtigungen, die mit der Klausel `WITH GRANT OPTION` erteilt werden, können vom Empfänger wiederum anderen Benutzern zugewiesen werden. Ohne diese Klausel der Empfänger die Berechtigung verwenden, aber nicht weitervergeben.

Für die Syntax gilt:

<i>object_priv</i>	Die zu erteilende Objektberechtigung
ALL	Alle Objektberechtigungen
<i>columns</i>	Die Spalte einer Tabelle oder View, für die Berechtigungen vergeben werden
ON <i>object</i>	Das Objekt, für das die Berechtigungen vergeben werden
TO	Der Empfänger der Berechtigung
PUBLIC	Alle Benutzer
WITH GRANT OPTION	Berechtigungsempfänger kann die Objektberechtigungen anderen Benutzern und Rollen erteilen

Hinweis: In der Syntax entspricht *schema* dem Namen des Eigentümers.

Objktberechtigungen erteilen

- Abfrageberechtigungen für die Tabelle `EMPLOYEES` erteilen:

```
GRANT  select
ON      employees
TO      demo;
```

- Benutzern und Rollen Berechtigungen zur Aktualisierung bestimmter Spalten zuweisen:

```
GRANT  update (department_name, location_id)
ON      departments
TO      demo, manager;
```

ORACLE

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

Richtlinien

- Damit Sie Berechtigungen für ein Objekt vergeben können, muss das Objekt in Ihrem Schema enthalten sein, oder Sie selbst müssen die Objektberechtigungen mit dem Zusatz `WITH GRANT OPTION` erhalten haben.
- Der Eigentümer eines Objekts kann jedem anderen Benutzer oder jeder Rolle der Datenbank alle Objektberechtigungen für dieses Objekt zuweisen.
- Der Eigentümer eines Objekts erhält automatisch alle Objektberechtigungen für das Objekt.

Im ersten Beispiel auf der Folie wird dem Benutzer `demo` die Berechtigung zur Abfrage Ihrer Tabelle `EMPLOYEES` erteilt. Im zweiten Beispiel werden dem Benutzer `demo` und der Rolle `manager` `UPDATE`-Berechtigungen für bestimmte Spalten in der Tabelle `DEPARTMENTS` erteilt.

Beispiel: Wenn Ihr Schema `oraxx` lautet und der Benutzer `demo` mit der Anweisung `SELECT` Daten aus Ihrer Tabelle `EMPLOYEES` abrufen möchte, muss er folgende Syntax verwenden:

```
SELECT * FROM oraxx.employees;
```

Alternativ kann der Benutzer `demo` ein Synonym für die Tabelle erstellen und dieses in der Anweisung `SELECT` verwenden:

```
CREATE SYNONYM emp FOR oraxx.employees;
SELECT * FROM emp;
```

Hinweis: Im Allgemeinen erteilen DBAs die Systemberechtigungen. Jeder Benutzer kann jedoch Objektberechtigungen für Objekte erteilen, deren Eigentümer er ist.

Berechtigungen weitergeben

- Benutzer autorisieren, Berechtigungen weiterzugeben:

```
GRANT  select, insert
ON     departments
TO     demo
WITH   GRANT OPTION;
```

- Allen Benutzern im System ermöglichen, Daten aus der Tabelle DEPARTMENTS abzufragen:

```
GRANT  select
ON     departments
TO     PUBLIC;
```

ORACLE

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

Schlüsselwort WITH GRANT OPTION

Der Berechtigungsempfänger kann eine Berechtigung, die ihm mit der Klausel WITH GRANT OPTION erteilt wurde, an andere Benutzer und Rollen weitergeben. Objektberechtigungen, die mit der Klausel WITH GRANT OPTION erteilt wurden, werden entzogen, wenn dem Berechtigungsverleiher die Berechtigung entzogen wird. Die Klausel WITH GRANT OPTION ist für Benutzer oder PUBLIC gültig, nicht für Rollen.

Der Berechtigungsverleiher muss mindestens eines der folgenden Kriterien erfüllen:

- Er ist der Objekteigentümer, oder es wurde ihm mit der Klausel GRANT OPTION Objektzugriff erteilt.
- Er verfügt über die Systemberechtigung GRANT ANY OBJECT PRIVILEGE und über eine Objektberechtigung für dieses Objekt.

Im Beispiel auf der Folie erhält der Benutzer demo Zugriff auf Ihre Tabelle DEPARTMENTS mit den Berechtigungen, die Tabelle abfragen und Zeilen hinzufügen zu können. Der Benutzer demo kann diese Berechtigungen an andere Benutzer weitergeben.

Schlüsselwort PUBLIC

Tabelleneigentümer können mit dem Schlüsselwort PUBLIC allen Benutzern Zugriff auf die Tabelle gewähren. Im zweiten Beispiel dürfen alle Benutzer im System Daten aus der Tabelle DEPARTMENTS abfragen.

Erteilte Berechtigungen prüfen

Data Dictionary View	Beschreibung
ROLE_SYS_PRIVS	Rollen zugewiesene Systemberechtigungen
ROLE_TAB_PRIVS	Rollen zugewiesene Tabellenberechtigungen
USER_ROLE_PRIVS	Für den Benutzer zugreifbare Rollen
USER_SYS_PRIVS	Dem Benutzer zugewiesene Systemberechtigungen
USER_TAB_PRIVS_MADE	Objekten des Benutzers zugewiesene Objektberechtigungen
USER_TAB_PRIVS_RECD	Dem Benutzer zugewiesene Objektberechtigungen
USER_COL_PRIVS_MADE	Objektberechtigungen für Spalten in Benutzerobjekten
USER_COL_PRIVS_RECD	Objektberechtigungen für bestimmte Spalten

ORACLE

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

Der Oracle-Server lässt keine Vorgänge zu, für deren Ausführung Sie nicht entsprechend autorisiert sind. (Sie können beispielsweise keine Zeile aus einer Tabelle löschen, für die Sie keine `DELETE`-Berechtigung haben.)

Die Oracle-Server-Fehlermeldung "Table or view does not exist" weist auf einen der folgenden Fehler hin:

- Sie haben eine Tabelle oder View angegeben, die nicht vorhanden ist.
- Sie haben versucht, einen Vorgang für eine Tabelle oder View auszuführen, für die Sie nicht die entsprechende Berechtigung haben.

Das Data Dictionary enthält Informationen zur Datenbank, die in Tabellen und Views angeordnet sind. Im Data Dictionary können Sie Ihre Berechtigungen einsehen. In der Tabelle auf der Folie sind verschiedene Data Dictionary Views beschrieben.

Weitere Informationen zu den Data Dictionary Views erhalten Sie in der Lektion "Data Dictionary Views – Einführung".

Hinweis: Die Dictionary View `ALL_TAB_PRIVS_MADE` beschreibt alle Objektberechtigungen, die vom Benutzer erteilt oder für die Objekte in seinem Eigentum vergeben wurden.

Lektionsagenda

- Systemberechtigungen
- Rollen erstellen
- Objektberechtigungen
- Objektberechtigungen entziehen

ORACLE®

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

Objektberechtigungen entziehen

- Mit der Anweisung `REVOKE` anderen Benutzern erteilte Berechtigungen entziehen
- Berechtigungen, die anderen Benutzern mit der Klausel `WITH GRANT OPTION` erteilt wurden, werden ebenfalls entzogen.

```
REVOKE {privilege [, privilege...]|ALL}
ON      object
FROM    {user[, user...]|role|PUBLIC}
[CASCADE CONSTRAINTS];
```

ORACLE

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

Berechtigungen, die Sie anderen Benutzern erteilt haben, können Sie mit der Anweisung `REVOKE` wieder entziehen. Die Anweisung `REVOKE` entzieht die von Ihnen angegebenen Berechtigungen sowohl dem genannten Benutzer als auch allen anderen Benutzern, denen die betreffenden Berechtigungen von diesem Benutzer zugewiesen wurden.

Für die Syntax gilt:

`CASCADE` Erforderlich, um alle referenziellen Integritäts-Constraints zu entfernen, die mit der Berechtigung `REFERENCES` auf das Objekt `CONSTRAINTS` angewendet wurden.

Weitere Informationen finden Sie in der *Oracle Database SQL Language Reference* für Oracle Database 12c.

Hinweis: Entziehen Sie einem Benutzer Berechtigungen, der das Unternehmen verlassen hat, müssen Sie alle Berechtigungen neu vergeben, die dieser Benutzer anderen Benutzern erteilt hat. Falls Sie den Benutzeraccount löschen, ohne die Berechtigungen zu entziehen, sind die Systemberechtigungen, die anderen Benutzern von diesem Benutzer erteilt wurden, nicht von dieser Aktion betroffen.

Objektberechtigungen entziehen

An Benutzer `demo` erteilte `SELECT`- und `INSERT`-Berechtigungen für die Tabelle `DEPARTMENTS` entziehen

```
REVOKE select, insert
ON      departments
FROM    demo;
```

REVOKE succeeded.

ORACLE

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

Im Beispiel auf der Folie werden die dem Benutzer `demo` erteilten `SELECT`- und `INSERT`-Berechtigungen für die Tabelle `DEPARTMENTS` entzogen.

Hinweis: Benutzer, die ihre Berechtigungen mit der Klausel `WITH GRANT OPTION` erhalten, können die Berechtigungen mit derselben Klausel weitergeben. Auf diese Weise kann eine lange Kette von Berechtigungsempfängern entstehen, wobei jedoch keine zyklischen Ketten (Weitergabe an Vorgänger) zulässig sind. Entzogene Berechtigungen, die weitergegeben wurden, werden auch allen in der Kette nachfolgenden Benutzern entzogen.

Beispiel: Benutzer `A` erteilt Benutzer `B` mit der Klausel `WITH GRANT OPTION` die Berechtigung `SELECT` für eine Tabelle. In diesem Fall kann Benutzer `B` die Berechtigung ebenfalls mit der Klausel `WITH GRANT OPTION` an Benutzer `C` weitergeben, Benutzer `C` wiederum an Benutzer `D` und so weiter. Entzieht Benutzer `A` nun Benutzer `B` die Berechtigung, wird sie auch den Benutzern `C` und `D` entzogen.

Quiz

Welche der folgenden Aussagen treffen zu?

- a. Nachdem Benutzer ein Objekt erstellt haben, können Sie die verfügbaren Objektberechtigungen mit der Anweisung `GRANT` an andere Benutzern weitergeben.
- b. Benutzer können Rollen mit der Anweisung `CREATE ROLE` erstellen und damit eine Zusammenstellung der System- und Objektberechtigungen an andere Benutzer weitergeben.
- c. Benutzer können ihre eigenen Kennwörter ändern.
- d. Benutzer können die Berechtigungen anzeigen, die ihnen und Ihren Objekte erteilt wurden.

ORACLE

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

Richtige Antwort: a, c, d

Zusammenfassung

In dieser Lektion haben Sie Folgendes gelernt:

- Systemberechtigungen von Objektberechtigungen unterscheiden
- Berechtigungen für Tabellen erteilen
- Rollen zuweisen
- Zwischen Berechtigungen und Rollen unterscheiden

ORACLE

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

DBAs richten die grundlegende Datenbanksicherheit für Benutzer ein, indem sie den Benutzern Berechtigungen erteilen.

- Der DBA erstellt Benutzer, die über ein Kennwort verfügen müssen. Er ist auch für die Einrichtung der ersten Systemberechtigungen von Benutzern zuständig.
- Benutzer können die verfügbaren Objektberechtigungen der von ihnen erstellten Objekte mit der Anweisung `GRANT` an andere oder an alle Benutzer weitergeben.
- DBAs können mit der Anweisung `CREATE ROLE` Rollen erstellen und auf diese Weise eine Zusammenstellung der System- und Objektberechtigungen an mehrere Benutzer weitergeben. Rollen vereinfachen Vergabe und Entzug von Berechtigungen.
- Benutzer können ihr Kennwort mit der Anweisung `ALTER USER` ändern.
- Sie können anderen Benutzern mit der Anweisung `REVOKE` Berechtigungen entziehen.
- In den Data Dictionary Views können Benutzer einsehen, über welche Berechtigungen sie verfügen und welche Berechtigungen für ihre Objekte erteilt wurden.

Übungen zu Lektion 8 – Überblick

Diese Übung behandelt folgende Themen:

- Anderen Benutzern Berechtigungen für Ihre Tabelle erteilen
- Tabellen anderer Benutzer mit den Ihnen erteilten Berechtigungen ändern

ORACLE

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

In diesen Übungen weisen Sie anderen Benutzern Objektberechtigungen zu und ändern die Tabellen anderer Benutzer mit den Ihnen erteilten Berechtigungen.

9

Daten bearbeiten

ORACLE®

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

Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- In den `INSERT`- und `UPDATE`-Anweisungen explizite Standardwerte angeben
- Features von `INSERT`-Anweisungen für mehrere Tabellen beschreiben
- Folgende Typen von `INSERT`-Anweisungen für mehrere Tabellen verwenden:
 - `INSERT` ohne Bedingung
 - `INSERT ALL` mit Bedingung
 - `INSERT FIRST` mit Bedingung
 - `INSERT` mit Pivoting
- Zeilen in einer Tabelle zusammenführen
- Flashback-Vorgänge durchführen
- Datenänderungen über einen längeren Zeitraum überwachen

ORACLE

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

In dieser Lektion lernen Sie, wie Sie mit dem Schlüsselwort `DEFAULT` in den `INSERT`- und `UPDATE`-Anweisungen den Standardwert einer Spalte kennzeichnen. Darüber hinaus werden `INSERT`-Anweisungen für mehrere Tabellen, `MERGE`-Anweisungen, Flashback-Vorgänge und die Überwachung von Änderungen in der Datenbank erläutert.

Lektionsagenda

- In INSERT- und UPDATE-Anweisungen explizite Standardwerte angeben
- Folgende Typen von INSERT-Anweisungen für mehrere Tabellen verwenden:
 - INSERT ohne Bedingung
 - INSERT ALL mit Bedingung
 - INSERT FIRST mit Bedingung
 - INSERT mit Pivoting
- Zeilen in einer Tabelle zusammenführen
- Flashback-Vorgänge ausführen
- Datenänderungen über einen längeren Zeitraum überwachen

ORACLE

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

Explizites Standardfeature – Überblick

- Schlüsselwort `DEFAULT` als Spaltenwert angeben, wenn ein Standardspaltenwert erwünscht ist
- So können Benutzer steuern, wo und wann der Standardwert auf Daten angewendet werden soll.
- Explizite Standardwerte können in `INSERT`- und `UPDATE`-Anweisungen verwendet werden.

ORACLE

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

Das Schlüsselwort `DEFAULT` kann in `INSERT`- und `UPDATE`-Anweisungen angegeben werden, um einen Standardspaltenwert zu kennzeichnen. Ist kein Standardwert vorhanden, wird ein Nullwert verwendet.

Mit der Option `DEFAULT` umgehen Sie die vor Einführung des Features übliche Hartcodierung von Standardwerten in Ihren Programmen oder die Abfrage des Dictionarys. Die Hartcodierung ist problematisch, wenn sich der Standardwert ändert, da in diesem Fall der Code angepasst werden muss. In der Regel wird in Anwendungen nicht auf das Dictionary zugegriffen. Explizite Standardwerte sind daher ein äußerst wichtiges Feature.

Explizite Standardwerte

- DEFAULT mit INSERT-Anweisung:

```
INSERT INTO deptm3
  (department_id, department_name, manager_id)
VALUES (300, 'Engineering', DEFAULT);
```

- DEFAULT mit UPDATE-Anweisung:

```
UPDATE deptm3
SET manager_id = DEFAULT
WHERE department_id = 10;
```

ORACLE

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

Mit `DEFAULT` legen Sie für die Spalte den Wert fest, der zuvor als Standardwert angegeben wurde. Wurde für die entsprechende Spalte kein Standardwert angegeben, legt der Oracle-Server einen Nullwert fest.

Im ersten Beispiel auf der Folie wird in der Anweisung `INSERT` für die Spalte `MANAGER_ID` ein Standardwert verwendet. Falls für die Spalte kein Standardwert definiert ist, wird ein `NULL`-Wert eingefügt.

Im zweiten Beispiel wird in der Anweisung `UPDATE` für die Spalte `MANAGER_ID` ein Standardwert für Abteilung 10 festgelegt. Wurde für die Spalte kein Standardwert definiert, ändert sich der Wert in einen `NULL`-Wert.

Hinweis: Standardwerte für Spalten können bei der Erstellung einer Tabelle angegeben werden. Dieses Thema wird in *SQL Workshop I* behandelt.

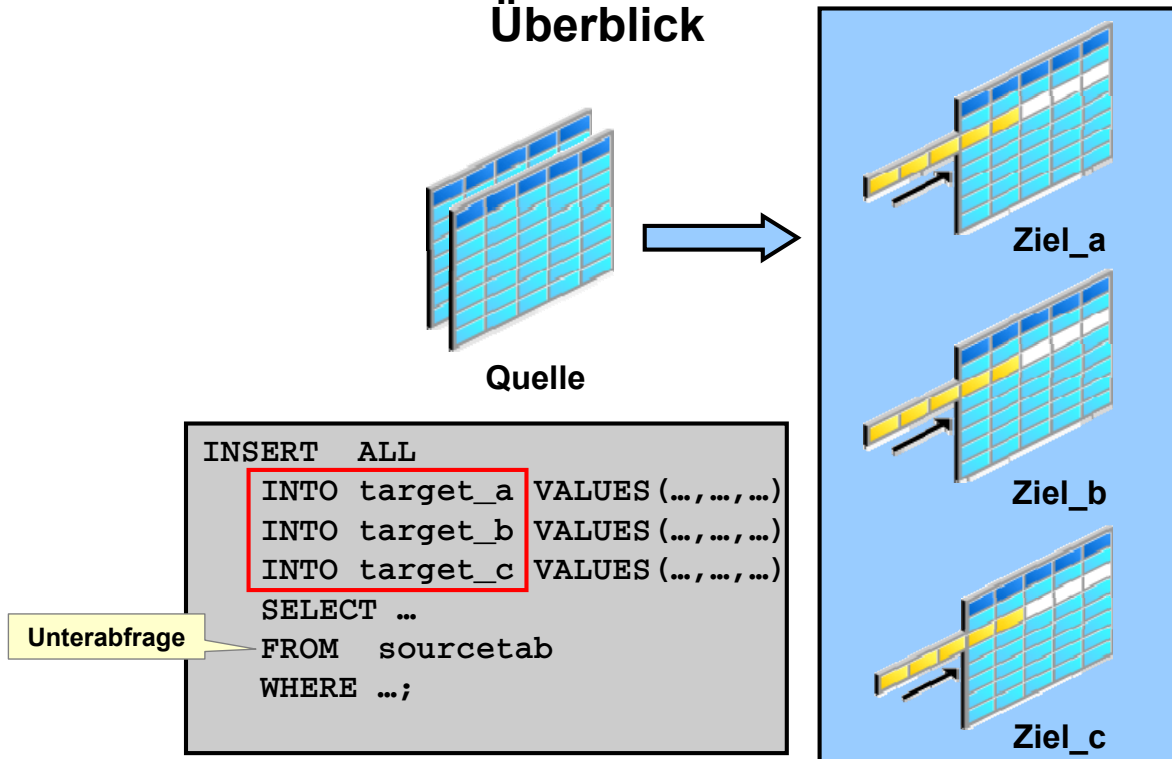
Lektionsagenda

- In INSERT- und UPDATE-Anweisungen explizite Standardwerte angeben
- Folgende Typen von INSERT-Anweisungen für mehrere Tabellen verwenden:
 - INSERT ohne Bedingung
 - INSERT ALL mit Bedingung
 - INSERT FIRST mit Bedingung
 - INSERT mit Pivoting
- Zeilen in einer Tabelle zusammenführen
- Flashback-Vorgänge ausführen
- Datenänderungen über einen längeren Zeitraum überwachen

ORACLE

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

INSERT-Anweisungen für mehrere Tabellen – Überblick



ORACLE®

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

Mit einer `INSERT`-Anweisung für mehrere Tabellen fügen Sie die ermittelten Zeilen, die durch die Auswertung einer Unterabfrage zurückgegeben wurden, in eine oder mehrere Tabellen ein.

`INSERT`-Anweisungen für mehrere Tabellen sind in einem Data Warehouse-Szenario sinnvoll. Sie müssen regelmäßig Daten in das Data Warehouse laden, damit es seinen Zweck, die Geschäftsanalyse, erfüllen kann. Dazu müssen Daten aus einem oder mehreren Betriebssystemen extrahiert und in das Warehouse kopiert werden. Der Prozess, bei dem Daten aus dem Quellsystem extrahiert und in das Data Warehouse geladen werden, heißt ETL (Extrahieren, Transformieren und Laden).

Beim Extrahieren müssen die gewünschten Daten ermittelt und aus vielen verschiedenen Quellen, zum Beispiel Datenbanksystemen und Anwendungen, extrahiert werden. Nach dem Extrahieren müssen die Daten zur weiteren Verarbeitung in das Zielsystem bzw. in ein zwischen-geschaltetes System transportiert werden. Je nach gewählter Transportmethode können dabei bestimmte Transformationen durchgeführt werden. Beispiel: Eine SQL-Anweisung, die über ein Gateway direkt auf ein Remote-Ziel zugreift, kann innerhalb der `SELECT`-Anweisung zwei Spalten verketteten.

Nachdem die Daten in eine Oracle-Datenbank geladen wurden, können mit SQL-Anweisungen Datentransformationen durchgeführt werden. Eine `INSERT`-Anweisung für mehrere Tabellen ist eine Methode zur Implementierung von SQL-Datentransformationen.

INSERT-Anweisungen für mehrere Tabellen – Überblick

- Mit `INSERT...SELECT`-Anweisungen in einer einzigen DML-Anweisung Zeilen in mehrere Tabellen einfügen
- `INSERT`-Anweisungen für mehrere Tabellen werden in Data Warehousing-Systemen verwendet, um Daten aus einer oder mehreren operativen Quellen in eine Gruppe von Zieltabellen zu übertragen.
- Die Anweisungen sorgen für deutliche Performancesteigerungen:
 - Einzelne DML-Anweisung gegenüber mehreren `INSERT . . . SELECT`-Anweisungen
 - Einzelne DML-Anweisung gegenüber mehreren Insert-Vorgängen mit der Syntax `IF . . . THEN`

ORACLE

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

Wenn mehrere Tabellen Ziele sind, bieten `INSERT`-Anweisungen für mehrere Tabellen die Vorteile der `INSERT . . . SELECT`-Anweisung. Ohne die `INSERT`-Anweisung für mehrere Tabellen müssten n unabhängige `INSERT . . . SELECT`-Anweisungen ausgeführt werden. Dabei müssten dieselben Quelldaten n -mal verarbeitet werden, und die Transformationslast würde um das n -Fache erhöht.

Wie bei der bestehenden `INSERT . . . SELECT`-Anweisung kann die neue Anweisung zur Performancesteigerung parallelisiert und mit dem Direct-Load-Mechanismus verwendet werden.

Jeder Datensatz aus einem Input Stream, zum Beispiel einer nicht relationalen Datenbanktabelle, kann jetzt für Umgebungen mit relationalen Datenbanktabellen in mehrere Datensätze konvertiert werden. Um diese Funktionalität alternativ zu implementieren, war es notwendig, mehrere `INSERT`-Anweisungen zu erstellen.

Typen von INSERT-Anweisungen für mehrere Tabellen

Verschiedene Typen von INSERT-Anweisungen für mehrere Tabellen:

- INSERT ohne Bedingung
- INSERT ALL mit Bedingung
- INSERT FIRST mit Bedingung
- INSERT mit Pivoting

ORACLE

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

Der Typ der auszuführenden INSERT-Anweisung wird mit verschiedenen Klauseln festgelegt.
Typen von INSERT-Anweisungen für mehrere Tabellen:

- **INSERT ohne Bedingung:** Für jede von der Unterabfrage zurückgegebene Zeile wird jeweils eine Zeile in die einzelnen Zieltabellen eingefügt.
- **INSERT ALL mit Bedingung:** Für jede von der Unterabfrage zurückgegebene Zeile wird jeweils eine Zeile in die einzelnen Zieltabellen eingefügt, falls die angegebene Bedingung erfüllt ist.
- **INSERT FIRST mit Bedingung:** Für jede von der Unterabfrage zurückgegebene Zeile wird eine Zeile in die erste Zieltabelle eingefügt, in der die Bedingung erfüllt ist.
- **INSERT mit Pivoting:** Dies ist ein Sonderfall von INSERT ALL ohne Bedingung.

INSERT-Anweisungen für mehrere Tabellen

- INSERT für mehrere Tabellen – Syntax:

```
{ ALL
{ insert_into_clause [ values_clause ]}...
| conditional_insert_clause
} subquery
```

- conditional_insert_clause:

```
[ ALL | FIRST ]
WHEN condition THEN insert_into_clause
                        [ values_clause ]
                        [ insert_into_clause [ values_clause ]]...
[ ELSE insert_into_clause
                        [ values_clause ]
                        [ insert_into_clause [ values_clause ]]...
]
```

ORACLE

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

Die Folie zeigt das allgemeine Format von INSERT-Anweisungen für mehrere Tabellen.

INSERT ohne Bedingung: ALL into_clause

Geben Sie ALL gefolgt von mehreren insert_into_clauses an, um INSERT-Vorgänge ohne Bedingung für mehrere Tabellen auszuführen. Der Oracle-Server führt jede insert_into_clause für jede von der Unterabfrage zurückgegebene Zeile einmal aus.

INSERT mit Bedingung: conditional_insert_clause

Geben Sie die conditional_insert_clause an, um INSERT-Vorgänge mit Bedingung für mehrere Tabellen durchzuführen. Der Oracle-Server filtert jede insert_into_clause durch die entsprechende Bedingung WHEN, die festlegt, ob diese insert_into_clause ausgeführt wird. Eine einzelne INSERT-Anweisung für mehrere Tabellen darf bis zu 127 WHEN-Klauseln enthalten.

INSERT mit Bedingung: ALL

Wenn Sie ALL angeben, wertet der Oracle-Server alle WHEN-Klauseln unabhängig von den Ergebnissen der Auswertung anderer WHEN-Klauseln aus. Für alle WHEN-Klauseln, deren Bedingung als TRUE ausgewertet wird, führt der Oracle-Server die entsprechende Liste mit INTO-Klauseln aus.

INSERT: FIRST mit Bedingung

Bei Angabe von `FIRST` wertet der Oracle-Server die `WHEN`-Klauseln in der Reihenfolge aus, in der sie in der Anweisung enthalten sind. Ergibt die Auswertung der ersten `WHEN`-Klausel `TRUE`, führt der Oracle-Server die entsprechende `INTO`-Klausel aus und überspringt in der gegebenen Zeile die nachfolgenden `WHEN`-Klauseln.

Klausel INSERT: ELSE mit Bedingung

Keine der `WHEN`-Klauseln der gegebenen Zeile ergibt `TRUE`:

- `ELSE`-Klausel vorhanden: Der Oracle-Server führt die Liste der `INTO`-Klauseln aus, die der Klausel `ELSE` zugeordnet ist.
- Keine `ELSE`-Klausel vorhanden: Der Oracle-Server führt keine Aktionen für die Zeile aus.

INSERT-Anweisung für mehrere Tabellen – Einschränkungen

- `INSERT`-Anweisungen für mehrere Tabellen können nur für Tabellen, nicht für Views oder Materialized Views ausgeführt werden.
- `INSERT`-Anweisungen für mehrere Tabellen können nicht für Tabellen in einer Remote-Datenbank ausgeführt werden.
- `INSERT`-Anweisungen für mehrere Tabellen können keine Tabellenliste enthalten.
- In `INSERT`-Anweisungen für mehrere Tabellen dürfen alle `insert_into_clauses` zusammen genommen nicht mehr als 999 Zielspalten angeben.

INSERT ALL ohne Bedingung

- Für Mitarbeiter, deren `EMPLOYEE_ID` größer als 200 ist, aus der Tabelle `EMPLOYEES` die Werte `EMPLOYEE_ID`, `HIRE_DATE`, `SALARY` und `MANAGER_ID` wählen
- Die zurückgegebenen Werte mit einer `INSERT`-Anweisung für mehrere Tabellen in die Tabellen `SAL_HISTORY` und `MGR_HISTORY` einfügen

```
INSERT ALL
  INTO sal_history VALUES (EMPID, HIREDATE, SAL)
  INTO mgr_history VALUES (EMPID, MGR, SAL)
SELECT employee_id EMPID, hire_date HIREDATE,
       salary SAL, manager_id MGR
FROM employees
WHERE employee_id > 200;
```

12 rows inserted

ORACLE

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

Im Beispiel auf der Folie werden Zeilen in die Tabellen `SAL_HISTORY` und `MGR_HISTORY` eingefügt.

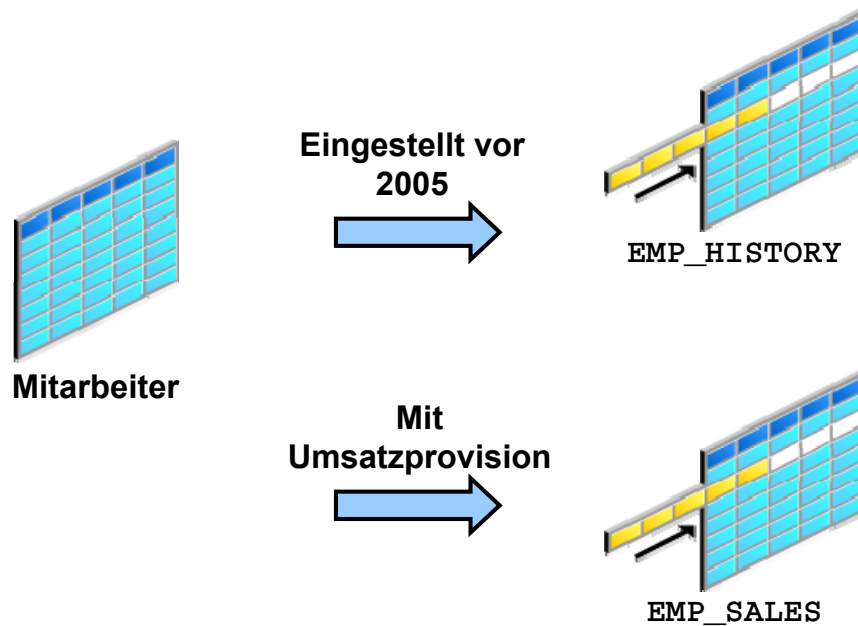
Die Anweisung `SELECT`-Anweisung ruft die Details Personalnummer, Einstellungsdatum, Gehalt und Managernummer der Mitarbeiter mit einer Personalnummer größer als 200 aus der Tabelle `EMPLOYEES` ab. Die Details Personalnummer, Einstellungsdatum und Gehalt werden in die Tabelle `SAL_HISTORY` eingefügt. Die Details Personalnummer, Managernummer und Gehalt werden in die Tabelle `MGR_HISTORY` eingefügt.

Die Anweisung `INSERT` wird als `INSERT` ohne Bedingung bezeichnet, da für die von der Anweisung `SELECT` abgerufenen Zeilen keine weitere Einschränkung gilt. Alle von der Anweisung `SELECT` abgerufenen Zeilen werden in die folgenden beiden Tabellen eingefügt: `SAL_HISTORY` und `MGR_HISTORY`. Die Klausel `VALUES` in den `INSERT`-Anweisungen gibt die Spalten aus der Anweisung `SELECT` an, die in die jeweiligen Tabellen eingefügt werden müssen. Für jede von der Anweisung `SELECT` zurückgegebene Zeile werden zwei Einfügungen ausgeführt: eine für die Tabelle `SAL_HISTORY` und eine für die Tabelle `MGR_HISTORY`.

Insgesamt wurden 12 Zeilen eingefügt:

```
SELECT COUNT(*) total_in_sal FROM sal_history;
SELECT COUNT(*) total_in_mgr FROM mgr_history;
```

INSERT ALL mit Bedingung – Beispiel



ORACLE

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

Aus der Tabelle `EMPLOYEES` aller Mitarbeiter sollen die Datensätze der Mitarbeiter, die vor 2005 eingestellt wurden, abgerufen und in die Mitarbeiterhistorie (`EMP_HISTORY`) eingefügt werden. Die Informationen der Mitarbeiter, die eine Umsatzprovision erhalten, sollen in die Tabelle `EMP_SALES` eingefügt werden. Die SQL-Anweisung finden Sie auf der nächsten Seite.

INSERT ALL mit Bedingung

```
INSERT ALL
  WHEN HIREDATE < '01-JAN-05' THEN
    INTO emp_history VALUES (EMPID, HIREDATE, SAL)
  WHEN COMM IS NOT NULL THEN
    INTO emp_sales VALUES (EMPID, COMM, SAL)
  SELECT employee_id EMPID, hire_date HIREDATE,
         salary SAL, commission_pct COMM
  FROM employees;
```

59 rows inserted.

ORACLE

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

Ähnlich wie im vorherigen Beispiel werden im Beispiel auf dieser Folie Zeilen in die Tabellen EMP_HISTORY und EMP_SALES eingefügt. Die Anweisung SELECT ruft aus der Tabelle EMPLOYEES Details wie Personalnummer, Einstellungsdatum, Gehalt und Provisionssatz für alle Mitarbeiter ab. Personalnummer, Einstellungsdatum und Gehalt werden in die Tabelle SAL_HISTORY eingefügt. Personalnummer, Provisionssatz und Gehalt werden in die Tabelle EMP_SALES eingefügt.

Die INSERT-Anweisung wird als INSERT mit Bedingung bezeichnet, da für die von der Anweisung SELECT abgerufenen Zeilen eine weitere Einschränkung gilt. Aus den von der Anweisung SELECT abgerufenen Zeilen werden nur die Zeilen mit einem Einstellungsdatum vor 2005 in die Tabelle EMP_HISTORY eingefügt. Analog werden nur die Zeilen in die Tabelle EMP_SALES eingefügt, in denen der Wert für den Provisionssatz kein Nullwert ist.

```
SELECT count(*) FROM emp_history;
```

Ergebnis: 24 rows fetched.

```
SELECT count(*) FROM emp_sales;
```

Ergebnis: 35 rows fetched.

In der Anweisung INSERT ALL können Sie optional die Klausel ELSE verwenden.

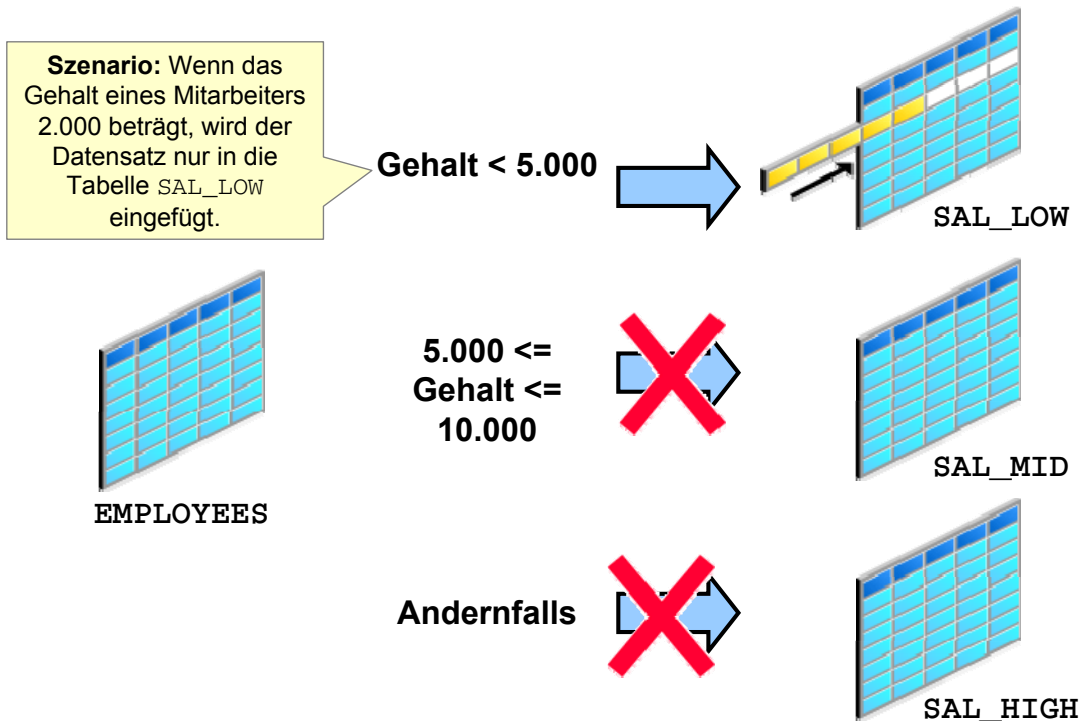
Beispiel:

```
INSERT ALL
  WHEN job_id IN
    (select job_id FROM jobs WHERE job_title LIKE '%Manager%') THEN
    INTO managers2(last_name,job_id,SALARY)
  VALUES (last_name,job_id,SALARY)
  WHEN SALARY>10000 THEN
    INTO richpeople(last_name,job_id,SALARY)
  VALUES (last_name,job_id,SALARY)
  ELSE
    INTO poorpeople VALUES (last_name,job_id,SALARY)
  SELECT * FROM employees;
```

Ergebnis:

```
116 rows inserted
```

INSERT FIRST mit Bedingung – Beispiel



ORACLE

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

Aus der Tabelle `EMPLOYEES` aller Mitarbeiter sollen die Daten der Mitarbeiter in die erste Zieltabelle eingefügt werden, die die Bedingung erfüllen. Beispiel: Wenn das Gehalt eines Mitarbeiters 2.000 beträgt, wird der Datensatz nur in die Tabelle `SAL_LOW` eingefügt. Die SQL-Anweisung finden Sie auf der nächsten Seite.

INSERT FIRST mit Bedingung

```
INSERT FIRST
WHEN salary < 5000 THEN
    INTO sal_low VALUES (employee_id, last_name, salary)
WHEN salary between 5000 and 10000 THEN
    INTO sal_mid VALUES (employee_id, last_name, salary)
ELSE
    INTO sal_high VALUES (employee_id, last_name, salary)
SELECT employee_id, last_name, salary
FROM employees;
```

107 rows inserted

ORACLE

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

Die Anweisung `SELECT` ruft für alle Mitarbeiter aus der Tabelle `EMPLOYEES` Details wie Personalnummer, Nachname und Gehalt ab. Jeder Mitarbeiterdatensatz wird in die erste Zieltabelle eingefügt, die die Bedingung erfüllt.

Diese `INSERT`-Anweisung wird als `INSERT FIRST` mit Bedingung bezeichnet. Die Bedingung `WHEN salary < 5000` wird zuerst ausgewertet. Ergibt die Auswertung der ersten `WHEN`-Klausel `TRUE`, führt der Oracle-Server die zugehörige Klausel `INTO` aus und fügt den Datensatz in die Tabelle `SAL_LOW` ein. Nachfolgende `WHEN`-Klauseln werden für diese Zeile übersprungen.

Für Zeilen, die die erste Bedingung `WHEN (WHEN salary < 5000)` nicht erfüllen, wird die nächste Bedingung (`WHEN salary between 5000 and 10000`) ausgewertet. Ergibt die Auswertung dieser Bedingung `TRUE`, wird der Datensatz in die Tabelle `SAL_MID` eingefügt. Die letzte Bedingung wird übersprungen.

Treffen weder die erste (`WHEN salary < 5000`) noch die zweite Bedingung (`WHEN salary between 5000 and 10000`) zu, führt der Oracle-Server die `INTO`-Klausel der Klausel `ELSE` aus.

Insgesamt wurden 107 Zeilen eingefügt:

```
SELECT count(*) low FROM sal_low;
```

49 rows fetched.

```
SELECT count(*) mid FROM sal_mid;
```

43 rows fetched.

```
SELECT count(*) high FROM sal_high;
```

15 rows fetched.

INSERT mit Pivoting

Gruppe von Umsatzdatensätzen aus der nicht relationalen Datenbanktabelle in das relationale Format konvertieren

Emp_ID	Week_ID	MON	TUES	WED	THUR	FRI
176	6	2000	3000	4000	5000	6000



Employee_ID	WEEK	SALES
176	6	2000
176	6	3000
176	6	4000
176	6	5000
176	6	6000

ORACLE

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

Im Rahmen von Pivoting müssen Sie eine Transformation erstellen, die die einzelnen Datensätze aus dem Input Stream, beispielsweise aus einer nicht relationalen Datenbanktabelle, in mehrere Datensätze konvertiert, die mit einer relationalen Datenbankumgebung kompatibel sind.

Beispiel: Sie erhalten eine Gruppe von Umsatzdatensätzen aus einer nicht relationalen Datenbanktabelle:

`SALES_SOURCE_DATA` in folgendem Format:

`EMPLOYEE_ID, WEEK_ID, SALES_MON, SALES_TUE, SALES_WED, SALES_THUR, SALES_FRI`

Sie möchten diese Datensätze in der Tabelle `SALES_INFO` in einem gängigeren relationalen Format speichern:

`EMPLOYEE_ID, WEEK, SALES`

Um diese Aufgabe zu lösen, erstellen Sie eine Transformation, die jeden Datensatz aus der ursprünglichen nicht relationalen Datenbanktabelle `SALES_SOURCE_DATA` in fünf Datensätze konvertiert, die für die Tabelle `SALES_INFO` des Data Warehouses passend sind. Dieser Vorgang wird allgemein als *Pivoting* bezeichnet.

Die Lösung der Aufgabe finden Sie auf der nächsten Seite.

INSERT mit Pivoting

```
INSERT ALL
  INTO sales_info VALUES (employee_id, week_id, sales_MON)
  INTO sales_info VALUES (employee_id, week_id, sales_TUE)
  INTO sales_info VALUES (employee_id, week_id, sales_WED)
  INTO sales_info VALUES (employee_id, week_id, sales_THUR)
  INTO sales_info VALUES (employee_id, week_id, sales_FRI)
SELECT EMPLOYEE_ID, week_id, sales_MON, sales_TUE,
       sales_WED, sales_THUR, sales_FRI
FROM sales_source_data;
```

5 rows inserted

ORACLE

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

Im Beispiel auf der Folie gehen die Umsatzdaten aus der nicht relationalen Datenbanktabelle `SALES_SOURCE_DATA` ein. Die Daten sind Details zu den Umsätzen, die Vertriebsmitarbeiter an den einzelnen Wochentagen einer bestimmten Kalenderwoche erzielt.

`DESC SALES_SOURCE_DATA`

DESC SALES_SOURCE_DATA		
Name	Null	Type

EMPLOYEE_ID		NUMBER(6)
WEEK_ID		NUMBER(2)
SALES_MON		NUMBER(8,2)
SALES_TUE		NUMBER(8,2)
SALES_WED		NUMBER(8,2)
SALES_THUR		NUMBER(8,2)
SALES_FRI		NUMBER(8,2)

```
SELECT * FROM SALES_SOURCE_DATA;
```

	EMPLOYEE_ID	WEEK_ID	SALES_MON	SALES_TUE	SALES_WED	SALES_THUR	SALES_FRI
1	178	6	1750	2200	1500	1500	3000

```
DESC SALES_INFO
```

desc sales_info	
Name	Null Type

EMPLOYEE_ID	NUMBER(6)
WEEK	NUMBER(2)
SALES	NUMBER(8,2)

```
SELECT * FROM sales_info;
```

	EMPLOYEE_ID	WEEK	SALES
1	178	6	1750
2	178	6	2200
3	178	6	1500
4	178	6	1500
5	178	6	3000

Im vorstehenden Beispiel wird mit der `INSERT`-Anweisung mit Pivoting eine Zeile aus der Tabelle `SALES_SOURCE_DATA` in fünf Datensätze für die relationale Tabelle `SALES_INFO` konvertiert.

Lektionsagenda

- In INSERT- und UPDATE-Anweisungen explizite Standardwerte angeben
- Folgende Typen von INSERT-Anweisungen für mehrere Tabellen verwenden:
 - INSERT ohne Bedingung
 - INSERT ALL mit Bedingung
 - INSERT FIRST mit Bedingung
 - INSERT mit Pivoting
- **Zeilen in einer Tabelle zusammenführen**
- Flashback-Vorgänge ausführen
- Datenänderungen über einen längeren Zeitraum überwachen

ORACLE

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

MERGE-Anweisungen

- Daten bedingungsabhängig in einer Datenbanktabelle aktualisieren, einfügen oder löschen
- UPDATE, wenn die Zeile vorhanden ist, INSERT, wenn die Zeile neu ist:
 - Vermeiden separate Aktualisierungen
 - Steigern Performance und Benutzerfreundlichkeit
 - Sind nützlich in Data Warehousing-Anwendungen

ORACLE

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

Der Oracle-Server unterstützt die Anweisung `MERGE` für die Vorgänge `INSERT`, `UPDATE` und `DELETE`. Mit dieser Anweisung können Sie eine Tabellenzeile bedingungsabhängig aktualisieren, einfügen oder löschen und dadurch die Ausführung mehrerer DML-Anweisungen vermeiden. Ob Daten in der Zieltabelle aktualisiert, eingefügt oder gelöscht werden, hängt von der Bedingung in der Klausel `ON` ab.

Sie müssen über die Objektberechtigungen `INSERT` und `UPDATE` für die Zieltabelle und die Objektberechtigung `SELECT` für die Quelltable verfügen. Um die Klausel `DELETE` der `merge_update_clause` anzugeben, benötigen Sie darüber hinaus die Objektberechtigung `DELETE` für die Zieltabelle.

Die `MERGE`-Anweisung ist deterministisch. Dies bedeutet, dass Sie dieselbe Zeile in der Zieltabelle nicht mehrmals in derselben `MERGE`-Anweisung aktualisieren können.

Eine alternative Methode sind PL/SQL-Schleifen und mehrere DML-Anweisungen. Die Anweisung `MERGE` ist jedoch benutzerfreundlicher und kann einfacher ausgedrückt werden als eine einzelne SQL-Anweisung.

Die Anweisung `MERGE` ist für verschiedene Data Warehousing-Anwendungen geeignet. In diesen Anwendungen arbeiten Sie teilweise mit Daten aus mehreren Quellen, sodass Duplikate auftreten können. Mit der Anweisung `MERGE` können Sie Zeilen bedingungsabhängig hinzufügen oder ändern.

MERGE-Anweisungen – Syntax

Mit der Anweisung `MERGE` Tabellenzeilen bedingungsabhängig einfügen, aktualisieren und löschen

```
MERGE INTO table_name table_alias
  USING (table/view/sub_query) alias
  ON (join condition)
  WHEN MATCHED THEN
    UPDATE SET
      col1 = col1_val,
      col2 = col2_val
  WHEN NOT MATCHED THEN
    INSERT (column_list)
    VALUES (column_values);
```

ORACLE

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

Zeilen zusammenführen

Mit der Anweisung `MERGE` können Sie Zeilen bedingungsabhängig aktualisieren und einfügen. Beim Aktualisieren von Zeilen in Tabellen können Sie gleichzeitig veraltete Zeilen löschen. Hierzu fügen Sie in der Syntax der Anweisung `MERGE` eine `DELETE`-Klausel mit eigener `WHERE`-Klausel ein.

Für die Syntax gilt:

Klausel <code>INTO</code>	Gibt die Zieltabelle an, deren Daten aktualisiert oder ergänzt werden
Klausel <code>USING</code>	Gibt die Quelle der zu aktualisierenden oder einzufügenden Daten an. Möglich sind Tabelle, View oder Unterabfrage
Klausel <code>ON</code>	Gibt die Bedingung für die Ausführung einer Aktualisierung oder Einfügung an
<code>WHEN MATCHED</code>	Gibt an, wie der Server auf die Ergebnisse der Join-Bedingung reagieren soll
<code>WHEN NOT MATCHED</code>	

Hinweis: Weitere Informationen finden Sie in der *Oracle Database SQL Language Reference* für Oracle Database 12c.

Zeilen zusammenführen – Beispiel

In der Tabelle COPY_EMP3 Zeilen einfügen oder aktualisieren, sodass sie der Tabelle EMPLOYEES entspricht

```
MERGE INTO copy_emp3 c
USING (SELECT * FROM EMPLOYEES ) e
ON (c.employee_id = e.employee_id)
WHEN MATCHED THEN
UPDATE SET
c.first_name = e.first_name,
c.last_name = e.last_name,
...

DELETE WHERE (E.COMMISSION_PCT IS NOT NULL)
WHEN NOT MATCHED THEN
INSERT VALUES (e.employee_id, e.first_name, e.last_name,
e.email, e.phone_number, e.hire_date, e.job_id,
e.salary, e.commission_pct, e.manager_id,
e.department_id);
```

107 rows merged.

ORACLE

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

```
MERGE INTO copy_emp3 c
USING (SELECT * FROM EMPLOYEES ) e
ON (c.employee_id = e.employee_id)
WHEN MATCHED THEN
UPDATE SET
c.first_name = e.first_name,
c.last_name = e.last_name,
c.email = e.email,
c.phone_number = e.phone_number,
c.hire_date = e.hire_date,
c.job_id = e.job_id,
c.salary = e.salary*2,
c.commission_pct = e.commission_pct,
c.manager_id = e.manager_id,
c.department_id = e.department_id
DELETE WHERE (E.COMMISSION_PCT IS NOT NULL)
WHEN NOT MATCHED THEN
```

```
INSERT VALUES(e.employee_id, e.first_name, e.last_name,  
e.email, e.phone_number, e.hire_date, e.job_id,  
e.salary, e.commission_pct, e.manager_id,  
e.department_id);
```

Die Tabelle COPY_EMP3 wird mit dem folgenden Code erstellt:

```
CREATE TABLE COPY_EMP3 AS SELECT * FROM EMPLOYEES  
WHERE SALARY<10000;
```

Fragen Sie anschließend die Tabelle COPY_EMP3 ab.

```
SELECT employee_id, salary, commission_pct FROM COPY_EMP3;
```

Sie sehen, dass einige Mitarbeiter die Bedingung SALARY < 10000 erfüllen und zwei Mitarbeiter COMMISSION_PCT beziehen.

Im Beispiel auf der Folie wird die Spalte EMPLOYEE_ID in der Tabelle COPY_EMP3 mit der Spalte EMPLOYEE_ID in der Tabelle EMPLOYEES abgeglichen. Wird eine Übereinstimmung gefunden, wird die Zeile in der Tabelle COPY_EMP3 an die Zeile in der Tabelle EMPLOYEES angepasst, und das Gehalt des Mitarbeiters wird verdoppelt. Die Datensätze der beiden Mitarbeiter, für die Werte in der Spalte COMMISSION_PCT vorhanden sind, werden gelöscht. Wird keine Übereinstimmung gefunden, werden Zeilen in die Tabelle COPY_EMP3 eingefügt.

Zeilen zusammenführen – Beispiel

```
TRUNCATE TABLE copy_emp3;  
SELECT * FROM copy_emp3;  
no rows selected
```

```
MERGE INTO copy_emp3 c  
USING (SELECT * FROM EMPLOYEES ) e  
ON (c.employee_id = e.employee_id)  
WHEN MATCHED THEN  
UPDATE SET  
c.first_name = e.first_name,  
c.last_name = e.last_name,  
...  
DELETE WHERE (E.COMMISSION_PCT IS NOT NULL)  
WHEN NOT MATCHED THEN  
INSERT VALUES(e.employee_id, e.first_name, ...
```

```
SELECT * FROM copy_emp3;  
107 rows selected.
```

ORACLE

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

Die Beispiele auf der Folie zeigen, dass die Tabelle `COPY_EMP3` leer ist. Die Bedingung `c.employee_id = e.employee_id` wird ausgewertet. Die Auswertung ergibt `FALSE`, d. h., es sind keine Übereinstimmungen vorhanden. Die Logik verarbeitet die Klausel `WHEN NOT MATCHED`, und der Befehl `MERGE` fügt die Zeilen der Tabelle `EMPLOYEES` in die Tabelle `COPY_EMP3` ein. Das bedeutet, dass in der Tabelle `COPY_EMP3` nun genau dieselben Daten enthalten sind wie in der Tabelle `EMPLOYEES`.

```
SELECT employee_id, salary, commission_pct from copy_emp3;
```

Lektionsagenda

- In INSERT- und UPDATE-Anweisungen explizite Standardwerte angeben
- Folgende Typen von INSERT-Anweisungen für mehrere Tabellen verwenden:
 - INSERT ohne Bedingung
 - INSERT ALL mit Bedingung
 - INSERT FIRST mit Bedingung
 - INSERT mit Pivoting
- Zeilen in einer Tabelle zusammenführen
- **Flashback-Vorgänge ausführen**
- Datenänderungen über einen längeren Zeitraum überwachen

ORACLE

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

FLASHBACK TABLE-Anweisungen

- Ermöglichen, den früheren Zustand von Tabellen mit einer einzigen Anweisung wiederherzustellen
- Stellen Tabellendaten sowie zugehörige Indizes und Constraints wieder her
- Ermöglichen, die Tabelle und ihren Inhalt in einen bestimmten früheren Zustand oder auf eine frühere System Change Number (SCN) zurückzusetzen



ORACLE

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

Mit Oracle Flashback Table können Sie den früheren Zustand von Tabellen mit einer einzigen Anweisung wiederherstellen. Sie können Tabellendaten sowie zugehörige Indizes und Constraints wiederherstellen, während die Datenbank online ist. Dabei werden nur in den angegebenen Tabellen Änderungen rückgängig gemacht.

Das Feature Flashback Table ist mit einem Selfservice-Wiederherstellungstool vergleichbar. Beispiel: Ein Benutzer hat versehentlich wichtige Zeilen in einer Tabelle gelöscht und möchte die gelöschten Zeilen wiederherstellen. Mit der Anweisung `FLASHBACK TABLE` können Sie den Zustand der Tabelle vor dem Löschvorgang wiederherstellen und die fehlenden Zeilen in der Tabelle anzeigen.

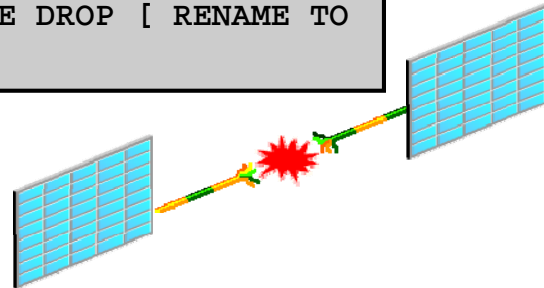
Mit der Anweisung `FLASHBACK TABLE` setzen Sie die Tabelle und ihren Inhalt in einen bestimmten früheren Zustand oder auf eine frühere SCN zurück.

Hinweis: Die System Change Number (SCN) ist eine ganze Zahl, die jeder Änderung in der Datenbank zugewiesen wird. Sie stellt eine eindeutige inkrementelle Zahl in der Datenbank dar. Bei jeder Transaktion, die festgeschrieben wird, speichert das System eine neue SCN.

FLASHBACK TABLE-Anweisungen

- Wiederherstellungstool für versehentliche Tabellenänderungen
 - Stellen den früheren Zustand einer Tabelle wieder her
 - Bieten Benutzerfreundlichkeit, Verfügbarkeit und schnelle Ausführung
 - Werden an Ort und Stelle ausgeführt
- Syntax:

```
FLASHBACK TABLE [ schema. ] table [, [ schema. ] table ]... TO { { { SCN | TIMESTAMP } expr |  
RESTORE POINT restore_point } [ { ENABLE |  
DISABLE } TRIGGERS ] | BEFORE DROP [ RENAME TO  
table ] } ;
```



ORACLE

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

Selfservice-Wiederherstellungstool

Oracle Database stellt in SQL den DDL-Befehl `FLASHBACK TABLE` bereit, mit dem Sie den früheren Zustand einer versehentlich gelöschten oder geänderten Tabelle wiederherstellen können. Der Befehl `FLASHBACK TABLE` ist ein Selfservice-Wiederherstellungstool, mit dem Sie die Daten einer Tabelle sowie ihre zugehörigen Attribute wie Indizes und Views wiederherstellen. Dabei werden nur die nachfolgenden Änderungen an der entsprechenden Tabelle zurückgerollt, während die Datenbank online ist. Gegenüber herkömmlichen Recovery-Methoden bietet dieses Feature wesentliche Vorteile wie Benutzerfreundlichkeit, Verfügbarkeit und Schnelligkeit beim Zurückschreiben. Darüber hinaus wird dem Datenbankadministrator die Aufgabe abgenommen, anwendungsspezifische Eigenschaften zu suchen und wiederherzustellen. Das Feature Flashback Table bietet keine Lösung bei physischen Fehlern aufgrund fehlerhafter Datenträger.

Syntax

Sie können einen `FLASHBACK TABLE`-Vorgang für eine oder mehrere Tabellen aufrufen. Die Tabellen können sich sogar in verschiedenen Schemas befinden. Geben Sie den Zeitpunkt, zu dem Sie zurückkehren möchten, mit einem gültigen Zeitstempel an. Datenbanktrigger werden für alle betreffenden Tabellen während des Flashback-Vorgangs standardmäßig deaktiviert. Sie können dieses Standardverhalten außer Kraft setzen, indem Sie die Klausel `ENABLE TRIGGERS` angeben.

Hinweis: Weitere Informationen zur Papierkorb- und Flashback-Semantik finden Sie im *Oracle Database Administrator's Guide* für Oracle Database 12c.

FLASHBACK TABLE-Anweisungen – Beispiel

```
DROP TABLE emp3;
```

```
table EMP3 dropped.
```

```
SELECT original_name, operation, droptime FROM  
recyclebin;
```

	ORIGINAL_NAME	OPERATION	DROPTIME
1	EMP3	DROP	2012-10-16:05:59:34

...

```
FLASHBACK TABLE emp3 TO BEFORE DROP;
```

```
table EMP3 succeeded.
```

ORACLE

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

Syntax und Beispiele

Im Beispiel wird der Zustand der Tabelle `EMP3` vor einer `DROP`-Anweisung wiederhergestellt.

Der Papierkorb (`recyclebin`) ist eine Data Dictionary-Tabelle, die Informationen zu gelöschten Objekten enthält. Gelöschte Tabellen und alle zugehörigen Objekte wie Indizes, Constraints und Nested Tables werden nicht entfernt und belegen weiterhin Speicherplatz. Sie werden so lange in den Speicherplatz-Quotas für Benutzer berücksichtigt, bis sie explizit dauerhaft aus dem Papierkorb gelöscht werden oder aufgrund von Tablespace-Speicherplatzbeschränkungen von der Datenbank dauerhaft gelöscht werden müssen.

Jeder Benutzer ist sozusagen Eigentümer eines Papierkorbs, da er nur auf die Objekte im Papierkorb zugreifen kann, die ihm selbst gehören, sofern er nicht über die Berechtigung `SYSDBA` verfügt. Mit der folgenden Anweisung kann der Benutzer seine Objekte im Papierkorb anzeigen:

```
SELECT * FROM RECYCLEBIN;
```

Wenn Sie einen Benutzer löschen, wird keines der Objekte, die diesem Benutzer gehören, in den Papierkorb verschoben, und alle Objekte im Papierkorb werden dauerhaft gelöscht.

Löschen Sie den Papierkorb mit der folgenden Anweisung dauerhaft:

```
PURGE RECYCLEBIN;
```

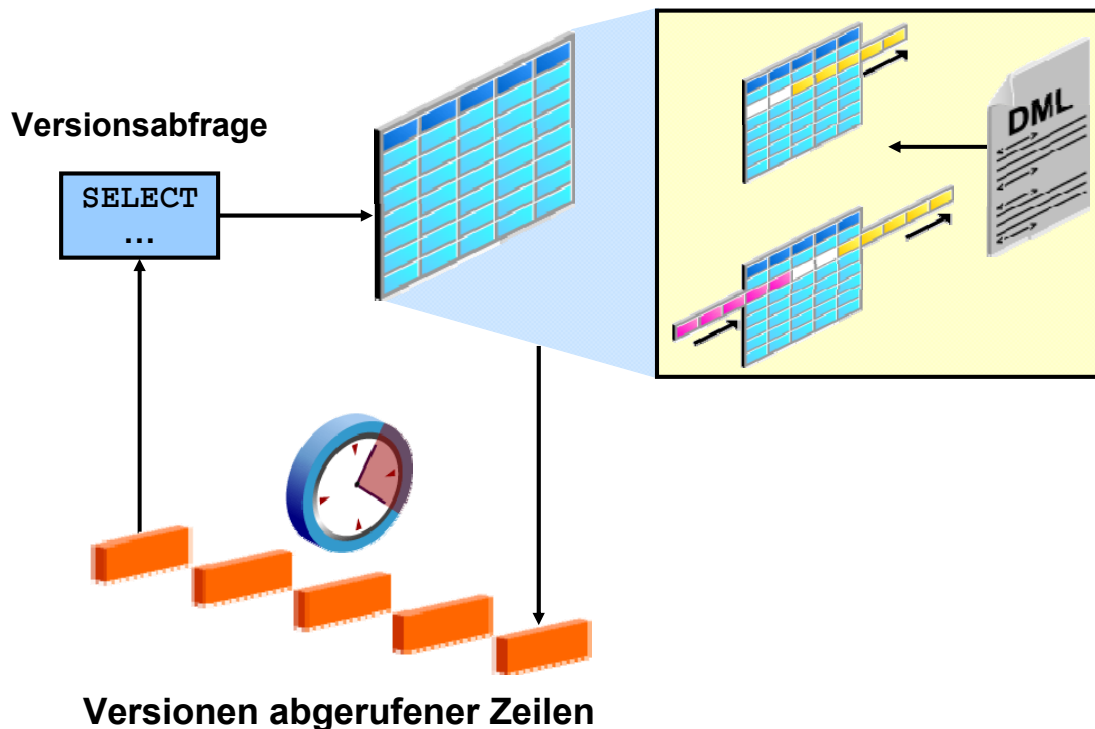
Lektionsagenda

- In INSERT- und UPDATE-Anweisungen explizite Standardwerte angeben
- Folgende Typen von INSERT-Anweisungen für mehrere Tabellen verwenden:
 - INSERT ohne Bedingung
 - INSERT ALL mit Bedingung
 - INSERT FIRST mit Bedingung
 - INSERT mit Pivoting
- Zeilen in einer Tabelle zusammenführen
- Flashback-Vorgänge ausführen
- Datenänderungen über einen längeren Zeitraum überwachen

ORACLE

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

Datenänderungen überwachen



ORACLE

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

Sie stellen fest, dass die Daten in einer Tabelle möglicherweise fehlerhaft geändert wurden. Um dies zu prüfen, können Sie mithilfe mehrerer Flashback-Abfragen Zeilendaten für bestimmte Zeitpunkte anzeigen. Mit Oracle Flashback Query können Sie Daten in einem früheren Zustand abrufen. Eine effizientere Möglichkeit bietet das Feature Flashback Version Query, das alle Änderungen an einer Zeile über einen längeren Zeitraum anzeigt. Mit diesem Feature können Sie eine `VERSIONS`-Klausel an eine `SELECT`-Anweisung anfügen, die einen SCN-(System Change Number-)Bereich oder den Zeitstempelbereich angibt, für den Sie die Änderungen an Zeilenwerten anzeigen möchten. Die Abfrage kann außerdem zugehörige Metadaten zurückgeben, zum Beispiel die für die Änderung verantwortliche Transaktion.

Nachdem Sie eine fehlerhafte Transaktion identifiziert haben, können Sie mit dem Feature Flashback Transaction Query auch andere durch die Transaktion vorgenommene Änderungen identifizieren. Sie haben dann die Möglichkeit, mit dem Feature Flashback Table den Zustand der Tabelle zu einem Zeitpunkt vor den Änderungen wiederherzustellen.

Mit einer Tabellenabfrage mit der Klausel `VERSIONS` können Sie alle Versionen sämtlicher Zeilen abfragen, die zwischen dem Zeitpunkt der Abfrage und dem Zeitraum `undo_retention` (in Sekunden) vor dem aktuellen Zeitpunkt vorhanden waren. `undo_retention` ist ein Initialisierungsparameter und damit ein automatisch optimierter Parameter. Eine Abfrage mit einer `VERSIONS`-Klausel wird als Versionsabfrage bezeichnet. Das Ergebnis einer Versionsabfrage entspricht dem Fall, dass die Klausel `WHERE` auf die Zeilenversionen angewendet wird. Die Versionsabfrage gibt nur transaktionsübergreifende Zeilenversionen zurück.

System change number (SCN): Der Oracle-Server weist eine SCN zu, um die Redo-Datensätze für jede festgeschriebene Transaktion zu identifizieren.

Flashback Query – Beispiel

```
SELECT salary FROM employees3  
WHERE last_name = 'Chung';
```

```
UPDATE employees3 SET salary = 4000  
WHERE last_name = 'Chung';
```

```
SELECT salary FROM employees3  
WHERE last_name = 'Chung';
```

```
SELECT salary FROM employees3  
AS OF TIMESTAMP (SYSTIMESTAMP - INTERVAL '1' MINUTE)  
WHERE last_name = 'Chung';
```

1

	SALARY
1	3800

2

	SALARY
1	4000

3

	SALARY
1	3800

ORACLE

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

Verwenden Sie Oracle Flashback Query in einer `SELECT`-Anweisung mit `AS OF`-Klausel. Oracle Flashback Query ruft Daten in einem früheren Zustand ab. Die Abfrage referenziert einen vergangenen Zeitpunkt explizit über einen Zeitstempel oder eine System Change Number (SCN). Sie gibt festgeschriebene Daten zurück, die zu diesem Zeitpunkt aktuell waren.

Im Beispiel auf der Folie wird das Gehalt für den Mitarbeiter Chung abgerufen (1). Das Gehalt für den Mitarbeiter Chung wird auf 4000 erhöht (2). Um zu erfahren, wie der Wert vor der Aktualisierung ausgesehen hat, können Sie Flashback Query verwenden (3).

Oracle Flashback Query kann in folgenden Szenarios verwendet werden:

- Sie können verlorene Daten wiederherstellen oder falsche, festgeschriebene Änderungen rückgängig machen. Beispiel: Wenn Sie versehentlich Zeilen löschen oder aktualisieren und sie dann festschreiben, können Sie diesen Fehler sofort rückgängig machen.
- Sie können aktuelle Daten mit den entsprechenden Daten zu einem Zeitpunkt in der Vergangenheit vergleichen. Beispiel: Sie können täglich einen Bericht ausführen, der die Änderung der Daten von gestern zeigt. Sie können einzelne Zeilen der Tabellendaten vergleichen und Schnittmengen oder Vereinigungsmengen von Zeilengruppen suchen.
- Sie können den Status von Transaktionsdaten zu einem bestimmten Zeitpunkt prüfen.

Flashback Version Query – Beispiel

```
SELECT salary FROM employees3  
WHERE employee_id = 107;
```

1

```
UPDATE employees3 SET salary = salary * 1.30  
WHERE employee_id = 107;
```

2

```
COMMIT;
```

```
SELECT salary FROM employees3  
VERSIONS BETWEEN SCN MINVALUE AND MAXVALUE  
WHERE employee_id = 107;
```

3

1

ROWID	SALARY
1	4200

3

ROWID	SALARY
1	5460
2	4200

ORACLE

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

Im Beispiel auf der Folie wird das Gehalt für den Mitarbeiter 107 abgerufen (1). Das Gehalt für den Mitarbeiter 107 wird um 30 Prozent erhöht, und diese Änderung wird festgeschrieben (2). Die unterschiedlichen Gehaltsversionen werden angezeigt (3).

Die Klausel `VERSIONS` ändert den Plan der Abfrage nicht. Beispiel: Wenn Sie eine Abfrage für eine Tabelle ausführen, die die Indexzugriffsmethode verwendet, wendet die gleiche Tabellenabfrage mit der Klausel `VERSIONS` die Indexzugriffsmethode weiterhin an. Die von der Versionsabfrage zurückgegebenen Zeilenversionen sind transaktionsübergreifende Zeilenversionen. Die Klausel `VERSIONS` wirkt sich nicht auf das Transaktionsverhalten einer Abfrage aus. Dies bedeutet, dass eine Tabellenabfrage mit einer `VERSIONS`-Klausel nach wie vor die Abfrageumgebung der laufenden Transaktion erbt.

Die Standardklausel `VERSIONS` kann als `VERSIONS BETWEEN {SCN|TIMESTAMP} MINVALUE AND MAXVALUE` angegeben werden. Die Klausel `VERSIONS` ist eine abfragenspezifische SQL-Erweiterung. DML- und DDL-Vorgänge können die Klausel `VERSIONS` in Unterabfragen verwenden. Die Zeilenversionsabfrage ruft alle festgeschriebenen Versionen der ausgewählten Zeilen ab. Änderungen durch die aktuelle aktive Transaktion werden nicht zurückgegeben. Die Versionsabfrage ruft alle Vorkommnisse der Zeilen ab. Dies bedeutet im Wesentlichen, dass die zurückgegebenen Versionen gelöschte und später wieder eingefügte Zeilenversionen beinhalten. Der Zeilenzugriff bei einer Versionsabfrage kann in folgende zwei Kategorien unterteilt werden:

- **ROWID-basierter Zeilenzugriff:** Bei einem ROWID-basierten Zugriff werden alle Versionen der angegebenen ROWID zurückgegeben. Der Zeileninhalt spielt dabei keine Rolle. Dies bedeutet im Wesentlichen, dass alle Versionen des Blockeintrags zurückgegeben werden, der durch die ROWID angegeben ist.
- **Sonstiger Zeilenzugriff:** Bei jedem anderen Zeilenzugriff werden alle Zeilenversionen zurückgegeben.

VERSIONS BETWEEN-Klauseln

```
SELECT versions_starttime "START_DATE",
       versions_endtime   "END_DATE",
       salary
FROM   employees
       VERSIONS BETWEEN SCN MINVALUE
       AND MAXVALUE
WHERE  last_name = 'Lorentz';
```

	START_DATE	END_DATE	SALARY
1	11-SEP-12 03.38.54.000000000 AM (null)		5460
2	(null)	11-SEP-12 03.38.54.000000000 AM	4200

```
SELECT salary FROM employees3
       VERSIONS BETWEEN SCN MINVALUE AND MAXVALUE
WHERE  employee_id = 107;
```

ORACLE

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

Mit der Klausel `VERSIONS BETWEEN` können Sie alle Zeilenversionen abrufen, die jemals zwischen dem Abfragezeitpunkt und einem zurückliegenden Zeitpunkt vorhanden waren.

Wenn die Undo-Erhaltungsfrist unter dem unteren zeitlichen Grenzwert oder der SCN der Klausel `BETWEEN` liegt, ruft die Abfrage nur Versionen bis zur Undo-Erhaltungsfrist ab. Das Zeitintervall der Klausel `BETWEEN` kann als SCN-Intervall oder als Uhrzeitintervall angegeben werden. Dieses Zeitintervall ist durch die Untergrenze und Obergrenze abgeschlossen.

Im Beispiel werden die Änderungen des Gehalts von Lorentz abgerufen. Der `NULL`-Wert für `END_DATE` für die erste Version gibt an, dass dies die vorhandene Version zum Zeitpunkt der Abfrage war. Der `NULL`-Wert für `START_DATE` für die letzte Version gibt an, dass diese Version zu einem Zeitpunkt vor der Undo-Erhaltungsfrist erstellt wurde.

Quiz

Wenn Sie das Schlüsselwort `DEFAULT` in den Befehlen `INSERT` und `UPDATE` angeben, müssen Sie in Ihren Programmen keinen Standardwert hartcodieren oder im Dictionary suchen.

- a. Richtig
- b. Falsch

ORACLE®

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

Richtige Antwort: a

Quiz

Wenn Sie den Befehl `DROP TABLE` ausführen, benennt die Datenbank die Tabelle in jedem Fall um und verschiebt sie in den Papierkorb. Von dort kann sie später mit der Anweisung `FLASHBACK TABLE` wiederhergestellt werden.

- 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:

- In den `INSERT`- und `UPDATE`-Anweisungen explizite Standardwerte angeben
- Features von `INSERT`-Anweisungen für mehrere Tabellen beschreiben
- Folgende Typen von `INSERT`-Anweisungen für mehrere Tabellen verwenden:
 - `INSERT` ohne Bedingung
 - `INSERT ALL` mit Bedingung
 - `INSERT FIRST` mit Bedingung
 - `INSERT` mit Pivoting
- Zeilen in einer Tabelle zusammenführen
- Flashback-Vorgänge durchführen
- Datenänderungen über einen längeren Zeitraum überwachen

ORACLE

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

In dieser Lektion wurden die Themen `INSERT`-Anweisungen für mehrere Tabellen, `MERGE`-Anweisungen und Überwachung von Änderungen in Datenbanken erläutert.

Übungen zu Lektion 9 – Überblick

Diese Übung behandelt folgende Themen:

- INSERT-Vorgänge für mehrere Tabellen ausführen
- MERGE-Vorgänge ausführen
- Flashback-Vorgänge ausführen
- Zeilenversionen überwachen

ORACLE

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

In diesen Übungen lernen Sie, wie Sie `INSERT`-Vorgänge für mehrere Tabellen, `MERGE`-Vorgänge und Flashback-Vorgänge ausführen sowie Zeilenversionen überwachen.

10

Daten in verschiedenen Zeitzonen verwalten

ORACLE®

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

Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- DATE-ähnliche Datentypen verwenden, die Sekundenbruchteile speichern und Zeitzonen überwachen
- Datentypen verwenden, die die Differenz zwischen zwei Datetime-Werten speichern
- Die folgenden Datetime-Funktionen verwenden:
 - CURRENT_DATE
 - CURRENT_TIMESTAMP
 - LOCALTIMESTAMP
 - DBTIMEZONE
 - SESSIONTIMEZONE
 - EXTRACT
 - TZ_OFFSET
 - FROM_TZ
 - TO_TIMESTAMP
 - TO_YMINTERVAL
 - TO_DSINTERVAL

ORACLE

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

In dieser Lektion lernen Sie, wie Sie mit DATE-ähnlichen Datentypen Sekundenbruchteile speichern und Zeitzonen überwachen. Außerdem werden einige der in Oracle Database verfügbaren Datetime-Funktionen beschrieben.

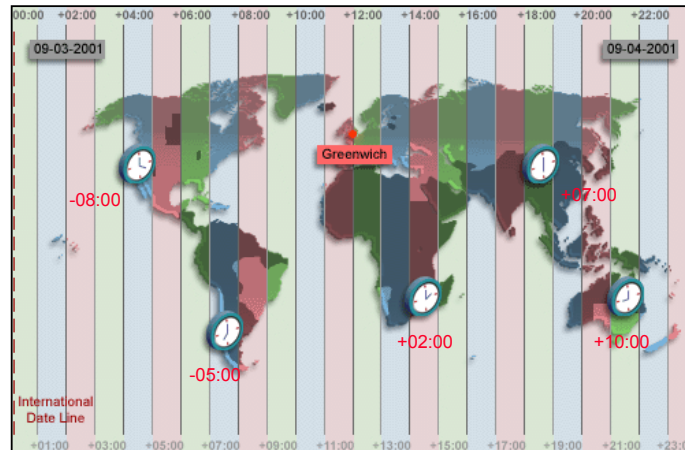
Lektionsagenda

- CURRENT_DATE, CURRENT_TIMESTAMP
und LOCALTIMESTAMP
- INTERVAL-Datentypen
- Folgende Funktionen verwenden:
 - EXTRACT
 - TZ_OFFSET
 - FROM_TZ
 - TO_TIMESTAMP
 - TO_YMINTERVAL
 - TO_DSINTERVAL

ORACLE®

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

Zeitzone



Das Bild zeigt die Uhrzeit der einzelnen Zeitzone relativ zu 12.00 Uhr GMT (mittlere Greenwich-Zeit).

ORACLE

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

Die Stunden eines Tages werden an der Erdumdrehung gemessen. Die aktuelle Uhrzeit hängt davon ab, wo Sie sich befinden. Wenn es in Greenwich in England 12 Uhr mittags ist, dann ist es entlang der internationalen Datumsgrenze Mitternacht. Die Erde ist in 24 Zeitzone eingeteilt, eine für jede Stunde des Tages. Die Zeitzone, die entlang des Nullmeridians im englischen Greenwich verläuft, wird als Greenwich Mean Time (GMT) bezeichnet. GMT ist heute von der koordinierten Weltzeit (Coordinated Universal Time (UTC)) abgelöst. UTC ist der Zeitstandard, auf den sich alle anderen Zeitzone der Welt beziehen. Diese Zeit bleibt das ganze Jahr hindurch gleich und unterliegt nicht der Sommerzeit. Die Meridianlinie ist eine imaginäre Linie, die vom Nordpol zum Südpol verläuft. Sie wird auch als Null-Längengrad bezeichnet und ist die Linie, von der aus alle anderen Längengrade gemessen werden. Alle Orte besitzen einen Breitengrad (ihre Entfernung vom Äquator in nördlicher oder südlicher Richtung) und einen Längengrad (ihre Entfernung vom Greenwich-Nullmeridian in östlicher oder westlicher Richtung). Die Zeit wird immer in Bezug auf die UTC gemessen.

Sessionparameter `TIME_ZONE`

`TIME_ZONE` kann auf folgende Werte eingestellt werden:

- Absolute Differenz
- Zeitzone der Datenbank
- Lokale Zeitzone des Betriebssystems
- Benannte Region

```
ALTER SESSION SET TIME_ZONE = '-05:00';  
ALTER SESSION SET TIME_ZONE = dbtimezone;  
ALTER SESSION SET TIME_ZONE = local;  
ALTER SESSION SET TIME_ZONE = 'America/New_York';
```

ORACLE

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

Oracle Database unterstützt die Aufnahme der Zeitzone in Ihre Datums- und Uhrzeitangaben und die Erfassung von Sekundenbruchteilen. Mit dem Befehl `ALTER SESSION` können Sie die Zeitzone in der Session eines Benutzers ändern. Die Zeitzone kann als absolute Differenz in Bezug auf die UTC, als benannte Zeitzone, als Datenbankzeitzone oder als lokale Zeitzone angegeben werden.

CURRENT_DATE, CURRENT_TIMESTAMP und LOCALTIMESTAMP

- CURRENT_DATE:
 - Gibt das aktuelle Datum von der Benutzersession zurück
 - Hat den Datentyp DATE
- CURRENT_TIMESTAMP:
 - Gibt das aktuelle Datum und die aktuelle Uhrzeit von der Benutzersession zurück
 - Hat den Datentyp TIMESTAMP WITH TIME ZONE
- LOCALTIMESTAMP:
 - Gibt das aktuelle Datum und die aktuelle Uhrzeit von der Benutzersession zurück
 - Hat den Datentyp TIMESTAMP

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. Der Datentyp von CURRENT_DATE ist DATE. Der Datentyp von CURRENT_TIMESTAMP ist TIMESTAMP WITH TIME ZONE. Die zurückgegebenen Werte zeigen die Zeitonenverschiebung für die SQL-Session an, in der die Funktionen ausgeführt werden. Die Zeitonenverschiebung ist die Differenz (in Stunden und Minuten) zwischen der lokalen und der UTC-Zeit. Der Datentyp TIMESTAMP WITH TIME ZONE hat das Format:

TIMESTAMP [(fractional_seconds_precision)] WITH TIME ZONE

Hierbei gibt fractional_seconds_precision optional im Datetime-Feld SECOND die Anzahl der Stellen für die Sekundenbruchteile an. Der Wertebereich ist 0 bis 9. Der Standardwert ist 6.

Die Funktion LOCALTIMESTAMP gibt das aktuelle Datum und die aktuelle Uhrzeit in der Sessionzeitzone zurück. Im Unterschied zu CURRENT_TIMESTAMP, das einen TIMESTAMP WITH TIME ZONE-Wert zurückgibt, gibt LOCALTIMESTAMP einen TIMESTAMP-Wert zurück.

Diese Funktionen richten sich nach dem National Language Support (NLS), das heißt, die Ergebnisse haben das aktuelle NLS-Kalenderformat und Datetime-Format.

Hinweis: Die Funktion SYSDATE gibt das aktuelle Datum und die aktuelle Uhrzeit als DATE-Datentyp zurück. Im Kurs *Oracle Database: SQL Workshop I* wurde die Verwendung der Funktion SYSDATE erläutert..

Datum und Uhrzeit in einer Sessionzeitzone vergleichen

Den Parameter `TIME_ZONE` auf `-5:00` einstellen und zum Vergleich der Unterschiede `SELECT`-Anweisungen für Datum und Uhrzeit ausführen:

```
ALTER SESSION  
SET NLS_DATE_FORMAT = 'DD-MON-YYYY HH24:MI:SS';  
ALTER SESSION SET TIME_ZONE = '-5:00';
```

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

1

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

2

```
SELECT SESSIONTIMEZONE, LOCALTIMESTAMP FROM DUAL;
```

3

ORACLE

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

Mit dem Befehl `ALTER SESSION` wird das Datumsformat der Session auf `'DD-MON-YYYY HH24:MI:SS'` festgelegt, also Tag (1-31)-Monatskürzel-vierstellige Jahreszahl Stunde (0-23):Minute (0-59):Sekunde (0-59).

Im Beispiel auf der Folie wird die Session geändert. Der Parameter `TIME_ZONE` wird auf den Wert `-5:00` eingestellt. Anschließend wird die Anweisung `SELECT` für `CURRENT_DATE`, `CURRENT_TIMESTAMP` und `LOCALTIMESTAMP` ausgeführt, um die Unterschiede im Format zu ermitteln.

Hinweis: Der Parameter `TIME_ZONE` gibt die Standardverschiebung der lokalen Zeitzone für die aktuelle SQL-Session an. `TIME_ZONE` ist ein reiner Sessionparameter und kein Initialisierungsparameter. Der Parameter `TIME_ZONE` wird wie folgt festgelegt:

```
TIME_ZONE = '[+ | -] hh:mm'
```

Die Formatmaske `([+ | -] hh:mm)` gibt die Stunden und Minuten vor und nach UTC an.

Datum und Uhrzeit in einer Sessionzeitzone vergleichen

Abfrageergebnisse:

session SET altered.

SESSIONTIMEZONE	CURRENT_DATE
1 -05:00	10-SEP-12 23:08:16

1

SESSIONTIMEZONE	CURRENT_TIMESTAMP
1 -05:00	10-SEP-12 11.10.21.183775000 PM -05:00

2

SESSIONTIMEZONE	LOCALTIMESTAMP
1 -05:00	10-SEP-12 11.10.43.871626000 PM

3

ORACLE

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

In diesem Beispiel gibt die Funktion `CURRENT_DATE` das aktuelle Datum in der Sessionzeitzone zurück. Die Funktion `CURRENT_TIMESTAMP` gibt das aktuelle Datum und die aktuelle Uhrzeit als Wert vom Datentyp `TIMESTAMP WITH TIME ZONE` in der Sessionzeitzone zurück. Die Funktion `LOCALTIMESTAMP` gibt das aktuelle Datum und die aktuelle Uhrzeit in der Sessionzeitzone zurück.

Hinweis: Die Ausgabe des Codebeispiels kann abhängig vom Ausführungsdatum abweichen.

DBTIMEZONE und SESSIONTIMEZONE

- Wert der Datenbankzeitzone anzeigen:

```
SELECT DBTIMEZONE FROM DUAL;
```

DBTIMEZONE
1 +00:00

- Wert der Sessionzeitzone anzeigen:

```
SELECT SESSIONTIMEZONE FROM DUAL;
```

SESSIONTIMEZONE
1 -05:00

ORACLE

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

Der Datenbankadministrator legt die Standardzeitzone der Datenbank fest, indem er in der Anweisung `CREATE DATABASE` die Klausel `SET TIME_ZONE` angibt. Wenn diese Klausel nicht angegeben wird, ist die Standardzeitzone der Datenbank die Zeitzone des Betriebssystems. Die Datenbankzeitzone für eine Session kann nicht mit der Anweisung `ALTER SESSION` geändert werden.

Die Funktion `DBTIMEZONE` gibt den Wert der Datenbankzeitzone zurück. Der Rückgabebetyp ist eine Zeitzonendifferenz (ein Zeichentyp im Format '`[+ | -] TZH:TZM`') oder der Name einer Zeitzone, je nachdem, wie der Benutzer den Wert der Datenbankzeitzone in der letzten `CREATE DATABASE`- oder `ALTER DATABASE`-Anweisung angegeben hat. Das Beispiel auf der Folie zeigt, dass die Datenbankzeitzone auf "-05:00" eingestellt ist, da der Parameter `TIME_ZONE` das folgende Format hat:

```
TIME_ZONE = '[+ | -] hh:mm'
```

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 Zeitzone, je nachdem, wie der Benutzer den Wert der Zeitzone der Session in der letzten `ALTER SESSION`-Anweisung angegeben hat. Das Beispiel auf der Folie zeigt, dass die Sessionzeitzone eine Differenz von -5 Stunden gegenüber der UTC hat. Die Datenbankzeitzone unterscheidet sich von der Zeitzone der aktuellen Session.

TIMESTAMP-Datentypen

Datentyp	Felder
TIMESTAMP	Jahr, Monat, Tag, Stunde, Minute, Sekunde mit Sekundenbruchteilen
TIMESTAMP WITH TIME ZONE	Entspricht dem Datentyp TIMESTAMP; enthält ebenfalls: TIMEZONE_HOUR und TIMEZONE_MINUTE oder TIMEZONE_REGION
TIMESTAMP WITH LOCAL TIME ZONE	Entspricht dem Datentyp TIMESTAMP; enthält ebenfalls eine Zeitzonendifferenz im Wert

ORACLE

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

Der Datentyp `TIMESTAMP` ist eine Erweiterung des Datentyps `DATE`.

`TIMESTAMP (fractional_seconds_precision)`

Dieser Datentyp enthält die Werte Jahr, Monat und Tag für das Datum sowie die Werte Stunde, Minute und Sekunde für die Uhrzeit, wobei `fractional_seconds_precision` die Anzahl der Stellen für die Sekundenbruchteile im Datetime-Feld `SECOND` angibt. Die zulässigen Werte für `fractional_seconds_precision` sind 0 bis 9. Der Standardwert ist 6.

`TIMESTAMP (fractional_seconds_precision) WITH TIME ZONE`

Dieser Datentyp enthält alle Werte von `TIMESTAMP` sowie den Wert für die Zeitzoneverschiebung.

`TIMESTAMP (fractional_seconds_precision) WITH LOCAL TIME ZONE`

Dieser Datentyp enthält alle Werte von `TIMESTAMP` mit folgenden Ausnahmen:

- Die Daten werden beim Speichern in der Datenbank auf die Datenbankzeitzone normalisiert.
- Beim Abrufen werden die Daten dem Benutzer in der Sessionzeitzone angezeigt.

TIMESTAMP-Felder

Datetime-Feld	Gültige Werte
YEAR	-4712 bis 9999 (ohne Jahr 0)
MONTH	01 bis 12
DAY	01 bis 31
HOURL	00 bis 23
MINUTE	00 bis 59
SECOND	00 bis 59,9(n), wobei 9(n) die Anzahl der Stellen ist
TIMEZONE_HOUR	-12 bis 14
TIMEZONE_MINUTE	00 bis 59

ORACLE

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

Jeder Datetime-Datentyp setzt sich aus mehreren dieser Felder zusammen. Datetime-Werte können nur miteinander verglichen und einander zugeordnet werden, wenn sie dieselben Datetime-Felder enthalten.

Unterschied zwischen DATE und TIMESTAMP

A

```
-- when hire_date is  
of type DATE  
  
SELECT hire_date  
FROM emp4;
```

	HIRE_DATE
1	17-JUN-03
2	21-SEP-05
3	13-JAN-01
4	03-JAN-06
5	21-MAY-07
6	25-JUN-05
7	05-FEB-06
8	07-FEB-07
9	17-AUG-02
10	16-AUG-02

...

B

```
ALTER TABLE emp4  
MODIFY hire_date  
TIMESTAMP(7);  
SELECT hire_date  
FROM emp4;
```

	HIRE_DATE
1	17-JUN-03 12.00.00.000000000 AM
2	21-SEP-05 12.00.00.000000000 AM
3	13-JAN-01 12.00.00.000000000 AM
4	03-JAN-06 12.00.00.000000000 AM
5	21-MAY-07 12.00.00.000000000 AM
6	25-JUN-05 12.00.00.000000000 AM
7	05-FEB-06 12.00.00.000000000 AM
8	07-FEB-07 12.00.00.000000000 AM
9	17-AUG-02 12.00.00.000000000 AM
10	16-AUG-02 12.00.00.000000000 AM

...

ORACLE

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

Datentyp TIMESTAMP – Beispiel

Beispiel A auf der Folie zeigt die Daten aus der Spalte `hire_date` der Tabelle `EMP4`, wenn der Datentyp der Spalte `DATE` ist. Im Beispiel B wird die Tabelle geändert. Der Datentyp der Spalte `hire_date` wird in `TIMESTAMP` geändert. Die Ausgabe zeigt die Unterschiede bei der Anzeige. Sie können von `DATE` in `TIMESTAMP` konvertieren, wenn die Spalte Daten enthält. Wenn die Spalte leer ist, können Sie nur von `DATE` oder `TIMESTAMP` in `TIMESTAMP WITH TIME ZONE` konvertieren.

Für `TIMESTAMP` können Sie eine Anzahl der Stellen für die Sekundenbruchteile angeben. Wenn wie im Beispiel oben kein Wert angegeben ist, wird der Standardwert 6 übernommen.

Die folgende Anweisung stellt die Anzahl der Stellen für die Sekundenbruchteile auf 7 ein:

```
ALTER TABLE emp4  
MODIFY hire_date TIMESTAMP(7);
```

Hinweis: Der von Oracle verwendete Datentyp `DATE` entspricht standardmäßig diesem Beispiel. Er kann jedoch auch zusätzliche Informationen wie Stunden, Minuten, Sekunden, AM und PM enthalten. Um das Datum in diesem Format auszugeben, können Sie eine Formatmaske oder eine Funktion auf den Datumswert anwenden.

TIMESTAMP-Datentypen vergleichen

```
CREATE TABLE web_orders  
(order_date TIMESTAMP WITH TIME ZONE,  
 delivery_time TIMESTAMP WITH LOCAL TIME ZONE);
```

```
INSERT INTO web_orders values  
(current_date, current_timestamp + 2);
```

```
SELECT * FROM web_orders;
```

ORDER_DATE	DELIVERY_TIME
1 10-SEP-12 11.30.20.000000000 PM -05:00	12-SEP-12 11.30.20.000000000 PM

ORACLE

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

Im Beispiel auf der Folie wird eine neue Tabelle `web_orders` erstellt, die eine Spalte vom Datentyp `TIMESTAMP WITH TIME ZONE` und eine Spalte vom Datentyp `TIMESTAMP WITH LOCAL TIME ZONE` enthält. Diese Tabelle wird gefüllt, wenn ein Webauftrag (`web_order`) erteilt wird. Zeitstempel und Zeitzone für den Benutzer, der den Auftrag erteilt, werden auf Basis des Wertes `CURRENT_DATE` eingefügt. Als Wert für den lokale Zeitstempel und die lokale Zeitzone werden bei jeder Auftragserteilung zwei Tage zum Wert von `CURRENT_TIMESTAMP` hinzugefügt. Auf diese Weise kann ein webbasiertes Unternehmen bei der Versandbestätigung die Lieferzeit auf der Basis der Zeitzone berechnen, die für die den Auftrag erteilende Person relevant ist.

Hinweis: Die Ausgabe des Codebeispiels kann in Abhängigkeit vom Ausführungsdatum abweichen.

Lektionsagenda

- CURRENT_DATE, CURRENT_TIMESTAMP und LOCALTIMESTAMP
- **INTERVAL-Datentypen**
- Folgende Funktionen verwenden:
 - EXTRACT
 - TZ_OFFSET
 - FROM_TZ
 - TO_TIMESTAMP
 - TO_YMINTERVAL
 - TO_DSINTERVAL

ORACLE®

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

INTERVAL-Datentypen

- INTERVAL-Datentypen dienen zum Speichern der Differenz zwischen zwei Datetime-Werten.
- Es gibt zwei Klassen von Intervallen:
 - Jahr-Monat
 - Tag-Uhrzeit
- Die Genauigkeit des Intervalls wird angegeben:
 - durch die tatsächliche Teilmenge der Felder im Intervall
 - im Intervallkennzeichner

Datentyp	Felder
INTERVAL YEAR TO MONTH	Jahr, Monat
INTERVAL DAY TO SECOND	Tage, Stunde, Minute, Sekunde mit Sekundenbruchteilen

ORACLE

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

INTERVAL-Datentypen dienen zum Speichern der Differenz zwischen zwei Datetime-Werten. Es gibt zwei Klassen von Intervallen: Jahr-Monat-Intervalle und Tag-Uhrzeit-Intervalle. Ein Jahr-Monat-Intervall besteht aus zusammenhängenden Bestandteilen der Felder `YEAR` und `MONTH`. Ein Tag-Uhrzeit-Intervall besteht aus zusammenhängenden Bestandteilen der Felder `DAY`, `hour`, `minute` und `second`. Die tatsächliche Teilmenge der Felder, die ein Intervall bilden, wird als Genauigkeit des Intervalls bezeichnet und im Intervallkennzeichner angegeben. Da die Anzahl der Tage in einem Jahr vom Kalender abhängt, sind Jahr-Monat-Intervalle NLS-abhängig, während Tag-Uhrzeit-Intervalle NLS-unabhängig sind.

Der Intervallkennzeichner kann ebenfalls die Genauigkeit, das heißt die Anzahl der Stellen im ersten oder einzigen Feld angeben. Wenn das letzte Feld `second` ist, kann der Kennzeichner auch die Genauigkeit der Sekundenbruchteile angeben, das heißt die Anzahl der Stellen für die Sekundenbruchteile im Wert `second`. Erfolgt keine Angabe, gelten der Standardwert 2 für die Genauigkeit des ersten Feldes und der Standardwert 6 für die Genauigkeit der Sekundenbruchteile.

INTERVAL YEAR (year_precision) TO MONTH

Dieser Datentyp speichert einen Zeitraum in Jahren und Monaten, wobei `year_precision` die Anzahl der Stellen im Datetime-Feld `YEAR` ist. Zulässige Werte sind 0 bis 9. Der Standardwert ist 6.

INTERVAL DAY (day_precision) TO SECOND (fractional_seconds_precision)

Dieser Datentyp speichert einen Zeitraum in Tagen, Stunden, Minuten und Sekunden, wobei `day_precision` die maximale Anzahl der Stellen im Datetime-Feld `DAY` ist. Zulässige Werte sind 0 bis 9. Der Standardwert ist 2. `fractional_seconds_precision` ist die Anzahl der Stellen für die Sekundenbruchteile im Feld `SECOND`. Zulässige Werte sind 0 bis 9. Der Standardwert ist 6.

INTERVAL-Felder

INTERVAL-Feld	Gültige Werte für Intervall
YEAR	Beliebige positive oder negative Ganzzahl
MONTH	00 bis 11
DAY	Beliebige positive oder negative Ganzzahl
HOURL	00 bis 23
MINUTE	00 bis 59
SECOND	00 bis 59,9(n), wobei 9(n) die Anzahl der Stellen ist

ORACLE

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

INTERVAL YEAR TO MONTH kann die Felder YEAR und MONTH enthalten.

INTERVAL DAY TO SECOND kann die Felder DAY, HOUR, MINUTE und SECOND enthalten.

Die tatsächliche Teilmenge der Felder, die ein Element einer dieser Intervalltypen bilden, wird durch einen Intervallkennzeichner definiert. Diese Teilmenge wird als Genauigkeit des Elements bezeichnet.

Jahr-Monat-Intervalle können nur mit anderen Jahr-Monat-Intervallen verglichen und diesen zugeordnet werden. Analog gilt dies für Tag-Uhrzeit-Intervalle.

INTERVAL YEAR TO MONTH – Beispiel

```
CREATE TABLE warranty
(prod_id number, warranty_time INTERVAL YEAR(3) TO
MONTH);
INSERT INTO warranty VALUES (123, INTERVAL '8' MONTH);
INSERT INTO warranty VALUES (155, INTERVAL '200'
YEAR(3));
INSERT INTO warranty VALUES (678, '200-11');
SELECT * FROM warranty;
```

	PROD_ID	WARRANTY_TIME
1	123	0-8
2	155	200-0
3	678	200-11

ORACLE

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

Der Datentyp `INTERVAL YEAR TO MONTH` speichert einen Zeitraum mit den Datetime-Feldern `YEAR` und `MONTH`. Geben Sie `INTERVAL YEAR TO MONTH` wie folgt an:

`INTERVAL YEAR [(year_precision)] TO MONTH`

Hierbei ist `year_precision` die Anzahl der Stellen im Datetime-Feld `YEAR`. Der Standardwert für `year_precision` ist 2.

Einschränkung: Das größere Feld muss dem kleineren Feld vorausgehen. Beispielsweise ist `INTERVAL '0-1' MONTH TO YEAR` ungültig.

Beispiele

- `INTERVAL '123-2' YEAR(3) TO MONTH`
Zeigt ein Intervall von 123 Jahren, 2 Monaten an
- `INTERVAL '123' YEAR(3)`
Zeigt ein Intervall von 123 Jahren, 0 Monaten an
- `INTERVAL '300' MONTH(3)`
Zeigt ein Intervall von 300 Monaten an
- `INTERVAL '123' YEAR`
Gibt einen Fehler zurück, da standardmäßig nur zwei Stellen zulässig sind und "123" drei Stellen enthält

Die Oracle-Datenbank unterstützt zwei Intervalldatentypen: `INTERVAL YEAR TO MONTH` und `INTERVAL DAY TO SECOND`. Spaltentyp, PL/SQL-Argument, Variable und Rückgabebetyp müssen einem dieser beiden Intervalldatentypen entsprechen. Für Intervalllitterale erkennt das System aber auch andere Intervalltypen vom American National Standards Institute (ANSI) wie `INTERVAL '2' YEAR` und `INTERVAL '10' HOUR`. In diesen Fällen werden die Intervalle in einen der beiden unterstützten Typen konvertiert.

Im Beispiel auf der Folie wird die Tabelle `WARRANTY` erstellt. Sie enthält die Spalte `warranty_time`, die Werte vom Datentyp `INTERVAL YEAR(3) TO MONTH` annimmt. Um die Jahre und Monate für verschiedene Produkte anzugeben, werden verschiedene Werte eingefügt. Beim Abrufen dieser Zeilen aus der Tabelle werden Jahres- und Monatswert durch ein Minuszeichen (-) voneinander getrennt angezeigt.

Datentyp INTERVAL DAY TO SECOND – Beispiel

```
CREATE TABLE lab
( exp_id number, test_time INTERVAL DAY(2) TO SECOND);

INSERT INTO lab VALUES (100012, '90 00:00:00');
INSERT INTO lab VALUES (56098,
    INTERVAL '6 03:30:16' DAY TO SECOND);
```

```
SELECT * FROM lab;
```

	EXP_ID	TEST_TIME
1	10001290	0:0:0.0
2	560986	3:30:16.0

ORACLE

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

Im Beispiel auf der Folie erstellen Sie die Tabelle `lab` mit einer `test_time` vom Datentyp `INTERVAL DAY TO SECOND`. Anschließend wird der Wert `'90 00:00:00'` eingefügt, der 90 Tage und 0 Stunden, 0 Minuten und 0 Sekunden angibt, sowie das Intervall `INTERVAL '6 03:30:16' DAY TO SECOND`, das 6 Tage, 3 Stunden, 30 Minuten und 16 Sekunden angibt. Die Anweisung `SELECT` zeigt, wie diese Daten in der Datenbank ausgegeben werden.

Lektionsagenda

- CURRENT_DATE, CURRENT_TIMESTAMP und LOCALTIMESTAMP
- INTERVAL-Datentypen
- Folgende Funktionen verwenden:
 - EXTRACT
 - TZ_OFFSET
 - FROM_TZ
 - TO_TIMESTAMP
 - TO_YMINTERVAL
 - TO_DSINTERVAL

ORACLE®

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

EXTRACT

- Alle Mitarbeiter anzeigen, die nach 2007 eingestellt wurden:

```
SELECT last_name, employee_id, hire_date
FROM employees
WHERE EXTRACT(YEAR FROM TO_DATE(hire_date, 'DD-MON-RR')) > 2007
ORDER BY hire_date;
```

- Die Komponente MONTH aus HIRE_DATE für Mitarbeiter mit der MANAGER_ID 100 anzeigen:

```
SELECT last_name, hire_date,
       EXTRACT (MONTH FROM HIRE_DATE)
FROM employees
WHERE manager_id = 100;
```

	LAST_NAME	HIRE_DATE	EXTRACT(MONTHFROMHIRE_DATE)
1	Kochhar	21-SEP-05	9
2	De Haan	13-JAN-01	1
3	Raphaely	07-DEC-02	12
4	Weiss	18-JUL-04	7
5	Fripp	10-APR-05	4
6	Kaufling	01-MAY-03	5

...

ORACLE

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

Der Ausdruck `EXTRACT` extrahiert den Wert eines angegebenen Datetime-Feldes aus einem Datetime- oder Intervallwert-Ausdruck und gibt ihn zurück. Mit der Funktion `EXTRACT` können Sie jede der in der folgenden Syntax enthaltenen Komponenten extrahieren. Die Syntax der Funktion `EXTRACT` lautet:

```
SELECT EXTRACT( { YEAR | MONTH | DAY | HOUR | MINUTE | SECOND
                | TIMEZONE_HOUR
                | TIMEZONE_MINUTE
                | TIMEZONE_REGION
                | TIMEZONE_ABBR }
FROM { expr } )
```

Wenn Sie die Komponente `TIMEZONE_REGION` oder `TIMEZONE_ABBR` (Abkürzung) extrahieren, ist der Rückgabewert eine Zeichenfolge, die den Namen oder die Abkürzung der entsprechenden Zeitzone enthält. Wenn Sie einen der anderen Werte extrahieren, entspricht der Rückgabewert einem Datum im gregorianischen Kalender. Wenn Sie Werte aus einem Datetime-Ausdruck mit einem Zeitzone-Wert extrahieren, wird ein UTC-Wert zurückgegeben.

Im ersten Beispiel auf der Folie wählt die Funktion `EXTRACT` alle Mitarbeiter, die nach 2007 eingestellt wurden. Im zweiten Beispiel extrahiert die Funktion `EXTRACT` für alle Mitarbeiter, die dem Manager mit der `EMPLOYEE_ID` 100 unterstellt sind, den Monat des Einstellungsdatums aus der Tabelle `EMPLOYEES`.

TZ_OFFSET

Zeitzonendifferenz für die Zeitzonen 'US/Eastern',
'Canada/Yukon' and 'Europe/London' anzeigen:

```
SELECT TZ_OFFSET('US/Eastern'),  
       TZ_OFFSET('Canada/Yukon'),  
       TZ_OFFSET('Europe/London')  
FROM DUAL;
```

	TZ_OFFSET('US/EASTERN')	TZ_OFFSET('CANADA/YUKON')	TZ_OFFSET('EUROPE/LONDON')
1	-04:00	-07:00	+01:00

ORACLE

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

Die Funktion `TZ_OFFSET` gibt die Zeitzonendifferenz entsprechend dem eingegebenen Wert zurück. Der Rückgabewert hängt vom Datum ab, an dem die Anweisung ausgeführt wird. Beispiel: Die Funktion `TZ_OFFSET` gibt den Wert `-08:00` zurück. Dies bedeutet, dass die Zeitzone, in der der Befehl ausgeführt wurde, acht Stunden hinter der UTC zurückliegt. Sie können einen gültigen Zeitzonennamen, eine Zeitzonendifferenz zu UTC (Eingabe und Rückgabewert sind identisch) oder das Schlüsselwort `SESSIONTIMEZONE` bzw. `DBTIMEZONE` eingeben. Die Syntax der Funktion `TZ_OFFSET` lautet:




```
TZ_OFFSET({ 'time_zone_name' | '{ + | - } hh : mi'  
          | SESSIONTIMEZONE  
          | DBTIMEZONE  
          })
```

Das Unternehmen Fold Motor Company hat seinen Hauptsitz in Michigan, USA, das zur Zeitzone "US/Eastern" gehört. Der Firmenchef, Mr. Fold, möchte eine Telefonkonferenz mit den Leitern der Geschäftsbereiche Kanada und Europa abhalten, die jeweils zu den Zeitzonen "Canada/Yukon" und "Europe/London" gehören. Mr. Fold möchte die Zeit an diesen Orten wissen, um sicherzustellen, dass seine Senior Manager an der Konferenz teilnehmen können. Sein Sekretär, Mr. Scott, unterstützt ihn und führt die im Beispiel gezeigten Abfragen aus. Er erhält folgende Ergebnisse:

- Die Zeitzone 'US/Eastern' liegt vier Stunden hinter der UTC.
- Die Zeitzone 'Canada/Yukon' liegt sieben Stunden hinter der UTC.
- Die Zeitzone 'Europe/London' liegt eine Stunde vor der UTC.

Um eine Liste der gültigen Werte für Zeitzonennamen anzuzeigen, können Sie die dynamische Performance-View `V$TIMEZONE_NAMES` abfragen.

```
SELECT * FROM V$TIMEZONE_NAMES;
```

	 TZNAME	 TZABBREV	 CON_ID
1	Africa/Abidjan	LMT	0
2	Africa/Abidjan	GMT	0
3	Africa/Accra	LMT	0
4	Africa/Accra	GMT	0
5	Africa/Accra	GHST	0

...

FROM_TZ

TIMESTAMP-Wert '2000-07-12 08:00:00' als TIMESTAMP WITH TIME ZONE-Wert für die Zeitzone region 'Australia/North' anzeigen:

```
SELECT FROM_TZ(TIMESTAMP
                '2000-07-12 08:00:00', 'Australia/North')
FROM DUAL;
```

FROM_TZ(TIMESTAMP'2000-07-12 08:00:00','AUSTRALIA/NORTH')
1 12-JUL-00 08.00.00.000000000 AM AUSTRALIA/NORTH

ORACLE

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

Die Funktion `FROM_TZ` konvertiert einen `TIMESTAMP`-Wert in einen `TIMESTAMP WITH TIME ZONE`-Wert.

Die Syntax der Funktion `FROM_TZ` lautet:

```
FROM_TZ(timestamp_value, time_zone_value)
```

Hierbei gilt: `time_zone_value` ist eine Zeichenfolge im Format 'TZH:TZM' oder ein Zeichen Ausdruck, der eine Zeichenfolge in TZR (Zeitzone region) mit optionalem Format TZD zurückgibt. TZD ist eine abgekürzte Zeichenfolge für eine Zeitzone mit Sommerzeitinformationen. TZR stellt die Zeitzone region in Datetime-Eingabezeichenfolgen dar. Beispiele: 'Australia/North', 'PST' für Pazifische Standardzeit (USA), 'PDT' für Pazifische Sommerzeit (USA) und so weiter.


Im Beispiel auf der Folie wird ein `TIMESTAMP`-Wert in einen `TIMESTAMP WITH TIME ZONE`-Wert konvertiert.

Hinweis: Um eine Liste der gültigen Werte für die Formatelemente TZR und TZD anzuzeigen, fragen Sie die dynamische Performance-View `V$TIMEZONE_NAMES` ab.

TO_TIMESTAMP

Zeichenfolge '2007-03-06 11:00:00' als TIMESTAMP-Wert anzeigen:

```
SELECT TO_TIMESTAMP ('2007-03-06 11:00:00',  
                     'YYYY-MM-DD HH:MI:SS')  
FROM DUAL;
```

 TO_TIMESTAMP('2007-03-06 11:00:00','YYYY-MM-DD HH:MI:SS')
1 06-MAR-07 11.00.00.000000000 AM

ORACLE

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

Die Funktion `TO_TIMESTAMP` konvertiert eine Zeichenfolge vom Datentyp `CHAR`, `VARCHAR2`, `NCHAR` oder `NVARCHAR2` in einen Wert vom Datentyp `TIMESTAMP`. Die Syntax der Funktion `TO_TIMESTAMP` lautet:

```
TO_TIMESTAMP(char [, fmt [, 'nlsparam' ] ])
```

Die optionale Formatmaske `fmt` gibt das Format von `char` an. Wenn Sie `fmt` nicht angeben, muss die Zeichenfolge das Standardformat vom Datentyp `TIMESTAMP` aufweisen. Der optionale Parameter `nlsparam` gibt die Sprache an, in der die Namen der Monate und Tage sowie deren Abkürzungen zurückgegeben werden. Die Syntax des Arguments kann folgendermaßen lauten:

```
'NLS_DATE_LANGUAGE = language'
```

Wenn Sie `nlsparams` nicht angeben, verwendet die Funktion die Standarddatumssprache für Ihre Session.

Im Beispiel auf der Folie wird eine Zeichenfolge in einen Wert vom Datentyp `TIMESTAMP` konvertiert.

Hinweis: Mit der Funktion `TO_TIMESTAMP_TZ` konvertieren Sie eine Zeichenfolge vom Datentyp `CHAR`, `VARCHAR2`, `NCHAR` oder `NVARCHAR2` in einen Wert vom Datentyp `TIMESTAMP WITH TIME ZONE`. Weitere Informationen zu dieser Funktion finden Sie im Handbuch *Oracle Database SQL Language Reference* für Oracle Database 12c.

TO_YMINTERVAL

Für Mitarbeiter der Abteilung mit `DEPARTMENT_ID` 20 das Datum anzeigen, das ein Jahr und zwei Monate nach dem Einstellungsdatum der einzelnen Mitarbeiter liegt:

```
SELECT hire_date,  
       hire_date + TO_YMINTERVAL('01-02') AS  
       HIRE_DATE_YMININTERVAL  
FROM   employees  
WHERE  department_id = 20;
```

	HIRE_DATE	HIRE_DATE_YMININTERVAL
1	17-FEB-04	17-APR-05
2	17-AUG-05	17-OCT-06

ORACLE

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

Die Funktion `TO_YMINTERVAL` konvertiert eine Zeichenfolge vom Datentyp `CHAR`, `VARCHAR2`, `NCHAR` oder `NVARCHAR2` in den Datentyp `INTERVAL YEAR TO MONTH`. Der Datentyp `INTERVAL YEAR TO MONTH` speichert einen Zeitraum mit den Datetime-Feldern `YEAR` und `MONTH`. Die Syntax von `INTERVAL YEAR TO MONTH` lautet wie folgt:

```
INTERVAL YEAR [(year_precision)] TO MONTH
```

Hierbei ist `year_precision` die Anzahl der Stellen im Datetime-Feld `YEAR`. Der Standardwert von `year_precision` lautet 2.

Die Syntax der Funktion `TO_YMINTERVAL` lautet:

```
TO_YMINTERVAL (char)
```

Hierbei ist `char` die zu konvertierende Zeichenfolge.

Im Beispiel auf der Folie wird ein Datum berechnet, das ein Jahr und zwei Monate nach dem Einstellungsdatum der Mitarbeiter aus Abteilung 20 der Tabelle `EMPLOYEES` liegt.

TO_DSINTERVAL

Datum anzeigen, das 100 Tage und 10 Stunden nach dem Einstellungsdatum der einzelnen Mitarbeiter liegt:

```
SELECT last_name,  
       TO_CHAR(hire_date, 'mm-dd-yy:hh:mi:ss') hire_date,  
       TO_CHAR(hire_date +  
               TO_DSINTERVAL('100 10:00:00'),  
               'mm-dd-yy:hh:mi:ss') hiredate2  
FROM employees;
```

	LAST_NAME	HIRE_DATE	HIREDATE2
1	King	06-17-03:12:00:00	09-25-03:10:00:00
2	Kochhar	09-21-05:12:00:00	12-30-05:10:00:00
3	De Haan	01-13-01:12:00:00	04-23-01:10:00:00
4	Hunold	01-03-06:12:00:00	04-13-06:10:00:00
5	Ernst	05-21-07:12:00:00	08-29-07:10:00:00
6	Austin	06-25-05:12:00:00	10-03-05:10:00:00
7	Pataballa	02-05-06:12:00:00	05-16-06:10:00:00
8	Lorentz	02-07-07:12:00:00	05-18-07:10:00:00
9	Greenberg	08-17-02:12:00:00	11-25-02:10:00:00
10	Faviet	08-16-02:12:00:00	11-24-02:10:00:00
11	Chen	09-28-05:12:00:00	01-06-06:10:00:00

...

ORACLE

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

TO_DSINTERVAL konvertiert eine Zeichenfolge vom Datentyp CHAR, VARCHAR2, NCHAR oder NVARCHAR2 in den Datentyp INTERVAL DAY TO SECOND.

Im Beispiel auf der Folie wird das Datum ausgegeben, das 100 Tage und 10 Stunden nach dem Einstellungsdatum liegt.

Sommerzeit (DST)

- Beginn der Sommerzeit:
 - Die Uhrzeit wird von 01:59:59 auf 03:00:00 umgestellt.
 - Werte von 02:00:00 bis 02:59:59 sind nicht gültig.
- Ende der Sommerzeit:
 - Die Uhrzeit wird von 02:00:00 auf 01:00:01 umgestellt.
 - Werte zwischen 01:00:01 und 02:00:00 sind nicht eindeutig, da dieses Zeitintervall wiederholt wird.

ORACLE

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

In den meisten westlichen Ländern wird die Uhr in den Sommermonaten eine Stunde vorgestellt. Dieser Zeitraum wird als Sommerzeit bezeichnet. In den meisten Teilen der USA, Mexikos und Kanadas dauert die Sommerzeit vom ersten Sonntag im April bis zum letzten Sonntag im Oktober. In den Staaten der Europäischen Union gilt ebenfalls die Sommerzeit. In Europa beginnt die Sommerzeit eine Woche früher als in Nordamerika, sie endet jedoch zum selben Zeitpunkt.

Oracle Database ermittelt automatisch für beliebige Zeitzonenregionen, ob die Sommerzeit gilt, und gibt entsprechende lokale Zeitwerte zurück. Außer in Grenzfällen benötigt Oracle Database nur den Datetime-Wert, um festzustellen, ob in einer bestimmten Region Sommerzeit gilt. Grenzfälle treten auf, wenn die Sommerzeit gerade beginnt oder endet. Beispiel: Wenn in Deutschland die Sommerzeit in Kraft tritt, ändert sich die Uhrzeit von 01:59:59 auf 03:00:00. Die Stunde zwischen 02:00:00 und 02:59:59 existiert dann nicht. Am Ende der Sommerzeit wird die Uhr von 02:00:00 auf 01:00:01 zurückgestellt, und das Stundenintervall zwischen 01:00:01 und 02:00:00 wird wiederholt.

ERROR_ON_OVERLAP_TIME

`ERROR_ON_OVERLAP_TIME` ist ein Sessionparameter. Er teilt dem System mit, dass ein Fehler ausgegeben werden soll, wenn ein Datetime-Wert erkannt wird, der in dem sich überschneidenden Zeitraum liegt und kein Zeitzonekürzel zur Unterscheidung des Zeitraums angegeben wurde.

Beispiel: Wenn die Sommerzeit am 31. Oktober um 02:00:01 Uhr endet, überschneiden sich folgende Zeiträume:

- 31/10/2009 01:00:01 bis 31/10/2009 02:00:00 (EDT)
- 31/10/2004 01:00:01 bis 31/10/2004 02:00:00 (EST)

Wenn Sie eine Datetime-Zeichenfolge eingeben, die in einem dieser beiden Zeiträume liegt, müssen Sie das Zeitzonekürzel (zum Beispiel EDT oder EST) in der Eingabezeichenfolge angeben, damit das System den Zeitraum bestimmen kann. Ohne Angabe des Zeitzonekürzels geht das System wie folgt vor:

Wenn der Parameter `ERROR_ON_OVERLAP_TIME` den Wert `FALSE` hat, geht das System davon aus, dass es sich bei der eingegebenen Zeit um die Standardzeit handelt (zum Beispiel EST). Andernfalls wird ein Fehler ausgegeben.

Quiz

Der Sessionparameter `TIME_ZONE` kann wie folgt festgelegt werden:

- a. Relative Differenz
- b. Zeitzone der Datenbank
- c. Lokale Zeitzone des Betriebssystems
- d. Benannte Region

ORACLE®

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

Richtige Antworten: b, c, d

Zusammenfassung

In dieser Lektion haben Sie Folgendes gelernt:

- DATE-ähnliche Datentypen verwenden, die Sekundenbruchteile speichern und Zeitzonen überwachen
- Datentypen verwenden, die die Differenz zwischen zwei Datetime-Werten speichern
- Die folgenden Datetime-Funktionen verwenden:
 - CURRENT_DATE
 - CURRENT_TIMESTAMP
 - LOCALTIMESTAMP
 - DBTIMEZONE
 - SESSIONTIMEZONE
 - EXTRACT
 - TZ_OFFSET
 - FROM_TZ
 - TO_TIMESTAMP
 - TO_YMINTERVAL
 - TO_DSINTERVAL

ORACLE

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

In dieser Lektion wurden einige Datetime-Funktionen beschrieben, die in der Oracle-Datenbank verfügbar sind.

Übungen zu Lektion 10 – Überblick

Diese Übung behandelt die Verwendung der Datetime-Funktionen.

ORACLE®

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

In dieser Übung zeigen Sie Zeitzonendifferenzen sowie `CURRENT_DATE`, `CURRENT_TIMESTAMP` und `LOCALTIMESTAMP` an. Außerdem stellen Sie Zeitzonen ein und verwenden die Funktion `EXTRACT`.

