ORACLE
UNIVERSITY

Oracle Database 12*c*: SQL Workshop II

Schulungsunterlagen – Band II

D80194DE11

Production 1.1 | Dezember 2014 | D88606

Learn more from Oracle University at **oracle.com/education/**

ORACLE

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

Inhalt

Tabellenbeschreibungen

Schemabeschreibung

Allgemeine Beschreibung

Das in den Beispielschemas von Oracle Database dargestellte Beispielunternehmen operiert weltweit und liefert viele unterschiedliche Produkte aus. Das Unternehmen hat drei Abteilungen:

- **Human Resources:** Überwacht Informationen zu Mitarbeitern und Einrichtungen
- **Order Entry:** Überwacht den Bestand und den Verkauf der Produkte über verschiedene Kanäle
- **Sales History:** Überwacht Unternehmensstatistiken, um Geschäftsentscheidungen zu vereinfachen

Jede dieser Abteilungen wird durch ein Schema dargestellt. In diesem Kurs hat der Benutzer Zugriff auf die Objekte in sämtlichen Schemas. Der Schwerpunkt in den Beispielen, Demonstrationen und Übungen liegt jedoch auf dem Schema `Human Resources` (`HR`).

Alle für die Erstellung der Beispielschemas erforderlichen Skripte befinden sich im Ordner `$ORACLE_HOME/demo/schema/`.

`Human Resources (HR)`

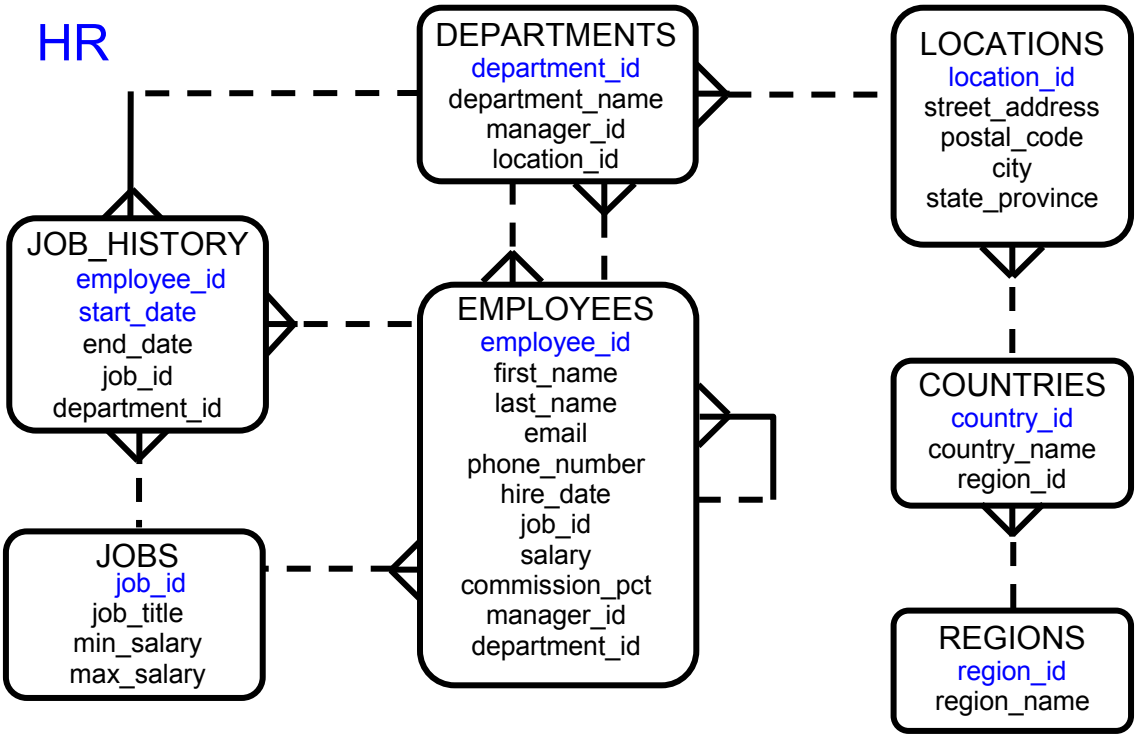
Dieses Schema wird im aktuellen Kurs verwendet. In den Datensätzen des Schemas `Human Resources` (`HR`) ist jeder Mitarbeiter mit einer ID-Nummer, einer E-Mail-Adresse, einem Tätigkeits-ID-Code, einem Gehalt und einem Manager verzeichnet. Neben ihrem Gehalt bekommen einige Mitarbeiter Provisionen.

Das Unternehmen erfasst außerdem Informationen über Tätigkeiten innerhalb des Unternehmens. Jede Tätigkeit ist durch eine ID und eine Bezeichnung gekennzeichnet. Außerdem werden Angaben zum Mindest- und Höchstgehalt für die Tätigkeit aufgezeichnet. Einige Mitarbeiter sind schon sehr lange im Unternehmen und hatten verschiedene Positionen inne. Wenn ein Mitarbeiter ausscheidet, werden seine Beschäftigungsdauer, die Tätigkeits-ID und die Abteilung aufgezeichnet.

Die Beispielunternehmen ist an verschiedenen Standorten tätig. Daher werden die Lager- und Abteilungsstandorte überwacht. Jeder Mitarbeiter ist einer Abteilung zugewiesen. Jede Abteilung wird entweder anhand einer eindeutigen Abteilungsnummer oder anhand einer Kurzbezeichnung identifiziert. Jede Abteilung ist einem Standort zugewiesen. Jeder Standort besitzt eine vollständige Adresse mit Straßennamen, Postleitzahl, Ort, Bundesland, Bundesstaat, Provinz oder Kanton sowie Länderkennung.

An den Abteilungs- und Lagerstandorten zeichnet das Unternehmen Details wie den Namen des Landes, das Währungssymbol, den Namen der Währung sowie die Region auf, in der das Land geografisch angesiedelt ist.

`HR` – Entity-Relationship-Diagramm



Human Resources (HR) – Tabellenbeschreibungen

```
DESCRIBE countries
```

```
Name                           Null     Type
------------------------------ -------- -------------
COUNTRY_ID                     NOT NULL CHAR(2)
COUNTRY_NAME                            VARCHAR2(40)
REGION_ID                               NUMBER
```

```
SELECT * FROM countries
```

	COUNTRY_ID	COUNTRY_NAME	REGION_ID
1	CA	Canada	2
2	DE	Germany	1
3	UK	United Kingdom	1
4	US	United States of America	2

```
DESCRIBE departments
```

```
Name                          Null     Type
----------------------------- -------- -------------
DEPARTMENT_ID                 NOT NULL NUMBER(4)
DEPARTMENT_NAME               NOT NULL VARCHAR2(30)
MANAGER_ID                             NUMBER(6)
LOCATION_ID                            NUMBER(4)
```

```
SELECT * FROM departments
```

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

```
DESCRIBE employees
```

```
Name                           Null     Type
------------------------------ -------- ----------------
EMPLOYEE_ID                    NOT NULL NUMBER(6)
FIRST_NAME                              VARCHAR2(20)
LAST_NAME                      NOT NULL VARCHAR2(25)
EMAIL                          NOT NULL VARCHAR2(25)
PHONE_NUMBER                            VARCHAR2(20)
HIRE_DATE                      NOT NULL DATE
JOB_ID                         NOT NULL VARCHAR2(10)
SALARY                                  NUMBER(8,2)
COMMISSION_PCT                          NUMBER(2,2)
MANAGER_ID                              NUMBER(6)
DEPARTMENT_ID                           NUMBER(4)
```

```
SELECT * FROM employees
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
100	Steven	King	SKING	515.123.4567	17-JUN-03	AD_PRES	24000	(null)	(null)	90
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-05	AD_VP	17000	(null)	100	90
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-01	AD_VP	17000	(null)	100	90
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-06	IT_PROG	9000	(null)	102	60
104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-07	IT_PROG	6000	(null)	103	60
107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-07	IT_PROG	4200	(null)	103	60
124	Kevin	Mourgos	KMOURGOS	650.123.5234	16-NOV-07	ST_MAN	5800	(null)	100	50
141	Trenna	Rajs	TRAJS	650.121.8009	17-OCT-03	ST_CLERK	3500	(null)	124	50
142	Curtis	Davies	CDAVIES	650.121.2994	29-JAN-05	ST_CLERK	3100	(null)	124	50
143	Randall	Matos	RMATOS	650.121.2874	15-MAR-06	ST_CLERK	2600	(null)	124	50
144	Peter	Vargas	PVARGAS	650.121.2004	09-JUL-06	ST_CLERK	2500	(null)	124	50
149	Eleni	Zlotkey	EZLOTKEY	011.44.1344.429018	29-JAN-08	SA_MAN	10500	0.2	100	80
174	Ellen	Abel	EABEL	011.44.1644.429267	11-MAY-04	SA_REP	11000	0.3	149	80
176	Jonathon	Taylor	JTAYLOR	011.44.1644.429265	24-MAR-06	SA_REP	8600	0.2	149	80
178	Kimberely	Grant	KGRANT	011.44.1644.429263	24-MAY-07	SA_REP	7000	0.15	149	(null)
200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-03	AD_ASST	4400	(null)	101	10
201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-04	MK_MAN	13000	(null)	100	20
202	Pat	Fay	PFAY	603.123.6666	17-AUG-05	MK_REP	6000	(null)	201	20
205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-02	AC_MGR	12008	(null)	101	110
206	William	Gietz	WGIETZ	515.123.8181	07-JUN-02	AC_ACCOUNT	8300	(null)	205	110

```
DESCRIBE job_history
```

```
Name                          Null     Type
----------------------------- -------- -------------
EMPLOYEE_ID                   NOT NULL NUMBER(6)
START_DATE                    NOT NULL DATE
END_DATE                      NOT NULL DATE
JOB_ID                        NOT NULL VARCHAR2(10)
DEPARTMENT_ID                          NUMBER(4)
```

```
SELECT * FROM job_history
```

	EMPLOYEE_ID	START_DATE	END_DATE	JOB_ID	DEPARTMENT_ID
1	102	13-JAN-01	24-JUL-06	IT_PROG	60
2	101	21-SEP-97	27-OCT-01	AC_ACCOUNT	110
3	101	28-OCT-01	15-MAR-05	AC_MGR	110
4	201	17-FEB-04	19-DEC-07	MK_REP	20
5	114	24-MAR-06	31-DEC-07	ST_CLERK	50
6	122	01-JAN-07	31-DEC-07	ST_CLERK	50
7	200	17-SEP-95	17-JUN-01	AD_ASST	90
8	176	24-MAR-06	31-DEC-06	SA_REP	80
9	176	01-JAN-07	31-DEC-07	SA_MAN	80
10	200	01-JUL-02	31-DEC-06	AC_ACCOUNT	90

```
DESCRIBE jobs
```

```
Name                          Null     Type
----------------------------- -------- --------------
JOB_ID                        NOT NULL VARCHAR2(10)
JOB_TITLE                     NOT NULL VARCHAR2(35)
MIN_SALARY                             NUMBER(6)
MAX_SALARY                             NUMBER(6)
```

```
SELECT * FROM jobs
```

	JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
1	AD_PRES	President	20080	40000
2	AD_VP	Administration Vice President	15000	30000
3	AD_ASST	Administration Assistant	3000	6000
4	AC_MGR	Accounting Manager	8200	16000
5	AC_ACCOUNT	Public Accountant	4200	9000
6	SA_MAN	Sales Manager	10000	20080
7	SA_REP	Sales Representative	6000	12008
8	ST_MAN	Stock Manager	5500	8500
9	ST_CLERK	Stock Clerk	2008	5000
10	IT_PROG	Programmer	4000	10000
11	MK_MAN	Marketing Manager	9000	15000
12	MK_REP	Marketing Representative	4000	9000

```
DESCRIBE locations
```

Name	Null	Type
LOCATION_ID	NOT NULL	NUMBER(4)
STREET_ADDRESS		VARCHAR2(40)
POSTAL_CODE		VARCHAR2(12)
CITY	NOT NULL	VARCHAR2(30)
STATE_PROVINCE		VARCHAR2(25)
COUNTRY_ID		CHAR(2)

```
SELECT * FROM locations
```

	LOCATION_ID	STREET_ADDRESS	POSTAL_CODE	CITY	STATE_PROVINCE	COUNTRY_ID
1	1400	2014 Jabberwocky Rd	26192	Southlake	Texas	US
2	1500	2011 Interiors Blvd	99236	South San Francisco	California	US
3	1700	2004 Charade Rd	98199	Seattle	Washington	US
4	1800	460 Bloor St. W.	ON M5S 1X8	Toronto	Ontario	CA
5	2500	Magdalen Centre, The Oxford Science Park	OX9 9ZB	Oxford	Oxford	UK

```
DESCRIBE regions
```

```
Name                          Null     Type
----------------------------- -------- --------------
REGION_ID                     NOT NULL NUMBER
REGION_NAME                            VARCHAR2(25)
```

```
SELECT * FROM regions
```

	REGION_ID	REGION_NAME
1	1	Europe
2	2	Americas
3	3	Asia
4	4	Middle East and Africa

B

SQL Developer

Ziele

Nach Ablauf dieses Anhangs haben Sie folgende Ziele erreicht:

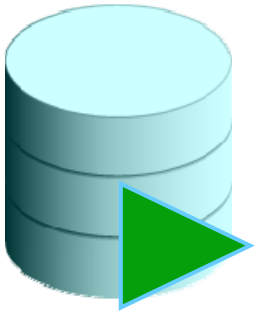
- Schlüsselfeatures von Oracle SQL Developer auflisten
- Menüoptionen von Oracle SQL Developer angeben
- Datenbankverbindungen erstellen
- Datenbankobjekte verwalten
- SQL Worksheet verwenden
- SQL-Skripte speichern und ausführen
- Berichte erstellen und speichern
- Data Modeler-Optionen in SQL Developer durchsuchen

ORACLE

In diesem Anhang wird das grafische Tool SQL Developer vorgestellt. Sie lernen, wie Sie SQL Developer für Aufgaben im Bereich der Datenbankentwicklung verwenden. Außerdem wird beschrieben, wie Sie SQL-Anweisungen und SQL-Skripte mit dem SQL Worksheet ausführen.

Was ist Oracle SQL Developer?

- Oracle SQL Developer ist ein grafisches Tool, das die Produktivität erhöht und Aufgaben der Datenbankentwicklung vereinfacht.
- Sie können sich mit der standardmäßigen Oracle-Datenbankauthentifizierung bei jedem Oracle-Zieldatenbankschema anmelden.



SQL Developer

Oracle SQL Developer ist ein kostenloses grafisches Tool, das Ihre Produktivität steigert und Routineaufgaben der Datenbankentwicklung vereinfacht. Mit wenigen Mausklicks können Sie problemlos Stored Procedures erstellen und debuggen, SQL-Anweisungen testen und Optimizer-Pläne anzeigen.

Mit SQL Developer, dem visuellen Tool zur Datenbankentwicklung, werden folgende Aufgaben vereinfacht:

- Datenbankobjekte durchsuchen und verwalten
- SQL-Anweisungen und -Skripte ausführen
- PL/SQL-Anweisungen bearbeiten und debuggen
- Berichte erstellen

Mit der standardmäßigen Oracle-Datenbankauthentifizierung können Sie sich bei jedem Oracle-Zieldatenbankschema anmelden. Nach der Anmeldung können Sie Vorgänge für die Objekte in der Datenbank ausführen.

SQL Developer ist die Verwaltungsschnittstelle für den Oracle Application Express Listener. Über die neue Schnittstelle können Sie globale Einstellungen und Einstellungen für mehrere Datenbanken mit unterschiedlichen Datenbankverbindungen für den Application Express Listener festlegen. Objekte lassen sich in SQL Developer nach Tabellen- oder Spaltenname mit Drag & Drop in das Worksheet ziehen. Neben verbesserten DB Diff-Vergleichsoptionen werden auch `GRANT`-Anweisungen im SQL-Editor und DB Doc-Berichte unterstützt. Zusätzlich unterstützt SQL Developer Funktionen aus Oracle Database 12*c*.

SQL Developer – Spezifikationen

- Im Lieferumfang von Oracle Database 12*c* Release 1 enthalten
- In Java entwickelt
- Unterstützt die Plattformen Windows, Linux und Mac OS X
- Standardkonnektivität durch den JDBC-Thin-Treiber
- Anmeldung bei Oracle Database Version 9.2.0.1 oder höher möglich

ORACLE

Oracle SQL Developer ist standardmäßig im Lieferumfang von Oracle Database 12*c* Release 1 enthalten. SQL Developer wurde mithilfe der Oracle JDeveloper-IDE (Integrated Development Environment, integrierte Entwicklungsumgebung) in Java entwickelt und ist damit ein plattformübergreifendes Tool, das unter Windows, Linux und Mac OS X ausgeführt werden kann.

Die Standardkonnektivität zur Datenbank erfolgt über den Java Database Connectivity-(JDBC-)Thin-Treiber, sodass kein Oracle Home erforderlich ist. Für SQL Developer benötigen Sie kein Installationsprogramm. Sie dekomprimieren einfach die heruntergeladene Datei. Mit SQL Developer können sich Benutzer bei Oracle Database 9.2.0.1 oder höher sowie bei allen Oracle-Datenbankeditionen einschließlich Express Edition anmelden.

Hinweis:

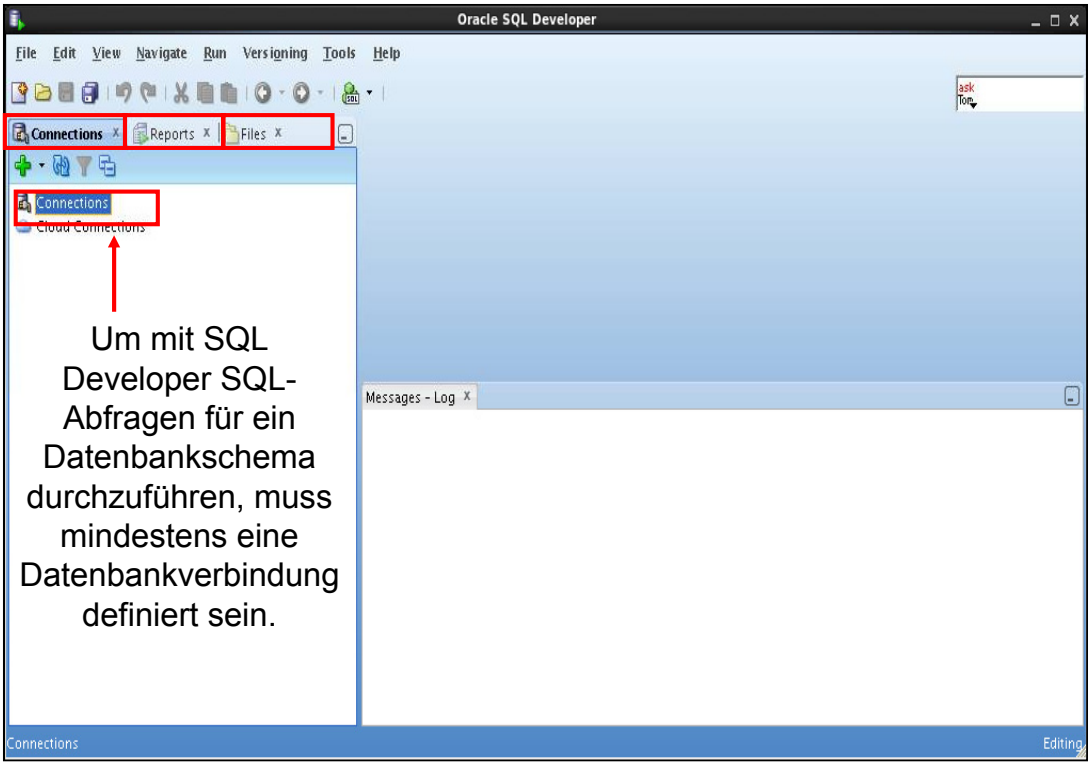
Für Oracle Database 12*c* Release 1 müssen Sie SQL Developer herunterladen und installieren. SQL Developer kann kostenlos über folgenden Link heruntergeladen werden:

http://www.oracle.com/technetwork/developer-tools/sql-developer/downloads/index.html

Anweisungen zur Installation von SQL Developer finden Sie auf der Website unter:

http://www.oracle.com/technetwork/developer-tools/sql-developer/overview/index.html

SQL Developer 3.2 – Benutzeroberfläche



Die Benutzeroberfläche von SQL Developer umfasst (von links nach rechts) drei Hauptregisterkarten zur Navigation:

- **Connections:** In dieser Registerkarte können Sie Datenbankobjekte und Benutzer durchsuchen, auf die Sie Zugriff haben.
- **Reports:** In dieser durch das Symbol **Reports** dargestellten Registerkarte führen Sie vordefinierte Berichte aus oder erstellen und fügen eigene Berichte hinzu.
- **Files:** In dieser durch das Ordnersymbol **Files** dargestellten Registerkarte können Sie von Ihrem lokalen Rechner aus auf Dateien zugreifen, ohne über das Menü **File** > **Open** navigieren zu müssen.

Allgemeine Navigation und Verwendung

Bei SQL Developer dient die linke Seite zur Navigation sowie zum Suchen und Wählen von Objekten. Auf der rechten Seite werden Informationen über die gewählten Objekte angezeigt. Über Voreinstellungen können Sie viele Aspekte der Darstellung und des Verhaltens von SQL Developer anpassen.

Hinweis: Sie müssen mindestens eine Verbindung definieren, damit Sie sich bei einem Datenbankschema anmelden und SQL-Abfragen bzw. Prozeduren oder Funktionen ausführen können.

Menüs

Folgende Menüs enthalten Standardeinträge sowie zusätzliche Einträge für spezifische Features von SQL Developer:

- **View:** Enthält Optionen, die bestimmen, was in der Benutzeroberfläche von SQL Developer angezeigt wird
- **Navigate:** Enthält Optionen für die Navigation zu verschiedenen Bereichen und die Ausführung von Unterprogrammen
- **Run:** Enthält die bei Auswahl einer Funktion oder Prozedur relevanten Optionen **Run File** und **Execution Profile** sowie Debugging-Optionen
- **Versioning:** Bietet integrierte Unterstützung für folgende Versionierungs- und Quell-kontrollsysteme: Concurrent Versions System (CVS) und Subversion
- **Tools:** Ruft SQL Developer-Tools wie SQL*Plus, Preferences oder SQL Worksheet auf und enthält Optionen für die Migration von Fremddatenbanken zu Oracle

Hinweis: Das Menü **Run** enthält auch Optionen, die relevant sind, wenn Funktionen oder Prozeduren für Debugging-Zwecke gewählt wurden.

Datenbankverbindungen erstellen

- Um SQL Developer verwenden zu können, ist mindestens eine Datenbankverbindung erforderlich.
- Sie können Verbindungen erstellen und testen für:
  - mehrere Datenbanken
  - mehrere Schemas
- SQL Developer importiert automatisch alle Verbindungen, die in der Datei `tnsnames.ora` auf dem System definiert sind.
- Verbindungen können in eine Extensible Markup Language-(XML-)Datei exportiert werden.
- Jede zusätzlich erstellte Datenbankverbindung wird in der Connections Navigator-Hierarchie aufgelistet.

ORACLE

Eine Verbindung ist ein SQL Developer-Objekt, das die nötigen Informationen angibt, mit denen Sie sich als bestimmter Benutzer bei einer bestimmten Datenbank anmelden können. Um SQL Developer zu verwenden, muss mindestens eine Datenbankverbindung vorhanden sein bzw. erstellt oder importiert werden.

Sie können Verbindungen für mehrere Datenbanken und mehrere Schemas erstellen und testen.

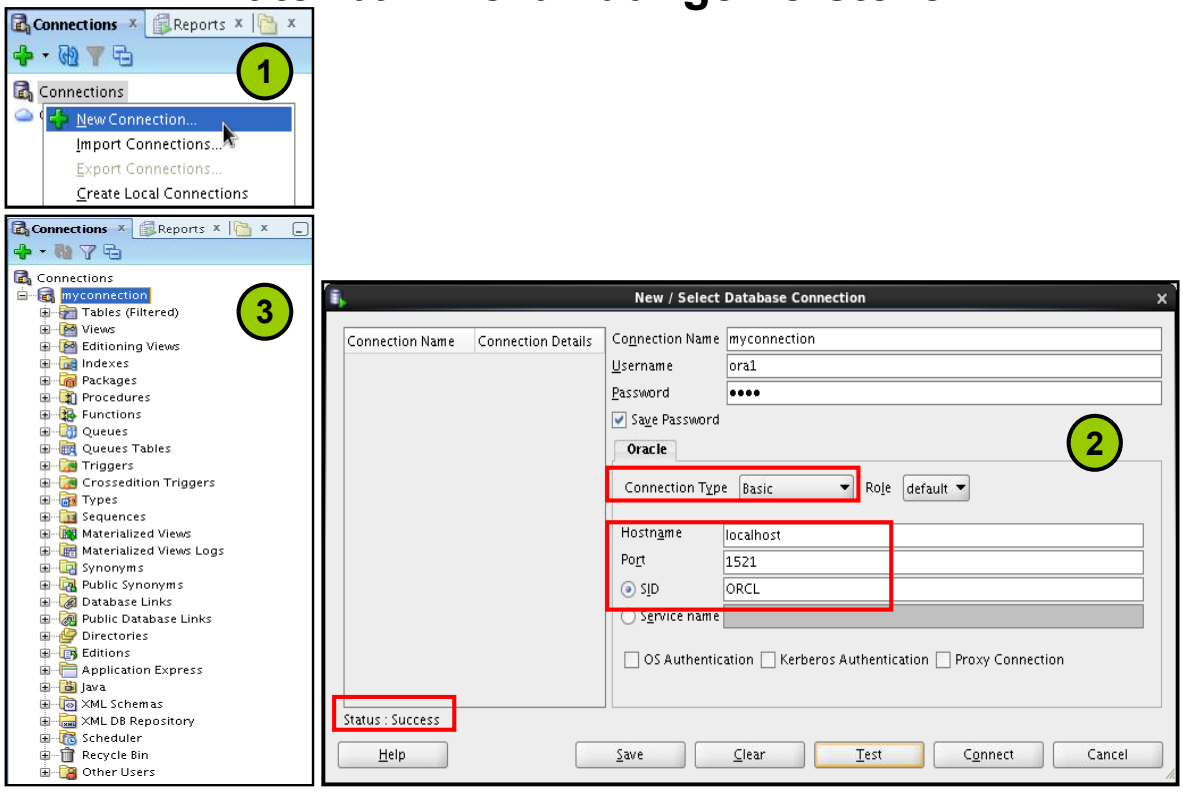
Standardmäßig befindet sich die Datei `tnsnames.ora` im Verzeichnis `$ORACLE_HOME/network/admin`. Die Datei kann jedoch auch in einem durch die Umgebungsvariable `TNS_ADMIN` oder durch einen Registrierungswert angegebenen Verzeichnis gespeichert werden. Wenn Sie SQL Developer starten und das Dialogfeld **Database Connections** anzeigen, importiert SQL Developer automatisch alle Verbindungen, die in der Datei `tnsnames.ora` auf dem System definiert sind.

Hinweis: Wenn die in der Datei `tnsnames.ora` definierten Verbindungen unter Windows nicht von SQL Developer verwendet werden, definieren Sie `TNS_ADMIN` als Systemumgebungsvariable.

Sie können Verbindungen zur späteren Wiederverwendung in eine XML-Datei exportieren.

Außerdem haben Sie die Möglichkeit, sich mehrfach als jeweils unterschiedlicher Benutzer bei derselben Datenbank anzumelden oder sich bei verschiedenen Datenbanken anzumelden.

Datenbankverbindungen erstellen



Um eine Datenbankverbindung zu erstellen, gehen Sie wie folgt vor:

1. Klicken Sie in der Registerkarte **Connections** mit der rechten Maustaste auf **Connections**, und wählen Sie **New Connection**.
2. Geben Sie im Fenster **New/Select Database Connection** den Verbindungsnamen ein. Geben Sie den Benutzernamen und das Kennwort des Schemas ein, bei dem Sie sich anmelden möchten.
   a. Wählen Sie in der Dropdown-Liste **Role** entweder *default* oder `SYSDBA`. (`SYSDBA` wird für den Benutzer `sys` oder einen Benutzer mit DBA-Berechtigungen gewählt.)
   b. Für den Verbindungstyp stehen folgende Optionen zur Wahl:

      Basic: Geben Sie für diesen Typ den Hostnamen und die SID für die Datenbank ein, bei der Sie sich anmelden möchten. **Port** ist bereits auf 1521 festgelegt. Sie können aber auch den Servicenamen direkt eingeben, wenn Sie eine Remote-Datenbankverbindung verwenden.

      TNS: Sie können jeden Datenbankalias wählen, der aus der Datei `tnsnames.ora` importiert wurde.

      LDAP: Sie können Datenbankservices in Oracle Internet Directory, einer Komponente von Oracle Identity Management, nachschlagen.

      Advanced: Sie können eine benutzerdefinierte Java Database Connectivity-(JDBC-) URL für die Anmeldung bei der Datenbank definieren.

Local/Bequeath: Befinden sich Client und Datenbank auf demselben Rechner, kann die Clientverbindung direkt an einen dedizierten Serverprozess übergeben werden, ohne dass der Listener einbezogen wird.

c. Klicken Sie auf **Test**, um sicherzustellen, dass die Verbindung richtig festgelegt wurde.

d. Klicken Sie auf **Connect**.

Bei Aktivierung des Kontrollkästchens **Save Password** wird das Kennwort in einer XML-Datei gespeichert. Wenn Sie dann die SQL Developer-Verbindung beenden und wieder öffnen, werden Sie nicht mehr aufgefordert, ein Kennwort einzugeben.

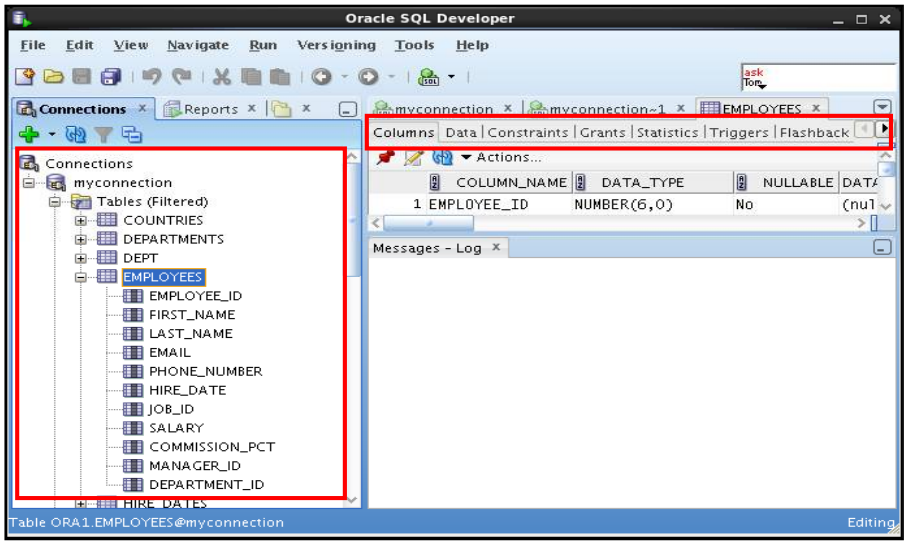
3. Die Verbindung wird dem Connections Navigator hinzugefügt. Sie können sie einblenden, um die Datenbankobjekte und Objektdefinitionen anzuzeigen, etwa Abhängigkeiten, Details und Statistiken.

Hinweis: Im Fenster **New/Select Database Connection** können Sie über die Registerkarten **Access**, **MySQL** und **SQL Server** auch Verbindungen zu Fremddatenquellen definieren. Diese Verbindungen sind jedoch Read-Only-Verbindungen, mit denen Sie nur Objekte und Daten in der jeweiligen Datenquelle durchsuchen können.

Datenbankobjekte durchsuchen

Mit dem Connections Navigator können Sie:

- viele Objekte in einem Datenbankschema durchsuchen
- Objektdefinitionen auf einen Blick prüfen



ORACLE

Wenn Sie eine Datenbankverbindung erstellt haben, können Sie mit dem Connections Navigator viele Objekte in einem Datenbankschema durchsuchen, darunter Tabellen, Views, Indizes, Packages, Prozeduren, Trigger und Typen.

Bei SQL Developer dient die linke Seite der Navigation, also dem Suchen und Wählen von Objekten. Auf der rechten Seite werden Informationen zu den gewählten Objekten angezeigt. Über Voreinstellungen können Sie zahlreiche Aspekte der Darstellung von SQL Developer anpassen.

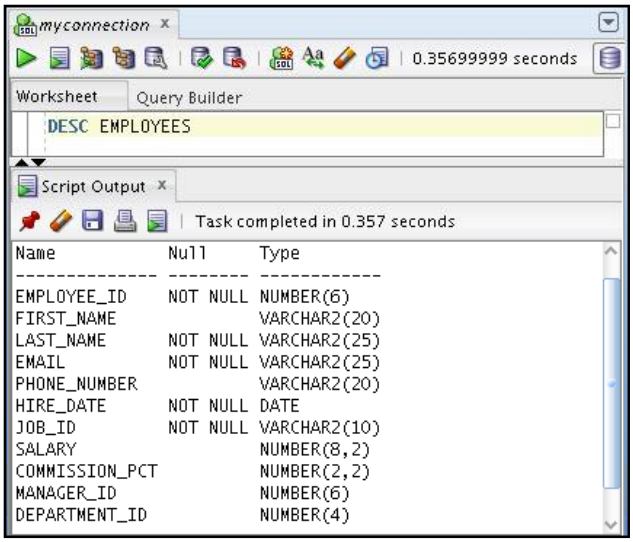
Die Definitionen der Objekte werden in verschiedenen Registerkarten angezeigt, die Informationen aus dem Data Dictionary enthalten. Wenn Sie beispielsweise eine Tabelle im Navigator wählen, werden alle Einzelheiten über Spalten, Constraints, Berechtigungen, Statistiken, Trigger usw. in einem übersichtlichen, in Registerkarten unterteilten Fenster angezeigt.

Um die Definition der Tabelle `EMPLOYEES` anzuzeigen (siehe Folie), gehen Sie wie folgt vor:

1. Blenden Sie im Connections Navigator den Knoten **Connections** ein.
2. Blenden Sie **Tables** ein.
3. Klicken Sie auf `EMPLOYEES`. Standardmäßig ist die Registerkarte **Columns** aktiviert, die die Spaltenbeschreibung der Tabelle enthält. Mit der Registerkarte **Data** können Sie die Tabellendaten anzeigen und außerdem neue Zeilen eingeben, Daten aktualisieren und diese Änderungen in der Datenbank festschreiben.

Tabellenstrukturen anzeigen

Struktur einer Tabelle mit dem Befehl `DESCRIBE` anzeigen:

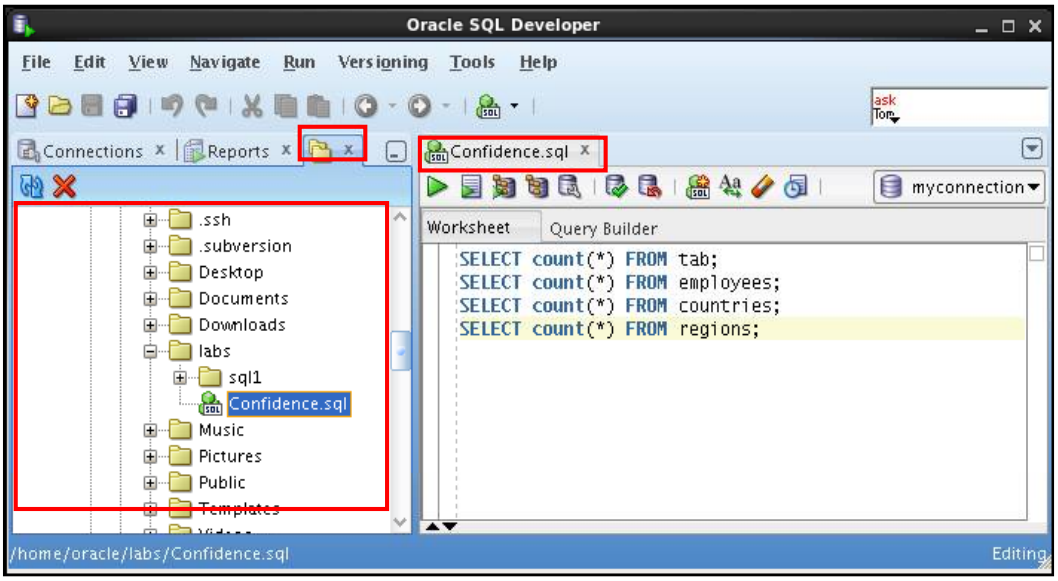


ORACLE

In SQL Developer können Sie die Struktur einer Tabelle mit dem Befehl `DESCRIBE` anzeigen. Daraufhin werden die Spaltennamen und Datentypen sowie ein Hinweis darauf angezeigt, ob eine Spalte Daten enthalten muss.

Dateien durchsuchen

Mit dem File Navigator das Dateisystem untersuchen und Systemdateien öffnen



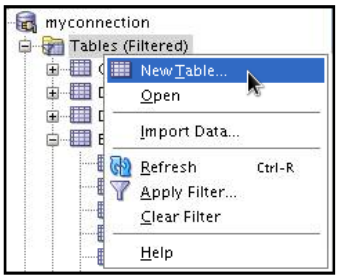
Datenbankobjekte durchsuchen

Mit dem File Navigator können Sie Systemdateien durchsuchen und öffnen.

- Um den File Navigator anzuzeigen, klicken Sie auf die Registerkarte **View** und wählen **Files**. Alternativ können Sie zu **View** > **Files** navigieren.
- Um den Inhalt einer Datei anzuzeigen, doppelklicken Sie auf einen Dateinamen. Der Inhalt der Datei wird dann im SQL Worksheet-Bereich angezeigt.

Schemaobjekte erstellen

- SQL Developer unterstützt die Erstellung beliebiger Schemaobjekte:
  - durch Ausführung einer SQL-Anweisung im SQL Worksheet
  - mithilfe des Kontextmenüs
- Objekte in einem Bearbeitungsdialogfeld oder über eines der zahlreichen Kontextmenüs bearbeiten
- DDL (Data Definition Language) für Anpassungen wie die Erstellung eines neuen Objekts oder die Bearbeitung eines vorhandenen Schemaobjekts anzeigen



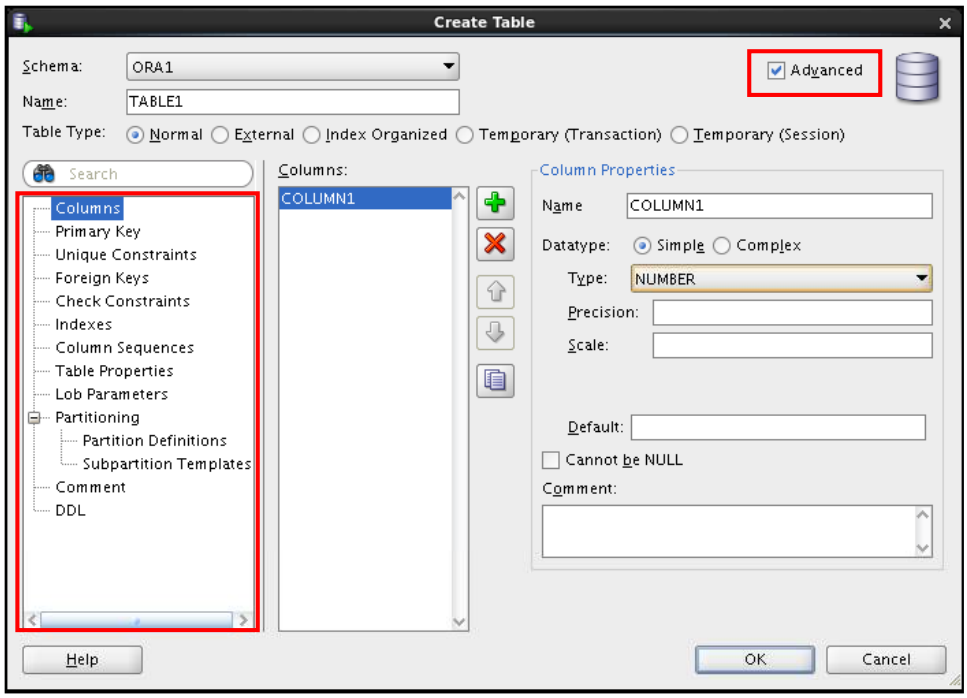
ORACLE

SQL Developer unterstützt die Erstellung beliebiger Schemaobjekte durch Ausführung einer SQL-Anweisung im SQL Worksheet. Alternativ können Sie Objekte mithilfe der Kontextmenüs erstellen. Nach ihrer Erstellung können Sie die Objekte in einem Bearbeitungsdialogfeld oder über eines der zahlreichen Kontextmenüs bearbeiten.

Bei der Erstellung neuer Objekte oder der Bearbeitung vorhandener Objekte kann für Prüfzwecke die DDL angezeigt werden. Die Option **Export DDL** ist verfügbar, wenn Sie den vollständigen DDL-Code für ein oder mehrere Objekte im Schema erstellen möchten.

Auf der Folie wird gezeigt, wie Sie eine Tabelle über das Kontextmenü erstellen. Um ein Dialogfeld zur Erstellung einer neuen Tabelle zu öffnen, klicken Sie mit der rechten Maustaste auf **Tables** und wählen **New Table**. Die Dialogfelder zum Erstellen und Bearbeiten von Datenbankobjekten verfügen über mehrere Registerkarten, die jeweils eine logische Zusammenstellung von Eigenschaften für den entsprechenden Objekttyp enthalten.

Neue Tabellen erstellen – Beispiel



Wenn Sie im Dialogfeld **Create Table** das Kontrollkästchen **Advanced** deaktivieren, können Sie schnell eine Tabelle erstellen, indem Sie Spalten und einige häufig verwendete Features angeben.

Bei Aktivierung des Kontrollkästchens **Advanced** werden im Dialogfeld **Create Table** mehrere Optionen eingeblendet, mit denen Sie beim Erstellen der Tabelle erweiterte Features festlegen können.

Das Beispiel auf der Folie zeigt, wie die Tabelle `DEPENDENTS` durch Aktivierung des Kontrollkästchens **Advanced** erstellt wird.

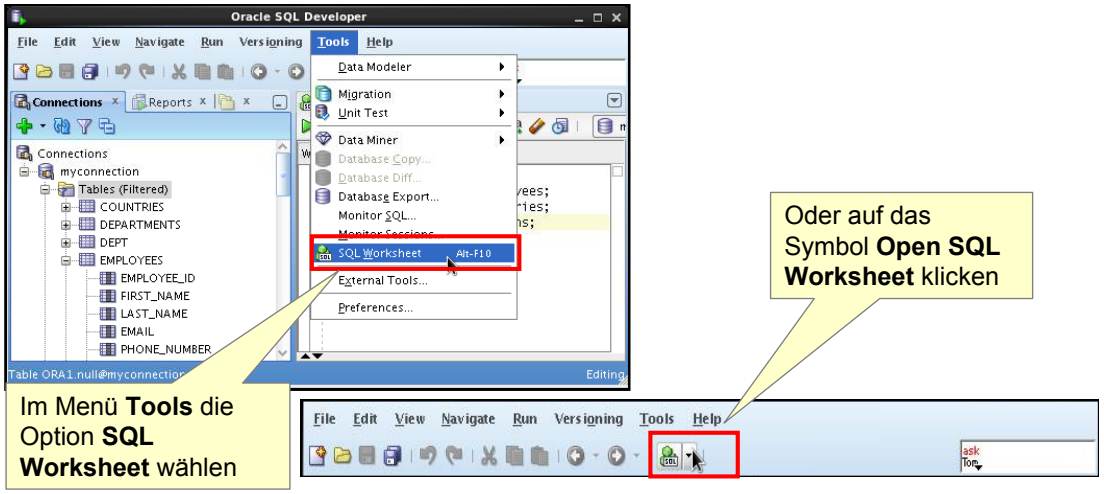
Um eine neue Tabelle zu erstellen, gehen Sie wie folgt vor:

1. Klicken Sie im Connections Navigator mit der rechten Maustaste auf **Tables**, und wählen Sie **Create TABLE**.
2. Aktivieren Sie im Dialogfeld **Create Table** das Kontrollkästchen **Advanced**.
3. Geben Sie Spalteninformationen an.
4. Klicken Sie auf **OK**.

Es empfiehlt sich, darüber hinaus in der Registerkarte **Primary Key** einen Primärschlüssel anzugeben. Um eine erstellte Tabelle zu bearbeiten, klicken Sie im Connections Navigator mit der rechten Maustaste auf die gewünschte Tabelle und wählen **Edit**.

SQL Worksheet

- Mit dem SQL Worksheet SQL-, PL/SQL- und SQL*Plus-Anweisungen eingeben und ausführen
- Aktionen angeben, die von der mit dem Worksheet verknüpften Datenbankverbindung verarbeitet werden können



Wenn Sie sich bei einer Datenbank anmelden, wird automatisch ein SQL Worksheet-Fenster für diese Verbindung geöffnet. Im SQL Worksheet können Sie SQL-, PL/SQL- und SQL*Plus-Anweisungen eingeben und ausführen. Das SQL Worksheet unterstützt nicht alle SQL*Plus-Anweisungen. Nicht unterstützte SQL*Plus-Anweisungen werden ignoriert und nicht an die Datenbank übergeben.

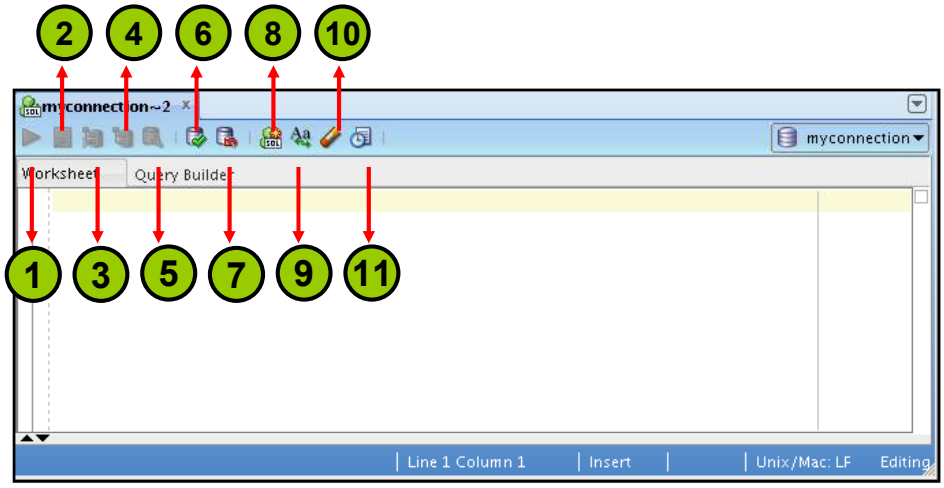
Sie können angeben, welche Aktionen die mit dem Worksheet verknüpfte Datenbankverbindung verarbeiten kann. Beispiele:

- Tabelle erstellen
- Daten einfügen
- Trigger erstellen und bearbeiten
- Daten aus einer Tabelle wählen
- Gewählte Daten in einer Datei speichern

Sie haben folgende Möglichkeiten, ein SQL Worksheet-Fenster anzuzeigen:

- Im Menü **Tools** die Option **SQL Worksheet** wählen
- Auf das Symbol **Open SQL Worksheet** klicken

SQL Worksheet



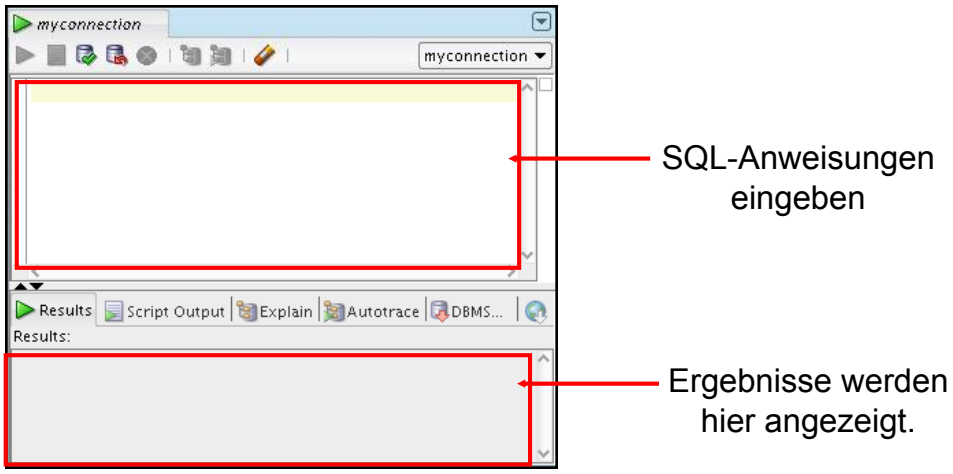
Möglicherweise möchten Sie Tastaturbefehle oder Symbole für bestimmte Aufgaben wie die Ausführung von SQL-Anweisungen und Skripten oder die Anzeige der Historie von bereits ausgeführten SQL-Anweisungen verwenden. Die Symbolleiste des SQL Worksheets enthält folgende Symbole:

1. **Run Statement:** Führt die Anweisung an der Cursorposition im Feld **Enter SQL Statement** aus. Sie können Bind-Variablen, jedoch keine Substitutionsvariablen in den SQL-Anweisungen verwenden.
2. **Run Script:** Führt mithilfe von Script Runner alle Anweisungen im Feld **Enter SQL Statement** aus. Sie können Substitutionsvariablen, jedoch keine Bind-Variablen in den SQL-Anweisungen verwenden.
3. **Autotrace:** Generiert Traceinformationen für die Anweisung
4. **Explain Plan:** Generiert den Ausführungsplan, den Sie in der Registerkarte **Explain** anzeigen können
5. **SQL Tuning Advisory:** Analysiert großvolumige SQL-Anweisungen und bietet Tuningempfehlungen
6. **Commit:** Schreibt alle Änderungen in der Datenbank fest und beendet die Transaktion
7. **Rollback:** Verwirft alle Änderungen in der Datenbank, ohne sie festzuschreiben, und beendet die Transaktion

8. **Unshared SQL Worksheet:** Erstellt ein separates, nicht gemeinsam genutztes SQL Worksheet für eine Verbindung
9. **To Upper/Lower/InitCap:** Ändert die Schreibweise des gewählten Textes in Groß- oder Kleinbuchstaben bzw. die Verwendung großer Anfangsbuchstaben
10. **Clear:** Löscht die Anweisungen im Feld **Enter SQL Statement**
11. **SQL History:** Zeigt ein Dialogfeld mit Informationen zu den bereits ausgeführten SQL-Anweisungen an

SQL Worksheet

- Mit dem SQL Worksheet SQL-, PL/SQL- und SQL*Plus-Anweisungen eingeben und ausführen
- Aktionen angeben, die von der mit dem Worksheet verknüpften Datenbankverbindung verarbeitet werden können

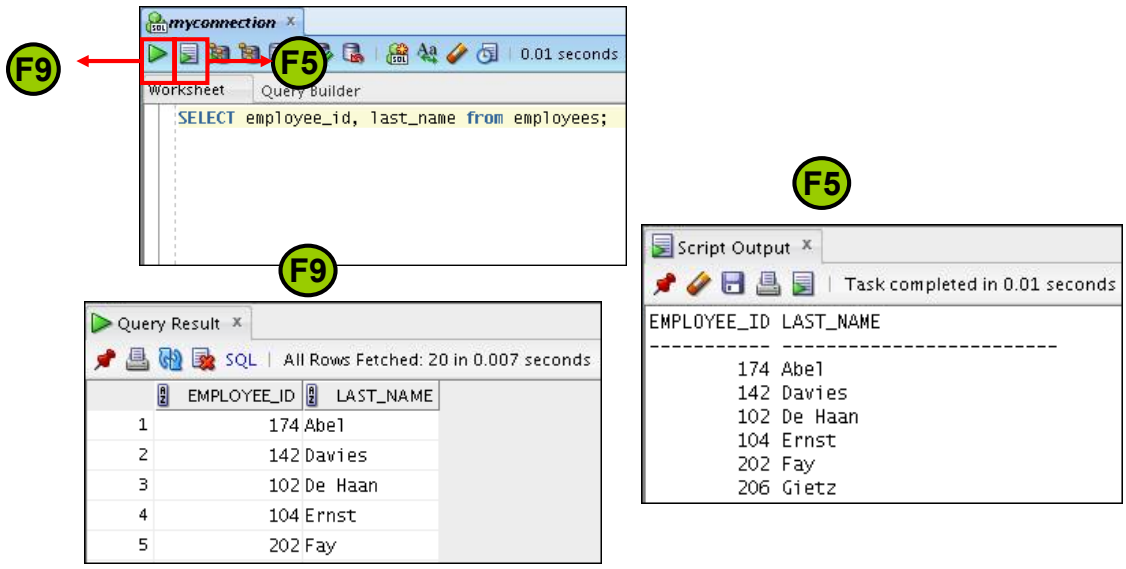


Wenn Sie sich bei einer Datenbank anmelden, wird automatisch ein SQL Worksheet-Fenster für diese Verbindung geöffnet. Im SQL Worksheet können Sie SQL-, PL/SQL- und SQL*Plus-Anweisungen eingeben und ausführen. Alle SQL- und PL/SQL-Befehle werden unterstützt und direkt vom SQL Worksheet an die Oracle-Datenbank übergeben. In SQL Developer verwendete SQL*Plus-Befehle müssen vor der Übergabe an die Datenbank vom SQL Worksheet interpretiert werden.

Das SQL Worksheet unterstützt derzeit eine Reihe von SQL*Plus-Befehlen. Nicht unterstützte SQL*Plus-Befehle werden ignoriert und nicht an die Oracle-Datenbank gesendet. Mit dem SQL Worksheet lassen sich SQL-Anweisungen und einige SQL*Plus-Befehle ausführen.

SQL-Anweisungen ausführen

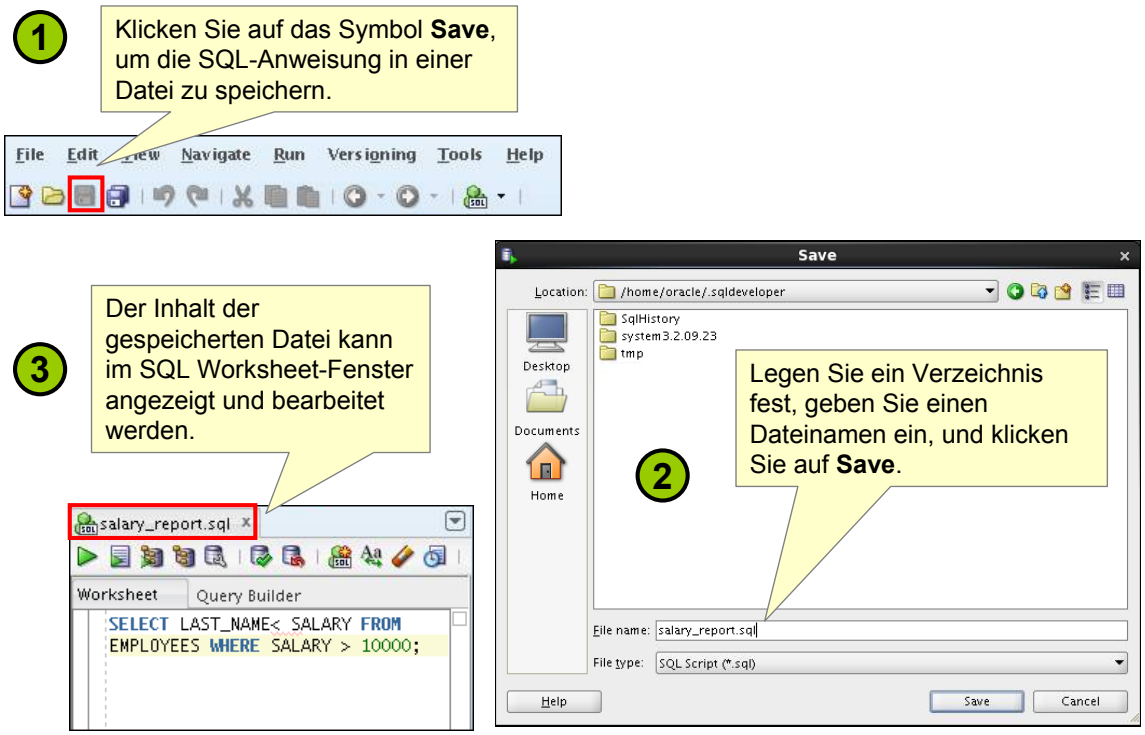
Im Feld **Enter SQL Statement** eine oder mehrere SQL-Anweisungen eingeben



ORACLE

Das Beispiel auf der Folie zeigt die unterschiedliche Ausgabe derselben Abfrage, einmal für die Taste F9 oder das Symbol **Execute Statement** und im Vergleich dazu für die Taste F5 oder das Symbol **Run Script**.

SQL-Skripte speichern



Sie können die SQL-Anweisungen im SQL Worksheet in einer Textdatei speichern. Um den Inhalt des Feldes **Enter SQL Statement** zu speichern, gehen Sie wie folgt vor:

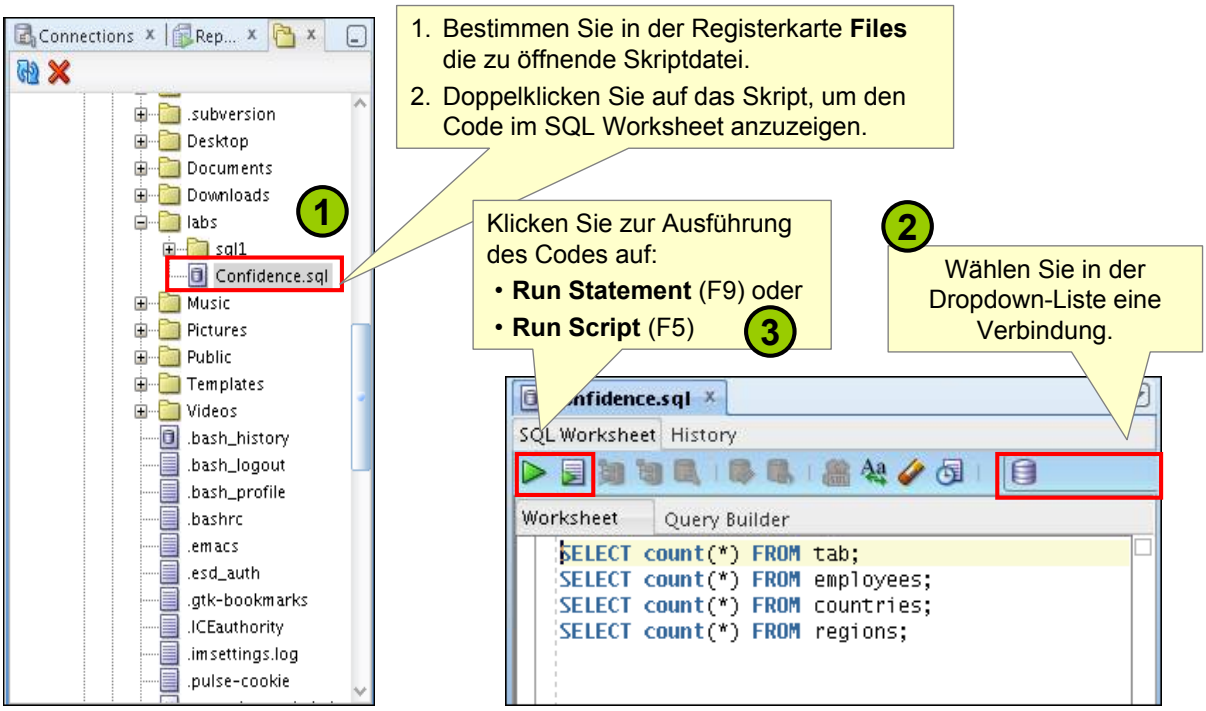
1. Klicken Sie auf das Symbol **Save**, oder wählen Sie die Menüoption **File** > **Save**.
2. Geben Sie im Dialogfeld **Save** einen Dateinamen und das gewünschte Verzeichnis für die Datei ein.
3. Klicken Sie auf **Save**.

Nachdem Sie den Inhalt in einer Datei gespeichert haben, wird im Fenster **Enter SQL Statement** eine Registerkarte mit dem Dateiinhalt angezeigt. Es können mehrere Dateien gleichzeitig geöffnet sein. Jede Datei wird in Form einer Registerkarte angezeigt.

Skriptpfad

Sie können einen Standardpfad für die Suche nach Skripten und deren Speicherung wählen. Geben Sie hierfür unter **Tools** > **Preferences** > **Database** > **Worksheet Parameters** einen Wert im Feld **Select default path to look for scripts** ein.

Gespeicherte Skriptdateien ausführen – Methode 1



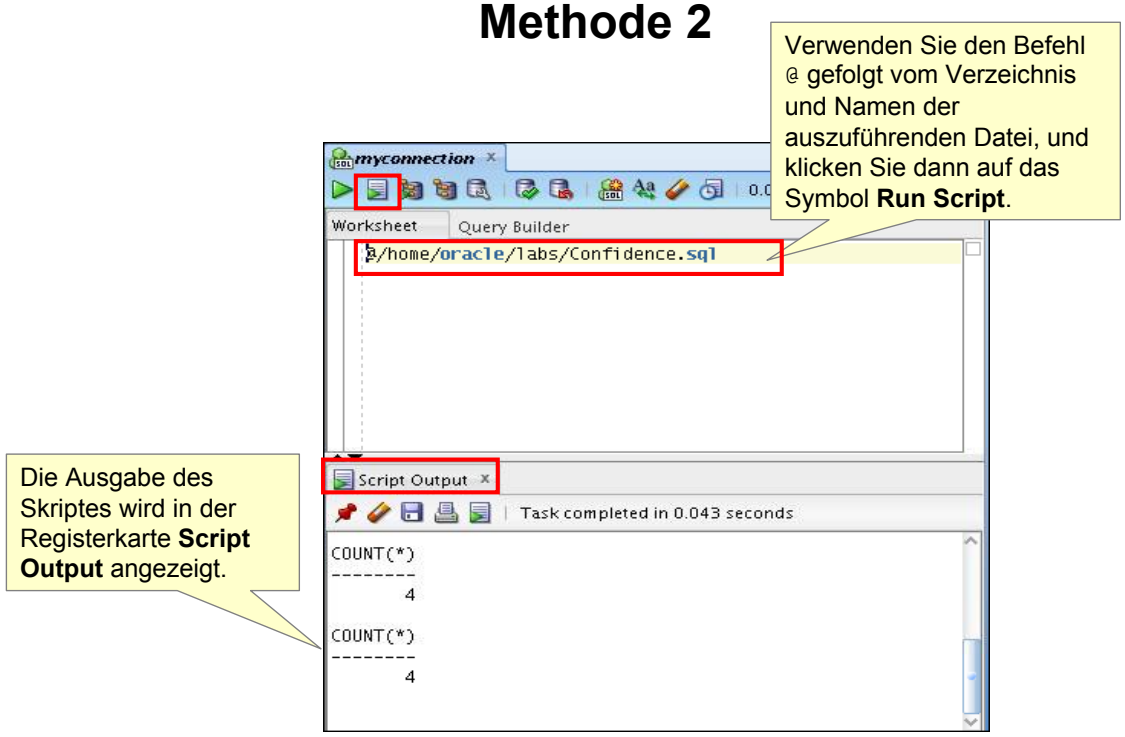
Um eine Skriptdatei zu öffnen und den Code im SQL Worksheet-Bereich anzuzeigen, gehen Sie wie folgt vor:

1. Wählen Sie im File Navigator die zu öffnende Skriptdatei, oder navigieren Sie zur gewünschten Datei.
2. Doppelklicken Sie auf die Datei, um sie zu öffnen. Der Code der Skriptdatei wird im SQL Worksheet-Bereich angezeigt.
3. Wählen Sie in der Dropdown-Liste **Connection** eine Verbindung.
4. Um den Code auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol **Run Script** (F5). Wenn Sie in der Dropdown-Liste **Connection** keine Verbindung gewählt haben, wird das Dialogfeld **Connection** angezeigt. Wählen Sie die gewünschte Verbindung für die Ausführung des Skriptes.

Alternativ können Sie auch wie folgt vorgehen:

1. Wählen Sie **File** > **Open**. Das Dialogfeld **Open** wird angezeigt.
2. Wählen Sie im Dialogfeld **Open** die zu öffnende Skriptdatei, oder navigieren Sie zur gewünschten Datei.
3. Klicken Sie auf **Open**. Der Code der Skriptdatei wird im SQL Worksheet-Bereich angezeigt.
4. Wählen Sie in der Dropdown-Liste **Connection** eine Verbindung.
5. Um den Code auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol **Run Script** (F5). Wenn Sie in der Dropdown-Liste **Connection** keine Verbindung gewählt haben, wird das Dialogfeld **Connection** angezeigt. Wählen Sie die gewünschte Verbindung für die Ausführung des Skriptes.

Gespeicherte Skriptdateien ausführen – Methode 2

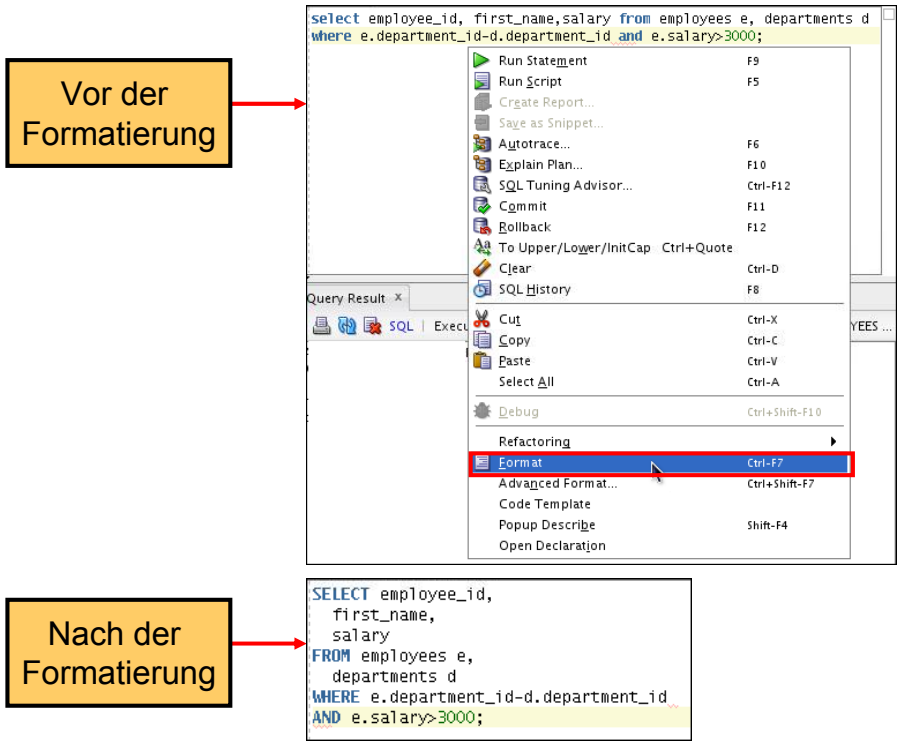


Um ein gespeichertes SQL-Skript auszuführen, gehen Sie wie folgt vor:

1. Verwenden Sie im Fenster **Enter SQL Statement** den Befehl @ gefolgt vom Verzeichnis und Namen der auszuführenden Datei.
2. Klicken Sie auf das Symbol **Run Script**.

Die Ergebnisse der Skriptausführung werden in der Registerkarte **Script Output** angezeigt. Sie können die Skriptausgabe auch speichern, indem Sie in der Registerkarte **Script Output** auf das Symbol **Save** klicken. Im angezeigten Dialogfeld **File Save** können Sie Name und Verzeichnis für die Datei angeben.

SQL-Code formatieren



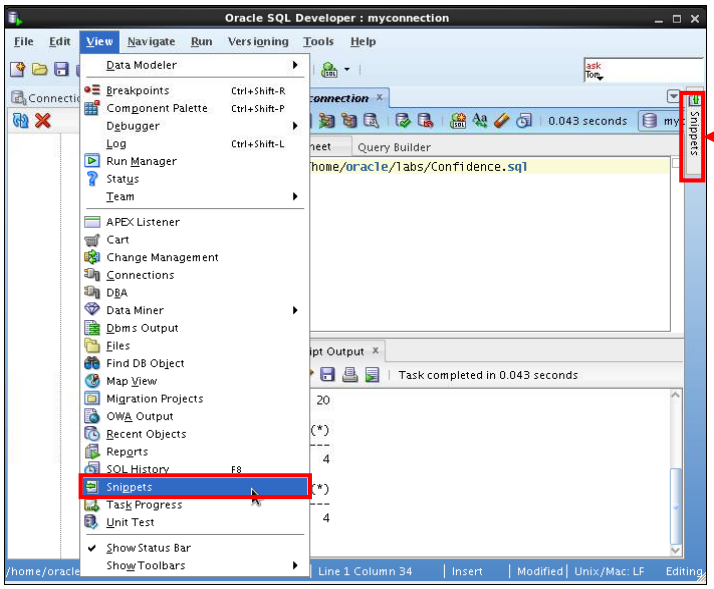
Es kann sinnvoll sein, die Einrückungen, Zeichenabstände, Groß-/Kleinschreibung und Zeilenabstände im SQL-Code übersichtlicher zu gestalten. SQL Developer bietet ein Feature zur Formatierung von SQL-Code.

Um SQL-Code zu formatieren, klicken Sie mit der rechten Maustaste in den Anweisungsbereich und wählen **Format SQL**.

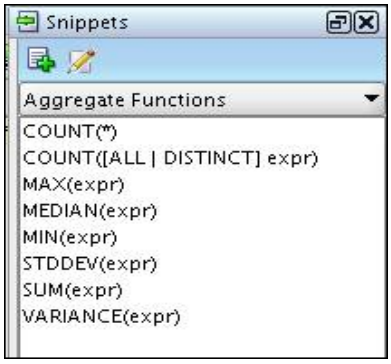
Das Beispiel auf der Folie zeigt, dass die Schlüsselwörter im SQL-Code vor der Formatierung nicht in Großbuchstaben dargestellt werden und die Anweisung nicht die richtigen Einrückungen aufweist. Nach der Formatierung ist der SQL-Code übersichtlicher, da die Schlüsselwörter groß geschrieben sind und die Anweisung mit den richtigen Einrückungen versehen ist.

Snippets

Snippets sind Codefragmente, die unter Umständen nur Syntax oder Beispiele enthalten.



Wenn Sie den Cursor hier platzieren, wird das Fenster **Snippets** angezeigt. Aus der Dropdown-Liste können Sie die gewünschte Funktionskategorie wählen.

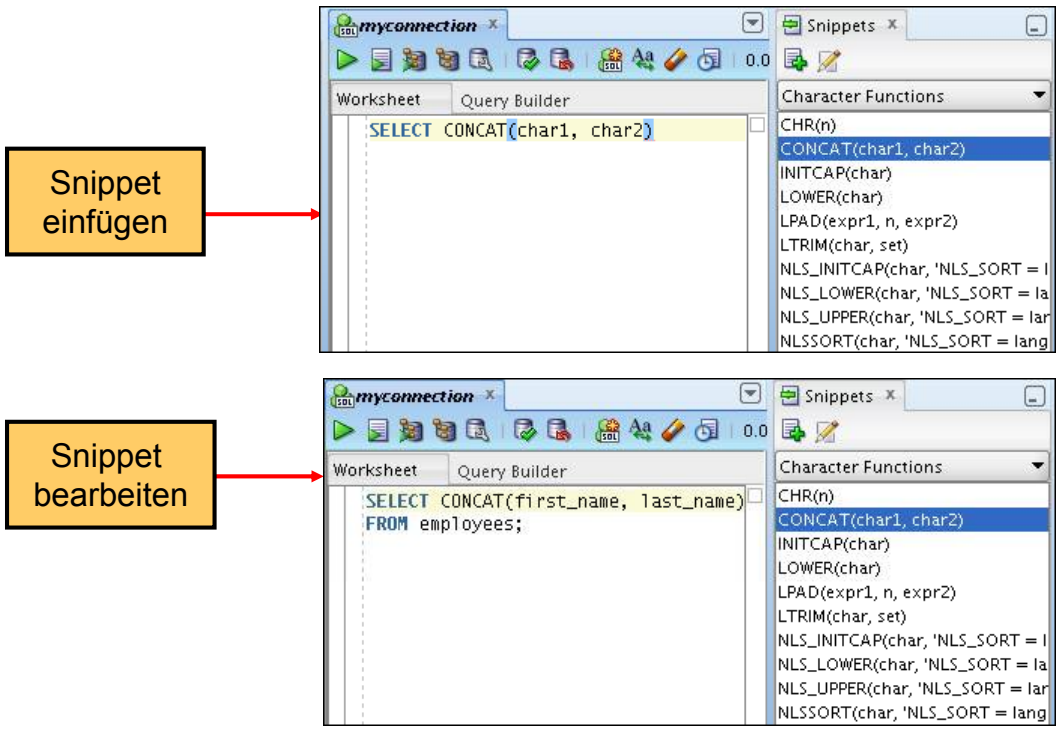


Bei der Arbeit mit dem SQL Worksheet bzw. der Erstellung oder Bearbeitung von PL/SQL-Funktionen oder -Prozeduren möchten Sie eventuell bestimmte Codefragmente verwenden. SQL Developer verfügt hierzu über das Feature Snippets. Snippets sind Codefragmente wie SQL-Funktionen, Optimizer Hints oder verschiedene PL/SQL-Programmiertechniken. Sie können Snippets in das Editorfenster ziehen.

Um Snippets anzuzeigen, wählen Sie **View** > **Snippets**.

Daraufhin wird rechts das Fenster **Snippets** angezeigt. Wählen Sie eine Gruppe aus der Dropdown-Liste. Über die Schaltfläche **Snippets** am rechten Fensterrand können Sie das ausgeblendete Fenster **Snippets** einblenden.

Snippets – Beispiel



ORACLE

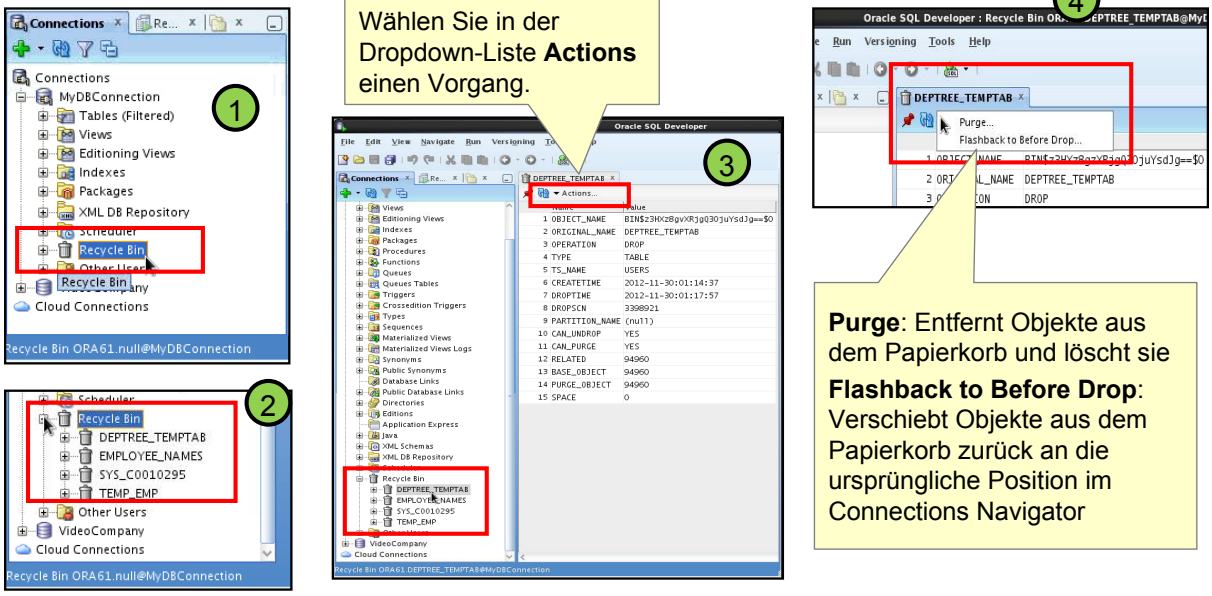
Um ein Snippet in den Code in einem SQL Worksheet oder in eine PL/SQL-Funktion oder -Prozedur einzufügen, ziehen Sie es aus dem Fenster **Snippets** an die gewünschte Stelle im Code. Danach können Sie die Syntax bearbeiten, sodass die SQL-Funktion im aktuellen Kontext gültig ist. Um eine kurze Beschreibung einer SQL-Funktion als QuickInfo anzuzeigen, platzieren Sie den Cursor über dem Funktionsnamen.

Im Beispiel auf der Folie wird `CONCAT(char1, char2)` aus der Gruppe **Character Functions** in das Fenster **Snippets** gezogen. Anschließend wird die Syntax der Funktion `CONCAT` bearbeitet und der restliche Teil der Anweisung wie folgt hinzugefügt:

```
SELECT CONCAT(first_name, last_name)
FROM employees;
```

Papierkorb

Im Papierkorb werden gelöschte Objekte aufbewahrt.



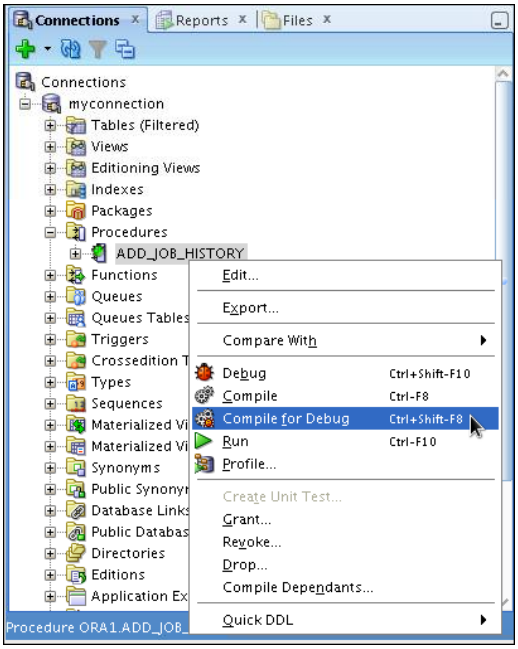
Der Papierkorb ist eine Data Dictionary-Tabelle, die Informationen zu gelöschten Objekten enthält. Gelöschte Tabellen und alle zugehörigen Objekte (wie Indizes, Constraints und Nested Tables) werden nicht entfernt und belegen weiterhin Speicherplatz. Sie werden so lange in den Speicherplatz-Quotas des Benutzers berücksichtigt, bis sie explizit dauerhaft aus dem Papierkorb gelöscht werden oder die unwahrscheinliche Situation eintritt, dass sie von der Datenbank aufgrund von Tablespace-Speicherplatzbeschränkungen dauerhaft gelöscht werden müssen.

Um den Papierkorb zu verwenden, gehen Sie wie folgt vor:

1. Wählen Sie im Connections Navigator den Papierkorb bzw. navigieren Sie zum Papierkorb.
2. Erweitern Sie den Papierkorb, und klicken Sie auf den Objektnamen. Die Objektdetails werden im SQL Worksheet-Bereich angezeigt.
3. Klicken Sie auf die Dropdow-Liste **Actions**, und wählen Sie den Vorgang, den Sie für das Objekt ausführen möchten.

Prozeduren und Funktionen debuggen

- Mit SQL Developer PL/SQL-Funktionen und -Prozeduren debuggen
- Mit der Option **Compile for Debug** eine PL/SQL-Kompilierung ausführen, sodass Sie die Prozedur debuggen können
- Mit der Menüoption **Debug** Breakpoints setzen und die Schritte **Step into** und **Step over** ausführen



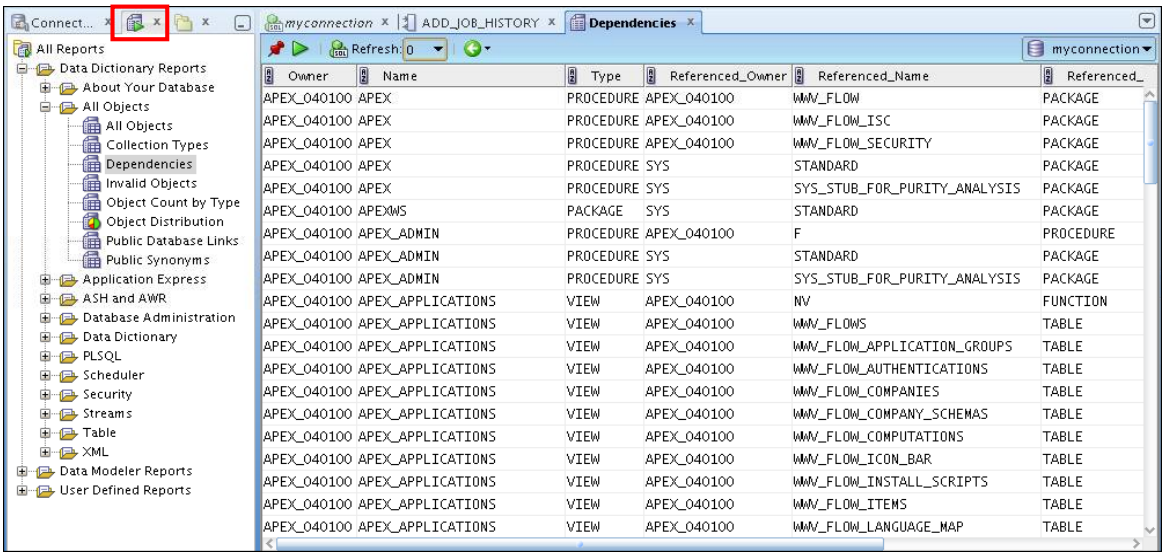
In SQL Developer können Sie PL/SQL-Prozeduren und -Funktionen debuggen. Mit den Menüoptionen zum Debuggen können Sie folgende Debugging-Aufgaben ausführen:

- **Find Execution Point** geht zum nächsten Ausführungspunkt.
- **Resume** setzt die Ausführung fort.
- **Step Over** umgeht die nächste Methode und geht zur nächsten Anweisung nach der Methode.
- **Step Into** geht zur ersten Anweisung in der nächsten Methode.
- **Step Out** verlässt die aktuelle Methode und geht zur nächsten Anweisung.
- **Step to End of Method** geht zur letzten Anweisung der aktuellen Methode.
- **Pause** unterbricht die Ausführung, beendet sie jedoch nicht. Sie können die Ausführung wieder aufnehmen.
- **Terminate** unterbricht die Ausführung und beendet sie. Sie können die Ausführung ab diesem Punkt nicht wieder aufnehmen. Um die Funktion oder Prozedur von Anfang an auszuführen oder zu debuggen, klicken Sie stattdessen in der Symbolleiste der Registerkarte **Source** auf das Symbol **Run** oder **Debug**.
- **Garbage Collection** entfernt ungültige Objekte aus dem Cache, um für gültige Objekte Platz zu machen, auf die häufiger zugegriffen wird.

Diese Optionen stehen auch als Symbole in der Symbolleiste **Debugging** zur Verfügung.

Datenbankberichte

SQL Developer bietet mehrere vordefinierte Berichte zur Datenbank und ihren Objekten.



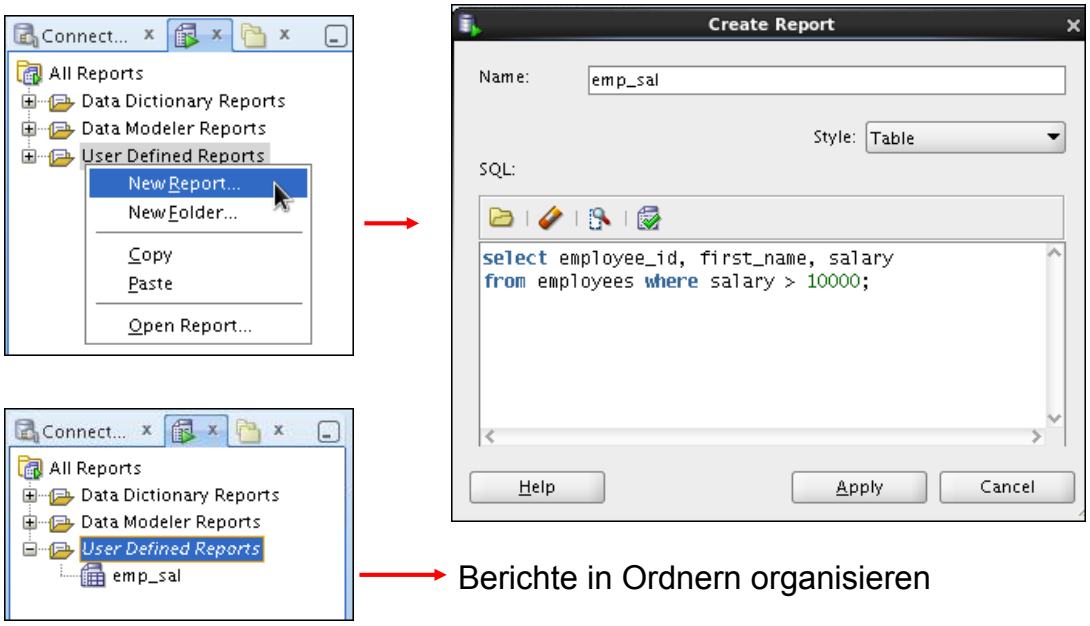
SQL Developer bietet zahlreiche Berichte zur Datenbank und ihren Objekten. Diese Berichte sind in folgende Kategorien unterteilt:

- About Your Database
- Database Administration
- Table
- PLSQL
- Security
- XML
- Jobs
- Streams
- All Objects
- Data Dictionary
- User-Defined Reports

Um Berichte anzuzeigen, klicken Sie links im Fenster auf die Registerkarte **Reports**. Die einzelnen Berichte werden rechts im Fenster in Registerkartenfenstern angezeigt. Bei jedem Bericht können Sie (in einer Dropdown-Liste) die Datenbankverbindung wählen, für die der Bericht angezeigt werden soll. Bei Berichten zu Objekten werden nur die Objekte angezeigt, die der Datenbankbenutzer mit der gewählten Datenbankverbindung sehen kann. Die Zeilen werden normalerweise nach dem Eigentümer (**Owner**) sortiert. Sie können auch eigene benutzerdefinierte Berichte erstellen.

Benutzerdefinierte Berichte erstellen

Benutzerdefinierte Berichte zur späteren Wiederverwendung erstellen und speichern



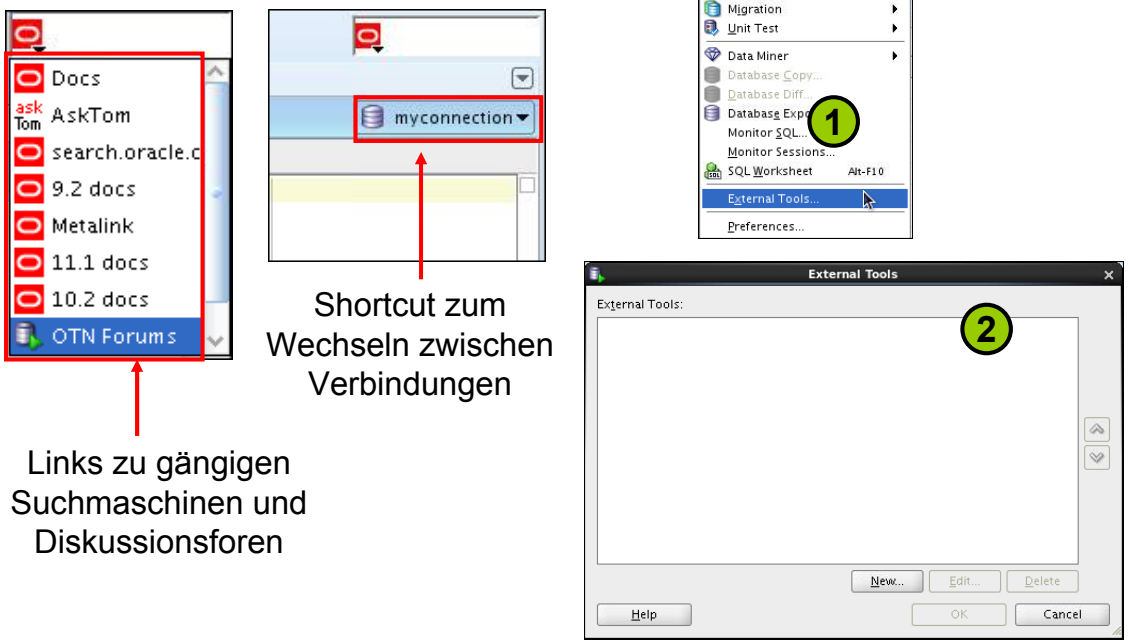
Benutzerdefinierte Berichte sind Berichte, die von SQL Developer-Benutzern erstellt werden. Um einen benutzerdefinierten Bericht zu erstellen, gehen Sie wie folgt vor:

1. Klicken Sie mit der rechten Maustaste unter **All Reports** auf den Knoten **User Defined Reports**, und wählen Sie **Add Report**.
2. Geben Sie im Dialogfeld **Create Report** den Berichtsnamen und die SQL-Abfrage an, um die Informationen für den Bericht abzurufen. Klicken Sie anschließend auf **Apply**.

Im Beispiel auf der Folie lautet der Name des Berichts `emp_sal`. Außerdem wurde eine optionale Beschreibung angegeben, aus der hervorgeht, dass der Bericht Details zu Mitarbeitern mit einem Gehalt von mindestens 10.000 enthält (`salary >= 10000`). Die vollständige SQL-Anweisung zum Abruf der Informationen für den benutzerdefinierten Bericht ist im Feld **SQL** angegeben. Sie können auch eine optionale QuickInfo aufnehmen, die angezeigt wird, wenn der Cursor in der Navigatoranzeige für **Reports** über dem Berichtsnamen platziert wird.

Sie können benutzerdefinierte Berichte in Ordnern organisieren und eine Hierarchie von Ordnern und Unterordnern erstellen. Um einen Ordner für benutzerdefinierte Berichte zu erstellen, klicken Sie mit der rechten Maustaste auf den Knoten **User Defined Reports** oder einen beliebigen Ordnernamen unter diesem Knoten und wählen **Add Folder**. Informationen über benutzerdefinierte Berichte, einschließlich der Ordner für diese Berichte, werden im Verzeichnis für benutzerspezifische Informationen in der Datei `UserReports.xml` gespeichert.

Suchmaschinen und externe Tools



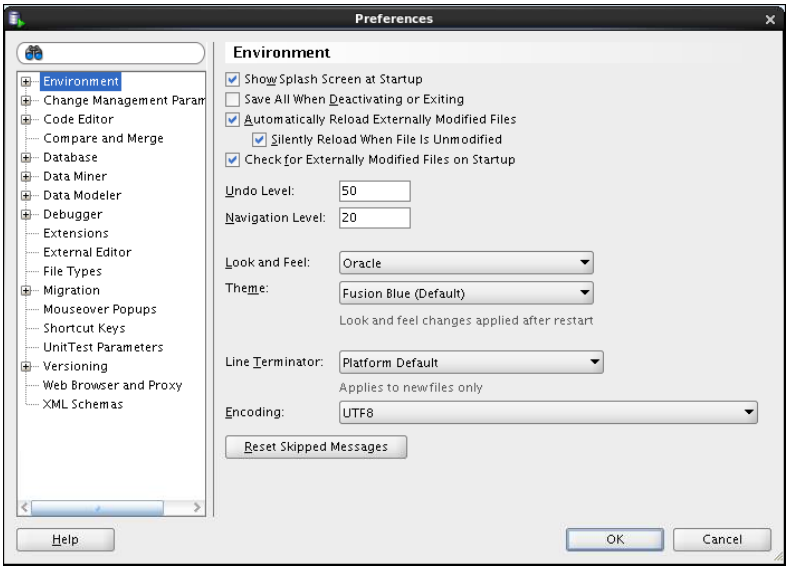
Um die Produktivität von SQL-Entwicklern zu erhöhen, wurde SQL Developer um Quicklinks zu gängigen Suchmaschinen und Diskussionsforen wie AskTom oder Google erweitert. Außerdem stehen Shortcut-Symbole zu häufig verwendeten Tools wie Notepad, Microsoft Word oder Dreamweaver zur Verfügung.

Sie können die vorhandene Liste um externe Tools ergänzen oder auch Shortcuts zu selten verwendeten Tools löschen. Gehen Sie dazu wie folgt vor:

1. Wählen Sie im Menü **Tools** die Option **External Tools**.
2. Um neue Tools hinzuzufügen, wählen Sie im Dialogfeld **External Tools** die Option **New**. Um Tools aus der Liste zu entfernen, wählen Sie **Delete**.

Voreinstellungen festlegen

- Benutzeroberfläche und Umgebung von SQL Developer anpassen
- Im Menü **Tools** die Option **Preferences** wählen



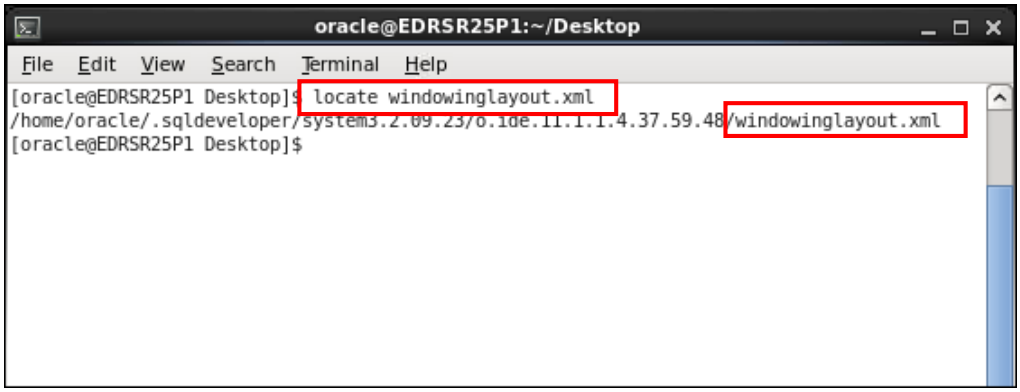
Sie können viele Aspekte der Benutzeroberfläche und der Umgebung von SQL Developer anpassen, indem Sie die SQL Developer-Voreinstellungen entsprechend Ihren Anforderungen ändern. Um SQL Developer-Voreinstellungen zu ändern, wählen Sie **Tools** und anschließend **Preferences**.

Die Voreinstellungen sind in folgende Kategorien eingeteilt:

- Environment
- Change Management Parameter
- Code Editor
- Compare and Merge
- Datenbank
- Data Miner
- Data Modeler
- Debugger
- Extensions
- External Editor
- File Types
- Migration

- Mouseover Popups
- Shortcut Keys
- Unit Test Parameters
- Versioning
- Web Browser and Proxy
- XML Schemas

SQL Developer-Layout zurücksetzen

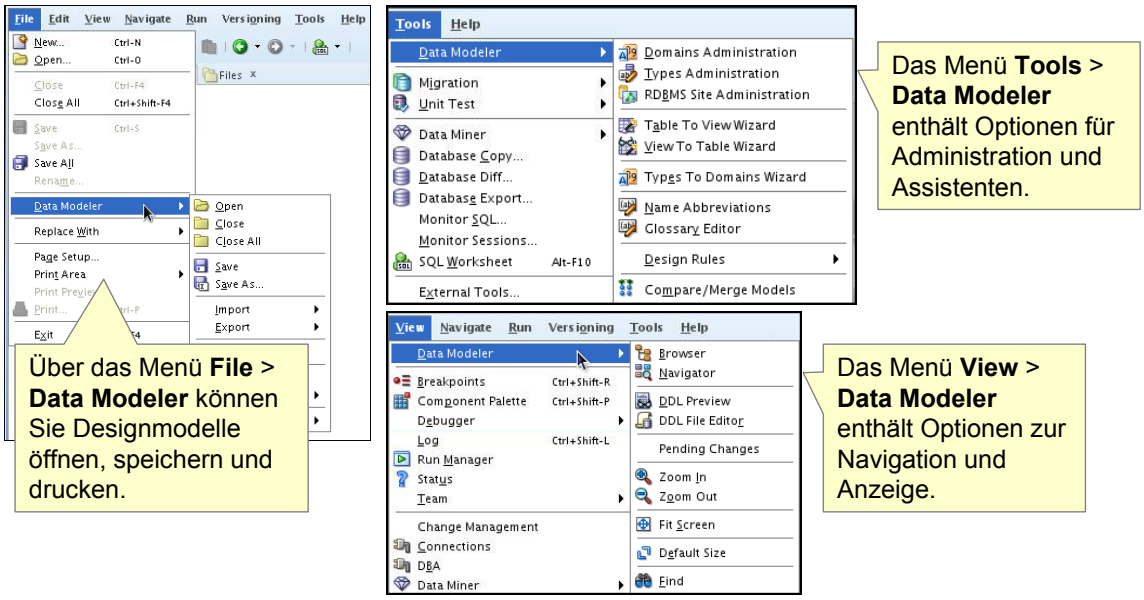


Wenn der Connections Navigator während der Arbeit mit SQL Developer ausgeblendet wird oder wenn Sie das Fenster **Log** nicht an seiner ursprünglichen Position verankern können, beheben Sie das Problem wie folgt:

1. Beenden Sie SQL Developer.
2. Öffnen Sie ein Terminalfenster, und verwenden Sie den Befehl `locate`, um den Speicherort von `windowinglayout.xml` zu ermitteln.
3. Navigieren Sie in das Verzeichnis, in dem sich `windowinglayout.xml` befindet, und löschen Sie diese Datei.
4. Starten Sie SQL Developer neu.

Data Modeler in SQL Developer

SQL Developer enthält eine integrierte Version von SQL Developer Data Modeler.



Mit der integrierten Version von SQL Developer Data Modeler können Sie:

- Datenbankdesigns erstellen, öffnen, importieren und speichern
- Data Modeler-Objekte erstellen, bearbeiten und löschen

Um Data Modeler in einem Bereich anzuzeigen, klicken Sie auf **Tools** und dann auf **Data Modeler**. Das Untermenü **Data Modeler** im Menü **Tools** enthält zusätzliche Befehle, etwa zur Festlegung von Designregeln und Voreinstellungen.

Zusammenfassung

In diesem Anhang haben Sie gelernt, mit SQL Developer folgende Aufgaben auszuführen:

- Datenbankobjekte durchsuchen, erstellen und bearbeiten
- SQL-Anweisungen und -Skripte im SQL Worksheet ausführen
- Benutzerdefinierte Berichte erstellen und speichern
- Data Modeler-Optionen in SQL Developer durchsuchen

ORACLE

Das kostenlose grafische Tool SQL Developer vereinfacht Aufgaben der Datenbankentwicklung. Mit SQL Developer können Sie Datenbankobjekte durchsuchen, erstellen und bearbeiten. Mit dem SQL Worksheet lassen sich SQL-Anweisungen und -Skripte ausführen. Darüber hinaus können Sie mit SQL Developer eigene spezielle Berichte erstellen und speichern, die Sie mehrfach wiederverwenden können.

C

SQL*Plus

Ziele

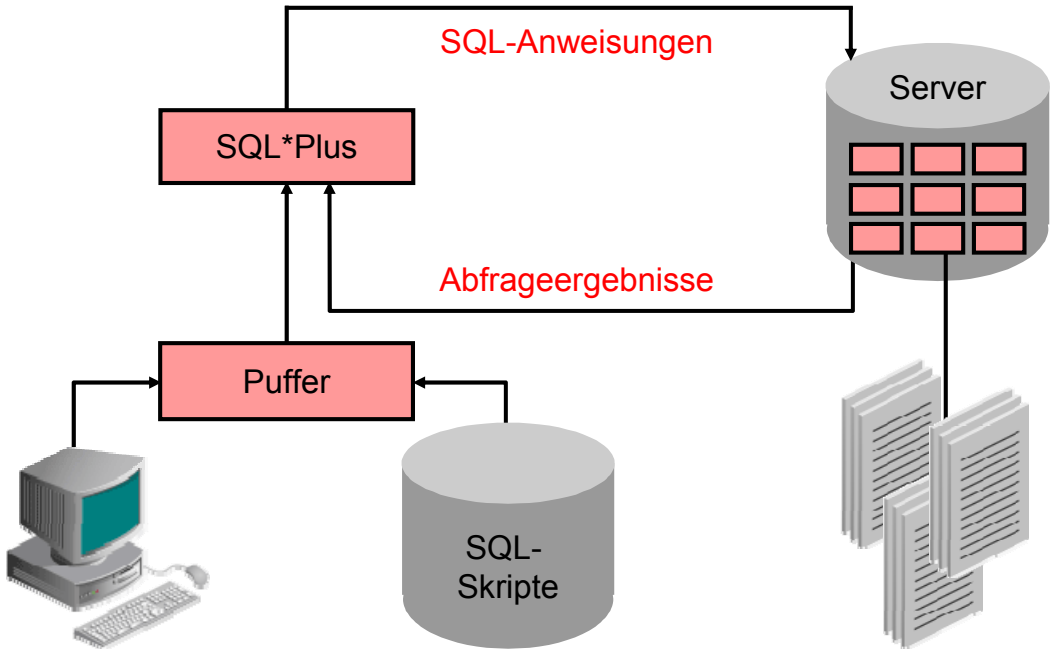
Nach Ablauf dieses Anhangs haben Sie folgende Ziele erreicht:

- Bei SQL*Plus anmelden
- SQL-Befehle bearbeiten
- Ausgabe mithilfe von SQL*Plus-Befehlen formatieren
- Mit Skriptdateien interagieren

ORACLE

Sie können `SELECT`-Anweisungen erstellen und mehrfach wiederverwenden. In diesem Anhang wird die Ausführung von SQL-Anweisungen mithilfe von SQL*Plus-Befehlen behandelt. Außerdem wird beschrieben, wie Sie die Ausgabe mit SQL*Plus-Befehlen formatieren, SQL-Befehle bearbeiten und Skripte in SQL*Plus speichern.

SQL und SQL*Plus – Interaktion



SQL und SQL*Plus

SQL ist eine Befehlssprache, mit deren Hilfe beliebige Tools und Anwendungen mit dem Oracle-Server kommunizieren können. Oracle SQL enthält zahlreiche Erweiterungen. Wenn Sie eine SQL-Anweisung eingeben, wird sie in einem Bereich des Speichers abgelegt, der als *SQL-Puffer* bezeichnet wird. Sie verbleibt dort, bis Sie eine neue SQL-Anweisung eingeben. SQL*Plus ist ein Oracle-Tool, das SQL-Anweisungen erkennt und zur Ausführung an den Oracle 9*i* Server weiterleitet. Es enthält eine eigene Befehlssprache.

SQL – Features

- Kann von allen Benutzern verwendet werden, auch von Benutzern ohne oder mit geringen Programmiererfahrungen
- Ist eine nicht prozedurale Sprache
- Reduziert den Zeitaufwand für Erstellung und Verwaltung von Systemen
- Ist an das Englische angelehnt

SQL*Plus – Features

- Akzeptiert die Ad-hoc-Eingabe von Anweisungen
- Akzeptiert die SQL-Eingabe aus Dateien
- Bietet einen Zeileneditor zum Ändern von SQL-Anweisungen
- Steuert die Umgebungseinstellungen
- Formatiert Abfrageergebnisse zu Basisberichten
- Greift auf lokale Datenbanken und Remote-Datenbanken zu

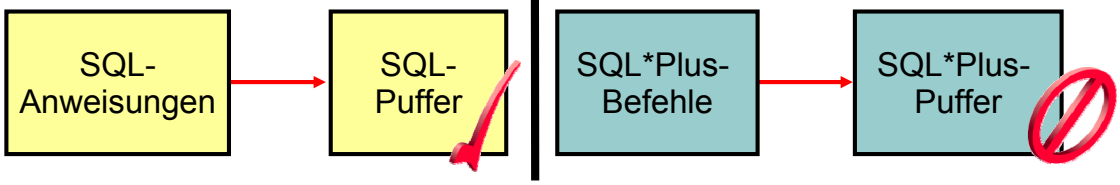
SQL-Anweisungen und SQL*Plus-Befehle – Vergleich

SQL

- Eine Sprache
- ANSI-Standard
- Schlüsselwörter dürfen nicht abgekürzt werden.
- Anweisungen bearbeiten Daten und Tabellen-definitionen in der Datenbank.

SQL*Plus

- Eine Umgebung
- Oracle-Tool
- Schlüsselwörter können abgekürzt werden.
- Befehle erlauben keine Bearbeitung von Werten in der Datenbank.



Die folgende Tabelle stellt SQL und SQL*Plus gegenüber:

SQL	SQL*Plus
Eine Sprache für die Kommunikation mit dem Oracle-Server beim Datenzugriff	Erkennt SQL-Anweisungen und sendet sie an den Server
Basiert auf Standard-SQL des American National Standards Institute (ANSI)	Oracle-eigene Schnittstelle zur Ausführung von SQL-Anweisungen
Bearbeitet Daten und Tabellendefinitionen in der Datenbank	Ermöglicht keine Bearbeitung von Werten in der Datenbank
Wird in einer oder mehreren Zeilen des SQL-Puffers eingegeben	Wird zeilenweise eingegeben und nicht im SQL-Puffer gespeichert
Hat kein Fortsetzungszeichen	Verwendet einen Bindestrich (–) als Fortsetzungszeichen, wenn der Befehl länger als eine Zeile ist
Kann nicht abgekürzt werden	Kann abgekürzt werden
Verwendet ein Abschlusszeichen, um Befehle sofort auszuführen	Benötigt kein Abschlusszeichen. Befehle werden sofort ausgeführt.
Verwendet Funktionen zur Ausführung von Formatierungen	Verwendet Befehle zum Formatieren von Daten

SQL*Plus – Überblick

- Bei SQL*Plus anmelden
- Tabellenstruktur beschreiben
- SQL-Anweisungen bearbeiten
- SQL über SQL*Plus ausführen
- SQL-Anweisungen in Dateien speichern und an Dateien anhängen
- Gespeicherte Dateien ausführen
- Befehle aus der Datei zur Bearbeitung in den Puffer laden

ORACLE

SQL*Plus

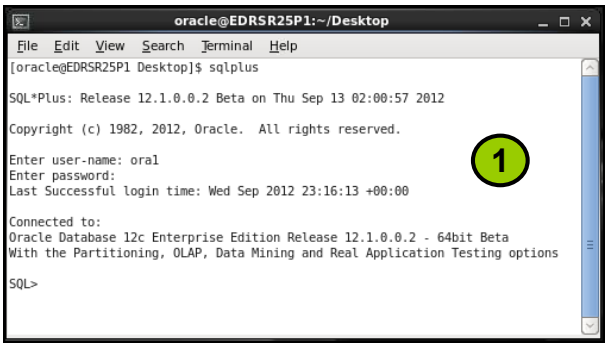
In der SQL*Plus-Umgebung können Sie folgende Aktionen ausführen:

- SQL-Anweisungen ausführen, um Daten aus der Datenbank abzurufen, zu ändern, hinzuzufügen und zu entfernen
- Abfrageergebnisse als Berichte formatieren, speichern und drucken sowie Berechnungen mit den Abfrageergebnissen ausführen
- Skriptdateien erstellen, um SQL-Anweisungen für die spätere Wiederverwendung zu speichern

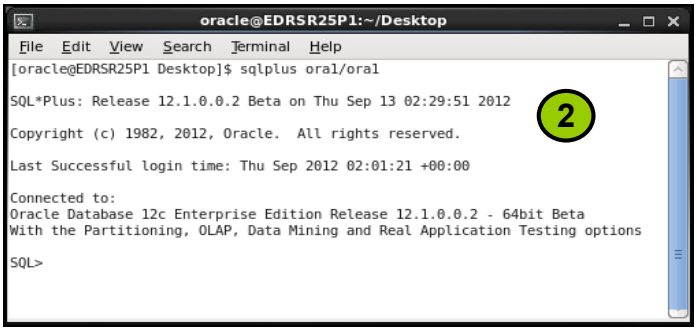
SQL*Plus-Befehle können in folgende Hauptkategorien unterteilt werden:

Kategorie	Zweck
Umgebung	Allgemeines Verhalten von SQL-Anweisungen für die Session beeinflussen
Format	Abfrageergebnisse formatieren
Dateibearbeitung	Skriptdateien speichern, laden und ausführen
Ausführung	SQL-Anweisungen aus dem SQL-Puffer an den Oracle-Server senden
Bearbeitung	SQL-Anweisungen im Puffer ändern
Interaktion	Variablen erstellen und an SQL-Anweisungen übergeben, Variablenwerte ausgeben und Meldungen auf dem Bildschirm anzeigen
Verschiedenes	Bei der Datenbank anmelden, SQL*Plus-Umgebung bearbeiten und Spaltendefinitionen anzeigen

Bei SQL*Plus anmelden




```
sqlplus [username[/password[@database]]]
```



Wie Sie SQL*Plus aufrufen, richtet sich nach dem Typ des Betriebssystems, auf dem Sie Oracle Database ausführen.

Um sich aus einer Linux-Umgebung anzumelden, gehen Sie wie folgt vor:

1. Klicken Sie mit der rechten Maustaste auf den Linux-Desktop, und wählen Sie **terminal**.
2. Geben Sie den auf der Folie gezeigten Befehl `sqlplus` ein.
3. Geben Sie Benutzername, Kennwort und Datenbanknamen ein.

Für die Syntax gilt:

username	Ihr Benutzername für die Datenbank
password	Ihr Kennwort für die Datenbank. (Das Kennwort ist hier bei der Eingabe sichtbar.)
@database	Die Verbindungszeichenfolge für die Datenbank

Hinweis: Um die Integrität des Kennwortes zu wahren, dürfen Sie es nicht in der Eingabeaufforderung des Betriebssystems eingeben. Geben Sie stattdessen nur Ihren Benutzernamen ein, und geben Sie das Kennwort in der Kennworteingabeaufforderung ein.

Tabellenstrukturen anzeigen

Struktur einer Tabelle mit dem SQL*Plus-Befehl DESCRIBE anzeigen:

```
DESC[RIBE] tablename
```


In SQL*Plus können Sie die Struktur einer Tabelle mit dem Befehl DESCRIBE anzeigen. Daraufhin werden die Spaltennamen und Datentypen sowie ein Hinweis darauf angezeigt, ob eine Spalte Daten enthalten muss.

Für die Syntax gilt:

tablename Der Name einer beliebigen vorhandenen Tabelle, einer View oder eines Synonyms, auf die bzw. das der Benutzer zugreifen kann

Die Tabelle DEPARTMENTS zeigen Sie mit folgendem Befehl an:

```
SQL> DESCRIBE DEPARTMENTS
Name                      Null     Type
------------------------- -------- ---------------

DEPARTMENT_ID             NOT NULL NUMBER(4)
DEPARTMENT_NAME           NOT NULL VARCHAR2(30)
MANAGER_ID                         NUMBER(6)
LOCATION_ID                        NUMBER(4)
```

Tabellenstrukturen anzeigen

```
DESCRIBE departments
```

```
Name                          Null     Type
----------------------------- -------- ------------
DEPARTMENT_ID                 NOT NULL NUMBER(4)
DEPARTMENT_NAME               NOT NULL VARCHAR2(30)
MANAGER_ID                             NUMBER(6)
LOCATION_ID                            NUMBER(4)
```


Das Beispiel auf der Folie zeigt Informationen über die Struktur der Tabelle `DEPARTMENTS`. Für das Ergebnis gilt:

- `Null`: Gibt an, ob eine Spalte Daten enthalten muss (`NOT NULL` gibt an, dass eine Spalte Daten enthalten muss.)
- `Type`: Zeigt den Datentyp einer Spalte an

SQL*Plus – Bearbeitungsbefehle

- A[PPEND] *text*
- C[HANGE] / *old* / *new*
- C[HANGE] / *text* /
- CL[EAR] BUFF[ER]
- DEL
- DEL *n*
- DEL *m n*

SQL*Plus-Befehle werden Zeile für Zeile eingegeben und nicht im SQL-Puffer gespeichert.

Befehl	Beschreibung
A[PPEND] *text*	Fügt Text an das Ende der aktuellen Zeile an
C[HANGE] / *old* / *new*	Ersetzt in der aktuellen Zeile alten Text (*old*) durch neuen Text (*new*)
C[HANGE] / *text* /	Löscht Text (*text*) aus der aktuellen Zeile
CL[EAR] BUFF[ER]	Löscht alle Zeilen aus dem SQL-Puffer
DEL	Löscht die aktuelle Zeile
DEL *n*	Löscht die Zeile *n*
DEL *m n*	Löscht die Zeilen *m* bis einschließlich *n*

Richtlinien

- Wenn Sie die EINGABETASTE drücken, bevor Sie einen Befehl abgeschlossen haben, zeigt SQL*Plus die Nummer der entsprechenden Befehlszeile an.
- Sie beenden den SQL-Puffer, indem Sie eines der Abschlusszeichen (Semikolon oder Schrägstrich) eingeben oder zweimal die EINGABETASTE drücken. Anschließend wird die SQL-Eingabeaufforderung angezeigt.

SQL*Plus – Bearbeitungsbefehle

- I[NPUT]
- I[NPUT] *text*
- L[IST]
- L[IST] *n*
- L[IST] *m n*
- R[UN]
- *n*
- *n text*
- 0 *text*

Befehl	Beschreibung
I[NPUT]	Fügt eine unbestimmte Anzahl von Zeilen ein
I[NPUT] *text*	Fügt eine Zeile mit Text (*text*) ein
L[IST]	Listet alle Zeilen im SQL-Puffer auf
L[IST] *n*	Listet eine Zeile auf (durch *n* angegeben)
L[IST] *m n*	Listet einen Bereich von Zeilen auf (*m* bis einschließlich *n*)
R[UN]	Zeigt die aktuell im Puffer befindliche SQL-Anweisung an und führt sie aus
n	Macht *n* zur aktuellen Zeile
n text	Ersetzt die Zeile *n* durch den Text (*text*)
0 *text*	Fügt eine Zeile vor Zeile 1 ein

Hinweis: Sie können nur jeweils einen SQL*Plus-Befehl pro SQL-Eingabeaufforderung eingeben. SQL*Plus-Befehle werden nicht im SQL-Puffer gespeichert. Um einen SQL*Plus-Befehl in der nächsten Zeile fortzusetzen, geben Sie am Ende der ersten Zeile einen Bindestrich (-) ein.

LIST, n und APPEND

```
LIST
  1  SELECT last_name
  2* FROM   employees
```

```
1
  1* SELECT last_name
```

```
A , job_id
  1* SELECT last_name, job_id
```

```
LIST
  1  SELECT last_name, job_id
  2* FROM   employees
```


- Mit dem Befehl `L[IST]` zeigen Sie den Inhalt des SQL-Puffers an. Das Sternchen (`*`) neben der 2. Zeile im Puffer gibt an, dass die 2. Zeile die aktuelle Zeile ist. Alle eingegebenen Bearbeitungsbefehle gelten für die aktuelle Zeile.
- Sie können die aktuelle Zeile wechseln, indem Sie die Nummer (`n`) der Zeile eingeben, die Sie bearbeiten möchten. Die neue aktuelle Zeile wird angezeigt.
- Mit dem Befehl `A[PPEND]` fügen Sie der aktuellen Zeile Text hinzu. Die neu bearbeitete Zeile wird angezeigt. Den neuen Inhalt des Puffers prüfen Sie mit dem Befehl `LIST`.

Hinweis: Viele SQL*Plus-Befehle, einschließlich `LIST` und `APPEND`, können mit ihrem ersten Buchstaben abgekürzt werden. `LIST` lässt sich mit `L` abkürzen, `APPEND` mit `A`.

Befehl CHANGE

```
LIST
  1* SELECT * from employees
```

```
c/employees/departments
  1* SELECT * from departments
```

```
LIST
  1* SELECT * from departments
```


- Mit `L[IST]` zeigen Sie den Inhalt des Puffers an.
- Mit dem Befehl `C[HANGE]` ändern Sie den Inhalt der aktuellen Zeile im SQL-Puffer. In diesem Fall ersetzen Sie die Tabelle `employees` durch die Tabelle `departments`. Die neue aktuelle Zeile wird angezeigt.
- Mit dem Befehl `L[IST]` prüfen Sie den neuen Inhalt des SQL-Puffers.

SQL*Plus – Dateibefehle

- SAVE *filename*
- GET *filename*
- START *filename*
- @ *filename*
- EDIT *filename*
- SPOOL *filename*
- EXIT

SQL-Anweisungen kommunizieren mit dem Oracle-Server. SQL*Plus-Befehle steuern die Umgebung, formatieren Abfrageergebnisse und verwalten Dateien. Sie können die in der folgenden Tabelle beschriebenen Befehle verwenden:

Befehl	Beschreibung
SAV[E] *filename* [.ext] [REP[LACE]APP[END]]	Speichert den aktuellen Inhalt des SQL-Puffers in einer Datei. Verwenden Sie APPEND, um den Pufferinhalt einer vorhandenen Datei hinzuzufügen, oder REPLACE, um eine vorhandene Datei zu ersetzen. Die Standarderweiterung ist .sql.
GET *filename* [.ext]	Schreibt den Inhalt einer zuvor gespeicherten Datei in den SQL-Puffer. Die Standarderweiterung für den Dateinamen ist .sql.
STA[RT] *filename* [.ext]	Führt eine zuvor gespeicherte Befehlsdatei aus
@ *filename*	Führt eine zuvor gespeicherte Befehlsdatei aus (identisch mit START)

Befehl	Beschreibung
ED[IT]	Ruft den Editor auf und speichert den Pufferinhalt in der Datei afiedt.buf
ED[IT] [*filename*[.ext]]	Ruft den Editor auf, um den Inhalt einer gespeicherten Datei zu bearbeiten
SPO[OL] [*filename*[.ext]|OFF|OUT]	Speichert die Abfrageergebnisse in einer Datei. OFF schließt die Spool-Datei. OUT schließt die Spool-Datei und sendet die Dateiergebnisse an den Drucker.
EXIT	Beendet SQL*Plus

Befehle SAVE und START

```
LIST
  1  SELECT last_name, manager_id, department_id
  2* FROM employees
```

```
SAVE my_query
  Created file my_query
```

```
START my_query

LAST_NAME                 MANAGER_ID DEPARTMENT_ID
------------------------- ---------- -------------
King                                           90
Kochhar                          100           90
...
107 rows selected.
```


SAVE

Mit dem Befehl `SAVE` speichern Sie den aktuellen Inhalt des Puffers in einer Datei. So können Sie häufig verwendete Skripte zur späteren Wiederverwendung speichern.

START

Mit dem Befehl `START` führen Sie ein Skript in SQL*Plus aus. Alternativ können Sie Skripte auch über das Symbol `@` ausführen.

```
@my_query
```

Befehl SERVEROUTPUT

- Mit dem Befehl SET SERVEROUT[PUT] steuern Sie, ob die Ausgabe von gespeicherten Prozeduren und PL/SQL-Blöcken in SQL*Plus angezeigt wird.
- Die Zeilenlängenbeschränkung für DBMS_OUTPUT wurde von 255 Byte auf 32767 Byte erhöht.
- Die Standardgröße ist nun unbegrenzt.
- Ist SERVEROUTPUT festgelegt, werden keine Ressourcen reserviert.
- Verwenden Sie UNLIMITED, da damit keine Performance-einbußen verbunden sind. Ausnahme: Sie möchten physischen Speicher einsparen.

```
SET SERVEROUT[PUT] {ON | OFF} [SIZE {n | UNL[IMITED]}]
    [FOR[MAT] {WRA[PPED] | WOR[D_WRAPPED] | TRU[NCATED]}]
```

ORACLE

Die meisten PL/SQL-Programme führen Ein- und Ausgaben über SQL-Anweisungen durch, um Daten in Datenbanktabellen zu speichern oder diese Tabellen abzufragen. Alle anderen PL/SQL-Eingaben/Ausgaben erfolgen über APIs, die mit anderen Programmen interagieren. Beispiel: Das Package DBMS_OUTPUT verfügt über Prozeduren wie PUT_LINE. Um das Ergebnis außerhalb von PL/SQL anzuzeigen, ist zum Lesen und Anzeigen der an DBMS_OUTPUT übergebenen Daten ein anderes Programm (etwa SQL*Plus) erforderlich.

SQL*Plus zeigt Daten von DBMS_OUTPUT nur an, wenn Sie zuvor den SQL*Plus-Befehl SET SERVEROUTPUT ON wie folgt abgesetzt haben:

```
SET SERVEROUTPUT ON
```

Hinweise

- SIZE legt die Anzahl der Ausgabebyte fest, die im Oracle Database-Server gepuffert werden können. Der Standardwert ist UNLIMITED. n darf nicht unter 2.000 oder über 1.000.000 liegen.
- Weitere Informationen zu SERVEROUTPUT finden Sie im *Oracle Database PL/SQL User's Guide and Reference 12c*.

SQL*Plus-Befehl SPOOL

```
SPO[OL] [file_name[.ext] [CRE[ATE] | REP[LACE] |
    APP[END]] | OFF | OUT]
```

Option	Beschreibung
file_name[.ext]	Schreibt die Ausgabe in die angegebene Datei
CRE[ATE]	Erstellt eine neue Datei mit dem angegebenen Namen
REP[LACE]	Ersetzt den Inhalt einer vorhandenen Datei. Ist die Datei nicht vorhanden, wird sie durch REPLACE erstellt.
APP[END]	Fügt den Pufferinhalt am Ende der angegebenen Datei ein
OFF	Stoppt das Spool-Verfahren
OUT	Stoppt das Spool-Verfahren und sendet die Datei an den Standarddrucker des Rechners

Der Befehl SPOOL speichert die Abfrageergebnisse in einer Datei oder sendet die Datei optional an einen Drucker. Der Befehl SPOOL wurde erweitert, sodass Sie jetzt Daten an das Ende einer Datei anhängen oder eine vorhandene Datei ersetzen können. Bisher konnten Sie mit SPOOL lediglich Dateien erstellen (und ersetzen). REPLACE ist der Standardwert.

Sie können die über Befehle generierte Ausgabe mit SET TERMOUT OFF in ein Skript schreiben, ohne die Ausgabe auf dem Bildschirm anzuzeigen. SET TERMOUT OFF hat keine Auswirkungen auf die Ausgabe von Befehlen, die interaktiv ausgeführt werden.

Dateinamen, die Leerzeichen enthalten, müssen in Anführungszeichen gesetzt werden. Um mit SPOOL APPEND-Befehlen eine gültige HTML-Datei zu erstellen, müssen Sie mit PROMPT oder einem ähnlichen Befehl den Header und Footer der HTML-Seite erstellen. Der Befehl SPOOL APPEND parst keine HTML-Tags. Um die Parameter CREATE, APPEND und SAVE zu deaktivieren, stellen Sie SQLPLUSCOMPAT[IBILITY] auf 9.2 oder früher ein.

Befehl AUTOTRACE

- Zeigt nach der erfolgreichen Ausführung von SQL-DML-Anweisungen wie SELECT, INSERT, UPDATE und DELETE einen Bericht an
- Der Bericht kann nun Ausführungsstatistiken und den Abfrageausführungspfad umfassen.

```
SET AUTOT[RACE] {ON | OFF | TRACE[ONLY]} [EXP[LAIN]]
    [STAT[ISTICS]]
```

```
SET AUTOTRACE ON
-- The AUTOTRACE report includes both the optimizer
-- execution path and the SQL statement execution
-- statistics
```

ORACLE

EXPLAIN zeigt den Abfrageausführungspfad durch Ausführung von EXPLAIN PLAN an. STATISTICS zeigt Statistiken zu SQL-Anweisungen an. Die Formatierung des AUTOTRACE-Berichts kann je nach Version des Servers, bei dem Sie angemeldet sind, und Serverkonfiguration variieren. Das Package DBMS_XPLAN ermöglicht Ihnen die einfache Anzeige der Ausgabe des Befehls EXPLAIN PLAN in verschiedenen vordefinierten Formaten.

Hinweise

- Weitere Informationen zum Package und zu Unterprogrammen finden Sie in der *Oracle Database PL/SQL Packages and Types Reference 12c*.
- Weitere Informationen zu EXPLAIN PLAN finden Sie in der *Oracle Database SQL Reference 12c*.
- Weitere Informationen zu Ausführungsplänen und Statistiken finden Sie im *Oracle Database Performance Tuning Guide 12c*.

Zusammenfassung

In diesem Anhang haben Sie gelernt, SQL*Plus als Umgebung für folgende Aufgaben zu verwenden:

- SQL-Anweisungen ausführen
- SQL-Anweisungen bearbeiten
- Ausgabe formatieren
- Mit Skriptdateien interagieren

ORACLE

SQL*Plus ist eine Ausführungsumgebung, mit der Sie SQL-Befehle an den Datenbankserver senden sowie SQL-Befehle bearbeiten und speichern. Die Ausführung der Befehle erfolgt über die SQL-Eingabeaufforderung oder eine Skriptdatei.

Häufig verwendete SQL-Befehle

Ziele

Nach Ablauf dieses Anhangs haben Sie folgende Ziele erreicht:

- Einfache `SELECT`-Anweisungen ausführen
- Mit DDL-Anweisungen Tabellen erstellen, ändern und löschen
- Mit DML-Anweisungen Zeilen aus einer oder mehreren Tabellen einfügen, aktualisieren und löschen
- Mit Anweisungen zur Transaktionskontrolle Savepoints festschreiben, zurückrollen und erstellen
- Join-Vorgänge für eine oder mehrere Tabellen durchführen

ORACLE

In dieser Lektion wird erläutert, wie Sie mit `SELECT`-Anweisungen Daten aus einer oder mehreren Tabellen abrufen, mit DDL-Anweisungen die Struktur von Datenobjekten ändern sowie mit DML-Anweisungen Daten in vorhandenen Schemaobjekten bearbeiten und die vorgenommenen Änderungen verwalten. Darüber hinaus lernen Sie, mithilfe von Joins und der SQL:1999-konformen Join-Syntax Daten aus mehreren Tabellen anzuzeigen.

Einfache SELECT-Anweisungen

- Mit SELECT-Anweisungen können Sie:
  - die anzuzeigenden Spalten bestimmen
  - Daten aus einzelnen oder mehreren Tabellen, Objekttabellen, Views, Objekt-Views und Materialized Views abrufen
- SELECT-Anweisungen werden auch als Abfragen bezeichnet, da sie eine Datenbank abfragen.
- Syntax:

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


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

- Eine SELECT-Klausel, in der die anzuzeigenden Spalten angegeben sind
- Eine FROM-Klausel, in der die Tabelle mit den in der Klausel SELECT aufgeführten Spalten angegeben ist

Für die Syntax gilt:

SELECT	Listet eine oder mehrere Spalten
*	Wählt alle Spalten
DISTINCT	Unterdrückt doppelte Werte
column|*expression*	Wählt die angegebene Spalte oder den angegebenen Ausdruck
alias	Weist den gewählten Spalten andere Überschriften zu
FROM *table*	Gibt die Tabelle an, die die Spalten enthält

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

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

SELECT-Anweisungen

- Alle Spalten wählen:

```
SELECT *
FROM   job_history;
```

	EMPLOYEE_ID	START_DATE	END_DATE	JOB_ID	DEPARTMENT_ID
1	102	13-JAN-01	24-JUL-06	IT_PROG	60
2	101	21-SEP-97	27-OCT-01	AC_ACCOUNT	110
3	101	28-OCT-01	15-MAR-05	AC_MGR	110
4	201	17-FEB-04	19-DEC-07	MK_REP	20
5	114	24-MAR-06	31-DEC-07	ST_CLERK	50
6	122	01-JAN-07	31-DEC-07	ST_CLERK	50
7	200	17-SEP-95	17-JUN-01	AD_ASST	90
8	176	24-MAR-06	31-DEC-06	SA_REP	80
9	176	01-JAN-07	31-DEC-07	SA_MAN	80
10	200	01-JUL-02	31-DEC-06	AC_ACCOUNT	90

- Bestimmte Spalten wählen:

```
SELECT manager_id, job_id
FROM   employees;
```

	MANAGER_ID	JOB_ID
1	(null)	AD_PRES
2	100	AD_VP
3	100	AD_VP
4	102	IT_PROG
5	103	IT_PROG
6	103	IT_PROG
7	100	ST_MAN
8	124	ST_CLERK
9	124	ST_CLERK
10	124	ST_CLERK

...

ORACLE

Um alle Datenspalten einer Tabelle anzuzeigen, können Sie nach dem Schlüsselwort `SELECT` ein Sternchen (`*`) angeben oder die Namen aller Spalten auflisten. Im ersten Beispiel auf der Folie werden alle Zeilen der Tabelle `job_history` angezeigt. Um bestimmte Spalten der Tabelle anzuzeigen, geben Sie die entsprechenden Spaltennamen durch Kommas getrennt an. Im zweiten Beispiel auf der Folie werden die Spalten `manager_id` und `job_id` aus der Tabelle `employees` angezeigt.

Geben Sie in der Klausel `SELECT` die Spalten in der Reihenfolge an, in der sie in der Ausgabe angezeigt werden sollen. Beispiel: Die folgende SQL-Anweisung zeigt die Spalte `location_id` vor der Spalte `department_id` an:

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

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

WHERE-Klauseln

- Mit der optionalen WHERE-Klausel:
  - Zeilen in einer Abfrage filtern
  - Teilmenge von Zeilen erstellen
- Syntax:

```
SELECT * FROM   table
[WHERE   condition];
```

- Beispiel:

```
SELECT location_id from departments
WHERE department_name = 'Marketing';
```


Die Klausel WHERE gibt eine Bedingung an, mit der Zeilen gefiltert werden und eine Teilmenge von Zeilen aus der Tabelle abgerufen wird. Bedingungen umfassen einen oder mehrere Ausdrücke und logische (boolesche) Operatoren. Sie geben als Ergebnis den Wert TRUE, FALSE oder NULL zurück. Im Beispiel auf der Folie wird die location_id der Marketingabteilung abgerufen.

Mit WHERE-Klauseln können Sie auch Daten aus der Datenbank aktualisieren oder löschen.

Beispiele:

```
UPDATE departments
SET  department_name = 'Administration'
WHERE department_id = 20;
```

und

```
DELETE from departments
WHERE department_id =20;
```

ORDER BY-Klauseln

- Die optionale ORDER BY-Klausel regelt die Reihenfolge der Zeilen.
- Syntax:

```
SELECT * FROM   table
[WHERE   condition]
[ORDER BY {<column>|<position> } [ASC|DESC] [, ...] ];
```

- Beispiel:

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

ORACLE

Mit der Klausel ORDER BY legen Sie fest, in welcher Reihenfolge die Zeilen angezeigt werden sollen. Zeilen können auf- oder absteigend sortiert werden. Die standardmäßige Sortierfolge für Zeilen ist aufsteigend.

Im Beispiel auf der Folie werden Zeilen aus der Tabelle employees abgerufen. Die Zeilen werden zuerst aufsteigend nach department_id und anschließend absteigend nach salary sortiert.

GROUP BY-Klauseln

- Mit der optionalen GROUP BY-Klausel fassen Sie Spalten mit übereinstimmenden Werten zu Untergruppen zusammen.
- In keiner Gruppe gibt es zwei Zeilen mit identischem Wert für die Spalte(n), nach der oder denen die Gruppierung erfolgt.
-
```
SELECT <column1, column2, ... column_n>
FROM  table
[WHERE   condition]
[GROUP BY <column> [, ...] ]
[ORDER BY <column> [, ...] ] ;
```
- Beispiel:

```
SELECT department_id, MIN(salary), MAX (salary)
FROM employees
GROUP BY department_id ;
```

ORACLE

Mit der Klausel GROUP BY werden gewählte Zeilen entsprechend den Werten eines oder mehrerer Ausdrücke zu einer Gruppe zusammengefasst. Die Klausel gruppiert Zeilen, garantiert jedoch nicht die Reihenfolge der Ergebnismenge. Um die Reihenfolge der Gruppen festzulegen, geben Sie die Klausel ORDER BY an.

Alle Elemente der SELECT-Liste, die nicht Teil von Aggregatfunktionen sind, müssen in die Elementliste von GROUP BY aufgenommen werden. Dies gilt sowohl für Spalten als auch für Ausdrücke. Die Datenbank gibt für jede Gruppe eine einzelne Zeile mit Aggregatinformationen zurück.

Im Beispiel auf der Folie wird für jede Abteilung aus der Tabelle employees das niedrigste und das höchste Gehalt zurückgegeben.

Data Definition Language (DDL)

- Mit DDL-Anweisungen können Sie Schemaobjekte definieren und löschen sowie die Struktur von Schemaobjekten ändern.
- Häufige DDL-Anweisungen:
  - `CREATE TABLE`, `ALTER TABLE` und `DROP TABLE`
  - `GRANT`, `REVOKE`
  - `TRUNCATE`

Mit DDL-Anweisungen können Sie die Attribute von Objekten ändern, ohne die Anwendungen zu ändern, die auf das Objekt zugreifen. Darüber hinaus können Sie die Struktur von Objekten ändern, während Benutzer in der Datenbank arbeiten. Häufige Verwendungsbereiche von DDL-Anweisungen:

- Schemaobjekte und andere Datenbankstrukturen (einschließlich der Datenbank selbst und der Datenbankbenutzer) erstellen, ändern und löschen
- Alle Daten in Schemaobjekten löschen, ohne die Struktur der Objekte zu entfernen
- Berechtigungen und Rollen erteilen und entziehen

Oracle Database schreibt die aktuelle Transaktion vor und nach jeder DDL-Anweisung implizit fest.

CREATE TABLE-Anweisungen

- Mit der Anweisung CREATE TABLE erstellen Sie Tabellen in der Datenbank.
- Syntax:

```
CREATE TABLE tablename (
{column-definition | Table-level constraint}
[ , {column-definition | Table-level constraint} ] * )
```

- Beispiel:

```
CREATE TABLE teach_dept (
department_id  NUMBER(3) PRIMARY KEY,
department_name VARCHAR2(10));
```


Mit der Anweisung CREATE TABLE erstellen Sie Tabellen in der Datenbank. Um Tabellen erstellen zu können, müssen Sie über die Berechtigung CREATE TABLE und einen Speicherbereich verfügen, in dem die Objekte erstellt werden.

Tabellen- und Datenbankeigentümer erhalten nach Erstellung der Tabelle automatisch folgende Berechtigungen:

- INSERT
- SELECT
- REFERENCES
- ALTER
- UPDATE

Tabellen- und Datenbankeigentümer können diese Berechtigungen auch anderen Benutzern erteilen.

ALTER TABLE-Anweisungen

- Mit der Anweisung ALTER TABLE ändern Sie die Definition einer vorhandenen Tabelle in der Datenbank.
- 1. Beispiel:

```
ALTER TABLE  teach_dept
ADD  location_id  NUMBER  NOT NULL;
```

- 2. Beispiel:

```
ALTER TABLE teach_dept
MODIFY department_name  VARCHAR2(30) NOT NULL;
```

ORACLE

Mit der Anweisung ALTER TABLE können Sie Änderungen an einer vorhandenen Tabelle vornehmen.

Sie können:

- Spalten zu Tabellen hinzufügen
- Constraints zu Tabellen hinzufügen
- vorhandene Spaltendefinitionen ändern
- Spalten aus Tabellen löschen
- vorhandene Constraints aus Tabellen löschen
- die Breite der Spalten VARCHAR und CHAR vergrößern
- Tabellen in den schreibgeschützten Status setzen

Im 1. Beispiel auf der Folie wird die Tabelle TEACH_DEPT um die neue Spalte LOCATION_ID erweitert.

Im 2. Beispiel wird die vorhandene Spalte department_name von VARCHAR2(10) in VARCHAR2(30) geändert und das Constraint NOT NULL hinzugefügt.

DROP TABLE-Anweisungen

- Mit der Anweisung DROP TABLE löschen Sie die Tabelle mit sämtlichen darin enthaltenen Daten aus der Datenbank.
- Beispiel:

```
DROP TABLE teach_dept;
```

- DROP TABLE mit der Klausel PURGE löscht die Tabelle und gibt den zugewiesenen Speicherplatz frei.

```
DROP TABLE teach_dept PURGE;
```

- Bei Verwendung der Klausel CASCADE CONSTRAINTS werden alle Constraints zur referenziellen Integrität aus der Tabelle gelöscht.

```
DROP TABLE teach_dept CASCADE CONSTRAINTS;
```

ORACLE

Mit der Anweisung DROP TABLE werden Tabellen mitsamt ihrem Inhalt aus der Datenbank gelöscht und in den Papierkorb verschoben. Beim Löschen einer Tabelle werden die abhängigen Objekte invalidiert und die Objektberechtigungen für die Tabelle entfernt.

Um den der Tabelle zugewiesenen Speicherplatz für den Tablespace wieder freizugeben, verwenden Sie die Anweisung DROP TABLE mit der Klausel PURGE. DROP TABLE-Anweisungen mit der Klausel PURGE können nicht zurückgerollt werden. Ebenso können Tabellen, die mit der Klausel PURGE gelöscht wurden, nicht wiederhergestellt werden.

Mit der Klausel CASCADE CONSTRAINTS können Sie die Referenz auf den Primärschlüssel und die eindeutigen Schlüssel in der gelöschten Tabelle aufheben.

GRANT-Anweisungen

- Die Anweisung GRANT erteilt Benutzern die Berechtigung zur Ausführung folgender Vorgänge:
  - Daten einfügen und löschen
  - Fremdschlüsselreferenz auf die benannte Tabelle oder eine Teilmenge von Spalten aus der Tabelle erstellen
  - Daten, Views oder eine Teilmenge von Spalten aus der Tabelle wählen
  - Trigger für Tabellen erstellen
  - Angegebene Funktionen oder Prozeduren ausführen
- Beispiel:

```
GRANT SELECT any table to PUBLIC;
```

ORACLE

Mit der Anweisung GRANT können Sie:

- bestimmten Benutzern bzw. Rollen oder allen Benutzern Berechtigungen zur Ausführung von Aktionen an Datenbankobjekten erteilen
- Benutzern, PUBLIC oder anderen Rollen eine Rolle zuweisen

Prüfen Sie vor dem Absetzen einer GRANT-Anweisung, ob für die Autorisierungseigenschaft derby.database.sql der Wert True eingestellt ist. Mit dieser Eigenschaft wird der SQL-Autorisierungsmodus aktiviert. Wenn Sie der Eigentümer der Datenbank sind, können Sie Berechtigungen für ein Objekt erteilen.

Um allen Benutzern Berechtigungen zu erteilen, verwenden Sie das Schlüsselwort PUBLIC. Ist PUBLIC angegeben, gelten die Berechtigungen oder Rollen für alle aktuellen und künftigen Benutzer.

Typen von Berechtigungen

- Folgende Berechtigungen mit der Anweisung `GRANT` zuweisen:
  - `ALL PRIVILEGES`
  - `DELETE`
  - `INSERT`
  - `REFERENCES`
  - `SELECT`
  - `UPDATE`

Oracle Database bietet verschiedene Berechtigungstypen für Benutzer und Rollen:

- Mit dem Berechtigungstyp `ALL PRIVILEGES` erteilen Sie Benutzern und Rollen sämtliche Berechtigungen für die angegebene Tabelle.
- Mit dem Berechtigungstyp `DELETE` erteilen Sie die Berechtigung, Zeilen aus der angegebenen Tabelle zu löschen.
- Mit dem Berechtigungstyp `INSERT` erteilen Sie die Berechtigung, Zeilen in die angegebene Tabelle einzufügen.
- Mit dem Berechtigungstyp `REFERENCES` erteilen Sie die Berechtigung zur Erstellung einer Fremdschlüsselreferenz auf die angegebene Tabelle.
- Mit dem Berechtigungstyp `SELECT` erteilen Sie die Berechtigung zur Ausführung von `SELECT`-Anweisungen für eine Tabelle oder View.
- Mit dem Berechtigungstyp `UPDATE` erteilen Sie die Berechtigung zur Verwendung von `UPDATE`-Anweisungen für die angegebene Tabelle.

REVOKE-Anweisungen

- Mit REVOKE-Anweisungen entziehen Sie Benutzern Berechtigungen zur Ausführung von Aktionen an Datenbankobjekten.
- Benutzern eine *Systemberechtigung* entziehen:

```
REVOKE DROP ANY TABLE
FROM hr;
```

- Benutzern eine *Rolle* entziehen:

```
REVOKE dw_manager
FROM sh;
```

ORACLE

Mit der Anweisung REVOKE entziehen Sie bestimmten oder allen Benutzern oder Rollen die Berechtigungen zur Ausführung von Aktionen an Datenbankobjekten. Folgende Vorgänge sind möglich:

- Benutzern, PUBLIC oder anderen Rollen eine Rolle entziehen
- Objekten Berechtigungen entziehen, wenn Sie der Eigentümer des Objekts oder der Datenbank sind

Hinweis: Um Rollen oder Systemberechtigungen entziehen zu können, müssen Sie über eine Berechtigung mit der ADMIN OPTION verfügen.

TRUNCATE TABLE-Anweisungen

- Mit TRUNCATE TABLE-Anweisungen löschen Sie alle Zeilen einer Tabelle.
- Beispiel:

```
TRUNCATE TABLE employees_demo;
```

- Oracle Database führt standardmäßig folgende Aufgaben durch:
  - Von gelöschten Zeilen belegten Speicherplatz freigeben
  - Speicherparameter NEXT auf die Größe des letzten Extents einstellen, das durch den Leerungsprozess aus dem Segment entfernt wurde

ORACLE

Mit der Anweisung TRUNCATE TABLE löschen Sie alle Zeilen aus einer angegebenen Tabelle. Diese Vorgehensweise ist häufig effizienter, als die Tabelle vollständig zu löschen und neu zu erstellen. Beim Löschen und Neuerstellen von Tabellen gilt:

- Die abhängigen Objekte der Tabelle werden invalidiert.
- Objektberechtigungen müssen neu vergeben werden.
- Indizes, Integritäts-Constraints und Trigger müssen neu erstellt werden.
- Speicherparameter müssen neu festgelegt werden.

Bei Verwendung der Anweisung TRUNCATE TABLE können Sie auf diese Arbeitsschritte verzichten.

Hinweis: TRUNCATE TABLE-Anweisungen können nicht zurückgerollt werden.

Data Manipulation Language (DML)

- Mit DML-Anweisungen können Sie Daten in vorhandenen Schemaobjekten abfragen und bearbeiten.
- DML-Anweisungen werden ausgeführt, wenn Sie:
  - mit der Anweisung `INSERT` neue Zeilen in Tabellen einfügen
  - mit der Anweisung `UPDATE` vorhandene Zeilen in Tabellen ändern
  - mit der Anweisung `DELETE` vorhandene Zeilen aus Tabellen löschen
- Eine *Transaktion* enthält eine Zusammenstellung von DML-Anweisungen, die eine logische Arbeitseinheit bilden.

ORACLE

Mit DML-Anweisungen können Sie den Inhalt vorhandener Schemaobjekte abfragen oder ändern. Häufige Verwendungsbereiche von DDL-Anweisungen:

- Neue Datenzeilen in Tabellen oder Views einfügen (durch Angabe einer Liste von Spaltenwerten oder durch Auswahl und Bearbeitung vorhandener Daten mithilfe einer Unterabfrage)
- Spaltenwerte in den vorhandenen Zeilen einer Tabelle oder View ändern
- Zeilen aus Tabellen oder Views entfernen

Eine Zusammenstellung von DML-Anweisungen, die eine logische Arbeitseinheit bilden, wird als Transaktion bezeichnet. Im Gegensatz zu DDL-Anweisungen schreiben DML-Anweisungen die aktuelle Transaktion nicht implizit fest.

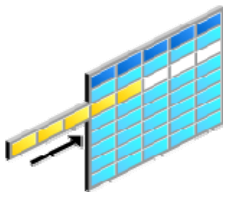
INSERT-Anweisungen

- Mit INSERT-Anweisungen fügen Sie Tabellen neue Zeilen hinzu.
- Syntax:

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

- Beispiel:

```
INSERT INTO  departments
VALUES     (200,'Development',104,1400);
1 rows inserted.
```



ORACLE

Mit der Anweisung INSERT fügen Sie einer Tabelle neue Zeilen hinzu. Die neuen Zeilen müssen Werte für die einzelnen Spalten enthalten und die Werte in der für die Spalten der Tabelle gültigen Standardreihenfolge aufgeführt sein. Optional können Sie die Spalten auch in der Anweisung INSERT auflisten.

Beispiel:

```
INSERT INTO job_history (employee_id, start_date,  end_date,
   job_id)
VALUES (120,'25-JUL-06','12-FEB_08','AC_ACCOUNT');
```

Mit der auf der Folie gezeigten Syntax können Sie jeweils eine einzelne Zeile einfügen. Das Schlüsselwort VALUES weist die Werte von Ausdrücken den entsprechenden Spalten in der Spaltenliste zu.

UPDATE-Anweisungen – Syntax

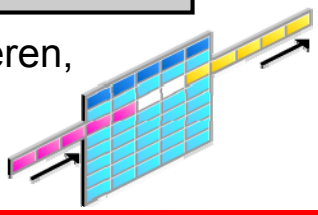
- Mit der Anweisung UPDATE ändern Sie die vorhandenen Zeilen in einer Tabelle.
- Falls erforderlich, können mehrere Zeilen gleichzeitig aktualisiert werden.

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

- Beispiel:

```
UPDATE     copy_emp
SET
22 rows updated
```

- Um einen Spaltenwert mit NULL zu aktualisieren, geben Sie SET *column_name*= NULL an.



ORACLE

Mit der Anweisung UPDATE ändern Sie die in einer Tabelle vorhandenen Werte. Sie können den Aktualisierungsvorgang prüfen, indem Sie die Tabelle abfragen und die aktualisierten Zeilen anzeigen. Um nur bestimmte Zeilen zu ändern, geben Sie die Klausel WHERE an.

Beispiel:

```
UPDATE employees
SET    salary = 17500
WHERE  employee_id = 102;
```

Im Allgemeinen identifizieren Sie die zu aktualisierende Zeile in der Klausel WHERE mit der Primärschlüsselspalte. Beispiel: Sie möchten eine bestimmte Zeile in der Tabelle employee_id aktualisieren. In diesem Fall geben Sie die Zeile nicht mit employee_name, sondern mit employee_id an, da eventuell mehrere Mitarbeiter denselben Namen haben können.

Hinweis: Das Schlüsselwort condition setzt sich in der Regel aus Spaltennamen, Ausdrücken, Konstanten, Unterabfragen und Vergleichsoperatoren zusammen.

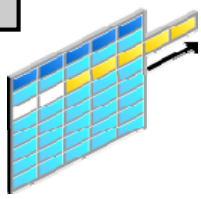
DELETE-Anweisungen

- Mit der Anweisung DELETE löschen Sie vorhandene Zeilen aus einer Tabelle.
- Syntax:

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

- Um bestimmte Zeilen aus einer Tabelle zu löschen, geben Sie die Anweisung DELETE mit der Klausel WHERE an.

```
DELETE FROM departments
WHERE  department_name = 'Finance';
1 rows deleted
```



Mit der Anweisung DELETE entfernen Sie vorhandene Zeilen aus einer Tabelle. Um bestimmte Zeilen auf Basis einer Bedingung zu löschen, verwenden Sie die Klausel WHERE. Die Bedingung (condition) gibt die zu löschenden Zeilen an. Sie kann Spaltennamen, Ausdrücke, Konstanten, Unterabfragen und Vergleichsoperatoren enthalten.

Im ersten Beispiel auf der Folie wird die Finanzabteilung aus der Tabelle departments gelöscht. Um den Löschvorgang zu prüfen, fragen Sie die Tabelle mit der Anweisung SELECT ab.

```
SELECT  *
FROM    departments
WHERE   department_name = 'Finance';
```

Wenn Sie die Klausel WHERE weglassen, werden alle Zeilen aus der Tabelle gelöscht. Beispiel:

```
DELETE FROM  copy_emp;
```

Im vorstehenden Beispiel werden alle Zeilen aus der Tabelle COPY_EMP gelöscht.

Anweisungen zur Transaktionskontrolle

- Mit Anweisungen zur Transaktionskontrolle werden die von DML-Anweisungen vorgenommenen Änderungen verwaltet.
- Die DML-Anweisungen werden zu Transaktionen zusammengefasst.
- Anweisungen zur Transaktionskontrolle:
  - `COMMIT`
  - `ROLLBACK`
  - `SAVEPOINT`

ORACLE

Eine Transaktion besteht aus einer Folge von SQL-Anweisungen, die von Oracle Database als Einheit behandelt werden. Anweisungen zur Transaktionskontrolle werden in einer Datenbank verwendet, um die über DML-Anweisungen vorgenommenen Änderungen zu verwalten und die Anweisungen zu Transaktionen zusammenzufassen.

Jeder Transaktion wird eine eindeutige `transaction_id` zugewiesen. Die in der Transaktion enthaltenen SQL-Anweisungen werden entweder alle festgeschrieben (das heißt, auf die Datenbank angewendet) oder alle zurückgerollt (das heißt, in der Datenbank rückgängig gemacht).

COMMIT-Anweisungen

- Mit COMMIT-Anweisungen können Sie:
  - die während der aktuellen Transaktion vorgenommenen Änderungen endgültig speichern
  - alle Savepoints in der Transaktion löschen
  - Transaktionssperren aufheben
- Beispiel:

```
INSERT INTO  departments
VALUES     (201, 'Engineering', 106, 1400);
COMMIT;
```

```
1 rows inserted.
commited.
```

ORACLE

Wenn Sie die aktuelle Transaktion mit der Anweisung COMMIT beenden, werden alle noch nicht gespeicherten Datenänderungen dauerhaft festgeschrieben. Dabei werden alle Zeilen- und Tabellensperren aufgehoben sowie alle seit dem letzten Commit- oder Rollback-Vorgang eventuell festgelegten Savepoints gelöscht. Die mit der Anweisung COMMIT vorgenommenen Änderungen sind für alle Benutzer sichtbar.

Oracle empfiehlt, alle in Ihren Anwendungsprogrammen ausgeführten Transaktionen (einschließlich der letzten Transaktion) explizit mit COMMIT oder ROLLBACK zu beenden, bevor Sie sich bei Oracle Database abmelden. Andernfalls wird im Falle eines Programmabsturzes die letzte nicht festgeschriebene Transaktion automatisch zurückgerollt.

Hinweis: Oracle Database setzt vor und nach jeder DDL-Anweisung implizite COMMIT-Anweisungen ab.

ROLLBACK-Anweisungen

- Mit der Anweisung ROLLBACK machen Sie die während der aktuellen Transaktion an der Datenbank vorgenommenen Änderungen rückgängig.
- Um nur den Teil der Transaktion nach dem Savepoint rückgängig zu machen, verwenden Sie die Klausel TO SAVEPOINT.
- Beispiel:

```
UPDATE          employees
SET             salary = 7000
WHERE           last_name = 'Ernst';
SAVEPOINT       Ernst_sal;

UPDATE          employees
SET             salary = 12000
WHERE           last_name = 'Mourgos';

ROLLBACK TO SAVEPOINT Ersnt_sal;
```

ORACLE

Mit der Anweisung ROLLBACK werden die in der aktuellen Transaktion durchgeführten Aktionen zurückgesetzt. Um die aktuelle Transaktion zurückzurollen, sind keine Berechtigungen erforderlich.

Mit ROLLBACK und der Klausel TO SAVEPOINT werden die folgenden Vorgänge durchgeführt:

- Nur den nach dem Savepoint ausgeführten Teil der Transaktion zurückrollen
- Alle nach dem angegebenen Savepoint erstellten Savepoints löschen. Der angegebene Savepoint bleibt erhalten, sodass Sie mehrere Rollbacks zu diesem Savepoint vornehmen können.

Mit ROLLBACK ohne Angabe der Klausel TO SAVEPOINT werden folgende Vorgänge durchgeführt:

- Transaktion beenden
- Alle während der aktuellen Transaktion vorgenommenen Änderungen rückgängig machen
- Alle Savepoints in der Transaktion löschen

SAVEPOINT-Anweisungen

- Mit der Anweisung SAVEPOINT benennen und markieren Sie den aktuellen Verarbeitungspunkt einer Transaktion.
- Geben Sie für jeden Savepoint einen Namen an.
- Vergeben Sie innerhalb einer Transaktion eindeutige Savepoint-Namen, um Überschreibungen zu vermeiden.
- Syntax:

```
SAVEPOINT savepoint;
```

ORACLE

Mit der Anweisung SAVEPOINT kennzeichnen Sie einen Punkt in der Transaktion, bis zu dem Sie die Transaktion im Falle eines Rollbacks zurücksetzen können. Jeder Savepoint muss über einen eindeutigen Namen verfügen. Wenn Sie einen zweiten Savepoint mit demselben Namen erstellen, wird der frühere Savepoint gelöscht.

Nach der Erstellung eines Savepoints können Sie entweder mit der Verarbeitung fortfahren, Ihre Arbeit festschreiben, die gesamte Transaktion zurückrollen oder alle nach dem Savepoint ausgeführten Aktionen zurückrollen.

Bei einem einfachen Rollback- oder Commit-Vorgang werden alle Savepoints gelöscht. Bei einem Rollback bis zu einem Savepoint werden alle späteren Savepoints gelöscht. Der Savepoint, bis zu dem die Transaktion zurückgerollt wurde, bleibt erhalten.

Wenn Savepoint-Namen innerhalb einer Transaktion mehrfach verwendet werden, verschiebt Oracle Database den Savepoint von der bisherigen an die aktuelle Position in der Transaktion und setzt damit den älteren Savepoint außer Kraft.

Joins

Mit Joins fragen Sie Daten aus mehreren Tabellen ab:

```
SELECT   table1.column, table2.column
FROM     table1, table2
WHERE    table1.column1 = table2.column2;
```

- Join-Bedingungen werden in der Klausel `WHERE` angegeben.
- Setzen Sie den Tabellennamen vor den Spaltennamen, wenn derselbe Spaltenname in mehreren Tabellen vorkommt.

Um Daten aus mehreren Tabellen einer Datenbank abzurufen, werden *Join*-Bedingungen verwendet. Zeilen aus zwei Tabellen werden in Abhängigkeit von gemeinsamen Werten verknüpft, die in den entsprechenden Spalten (in der Regel Primär- und Fremdschlüsselspalten) vorhanden sind.

Um Daten aus zwei oder mehr verknüpften Tabellen anzuzeigen, geben Sie in der Klausel `WHERE` eine einfache Join-Bedingung an.

Für die Syntax gilt:

table1.column	Tabelle und Spalte, aus der Daten abgerufen werden
table1.column1 = table2.column2	Bedingung für die Verknüpfung (oder Relation) der Tabellen

Richtlinien

- Um den Code übersichtlicher zu gestalten und den Datenbankzugriff zu vereinfachen, sollten Sie in `SELECT`-Anweisungen, die Tabellen verknüpfen, den Spaltennamen Tabellennamen voranstellen.
- Kommt derselbe Spaltenname in mehreren Tabellen vor, ist der Tabellenname als Präfix obligatorisch.
- Um *n* Tabellen miteinander zu verknüpfen, sind mindestens `n-1` Join-Bedingungen erforderlich. Beispiel: Um vier Tabellen zu verknüpfen, werden mindestens drei Joins benötigt. In Tabellen mit verkettetem Primärschlüssel trifft diese Regel möglicherweise nicht zu. In diesem Fall werden mehrere Spalten benötigt, um die einzelnen Zeilen eindeutig zu identifizieren.

Typen von Joins

- Natural Joins
- Equi Joins
- Non-Equi Joins
- Outer Joins
- Self Joins
- Cross Joins

ORACLE

Mit der Join-Syntax von Oracle können Sie Tabellen verknüpfen.

Hinweis: In Datenbankreleases bis Oracle 9*i* galt für Joins eine Oracle-spezifische Syntax. Die SQL:1999-konforme Join-Syntax bietet gegenüber der Oracle-spezifischen Join-Syntax keine Performancevorteile.

Mehrdeutige Spaltennamen eindeutig kennzeichnen

- Spaltennamen, die in mehreren Tabellen vorkommen, müssen durch ein Tabellenpräfix eindeutig gekennzeichnet werden.
- Tabellenpräfixe verbessern die Performance.
- Anstelle von vollständigen Tabellennamen können Sie Tabellenaliasnamen verwenden.
- Tabellenaliasnamen verkürzen den Tabellennamen:
  - Kürzerer SQL-Code, weniger Speicherbedarf
- Spaltenaliasnamen dienen der Unterscheidung von Spalten mit identischen Namen, die in verschiedenen Tabellen vorkommen.

ORACLE

Wenn Sie zwei oder mehr Tabellen miteinander verknüpfen, müssen die Spaltennamen durch den Tabellennamen gekennzeichnet werden, um Mehrdeutigkeit zu vermeiden. Ohne das Tabellenpräfix könnte sich der Spaltenname `DEPARTMENT_ID` in der `SELECT`-Liste auf die entsprechende Spalte in der Tabelle `DEPARTMENTS` oder in der Tabelle `EMPLOYEES` beziehen. Daher müssen Sie vor Ausführung der Abfrage das Tabellenpräfix hinzufügen. Gibt es in den Tabellen keine identischen Spaltennamen, besteht keine Notwendigkeit, die Spalten durch Angabe des Tabellennamens eindeutig zu kennzeichnen. Tabellenpräfixe verbessern jedoch die Performance, weil sie dem Oracle-Server genau angeben, wo sich die Spalten befinden.

Die eindeutige Kennzeichnung von Spaltennamen durch Tabellennamen kann insbesondere bei langen Tabellennamen zeitaufwendig sein. Sie können daher (analog zu Aliasnamen für Spalten) anstelle von Tabellennamen einen kürzeren *Tabellenalias* verwenden. Tabellenaliasnamen verkürzen den SQL-Code und sind daher weniger speicherintensiv.

Der Tabellenalias folgt auf den vollständig angegebenen Tabellennamen, getrennt durch ein Leerzeichen. Beispielsweise kann die Tabelle `EMPLOYEES` den Alias `e` und die Tabelle `DEPARTMENTS` den Alias `d` erhalten.

Richtlinien

- Tabellenaliasnamen können bis zu 30 Zeichen lang sein, kürzere Aliasnamen sind jedoch besser.
- Tabellenaliasnamen, die in der Klausel `FROM` bestimmte Tabellennamen ersetzen, müssen in der gesamten `SELECT`-Anweisung anstelle des Tabellennamens verwendet werden.
- Wählen Sie möglichst aussagekräftige Tabellenaliasnamen.
- Tabellenaliasnamen gelten nur für die aktuelle `SELECT`-Anweisung.

Natural Joins

- Die Klausel `NATURAL JOIN` verbindet zwei Tabellen auf der Basis von allen Spalten, die in beiden Tabellen denselben Namen haben.
- Die Klausel wählt die Zeilen, deren Spalten in den Tabellen identische Namen und Datenwerte aufweisen.
- Beispiel:

```
SELECT country_id, location_id, country_name,city
FROM countries NATURAL JOIN locations;
```

	COUNTRY_ID	LOCATION_ID	COUNTRY_NAME	CITY
1	US	1400	United States of America	Southlake
2	US	1500	United States of America	South San Francisco
3	US	1700	United States of America	Seattle
4	CA	1800	Canada	Toronto
5	UK	2500	United Kingdom	Oxford

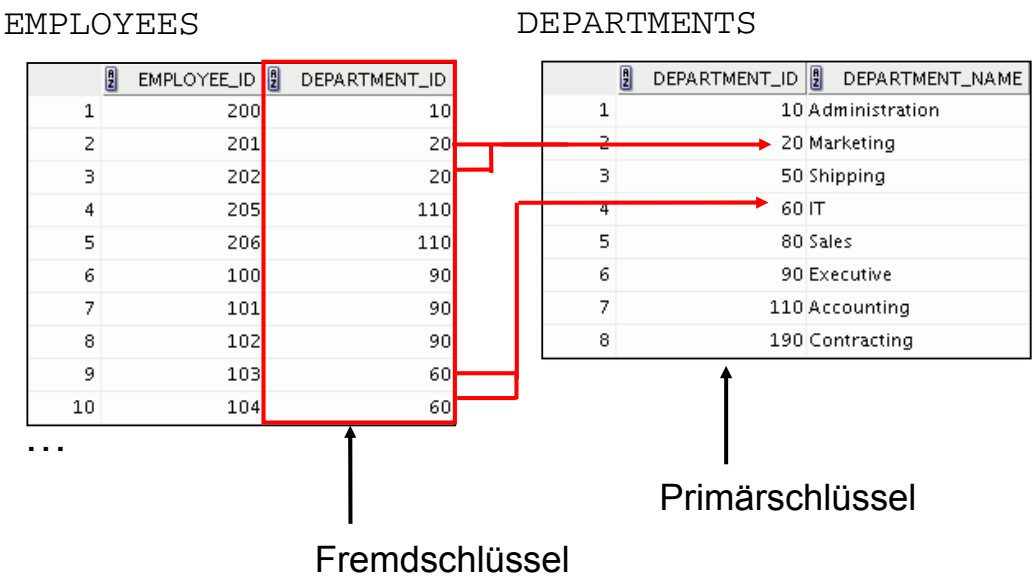
ORACLE

Sie können Tabellen automatisch verknüpfen, deren Spalten in beiden Tabellen denselben Namen und Datentyp aufweisen. Hierzu verwenden Sie die Schlüsselwörter `NATURAL JOIN`.

Hinweis: Joins sind nur für Spalten möglich, die in beiden Tabellen denselben Namen und denselben Datentyp aufweisen. Haben die Spalten denselben Namen, jedoch unterschiedliche Datentypen, verursacht die Syntax `NATURAL JOIN` einen Fehler.

Im Beispiel auf der Folie werden die Tabellen `COUNTRIES` und `LOCATIONS` über die Spalte `COUNTRY_ID` verknüpft, da dies der einzige Spaltenname ist, der in beiden Tabellen vorkommt. Wären weitere übereinstimmende Spalten vorhanden, würde der Join alle diese Spalten berücksichtigen.

Equi Joins



Ein **Equi Join** ist ein Join mit einer Join-Bedingung, die einen Gleichheitsoperator enthält. Equi Joins verbinden Zeilen, die in den angegebenen Spalten äquivalente Werte enthalten. Um die Abteilung eines Mitarbeiters zu ermitteln, vergleichen Sie die Werte der Spalte `DEPARTMENT_ID` in der Tabelle `EMPLOYEES` mit den Werten der Spalte `DEPARTMENT_ID` in der Tabelle `DEPARTMENTS`. Die Beziehung zwischen den Tabellen `EMPLOYEES` und `DEPARTMENTS` ist ein *Equi Join*, das heißt, die Werte der Spalte `DEPARTMENT_ID` müssen in beiden Tabellen gleich sein. Häufig sind bei dieser Art von Join sich ergänzende Primärschlüssel und Fremdschlüssel beteiligt.

Hinweis: Equi Joins werden auch als *einfache Joins* bezeichnet.

Datensätze mit Equi Joins abrufen

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e JOIN departments d
ON e.department_id = d.department_id;
```

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID_1	LOCATION_ID
1	200	Whalen	10	10	1700
2	201	Hartstein	20	20	1800
3	202	Fay	20	20	1800
4	144	Vargas	50	50	1500
5	143	Matos	50	50	1500
6	142	Davies	50	50	1500
7	141	Rajs	50	50	1500
8	124	Mourgos	50	50	1500
9	103	Hunold	60	60	1400
10	104	Ernst	60	60	1400
11	107	Lorentz	60	60	1400

...

Im Beispiel auf der Folie gilt:

- **Die Klausel `SELECT` gibt die abzurufenden Spaltennamen an:**
  - Nachname, Personalnummer und Abteilungsnummer des Mitarbeiters aus den entsprechenden Spalten der Tabelle `EMPLOYEES`
  - Abteilungs-ID und Standort-ID aus den entsprechenden Spalten der Tabelle `DEPARTMENTS`
- **Die Klausel `FROM` gibt die beiden Tabellen an, auf die die Datenbank zugreifen muss:**
  - `EMPLOYEES`
  - `DEPARTMENTS`
- **Die Klausel `WHERE` gibt an, wie die Tabellen verknüpft werden sollen:**

  `e.department_id = d.department_id`

Da die Spalte `DEPARTMENT_ID` in beiden Tabellen enthalten ist, muss dem Spaltennamen der Tabellenalias vorangestellt werden, um Mehrdeutigkeit zu vermeiden. Eine Kennzeichnung anderer Spalten, die nicht in beiden Tabellen vorkommen, durch einen Tabellenalias ist nicht erforderlich, empfiehlt sich jedoch aus Performancegründen.

Hinweis: Wenn Sie die Abfrage mit dem Symbol **Run Statement** ausführen, fügt SQL Developer zur Unterscheidung der beiden `DEPARTMENT_ID`s das Suffix "_1" an.

Zusätzliche Suchbedingungen mit den Operatoren AND und WHERE

```
SELECT  d.department_id, d.department_name, l.city
FROM    departments d JOIN locations l
ON  d.location_id = l.location_id
AND d.department_id IN (20, 50);
```

	DEPARTMENT_ID	DEPARTMENT_NAME	CITY
1	20	Marketing	Toronto
2	50	Shipping	South San Francisco

```
SELECT  d.department_id, d.department_name, l.city
FROM    departments d JOIN locations l
ON  d.location_id = l.location_id
WHERE d.department_id IN (20, 50);
```

ORACLE

Neben dem Join können Sie auch Kriterien für die Klausel WHERE angeben, um die für eine oder mehrere Tabellen im Join berücksichtigten Zeilen einzuschränken. Im Beispiel auf der Folie werden die Tabellen DEPARTMENTS und LOCATIONS verknüpft und darüber hinaus nur die Abteilungen mit der ID 20 oder 50 angezeigt. Um zusätzliche Bedingungen für die Klausel ON aufzunehmen, können Sie AND-Klauseln hinzufügen. Alternativ können Sie zusätzliche Bedingungen über eine WHERE-Klausel anwenden.

Beide Abfragen führen zum selben Ergebnis.

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Matos	50	Shipping

Datensätze mit Non-Equi Joins abrufen

```
SELECT e.last_name, e.salary, j.grade_level
FROM   employees e JOIN job_grades j
ON  e.salary
       BETWEEN j.lowest_sal AND j.highest_sal;
```

	LAST_NAME	SALARY	GRADE_LEVEL
1	Vargas	2500	A
2	Matos	2600	A
3	Davies	3100	B
4	Rajs	3500	B
5	Lorentz	4200	B
6	Whalen	4400	B
7	Fay	6000	C

...

Im Beispiel auf der Folie wird ein Non-Equi Join erstellt, um die Gehaltsstufe eines Mitarbeiters auszuwerten. Das Gehalt muss *zwischen* dem Mindest- und Höchstwert für die einzelnen Gehaltsbereiche liegen.

Beachten Sie, dass jeder Mitarbeiter bei der Ausführung dieser Abfrage genau einmal angezeigt wird. Kein Mitarbeiter wird in der Liste mehrfach angezeigt. Dafür gibt es zwei Gründe:

- Keine der Zeilen in der Tabelle `job_grades` enthält Gehaltsstufen, die sich überschneiden. Das bedeutet, der Gehaltswert für einen Mitarbeiter kann nur zwischen dem Mindest- und Höchstwert aus einer der Zeilen der Tabelle `job_grades` liegen.
- Alle Mitarbeitergehälter liegen innerhalb der durch die Tabelle job_grades festgelegten Grenzwerte. Kein Mitarbeiter verdient also weniger als das niedrigste Gehalt aus der Spalte `LOWEST_SAL` oder mehr als das höchste Gehalt aus der Spalte `HIGHEST_SAL`.

Hinweis: Es können auch andere Bedingungen verwendet werden, z. B. `<=` und `>=`. `BETWEEN` ist jedoch die einfachste Möglichkeit. Geben Sie zuerst den Mindest- und dann den Höchstwert an, wenn Sie den Operator `BETWEEN` verwenden. Der Oracle-Server übersetzt den Operator `BETWEEN` in ein `AND`-Bedingungspaar. Daher bietet `BETWEEN` keine Performancevorteile und dient nur der logischen Einfachheit.

Die Tabellenaliasnamen wurden im Beispiel auf der Folie aus Performancegründen verwendet, nicht wegen möglicher Mehrdeutigkeit.

Datensätze mit USING-Klauseln abrufen

- Um aus mehreren übereinstimmenden Spalten nur eine Spalte abzurufen, verwenden Sie die Klausel USING.
- Diese Klausel kann nicht mit einem NATURAL-Join angegeben werden.
- Kennzeichnen Sie den Spaltennamen nicht mit einem Tabellennamen oder -alias.
- Beispiel:

```
SELECT country_id, country_name, location_id, city
FROM   countries JOIN locations
USING (country_id) ;
```

	COUNTRY_ID	COUNTRY_NAME	LOCATION_ID	CITY
1	US	United States of America	1400	Southlake
2	US	United States of America	1500	South San Francisco
3	US	United States of America	1700	Seattle
4	CA	Canada	1800	Toronto
5	UK	United Kingdom	2500	Oxford

Im Beispiel auf der Folie werden die COUNTRY_ID-Spalten aus den Tabellen COUNTRIES und LOCATIONS verknüpft. Damit wird die LOCATION_ID für den Standort angezeigt, an dem ein Mitarbeiter arbeitet.

Datensätze mit ON-Klauseln abrufen

- Die Join-Bedingung für Natural Joins ist im Prinzip ein Equi Join aller Spalten mit gleichem Namen.
- Mit der Klausel ON können Sie beliebige Bedingungen oder zu verknüpfende Spalten angeben.
- Die Klausel ON macht den Code verständlicher.

```
SELECT e.employee_id, e.last_name, j.department_id,
FROM   employees e JOIN job_history j
ON     (e.employee_id = j.employee_id);
```

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	101	Kochhar	110
2	101	Kochhar	110
3	102	De Haan	60
4	176	Taylor	80
5	176	Taylor	80
6	200	Whalen	90
7	200	Whalen	90
8	201	Hartstein	20

ORACLE

Mit der Klausel ON können Sie Join-Bedingungen getrennt von Such- oder Filterkriterien in der Klausel WHERE angeben.

In diesem Beispiel werden die EMPLOYEE_ID-Spalten aus den Tabellen EMPLOYEES und JOB_HISTORY mit der Klausel ON verknüpft. Immer wenn eine Personalnummer in der Tabelle EMPLOYEES einer Personalnummer in der Tabelle JOB_HISTORY entspricht, wird die Zeile zurückgegeben. Der Tabellenalias ist erforderlich, um die übereinstimmenden Spaltennamen zu kennzeichnen.

Mit der Klausel ON können Sie auch Spalten mit unterschiedlichen Namen verknüpfen. Die Klammern um die verknüpften Spalten im Beispiel auf der Folie (e.employee_id = j.employee_id) sind optional. Entsprechend wäre auch ON e.employee_id = j.employee_id korrekt.

Hinweis: Wenn Sie die Abfrage mit dem Symbol **Run Statement** ausführen, fügt SQL Developer zur Unterscheidung der beiden employee_ids das Suffix "_1" an.

Left Outer Joins

- Ein Join zwischen zwei Tabellen, der alle übereinstimmenden Zeilen sowie die nicht übereinstimmenden Zeilen aus der linken Tabelle zurückgibt, wird als `LEFT OUTER JOIN` bezeichnet.
- Beispiel:

```
SELECT c.country_id, c.country_name, l.location_id, l.city
FROM   countries c LEFT OUTER JOIN locations l
ON    (c.country_id = l.country_id) ;
```

	COUNTRY_ID	COUNTRY_NAME	LOCATION_ID	CITY
1	CA	Canada	1800	Toronto
2	DE	Germany	(null)	(null)
3	UK	United Kingdom	2500	Oxford
4	US	United States of America	1400	Southlake
5	US	United States of America	1500	South San Francisco
6	US	United States of America	1700	Seattle

ORACLE

Diese Abfrage ruft alle Zeilen aus der Tabelle `COUNTRIES` (linke Tabelle) ab, auch wenn keine Übereinstimmung in der Tabelle `LOCATIONS` gegeben ist.

Right Outer Joins

- Ein Join zwischen zwei Tabellen, der alle übereinstimmenden Zeilen sowie die nicht übereinstimmenden Zeilen aus der rechten Tabelle zurückgibt, wird als `RIGHT OUTER JOIN` bezeichnet.
- Beispiel:

```
SELECT e.last_name, d.department_id, d.department_name
FROM   employees e RIGHT OUTER JOIN departments d
ON     (e.department_id = d.department_id) ;
```

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Hartstein	20	Marketing
3	Fay	20	Marketing
4	Davies	50	Shipping

...

18	Higgins	110	Accounting
19	Gietz	110	Accounting
20	(null)	190	Contracting

Diese Abfrage ruft alle Zeilen aus der Tabelle `DEPARTMENTS` (rechte Tabelle) ab, auch wenn keine Übereinstimmung in der Tabelle `EMPLOYEES` gefunden wird.

Full Outer Joins

- Ein Join zwischen zwei Tabellen, der neben allen übereinstimmenden Zeilen auch die nicht übereinstimmenden Zeilen aus beiden Tabellen zurückgibt, wird als `FULL OUTER JOIN` bezeichnet.
- Beispiel:

```
SELECT e.last_name, d.department_id, d.manager_id,
 d.department_name
FROM   employees e FULL OUTER JOIN departments d
ON    (e.manager_id = d.manager_id) ;
```

	LAST_NAME	DEPARTMENT_ID	MANAGER_ID	DEPARTMENT_NAME
1	King	(null)	(null)	(null)
2	Kochhar	90	100	Executive
3	De Haan	90	100	Executive
4	Hunold	(null)	(null)	(null)

...

19	Higgins	(null)	(null)	(null)
20	Gietz	110	205	Accounting
21	(null)	190	(null)	Contracting
22	(null)	10	200	Administration

ORACLE

Diese Abfrage ruft alle Zeilen aus der Tabelle `EMPLOYEES` ab, auch wenn in der Tabelle `DEPARTMENTS` keine Übereinstimmung gefunden wird. Außerdem werden alle Zeilen aus der Tabelle `DEPARTMENTS` abgerufen, unabhängig davon, ob Übereinstimmungen in der Tabelle `EMPLOYEES` vorhanden sind.

Self Joins – Beispiel

```
SELECT worker.last_name || ' works for '
       || manager.last_name
FROM   employees worker JOIN employees manager
ON worker.manager_id = manager.employee_id
ORDER BY worker.last_name;
```

	WORKER.LAST_NAME||'WORKSFOR'||MANAGER.LAST_NAME
1	Abel works for Zlotkey
2	Davies works for Mourgos
3	De Haan works for King
4	Ernst works for Hunold
5	Fay works for Hartstein
6	Gietz works for Higgins
7	Grant works for Zlotkey
8	Hartstein works for King
9	Higgins works for Kochhar

...

In manchen Fällen müssen Sie eine Tabelle mit sich selbst verknüpfen. Um für jeden Mitarbeiter den Namen des zuständigen Managers zu ermitteln, verknüpfen Sie die Tabelle `EMPLOYEES` mit sich selbst und führen damit einen Self Join durch. Im Beispiel auf der Folie wird die Tabelle `EMPLOYEES` mit sich selbst verknüpft. Um in der Klausel `FROM` zwei Tabellen zu simulieren, werden für dieselbe Tabelle (`EMPLOYEES`) zwei Aliasnamen verwendet: worker und manager.

In diesem Beispiel enthält die Klausel `WHERE` den Join, der wie folgt interpretiert wird: "wobei die Managernummer eines Mitarbeiters der Personalnummer des Managers entspricht".

Cross Joins

- Ein `CROSS JOIN` ist ein `JOIN`-Vorgang, der das kartesische Produkt aus zwei Tabellen zurückgibt.
- Beispiel:

```
SELECT department_name, city
FROM department CROSS JOIN location;
```

	DEPARTMENT_NAME	CITY
1	Administration	Oxford
2	Administration	Seattle
3	Administration	South San Francisco
4	Administration	Southlake
5	Administration	Toronto
6	Marketing	Oxford
7	Marketing	Seattle
8	Marketing	South San Francisco
9	Marketing	Southlake
10	Marketing	Toronto

...

ORACLE

Die Syntax für `CROSS JOIN` gibt das Kreuzprodukt an, das auch als kartesisches Produkt bezeichnet wird. Ein Cross Join gibt das Kreuzprodukt aus zwei Relationen zurück und entspricht im Prinzip einer durch Kommas getrennten Oracle Database-Notation.

Zwischen den beiden Tabellen im `CROSS JOIN` geben Sie keine `WHERE`-Bedingungen an.

Zusammenfassung

In diesem Anhang haben Sie Folgendes gelernt:

- Mit `SELECT`-Anweisungen Zeilen aus einzelnen oder mehreren Tabellen abrufen
- Mit DDL-Anweisungen die Struktur von Objekten ändern
- Mit DML-Anweisungen Daten in vorhandenen Schemaobjekten abfragen oder bearbeiten
- Mit Anweisungen zur Transaktionskontrolle die über DML-Anweisungen vorgenommenen Änderungen verwalten
- Mithilfe von Joins Daten aus mehreren Tabellen anzeigen

ORACLE

Zu den zahlreichen häufig verwendeten SQL-Anweisungen und -Befehlen gehören DDL-, DML- und Transaktionskontrollanweisungen sowie Joins.

Berichte durch Gruppierung zusammenhängender Daten generieren

Ziele

Nach Ablauf dieses Anhangs haben Sie folgende Ziele erreicht:

- Mit dem Operator `ROLLUP` Zwischensummenwerte erstellen
- Mit dem Operator `CUBE` Kreuztabellenwerte erstellen
- Mit der Funktion `GROUPING` von `ROLLUP` oder `CUBE` erstellte Zeilenwerte identifizieren
- Mit `GROUPING SETS` eine einzige Ergebnismenge erstellen

ORACLE

In diesem Anhang lernen Sie Folgendes:

- Daten gruppieren, um mithilfe des Operators `ROLLUP` Zwischensummenwerte zu erhalten
- Daten gruppieren, um mithilfe des Operators `CUBE` Kreuztabellenwerte zu erhalten
- `GROUPING`-Funktion verwenden, um die Aggregationsebene in der vom `ROLLUP`- oder `CUBE`-Operator erstellten Ergebnismenge zu identifizieren
- Mit `GROUPING SETS` die selbe einzelne Ergebnismenge wie mit `UNION ALL` erzeugen

Gruppenfunktionen – Wiederholung

- Gruppenfunktionen werden auf Zeilengruppen angewendet und geben ein Ergebnis pro Gruppe zurück.

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

- Beispiel:

```
SELECT AVG(salary), STDDEV(salary),
       COUNT(commission_pct),MAX(hire_date)
FROM   employees
WHERE  job_id LIKE 'SA%';
```

ORACLE

Teilen Sie mit der Klausel GROUP BY die Zeilen einer Tabelle in Gruppen auf. Anschließend können Sie mit den Gruppenfunktionen Aggregatinformationen für jede Gruppe zurückgeben. Gruppenfunktionen können in SELECT-Listen und in ORDER BY- und HAVING-Klauseln verwendet werden. Der Oracle-Server wendet die Gruppenfunktionen auf jede Zeilengruppe an und gibt für jede Gruppe eine einzelne Ergebniszeile zurück.

Typen von Gruppenfunktionen: Jede der Gruppenfunktionen AVG, SUM, MAX, MIN, COUNT, STDDEV und VARIANCE akzeptiert ein Argument. Die Funktionen AVG, SUM, STDDEV und VARIANCE bearbeiten nur numerische Werte. MAX und MIN können auf numerische, Zeichen- und Datumsdatenwerte angewendet werden. COUNT gibt die Anzahl der nicht leeren Zeilen für den angegebenen Ausdruck zurück. Im Beispiel auf der Folie werden das Durchschnittsgehalt, die Standardabweichung des Gehalts, die Anzahl der provisionsberechtigten Mitarbeiter und das späteste Einstellungsdatum für Mitarbeiter berechnet, deren JOB_ID mit SA beginnt.

Gruppenfunktionen – Richtlinien

- Mögliche Datentypen für die Argumente sind CHAR, VARCHAR2, NUMBER und DATE.
- Alle Gruppenfunktionen außer COUNT(*) ignorieren Nullwerte. Um Nullwerte durch einen Wert zu ersetzen, verwenden Sie die Funktion NVL. COUNT gibt entweder eine Zahl oder null (0) zurück.
- Wenn Sie eine GROUP BY-Klausel verwenden, sortiert der Oracle-Server die Ergebnismenge implizit in aufsteigender Reihenfolge nach den angegebenen Gruppenspalten. Um diese Standardsortierung außer Kraft zu setzen, können Sie DESC in der Klausel ORDER BY verwenden.

Klausel GROUP BY – Wiederholung

- Syntax:

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

- Beispiel:

```
SELECT   department_id, job_id, SUM(salary),
         COUNT(employee_id)
FROM     employees
GROUP BY department_id, job_id ;
```

ORACLE

Der Oracle-Server wertet das auf der Folie dargestellte Beispiel wie folgt aus:

- Die Klausel SELECT gibt an, dass folgende Spalten abgerufen werden sollen:
  - Abteilungs-ID und Tätigkeits-ID aus der Tabelle EMPLOYEES
  - die Summe aller Gehälter und die Anzahl der Angestellten in jeder Gruppe, die Sie in der Klausel GROUP BY angegeben haben
- Die Klausel GROUP BY gibt an, wie die Zeilen in der Tabelle gruppiert werden sollen. Das Gesamtgehalt und die Mitarbeiterzahl werden für Tätigkeits-IDs aller Abteilungen berechnet. Die Zeilen werden nach Abteilungs-ID und dann innerhalb jeder Abteilung nach Tätigkeit gruppiert.

Klausel HAVING – Wiederholung

- Mit der Klausel HAVING angeben, welche Gruppen angezeigt werden sollen
- Gruppen auf der Basis einer einschränkenden Bedingung weiter eingrenzen

```
SELECT        [column,] group_function(column)...
FROM          table
[WHERE        condition]
[GROUP BY     group_by_expression]
[HAVING       having_expression]
[ORDER BY     column];
```


HAVING-Klausel

Gruppen werden gebildet und Gruppenfunktionen berechnet, bevor die Klausel HAVING auf die Gruppen angewendet wird. Die Klausel HAVING kann vor der Klausel GROUP BY stehen. Es wird jedoch empfohlen, die Klausel GROUP BY zuerst anzugeben, da die Klausel GROUP BY logischer als die Klausel HAVING ist.

Der Oracle-Server führt die folgenden Schritte aus, wenn Sie die Klausel HAVING verwenden:

1. Er gruppiert Zeilen.
2. Er wendet die Gruppenfunktionen auf die Gruppen an und zeigt die Gruppen an, die die Kriterien in der Klausel HAVING erfüllen.

GROUP BY mit den Operatoren ROLLUP und CUBE

- ROLLUP oder CUBE mit GROUP BY verwenden, um Zeilen mit Superaggregaten durch das Erstellen einer Obermenge von Spalten zu erzeugen
- Bei der Gruppierung mit ROLLUP wird eine Ergebnismenge erstellt, die die normal gruppierten Zeilen und die Zwischensummenwerte enthält.
- Bei der Gruppierung mit CUBE wird eine Ergebnismenge erstellt, die die Zeilen aus dem ROLLUP-Vorgang und Kreuztabellenzeilen enthält.

Sie geben die ROLLUP- und CUBE-Operatoren in der Klausel GROUP BY einer Abfrage an. Bei der Gruppierung mit ROLLUP wird eine Ergebnismenge erstellt, die die normal gruppierten Zeilen und Zwischensummenzeilen enthält. Der Operator ROLLUP berechnet auch eine Gesamtsumme. Bei Verwendung des Vorgangs CUBE in der Klausel GROUP BY werden die ausgewählten Zeilen auf der Basis der Werte aller möglichen Kombinationen der Ausdrücke in der Spezifikation gruppiert, und für jede Gruppe wird eine Zeile mit Aggregatinformationen zurückgegeben. Mit dem Operator CUBE können Sie Kreuztabellenzeilen erstellen.

Hinweis: Stellen Sie bei der Verwendung von ROLLUP und CUBE sicher, dass zwischen den nach der Klausel GROUP BY angegebenen Spalten sinnvolle und reale Beziehungen bestehen, da die Operatoren andernfalls irrelevante Informationen zurückgeben.

Operator ROLLUP

- ROLLUP ist eine Erweiterung der Klausel GROUP BY.
- Mit dem Operator ROLLUP können Sie kumulative Aggregate wie Zwischensummen erstellen.

```
SELECT      [column, ]group_function(column). . .
FROM        table
[WHERE      condition]
[GROUP BY   ROLLUP group_by_expression]
[HAVING     having_expression];
[ORDER BY   column];
```


Der Operator ROLLUP erzeugt Aggregate und Superaggregate für Ausdrücke in einer GROUP BY-Anweisung. Der Operator ROLLUP kann bei der Berichterstellung eingesetzt werden, um Statistik- und Aggregatinformationen aus Ergebnismengen zu extrahieren. Die kumulativen Aggregate können in Berichten, Diagrammen und graphischen Darstellungen verwendet werden.

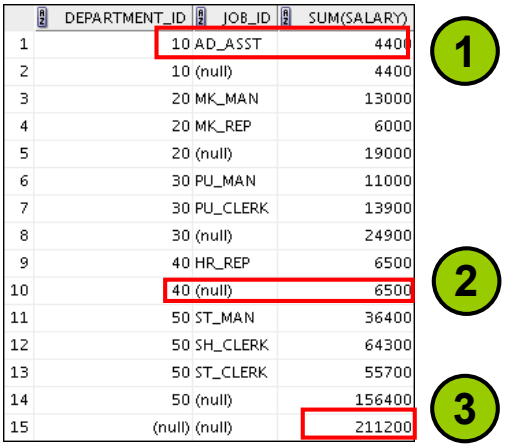
Der Operator ROLLUP erstellt Gruppierungen, indem er die Liste der in der Klausel GROUP BY angegebenen Spalten in einer Richtung (von rechts nach links) bearbeitet. Anschließend wendet er die Aggregatfunktion auf diese Gruppierungen an.

Hinweis:

- Um Zwischensummen in *n* Dimensionen (d. h. *n* Spalten in der Klausel GROUP BY) ohne den Operator ROLLUP zu erstellen, müssen *n*+1 SELECT-Anweisungen mit UNION ALL verknüpft werden. Dies führt zu einer ineffizienten Abfrageausführung, da bei jeder SELECT-Anweisung ein Tabellenzugriff erfolgt. Der Operator ROLLUP ruft seine Ergebnisse mit nur einem Tabellenzugriff ab. Die Verwendung des Operators ROLLUP ist hilfreich, wenn viele Spalten an der Erstellung der Zwischensummen beteiligt sind.
- Zwischensummen und Summen werden mit ROLLUP erstellt. CUBE erstellt ebenfalls Summen, fasst jedoch Daten in allen möglichen Richtungen zusammen und generiert damit Kreuztabellenwerte.

Operator ROLLUP – Beispiel

```
SELECT   department_id, job_id, SUM(salary)
FROM     employees
WHERE    department_id < 60
GROUP BY ROLLUP(department_id, job_id);
```



	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	10	AD_ASST	4400
2	10	(null)	4400
3	20	MK_MAN	13000
4	20	MK_REP	6000
5	20	(null)	19000
6	30	PU_MAN	11000
7	30	PU_CLERK	13900
8	30	(null)	24900
9	40	HR_REP	6500
10	40	(null)	6500
11	50	ST_MAN	36400
12	50	SH_CLERK	64300
13	50	ST_CLERK	55700
14	50	(null)	156400
15	(null)	(null)	211200

Im Beispiel auf der Folie gilt:

- Die Klausel `GROUP BY` zeigt für Abteilungen, deren Abteilungs-ID kleiner als 60 ist, das Gesamtgehalt für jede Tätigkeits-ID an.
- Der Operator `ROLLUP` zeigt Folgendes an:
  - Gesamtgehalt für jede Abteilung, deren Abteilungs-ID kleiner als 60 ist
  - Gesamtgehalt für die Abteilungen, deren Abteilungs-ID kleiner als 60 ist, wobei die Tätigkeits-IDs nicht berücksichtigt werden

In diesem Beispiel kennzeichnet 1 eine Gruppe, die nach `DEPARTMENT_ID` und `JOB_ID` zusammengefasst wurde. 2 kennzeichnet eine Gruppe, die nur nach `DEPARTMENT_ID` zusammengefasst wurde, und 3 kennzeichnet die Gesamtsumme.
Der Operator `ROLLUP` erstellt entsprechend der in der Klausel `GROUP BY` angegebenen Gruppierungsliste Zwischensummen. Dabei beginnt er bei der detailliertesten Ebene und fasst die Zwischensummen schließlich zu einer Gesamtsumme zusammen. Zuerst werden die Standard-Aggregatwerte für die in der Klausel `GROUP BY` angegebenen Gruppen berechnet (im Beispiel ist dies die Summe der Gehälter, die innerhalb der einzelnen Abteilungen nach Job gruppiert sind). Dann werden auf immer höheren Ebenen Zwischensummen gebildet, wobei die Liste der Gruppierungsspalten von rechts nach links bearbeitet wird. (Im Beispiel wird die Summe der Gehälter für jede Abteilung und dann die Summe der Gehälter für alle Abteilungen berechnet.)

- Bei der Angabe von *n* Ausdrücken im Operator `ROLLUP` der Klausel `GROUP BY` führt der Vorgang zu *n* + 1 Gruppierungen (in diesem Fall 2 + 1 = 3).
- Zeilen, die auf den Werten der ersten *n* Ausdrücke basieren, werden als Zeilen oder normale Zeilen bezeichnet. Die übrigen Zeilen werden als Zeilen mit Superaggregaten (superaggregate rows) bezeichnet.

Operator CUBE

- CUBE ist eine Erweiterung der Klausel GROUP BY.
- Mit dem Operator CUBE können Sie in einer einzelnen SELECT-Anweisung Kreuztabellenwerte (cross-tabulation values) erstellen.

```
SELECT        [column, ] group_function(column)...
FROM          table
[WHERE        condition]
[GROUP BY     CUBE group_by_expression]
[HAVING       having_expression]
[ORDER BY     column];
```


Der Operator CUBE ist ein zusätzlicher Schalter in der Klausel GROUP BY einer SELECT-Anweisung. CUBE kann auf alle Aggregatfunktionen angewendet werden, einschließlich AVG, SUM, MAX, MIN und COUNT. Er dient zur Erstellung von Ergebnismengen, die in der Regel für Kreuztabellenberichte verwendet werden. ROLLUP erstellt nur einen Bruchteil der möglichen Zwischensummenkombinationen. CUBE generiert dagegen Zwischensummen für alle möglichen Kombinationen der in der Klausel GROUP BY angegebenen Gruppierungen sowie eine Gesamtsumme.

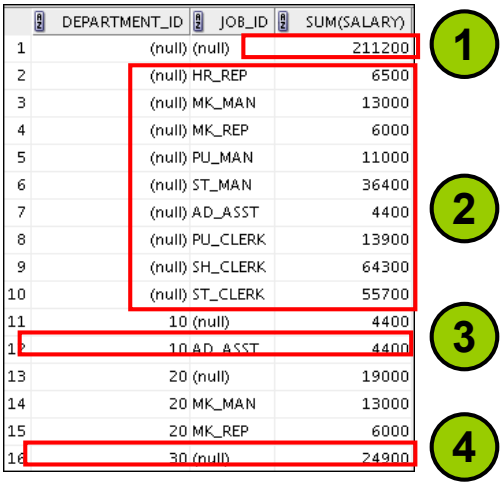
Der Operator CUBE wird mit einer Aggregatfunktion verwendet, um zusätzliche Zeilen in einer Ergebnismenge zu generieren. In der Klausel GROUP BY enthaltene Spalten werden intern verknüpft, um eine Obermenge von Gruppen zu erzeugen. Auf diese Gruppen wird die in der SELECT-Liste angegebene Aggregatfunktion angewendet, um Summenwerte für die zusätzlichen Zeilen mit Superaggregaten zu erstellen. Die Anzahl der zusätzlichen Gruppen in der Ergebnismenge wird durch die Anzahl der Spalten in der Klausel GROUP BY bestimmt.

Tatsächlich wird zur Erstellung von Superaggregaten jede mögliche Kombination der Spalten oder Ausdrücke in der Klausel GROUP BY verwendet. Wenn die GROUP BY-Klausel *n* Spalten oder Ausdrücke enthält, gibt es 2^n mögliche Superaggregat-Kombinationen. Mathematisch gesehen bilden diese Kombinationen einen *n*-dimensionalen Würfel (daher auch der Operatorname CUBE = Englisch für Würfel).

Mit Anwendungs- oder Programmiertools können diese Superaggregat-Werte in Diagramme und Grafiken geladen werden, die Ergebnisse und Beziehungen effektiv visuell darstellen.

Operator CUBE – Beispiel

```
SELECT   department_id, job_id, SUM(salary)
FROM     employees
WHERE    department_id < 60
GROUP BY CUBE (department_id, job_id) ;
```



	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	(null)	(null)	211200
2	(null)	HR_REP	6500
3	(null)	MK_MAN	13000
4	(null)	MK_REP	6000
5	(null)	PU_MAN	11000
6	(null)	ST_MAN	36400
7	(null)	AD_ASST	4400
8	(null)	PU_CLERK	13900
9	(null)	SH_CLERK	64300
10	(null)	ST_CLERK	55700
11	10	(null)	4400
12	10	AD_ASST	4400
13	20	(null)	19000
14	20	MK_MAN	13000
15	20	MK_REP	6000
16	30	(null)	24900

ORACLE

Die Ausgabe der Anweisung SELECT im Beispiel kann folgendermaßen interpretiert werden:

- Gesamtgehalt für jeden Job in einer Abteilung (für Abteilungen, deren Abteilungsnummer kleiner als 60 ist)
- Gesamtgehalt für jede Abteilung, deren Abteilungs-ID kleiner als 60 ist
- Gesamtgehalt für jede Tätigkeit, unabhängig von der Abteilung
- Gesamtgehalt für die Abteilungen, deren Abteilungs-ID kleiner als 60 ist, wobei die Tätigkeitsbezeichnungen nicht berücksichtigt werden

In diesem Beispiel kennzeichnet 1 die Gesamtsumme, 2 die nur nach JOB_ID zusammengefassten Zeilen, 3 einige der nach DEPARTMENT_ID und JOB_ID zusammengefassten Zeilen und 4 einige der nur nach DEPARTMENT_ID zusammengefassten Zeilen.

Der Operator CUBE hat zudem den Vorgang ROLLUP durchgeführt, um die Zwischensummen und das Gesamtgehalt für die Abteilungen mit einer Abteilungs-ID kleiner als 60 anzuzeigen, wobei die Tätigkeitsbezeichnungen nicht berücksichtigt wurden. Darüber hinaus zeigt der Operator CUBE das Gesamtgehalt für jeden Job unabhängig von der Abteilung an.

Hinweis: Wie beim Operator ROLLUP müssen zum Erstellen von Zwischensummen in *n* Dimensionen (d. h. *n* Spalten in der Klausel GROUP BY) ohne einen CUBE-Operator 2^n SELECT-Anweisungen mit UNION ALL verknüpft werden. Folglich müssen für einen Bericht mit drei Dimensionen $2^3 = 8$ SELECT-Anweisungen mit UNION ALL verknüpft werden.

Funktion GROUPING

Die Funktion GROUPING:

- wird mit dem Operator CUBE oder ROLLUP verwendet
- dient zur Ermittlung der Gruppen, die die Zwischensumme in einer Zeile bilden
- unterscheidet gespeicherte NULL-Werte von NULL-Werten, die mit ROLLUP oder CUBE erstellt wurden
- Gibt 0 oder 1 zurück

```
SELECT    [column,] group_function(column) .. ,
          GROUPING(expr)
FROM      table
[WHERE    condition]
[GROUP BY [ROLLUP][CUBE] group_by_expression]
[HAVING   having_expression]
[ORDER BY column];
```


Sie können die Funktion GROUPING mit dem Operator CUBE oder ROLLUP verwenden, um festzustellen, wie ein Summenwert ermittelt wurde.

Die Funktion GROUPING verwendet eine einzelne Spalte als Argument. Das Argument *expr* in der Funktion GROUPING muss mit einem der Ausdrücke in der Klausel GROUP BY übereinstimmen. Die Funktion gibt den Wert 0 oder 1 zurück.

Anhand der von GROUPING zurückgegebenen Werte können Sie:

- die Aggregationsebene einer bestimmten Zwischensumme bestimmen (das heißt die Gruppen, auf denen die Zwischensumme basiert)
- NULL-Werte in der Ausdrucksspalte einer Zeile der Ergebnismenge einer der beiden folgenden Kategorien zuordnen:
  - NULL-Wert aus der Basistabelle (gespeicherter NULL-Wert)
  - NULL-Wert, der von ROLLUP oder CUBE erstellt wurde (als Ergebnis der Bearbeitung dieses Ausdrucks durch eine Gruppenfunktion)

Ein von der Funktion GROUPING auf Basis eines Ausdrucks zurückgegebener Wert 0 weist auf eine der folgenden Situationen hin:

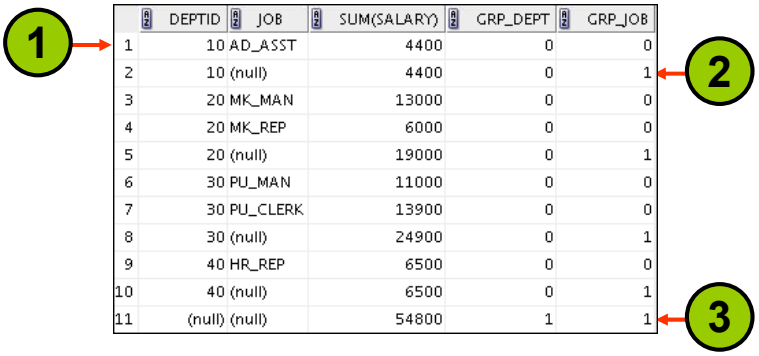
- Der Ausdruck wurde zum Berechnen des Aggregatwertes verwendet.
- Der NULL-Wert in der Ausdrucksspalte ist ein gespeicherter NULL-Wert.

Ein von der Funktion GROUPING auf Basis eines Ausdrucks zurückgegebener Wert 1 weist auf eine der folgenden Situationen hin:

- Der Ausdruck wurde nicht zur Berechnung des Aggregatwertes verwendet.
- Der NULL-Wert in der Ausdrucksspalte wurde von ROLLUP oder CUBE als Ergebnis einer Gruppierung erstellt.

Funktion GROUPING – Beispiel

```
SELECT   department_id DEPTID, job_id JOB,
         SUM(salary),
         GROUPING(department_id) GRP_DEPT,
         GROUPING(job_id) GRP_JOB
FROM     employees
WHERE    department_id < 50
GROUP BY ROLLUP(department_id, job_id);
```



	DEPTID	JOB	SUM(SALARY)	GRP_DEPT	GRP_JOB
1	10	AD_ASST	4400	0	0
2	10	(null)	4400	0	1
3	20	MK_MAN	13000	0	0
4	20	MK_REP	6000	0	0
5	20	(null)	19000	0	1
6	30	PU_MAN	11000	0	0
7	30	PU_CLERK	13900	0	0
8	30	(null)	24900	0	1
9	40	HR_REP	6500	0	0
10	40	(null)	6500	0	1
11	(null)	(null)	54800	1	1

Im Beispiel auf der Folie enthält die erste Zeile den Summenwert 4400 (mit 1 gekennzeichnet). Dieser Summenwert ist das Gesamtgehalt für die Tätigkeits-ID von `AD_ASST` in Abteilung 10. Für die Berechnung dieses Summenwertes wurden die beiden Spalten `DEPARTMENT_ID` und `JOB_ID` berücksichtigt. Daher wird für die beiden Ausdrücke `GROUPING(department_id)` und `GROUPING(job_id)` der Wert `0` zurückgegeben.

Betrachten Sie nun den Summenwert 4400 in der zweiten Zeile (mit 2 gekennzeichnet). Dieser Wert ist das Gesamtgehalt für Abteilung 10, das unter Berücksichtigung der Spalte `DEPARTMENT_ID` berechnet wurde. Daher wurde von `GROUPING(department_id)` der Wert `0` zurückgegeben. Da die Spalte `JOB_ID` bei der Berechnung dieses Wertes nicht berücksichtigt wurde, hat `GROUPING(job_id)` den Wert `1` zurückgegeben. In der fünften Zeile finden Sie ähnliche Ausgabewerte.

Die letzte Zeile enthält den Summenwert 54800 (mit 3 gekennzeichnet). Dieser Wert ist das Gesamtgehalt für alle Tätigkeitsbezeichnungen und die Abteilungen, deren Abteilungs-ID kleiner als 50 ist. Bei der Berechnung dieses Summenwertes wurden die Spalten `DEPARTMENT_ID` und `JOB_ID` nicht berücksichtigt. Daher wird für die beiden Ausdrücke `GROUPING(department_id)` und `GROUPING(job_id)` der Wert `1` zurückgegeben.

GROUPING SETS

- Mit der Syntax GROUPING SETS mehrere Gruppierungen in derselben Abfrage definieren
- Alle in der Klausel GROUPING SETS angegebenen Gruppierungen werden berechnet und die Ergebnisse der einzelnen Gruppierungen mit UNION ALL kombiniert
- Gruppierungssätze sind effizient:
  - Die Basistabelle muss nur einmal durchsucht werden.
  - Es müssen keine komplexen UNION-Anweisungen geschrieben werden.
  - Je mehr Elemente die GROUPING SETS enthalten, desto höher ist die Performance.

ORACLE

GROUPING SETS ist eine Erweiterung der Klausel GROUP BY, mit der Sie mehrere Datengruppierungen angeben können. Dies ermöglicht die effiziente Aggregation und erleichtert damit die Analyse von Daten über mehrere Dimensionen.

Sie können jetzt mit einer SELECT-Anweisung mit GROUPING SETS verschiedene Gruppierungen angeben (die ROLLUP- oder CUBE-Operatoren enthalten können) und müssen nicht mehr mehrere SELECT-Anweisungen mit UNION ALL-Operatoren verknüpfen. Beispiel:

```
SELECT department_id, job_id, manager_id, AVG(salary)
 FROM employees
 GROUP BY
 GROUPING SETS
 ((department_id, job_id, manager_id),
 (department_id, manager_id),(job_id, manager_id));
```

Diese Anweisung berechnet Aggregate über drei Gruppierungen:

```
(department_id, job_id, manager_id), (department_id,
  manager_id)and (job_id, manager_id)
```

Ohne dieses Feature sind mehrere durch UNION ALL verknüpfte Abfragen erforderlich, um die Ausgabe der oben stehenden SELECT-Anweisung zu erhalten. Die Verwendung mehrerer Abfragen ist ineffizient, da die gleichen Daten mehrmals durchsucht werden müssen.

Vergleichen Sie das vorherige Beispiel mit der folgenden Alternative:

```
SELECT department_id, job_id, manager_id, AVG(salary)
FROM employees
GROUP BY CUBE(department_id, job_id, manager_id);
```

Diese Anweisung ermittelt alle 8 (2 x 2 x 2) Gruppierungen, obwohl nur die Gruppen `(department_id, job_id, manager_id)`, `(department_id, manager_id)` und `(job_id, manager_id)` für Sie relevant sind.

Eine weitere Alternative ist die folgende Anweisung:

```
SELECT department_id, job_id, manager_id, AVG(salary)
FROM employees
GROUP BY department_id, job_id, manager_id
UNION ALL
SELECT department_id, NULL, manager_id, AVG(salary)
FROM employees
GROUP BY department_id, manager_id
UNION ALL
SELECT NULL, job_id, manager_id, AVG(salary)
FROM employees
GROUP BY job_id, manager_id;
```

Da bei dieser Anweisung die Basistabelle dreimal durchsucht werden muss, ist diese Methode ineffizient.

`CUBE` und `ROLLUP` sind eine Art Gruppierungsgruppe mit sehr spezifischer Semantik und Ergebnissen. Die folgende Übersicht verdeutlicht dies:

`CUBE(a, b, c)` entspricht	`GROUPING SETS ((a, b, c), (a, b), (a, c), (b, c), (a), (b), (c), ())`
`ROLLUP(a, b,c)` entspricht	`GROUPING SETS ((a, b, c), (a, b),(a), ())`

GROUPING SETS – Beispiel

```
SELECT   department_id, job_id,
         manager_id,AVG(salary)
FROM     employees
GROUP BY GROUPING SETS
((department_id,job_id), (job_id,manager_id));
```

	DEPARTMENT_ID	JOB_ID	MANAGER_ID	AVG(SALARY)
1	(null)	SH_CLERK	122	3200
2	(null)	AC_MGR	101	12000
3	(null)	ST_MAN	100	7280
4	...	(null)	ST_CLERK	121	2675

1

	DEPARTMENT_ID	JOB_ID	MANAGER_ID	AVG(SALARY)
39	110	AC_MGR	(null)	12000
40	90	AD_PRES	(null)	24000
41	60	IT_PROG	(null)	5760
42	100	FI_MGR	(null)	12000

2

...

ORACLE

In der Abfrage auf der Folie werden Aggregate über zwei Gruppierungen berechnet. Die Tabelle ist in die folgenden Gruppen unterteilt:

- Abteilungs-ID, Tätigkeits-ID
- Tätigkeits-ID, Manager-ID

Für jede dieser Gruppen werden die Durchschnittsgehälter berechnet. Die Ergebnismenge zeigt das Durchschnittsgehalt für jede der zwei Gruppen an.

Die in der Ausgabe mit 1 gekennzeichnete Gruppe kann folgendermaßen interpretiert werden:

- Das Durchschnittsgehalt aller Mitarbeiter mit der Tätigkeits-ID `SH_CLERK` unter Manager 122 beträgt 3.200.
- Das Durchschnittsgehalt aller Mitarbeiter mit der Tätigkeits-ID `AC_MGR` unter Manager 101 beträgt 12.000 und so weiter.

Die in der Ausgabe mit 2 gekennzeichnete Gruppe wird folgendermaßen interpretiert:

- Das Durchschnittsgehalt aller Mitarbeiter mit der Tätigkeits-ID `AC_MGR` in Abteilung 110 beträgt 12.000.
- Das Durchschnittsgehalt aller Mitarbeiter mit der Tätigkeits-ID `AD_PRES` in Abteilung 90 beträgt 24.000 und so weiter.

Das Beispiel auf der Folie kann auch folgendermaßen erstellt werden:

```
SELECT department_id, job_id, NULL as manager_id,
       AVG(salary) as AVGSAL
FROM employees
GROUP BY department_id, job_id
UNION ALL
SELECT NULL, job_id, manager_id, avg(salary) as AVGSAL
FROM employees
GROUP BY job_id, manager_id;
```

Ohne Optimizer, der zur Generierung des Ausführungsplans Abfrageblöcke durchgeht, müsste bei der vorstehenden Abfrage die Basistabelle `EMPLOYEES` zweimal durchsucht werden. Diese Methode kann sehr ineffizient sein. Aus diesem Grund empfiehlt sich die Verwendung der Anweisung `GROUPING SETS`.

Zusammengesetzte Spalten

- Eine zusammengesetzte Spalte ist eine Gruppe von Spalten, die als Einheit behandelt werden.

  `ROLLUP (a, (b, c), d)`
- Spalten mithilfe von Klammern in der Klausel `GROUP BY` gruppieren, damit sie beim Ausführen von `ROLLUP` oder `CUBE` als Einheit behandelt werden
- Mit `ROLLUP` oder `CUBE` bedeuten zusammengesetzte Spalten, dass bei der Aggregation bestimmte Ebenen übersprungen werden.

Eine zusammengesetzte Spalte ist eine Gruppe von Spalten, die bei der Berechnung von Gruppierungen als Einheit behandelt werden. Sie geben die Spalten wie in der folgenden Anweisung in Klammern an: `ROLLUP (a, (b, c), d)`

In diesem Beispiel stellt `(b, c)` eine zusammengesetzte Spalte dar, die als Einheit behandelt wird. Die Verwendung von zusammengesetzten Spalten ist im Allgemeinen in `ROLLUP`-, `CUBE`- und `GROUPING SETS`-Anweisungen sinnvoll. Bei der Verwendung mit `CUBE` oder `ROLLUP` bedeuten zusammengesetzte Spalten beispielsweise, dass bei der Aggregation bestimmte Ebenen übersprungen werden.

`GROUP BY ROLLUP(a, (b, c) )` entspricht demnach:

```
GROUP BY a, b, c UNION ALL
GROUP BY a UNION ALL
GROUP BY ()
```

In dieser Anweisung wird `(b, c)` als Einheit behandelt, sodass kein `ROLLUP` über `(b, c)` durchgeführt wird. Dies ist vergleichbar mit der Verwendung eines Alias. Beispiel: `z` dient als Alias für `(b, c)`, sodass der Ausdruck `GROUP BY` auf Folgendes reduziert wird: `GROUP BY ROLLUP(a, z)`.

Hinweis: `GROUP BY( )` ist in der Regel eine `SELECT`-Anweisung mit `NULL`-Werten für die Spalten `a` und `b` und nur der Aggregatfunktion. Mit ihr werden im Allgemeinen die Gesamtsummen generiert.

```
SELECT   NULL, NULL, aggregate_col
FROM     <table_name>
GROUP BY ( );
```

Vergleichen Sie dies mit dem normalen ROLLUP wie bei:

```
GROUP BY ROLLUP(a, b, c)
```

Dies wäre:

```
GROUP BY a, b, c UNION ALL
GROUP BY a, b UNION ALL
GROUP BY a UNION ALL
GROUP BY ()
```

Analog dazu entspricht

```
GROUP BY CUBE((a, b), c)
```

den folgenden Anweisungen:

```
GROUP BY a, b, c UNION ALL
GROUP BY a, b UNION ALL
GROUP BY c UNION ALL
GROUP By ()
```

Die folgende Tabelle enthält die Spezifikation für GROUPING SETS und die äquivalente Spezifikation für GROUP BY.

GROUPING SETS-Anweisungen	Äquivalente GROUP BY-Anweisungen

`GROUP BY GROUPING SETS(a, b, c)`	`GROUP BY a UNION ALL`
	`GROUP BY b UNION ALL`
	`GROUP BY c`

`GROUP BY GROUPING SETS(a, b,(b, c))`
(Der Ausdruck GROUPING SETS enthält eine zusammengesetzte Spalte.)
	`GROUP BY a UNION ALL`
	`GROUP BY b UNION ALL`
	`GROUP BY b, c`

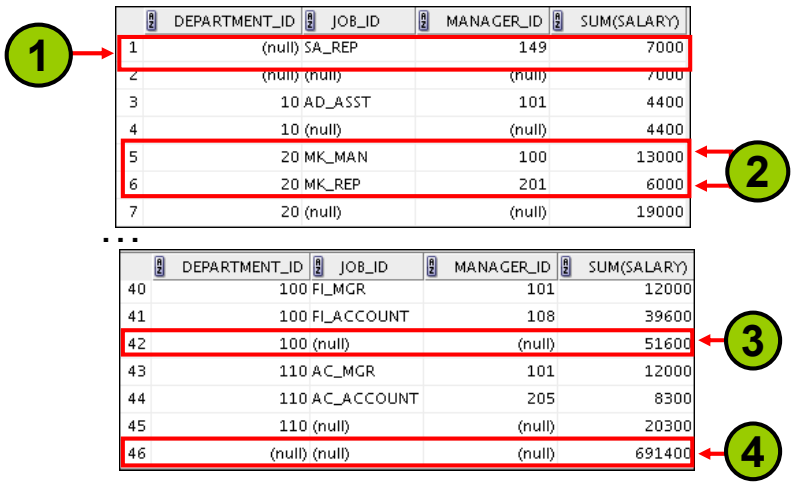
`GROUP BY GROUPING SETS((a, b, c))`	`GROUP BY a, b, c`

`GROUP BY GROUPING SETS(a, (b), ())`	`GROUP BY a UNION ALL`
	`GROUP BY b UNION ALL`
	`GROUP BY ()`

`GROUP BY GROUPING SETS`
`(a,ROLLUP(b, c))`
(Der Ausdruck GROUPING SETS enthält eine zusammengesetzte Spalte.)
	`GROUP BY a UNION ALL`
	`GROUP BY ROLLUP(b, c)`

Zusammengesetzte Spalten – Beispiel

```
SELECT   department_id, job_id, manager_id,
         SUM(salary)
FROM     employees
GROUP BY ROLLUP( department_id,(job_id, manager_id));
```



	DEPARTMENT_ID	JOB_ID	MANAGER_ID	SUM(SALARY)
1	(null)	SA_REP	149	7000
2	(null)	(null)	(null)	7000
3	10	AD_ASST	101	4400
4	10	(null)	(null)	4400
5	20	MK_MAN	100	13000
6	20	MK_REP	201	6000
7	20	(null)	(null)	19000

...

	DEPARTMENT_ID	JOB_ID	MANAGER_ID	SUM(SALARY)
40	100	FI_MGR	101	12000
41	100	FI_ACCOUNT	108	39600
42	100	(null)	(null)	51600
43	110	AC_MGR	101	12000
44	110	AC_ACCOUNT	205	8300
45	110	(null)	(null)	20300
46	(null)	(null)	(null)	691400

Betrachten Sie das folgende Beispiel:

```
SELECT department_id, job_id,manager_id, SUM(salary)
 FROM   employees
 GROUP BY ROLLUP( department_id,job_id, manager_id);
```

Bei dieser Abfrage ermittelt der Oracle-Server die folgenden Gruppierungen:

- `(job_id, manager_id)`
- `(department_id, job_id, manager_id)`
- `(department_id)`
- Gesamtsumme

Wenn für Sie nur bestimmte Gruppen relevant sind, können Sie die Berechnung nur mithilfe zusammengesetzter Spalten auf diese Gruppierungen beschränken. Mit zusammengesetzten Spalten können die Spalten `JOB_ID` und `MANAGER_ID` beim Rollup-Vorgang als eine Einheit behandelt werden. In Klammern gesetzte Spalten werden bei der Verarbeitung von `ROLLUP` und `CUBE` als Einheit behandelt. Das Beispiel auf der Folie verdeutlicht dies. Indem Sie die Spalten `JOB_ID` und `MANAGER_ID` in Klammern setzen, weisen Sie den Oracle-Server an, `JOB_ID` und `MANAGER_ID` als Einheit (als zusammengesetzte Spalte) zu behandeln.

Im Beispiel auf der Folie werden folgende Gruppierungen ermittelt:

- `(department_id, job_id, manager_id)`
- `(department_id)`
- `( )`

Das Beispiel auf der Folie zeigt folgende Ergebnisse an:

- Gesamtgehalt für jeden Job und Manager (mit 1 gekennzeichnet)
- Gesamtgehalt für jede Abteilung, jeden Job und jeden Manager (mit 2 gekennzeichnet)
- Gesamtgehalt für jede Abteilung (mit 3 gekennzeichnet)
- Gesamtsumme (mit 4 gekennzeichnet)

Das Beispiel auf der Folie kann auch folgendermaßen erstellt werden:

```
SELECT      department_id, job_id, manager_id, SUM(salary)
FROM        employees
GROUP       BY department_id,job_id, manager_id
UNION       ALL
SELECT      department_id, TO_CHAR(NULL),TO_NUMBER(NULL),    SUM(salary)
FROM        employees
GROUP BY    department_id
UNION ALL
SELECT  TO_NUMBER(NULL), TO_CHAR(NULL),TO_NUMBER(NULL), SUM(salary)
FROM    employees
GROUP BY ();
```

Ohne Optimizer, der zur Generierung des Ausführungsplans Abfrageblöcke durchgeht, müsste bei der vorstehenden Abfrage die Basistabelle `EMPLOYEES` dreimal durchsucht werden. Diese Methode kann sehr ineffizient sein. Aus diesem Grund empfiehlt sich die Verwendung zusammengesetzter Spalten.

Verkettete Gruppierungen

- Verkettete Gruppierungen bieten eine präzise Methode für die Erstellung sinnvoller Kombinationen von Gruppierungen.
- Um verkettete Gruppierungssätze anzugeben, trennen Sie mehrere Gruppierungssätze, `ROLLUP`- und `CUBE`-Vorgänge durch Kommas, sodass sie vom Server zu einer einzigen `GROUP BY`-Klausel kombiniert werden.
- Das Ergebnis ist ein Kreuzprodukt aus Gruppierungen der einzelnen `GROUPING SET`.

```
GROUP BY GROUPING SETS(a, b), GROUPING SETS(c, d)
```

ORACLE

Verkettete Gruppierungen bieten eine präzise Methode für die Erstellung sinnvoller Kombinationen von Gruppierungen. Die verketteten Gruppierungen werden angegeben, indem mehrere Gruppierungssätze, `CUBE`- und `ROLLUP`-Vorgänge aufgelistet und durch Kommas getrennt werden. Beispiel für verkettete Gruppierungssätze:

```
GROUP BY GROUPING SETS(a, b), GROUPING SETS(c, d)
```

Dieses SQL-Beispiel definiert die folgenden Gruppierungen:

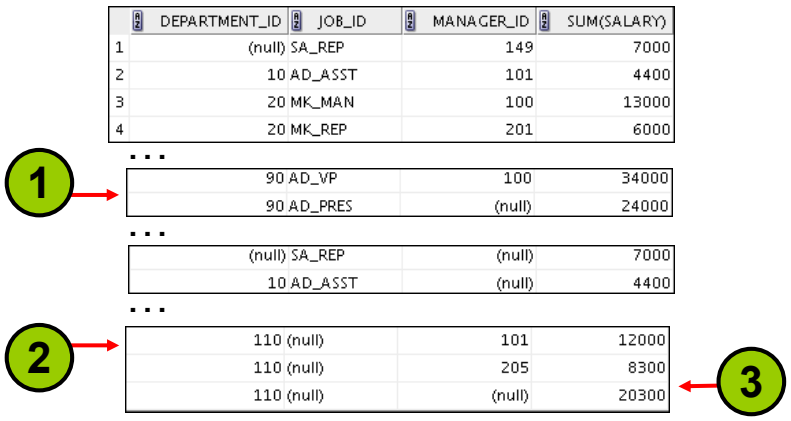
```
(a, c), (a, d), (b, c), (b, d)
```

Die Verkettung von Gruppierungssätzen ist aus folgenden Gründen sinnvoll:

- **Einfaches Erstellen von Abfragen:** Sie müssen nicht alle Gruppierungen manuell auflisten.
- **Verwendung durch Anwendungen:** Bei von OLAP-(Online Analytical Processing-) Anwendungen generiertem SQL-Code werden häufig Gruppierungssätze häufig verkettet, wobei jedes `GROUPING SET` die für eine Dimension benötigten Gruppierungen definiert.

Verkettete Gruppierungen – Beispiel

```
SELECT   department_id, job_id, manager_id,
         SUM(salary)
FROM     employees
GROUP BY department_id,
         ROLLUP(job_id),
         CUBE(manager_id);
```



	DEPARTMENT_ID	JOB_ID	MANAGER_ID	SUM(SALARY)
1	(null)	SA_REP	149	7000
2	10	AD_ASST	101	4400
3	20	MK_MAN	100	13000
4	20	MK_REP	201	6000
...				
1	90	AD_VP	100	34000
	90	AD_PRES	(null)	24000
...				
	(null)	SA_REP	(null)	7000
	10	AD_ASST	(null)	4400
...				
2	110	(null)	101	12000
	110	(null)	205	8300
	110	(null)	(null)	20300
				3

Im Beispiel auf der Folie werden folgende Gruppierungen erstellt:

- `(department_id,job_id,)` (1)
- `(department_id,manager_id)` (2)
- `(department_id)` (3)

Für jede dieser Gruppen wird das Gesamtgehalt berechnet.

Weiteres Beispiel für eine verkettete Gruppierung:

```
SELECT department_id, job_id, manager_id, SUM(salary) totsal
FROM employees
WHERE department_id<60
GROUP BY GROUPING SETS(department_id),
GROUPING SETS (job_id, manager_id);
```

Zusammenfassung

In diesem Anhang haben Sie Folgendes gelernt:

- Mit dem Operator ROLLUP Zwischensummenwerte erstellen
- Mit dem Operator CUBE Kreuztabellenwerte erstellen
- Mit der Funktion GROUPING von ROLLUP oder CUBE erstellte Zeilenwerte identifizieren
- Mit der Syntax für GROUPING SETS mehrere Gruppierungen in derselben Abfrage definieren
- Mit der Klausel GROUP BY Ausdrücke auf verschiedene Weise kombinieren:
  - Zusammengesetzte Spalten
  - Verkettete Gruppierungssätze

ORACLE

- ROLLUP und CUBE sind Erweiterungen der Klausel GROUP BY.
- ROLLUP wird verwendet, um Zwischensummen- und Gesamtsummenwerte anzuzeigen.
- CUBE wird verwendet, um Kreuztabellenwerte anzuzeigen.
- Mit der Funktion GROUPING können Sie feststellen, ob eine Zeile ein vom Operator CUBE oder ROLLUP erstelltes Aggregat ist.
- Mit der GROUPING SETS-Syntax können Sie mehrere Gruppierungen in derselben Abfrage definieren. GROUP BY ermittelt alle angegebenen Gruppierungen und kombiniert sie mithilfe von UNION ALL.
- Innerhalb der Klausel GROUP BY können Sie Ausdrücke auf verschiedene Weise kombinieren:
  - Um zusammengesetzte Spalten anzugeben, gruppieren Sie Spalten in Klammern, sodass sie bei der Verarbeitung von ROLLUP- oder CUBE-Vorgängen vom Oracle-Server als Einheit behandelt werden.
  - Um verkettete Grouping Sets anzugeben, trennen Sie mehrere Gruppierungssätze, ROLLUP- und CUBE-Vorgänge durch Kommas, sodass sie vom Server zu einer einzigen GROUP BY-Klausel kombiniert werden. Das Ergebnis ist ein Kreuzprodukt aus Gruppierungen der einzelnen Gruppierungssätze.

Hierarchische Datenabfragen

Ziele

Nach Ablauf dieses Anhangs haben Sie folgende Ziele erreicht:

- Konzept hierarchischer Abfragen interpretieren
- Berichte mit Baumstruktur erstellen
- Hierarchische Daten formatieren
- Verzweigungen der Baumstruktur ausblenden (Pruning)

ORACLE

In dieser Lektion wird beschrieben, wie Sie mithilfe hierarchischer Abfragen Berichte mit einer Baumstruktur erstellen.

Beispieldaten aus der Tabelle EMPLOYEES

	EMPLOYEE_ID	LAST_NAME	JOB_ID	MANAGER_ID
1	100	King	AD_PRES	(null)
2	101	Kochhar	AD_VP	100
3	102	De Haan	AD_VP	100
4	103	Hunold	IT_PROG	102
5	104	Ernst	IT_PROG	103
6	107	Lorentz	IT_PROG	103

...

16	200	Whalen	AD_ASST	101
17	201	Hartstein	MK_MAN	100
18	202	Fay	MK_REP	201
19	205	Higgins	AC_MGR	101
20	206	Gietz	AC_ACCOUNT	205

ORACLE

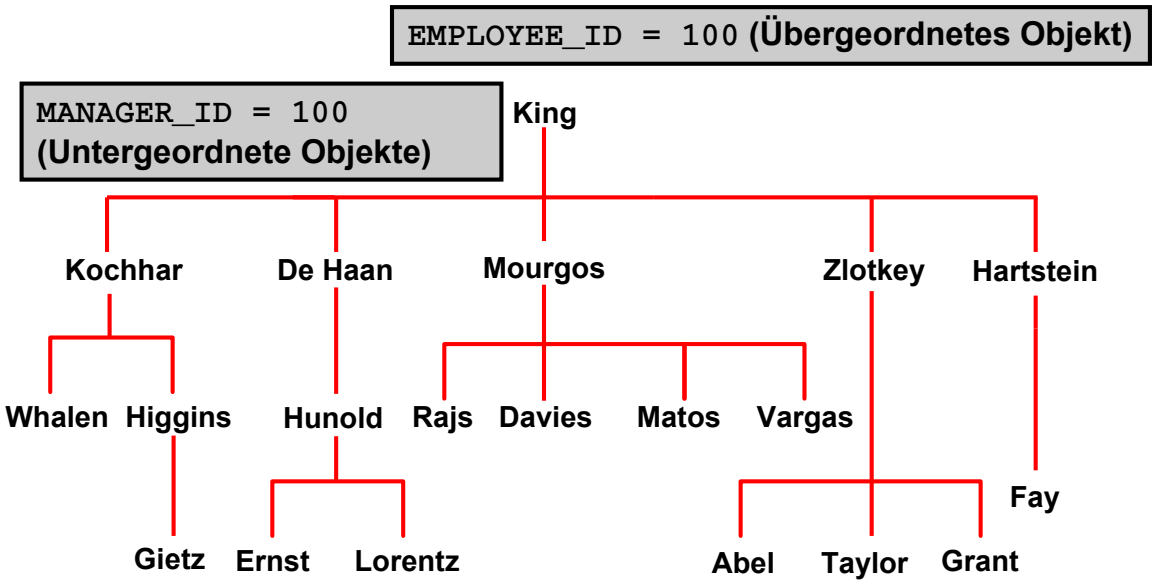
Mit hierarchischen Abfragen können Sie Daten auf der Basis einer natürlichen hierarchischen Beziehung zwischen Zeilen einer Tabelle abrufen. In relationalen Datenbanken werden Datensätze nicht auf hierarchische Weise gespeichert. Wenn jedoch zwischen den Zeilen einer Tabelle eine hierarchische Beziehung besteht, ermöglicht das so genannte *Durchlaufen der Baumstruktur* das Erstellen der Hierarchie. Hierarchische Abfragen sind eine Berichtsmethode, mit der die Zweige einer Baumstruktur in der korrekten Reihenfolge zurückgegeben werden.

Stellen Sie sich einen Stammbaum vor, bei dem sich die ältesten Familienmitglieder in der Nähe der Wurzel oder des Stammes befinden und die jüngsten Mitglieder die Zweige des Baumes darstellen. Diese Verzweigungen können wiederum verzweigt sein und so weiter.

Eine hierarchische Abfrage ist möglich, wenn zwischen Zeilen in einer Tabelle eine Beziehung besteht. Im Beispiel auf der Folie sind Kochhar, De Haan und Hartstein der `MANAGER_ID` 100 unterstellt, die der `EMPLOYEE_ID` von King entspricht.

Hinweis: Hierarchische Baumstrukturen werden in verschiedenen Bereichen verwendet, wie etwa in der Ahnenforschung (Stammbäume), der Viehzucht, dem Unternehmensmanagement (Managementhierarchien), der Fertigung (Produktmontage), der Evolutionsforschung (Artenentwicklung) und der wissenschaftlichen Forschung.

Natürliche Baumstrukturen



Die Tabelle `EMPLOYEES` besitzt eine Baumstruktur, die die Berichterstattungsstrukturen im Management darstellt. Für die Erstellung der Hierarchie kann die Beziehung zwischen äquivalenten Werten in den Spalten `EMPLOYEE_ID` und `MANAGER_ID` herangezogen werden. Diese Beziehung kann durch Verknüpfen der Tabelle mit sich selbst genutzt werden. Die Spalte `MANAGER_ID` enthält die Angestelltennummer des Managers des Angestellten.

Durch die Überordnung/Unterordnung in einer Baumstruktur steuern Sie:

- die Richtung, in der die Hierarchie durchlaufen wird
- den Ausgangspunkt innerhalb der Hierarchie

Hinweis: Auf der Folie ist eine invertierte Baumstruktur der Managementhierarchie der Angestellten in der Tabelle `EMPLOYEES` dargestellt.

Hierarchische Abfragen

```
SELECT [LEVEL], column, expr...
FROM   table
[WHERE condition(s)]
[START WITH condition(s)]
[CONNECT BY PRIOR condition(s)] ;
```

condition:

```
expr comparison_operator expr
```


Schlüsselwörter und Klauseln

Hierarchische Abfragen können durch das Vorhandensein der Klauseln `CONNECT BY` und `START WITH` identifiziert werden.

Für die Syntax gilt:

`SELECT`	Standardmäßige `SELECT`-Klausel
`LEVEL`	Pseudospalte, die für jede von einer hierarchischen Abfrage zurückgegebene Rootzeile 1, für eine untergeordnete Zeile der Rootzeile 2 und so weiter zurückgibt.
`FROM` *table*	Gibt die Tabelle, die View oder den Snapshot an, worin die Spalten enthalten sind. Sie können die `SELECT`-Anweisung nur für eine Tabelle ausführen.
`WHERE`	Schränkt die von der Abfrage zurückgegebenen Zeilen ein, ohne dass andere Zeilen der Hierarchie davon betroffen sind.
condition	Ein Vergleich mit Ausdrücken
`START WITH`	Die Rootzeilen (Ausgangszeilen) der Hierarchie. Diese Klausel ist für echte hierarchische Abfragen obligatorisch.
`CONNECT BY PRIOR`	Die Spalten, in denen die Beziehung zwischen übergeordneten und untergeordneten `PRIOR`-Zeilen existiert. Diese Klausel ist für hierarchische Abfragen obligatorisch.

Baumstruktur durchlaufen

Ausgangspunkt

- Gibt die zu erfüllende Bedingung an
- Akzeptiert jede gültige Bedingung

```
START WITH column1 = value
```

Beginnen Sie in der Tabelle EMPLOYEES mit dem Mitarbeiter, dessen Nachname Kochhar lautet.

```
...START WITH last_name = 'Kochhar'
```

ORACLE

Die Zeilen, die als Stamm der Baumstruktur verwendet werden sollen, werden mit der Klausel START WITH festgelegt. Die Klausel START WITH kann eine beliebige gültige Bedingung enthalten.

Beispiele

Beginnen Sie in der Tabelle EMPLOYEES bei King, dem Geschäftsführer des Unternehmens.

```
... START WITH manager_id IS NULL
```

Beginnen Sie in der Tabelle EMPLOYEES mit dem Mitarbeiter Kochhar. Die Bedingung START WITH kann eine Unterabfrage enthalten.

```
... START WITH employee_id = (SELECT employee_id
                              FROM   employees
                              WHERE  last_name = 'Kochhar')
```

Wenn Sie die Klausel START WITH nicht angeben, werden alle Zeilen in der Tabelle beim Durchlaufen der Baumstruktur als Rootzeilen betrachtet.

Hinweis: Die Klauseln CONNECT BY und START WITH entsprechen nicht dem ANSI-(American National Standards Institute-)Standard für SQL.

Baumstruktur durchlaufen

```
CONNECT BY PRIOR column1 = column2
```

Durchlaufen Sie die Baumstruktur mit der Tabelle EMPLOYEES von oben nach unten.

```
... CONNECT BY PRIOR employee_id = manager_id
```

Richtung

Von oben nach unten → Column1 = Übergeordneter Schlüssel
Column2 = Untergeordneter Schlüssel

Von unten nach oben → Column1 = Untergeordneter Schlüssel
Column2 = Übergeordneter Schlüssel

ORACLE

Die Richtung der Abfrage wird durch die Position der Spalte CONNECT BY PRIOR bestimmt. Um den Baum von oben nach unten zu durchlaufen, bezieht sich der Operator PRIOR auf die übergeordnete Zeile. Um den Baum von unten nach oben zu durchlaufen, bezieht sich der Operator PRIOR auf die untergeordnete Zeile. Um die untergeordneten Zeilen einer übergeordneten Zeile zu ermitteln, wertet der Oracle-Server den Ausdruck PRIOR für die übergeordnete Zeile und die anderen Ausdrücke für jede Zeile in der Tabelle aus. Zeilen, die die Bedingung erfüllen, sind untergeordnete Zeilen der übergeordneten Zeile. Der Oracle-Server wählt untergeordnete Zeilen immer dadurch aus, dass er die Bedingung CONNECT BY in Bezug auf eine aktuelle übergeordnete Zeile auswertet.

Beispiele

Durchlaufen Sie in der Tabelle EMPLOYEES die Baumstruktur von oben nach unten. Definieren Sie eine hierarchische Beziehung, in der der Wert von EMPLOYEE_ID der übergeordneten Zeile dem Wert von MANAGER_ID der untergeordneten Zeile entspricht.

```
... CONNECT BY PRIOR employee_id = manager_id
```

Durchlaufen Sie in der Tabelle EMPLOYEES die Baumstruktur von unten nach oben.

```
... CONNECT BY PRIOR manager_id = employee_id
```

Der Operator PRIOR muss im Code nicht zwangsläufig direkt nach CONNECT BY angegeben werden. Daher führt die folgende Klausel CONNECT BY PRIOR zum gleichen Ergebnis wie im vorhergehenden Beispiel:

```
... CONNECT BY employee_id = PRIOR manager_id
```

Hinweis: Die Klausel CONNECT BY darf keine Unterabfrage enthalten.

Baumstruktur durchlaufen – Von unten nach oben (Bottom-Up)

```
SELECT employee_id, last_name, job_id, manager_id
FROM   employees
START  WITH  employee_id = 101
CONNECT BY PRIOR manager_id = employee_id ;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	MANAGER_ID
1	101	Kochhar	AD_VP	100
2	100	King	AD_PRES	(null)

ORACLE

Im Beispiel auf der Folie wird eine Liste von Managern angezeigt, die mit dem Mitarbeiter mit der Personalnummer 101 beginnt.

Baumstruktur durchlaufen – Von oben nach unten (Top-Down)

```
SELECT  last_name||' reports to '||
PRIOR   last_name "Walk Top Down"
FROM    employees
START   WITH last_name = 'King'
CONNECT BY PRIOR employee_id = manager_id ;
```

	Walk Top Down
1	King reports to
2	King reports to
3	Kochhar reports to King
4	Greenberg reports to Kochhar
5	Faviet reports to Greenberg

...

105	Grant reports to Zlotkey
106	Johnson reports to Zlotkey
107	Hartstein reports to King
108	Fay reports to Hartstein

Zeigen Sie die Namen der Mitarbeiter und ihrer Manager an. Durchlaufen Sie dabei die Baumstruktur von oben nach unten. Verwenden Sie den Mitarbeiter King als Ausgangspunkt. Geben Sie nur eine Spalte aus.

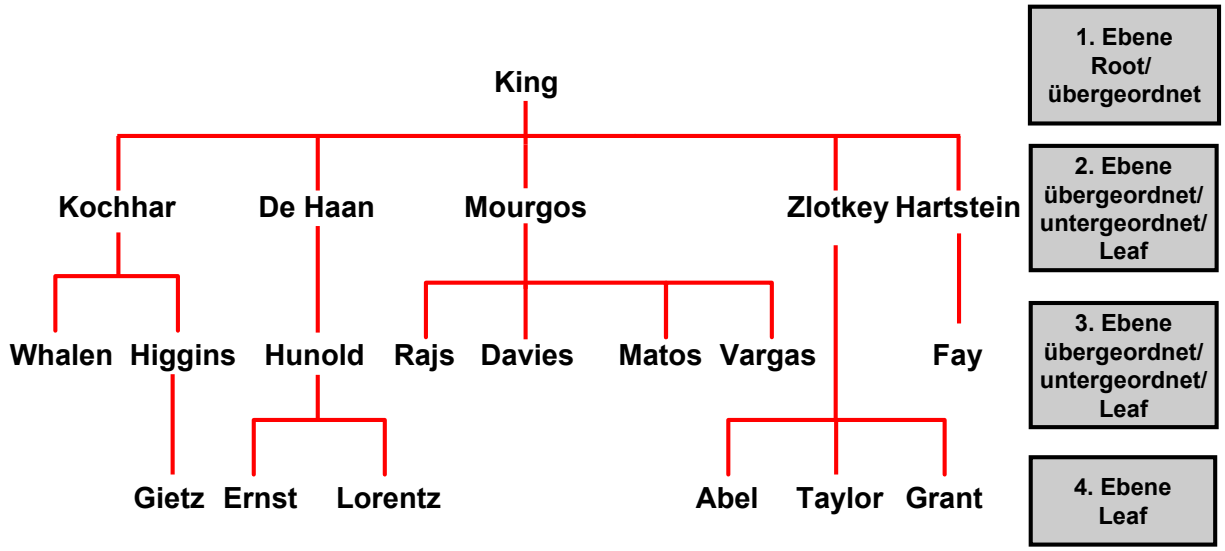
Beispiel

Im folgenden Beispiel werden die Werte von `EMPLOYEE_ID` für die übergeordnete Zeile und die Werte von `MANAGER_ID` und `SALARY` für die untergeordneten Zeilen ausgewertet. Der Operator `PRIOR` gilt nur für den Wert von `EMPLOYEE_ID`.

```
... CONNECT BY PRIOR employee_id = manager_id
                     AND salary > 15000;
```

Um als untergeordnete Zeile zu gelten, muss der Wert von `MANAGER_ID` dem Wert von `EMPLOYEE_ID` der übergeordneten Zeile entsprechen und der Wert von `SALARY` größer als $ 15.000 sein.

Rangfolge von Zeilen mit der Pseudospalte LEVEL festlegen



Mit der Pseudospalte LEVEL können Sie die Position oder die Ebene einer Zeile in der Hierarchie explizit anzeigen. Dadurch wird der Bericht übersichtlicher. Die Abzweigungen, an denen sich eine große Verzweigung in eine oder mehrere Verzweigungen teilt, werden als Knoten bezeichnet. Das Ende einer Verzweigung heißt Leaf oder Leaf-Knoten. Die Abbildung auf der Folie zeigt die Knoten der invertierten Baumstruktur mit ihren Werten vom Typ LEVEL. Beispiel: Der Mitarbeiter Higgins ist ein übergeordneter und ein untergeordneter Knoten. Der Mitarbeiter Davies ist dagegen ein untergeordneter Knoten und ein Leaf-Knoten.

Pseudospalte LEVEL

Wert	Ebene für Top-Down	Ebene für Bottom-Up
1	Ein Rootknoten	Ein Rootknoten
2	Untergeordneter Knoten eines Rootknotens	Übergeordneter Knoten eines Rootknotens
3	Untergeordneter Knoten eines untergeordneten Knotens usw.	Übergeordneter Knoten eines übergeordneten Knotens usw.

Auf der Folie ist King der Rootknoten oder übergeordnete Knoten (LEVEL = 1). Kochhar, De Haan, Mourgos, Zlotkey, Hartstein, Higgens und Hunold sind untergeordnete Knoten und gleichzeitig übergeordnete Knoten (LEVEL = 2). Whalen, Rajs, Davies, Matos, Vargas, Gietz, Ernst, Lorentz, Abel, Taylor, Grant und Fay sind untergeordnete und Leaf-Knoten (LEVEL = 3 and LEVEL = 4).

Hinweis: Ein *Rootknoten* ist der oberste Knoten in einer invertierten Baumstruktur. Ein *untergeordneter Knoten* ist jeder Knoten außer dem Rootknoten. Ein übergeordneter Knoten ist jeder Knoten, der über untergeordnete Knoten verfügt. Ein Leaf-Knoten besitzt keine untergeordneten Knoten. Die Anzahl der Ebenen, die eine hierarchische Abfrage zurückgibt, wird eventuell durch den verfügbaren Benutzerspeicher begrenzt.

Hierarchische Berichte mit LEVEL und LPAD formatieren

Bericht erstellen, der die Managementebenen des Unternehmens zeigt. Er beginnt mit der obersten Ebene, alle nachfolgenden Ebenen sind eingerückt.

```
COLUMN org_chart FORMAT A12
SELECT LPAD(last_name, LENGTH(last_name)+(LEVEL*2)-2,'_')
       AS org_chart
FROM   employees
START WITH first_name='Steven' AND last_name='King'
CONNECT BY PRIOR employee_id=manager_id
```

ORACLE

Den Knoten einer Baumstruktur werden ausgehend vom Rootknoten Ebenennummern zugewiesen. Verwenden Sie die Funktion `LPAD` in Verbindung mit der Pseudospalte `LEVEL`, um einen hierarchischen Bericht als Baumstruktur mit Einrückungen anzuzeigen.

Im Beispiel auf der Folie gilt:

- `LPAD(`*`char1`*`,`*`n`* `[,`*`char2`*`])` gibt *`char1`* zurück, nach links bis zur Länge *`n`* mit der Zeichenfolge in *`char2`* aufgefüllt. Das Argument *`n`* ist die Gesamtlänge des Rückgabewertes, mit der er auf dem Bildschirm angezeigt wird.
- `LPAD(last_name, LENGTH(last_name)+(LEVEL*2)-2,'_')` definiert das Anzeigeformat
- *`char1`* ist der Wert von `LAST_NAME`, *`n`* die Gesamtlänge des Rückgabewertes, d. h. die Länge von `LAST_NAME +(LEVEL*2)-2` und *`char2`* ist `'_'`.

Dieser Code weist SQL an, den `LAST_NAME`-Wert abzurufen und nach links mit dem `'_'`-Zeichen aufzufüllen, bis die Länge der Zeichenfolge dem durch `LENGTH(last_name)+(LEVEL*2)-2` festgelegten Wert entspricht.

Für King gilt `LEVEL = 1`. Daraus folgt (2 x 1) - 2 = 2 - 2 = 0. Somit wird "King" nicht mit dem Zeichen `'_'` aufgefüllt und wird in Spalte 1 angezeigt.

Für Kochhar gilt `LEVEL = 2`. Daraus folgt (2 x 2) - 2 = 4 - 2 = 2. Kochhar wird deshalb mit 2 `'_'`-Zeichen aufgefüllt und eingerückt angezeigt.
Die übrigen Datensätze in der Tabelle `EMPLOYEES` werden auf ähnliche Weise angezeigt.

	ORG_CHART
1	King
2	__Kochhar
3	____Greenberg
4	______Faviet
5	______Chen
6	______Sciarra
7	______Urman
8	______Popp
9	____Whalen
10	____Mavris
11	____Baer
12	____Higgins
13	______Gietz
14	__De Haan
15	____Hunold
16	______Ernst
17	______Austin

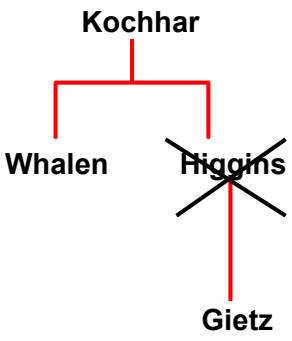
Verzweigungen ausblenden (Pruning)

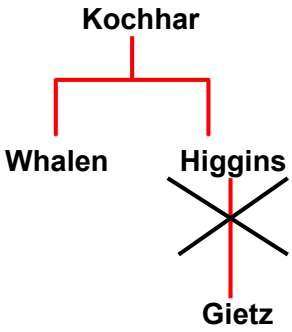
Mit WHERE-Klausel
Knoten entfernen

```
WHERE last_name != 'Higgins'
```

Mit CONNECT BY-Klausel
Verzweigung entfernen

```
CONNECT BY PRIOR
employee_id = manager_id
AND last_name != 'Higgins'
```





Mit den Klauseln WHERE und CONNECT BY können Sie Elemente der Baumstruktur ausblenden (das heißt steuern, welche Knoten oder Zeilen angezeigt werden). Das verwendete Prädikat fungiert als boolesche Bedingung.

Beispiele

Durchlaufen Sie die Hierarchie beginnend bei der Root von oben nach unten. Blenden Sie den Mitarbeiter Higgins aus dem Ergebnis aus, wobei die untergeordneten Zeilen jedoch verarbeitet werden sollen.

```
SELECT  department_id, employee_id,last_name, job_id, salary
FROM    employees
WHERE   last_name  != 'Higgins'
START   WITH manager_id IS NULL
CONNECT BY PRIOR employee_id = manager_id;
```

Durchlaufen Sie die Hierarchie ausgehend von der Root von oben nach unten, und blenden Sie den Mitarbeiter Higgins und alle untergeordneten Zeilen aus.

```
SELECT  department_id, employee_id,last_name, job_id, salary
FROM    employees
START   WITH manager_id IS NULL
CONNECT BY PRIOR employee_id = manager_id
AND     last_name != 'Higgins';
```

Zusammenfassung

In diesem Anhang haben Sie Folgendes gelernt:

- Mit hierarchischen Abfragen eine hierarchische Beziehung zwischen Zeilen einer Tabelle anzeigen
- Richtung und Ausgangspunkt der Abfrage angeben
- Knoten oder Verzweigungen der Baumstruktur ausblenden (Pruning)

ORACLE

Mit hierarchischen Abfragen können Sie Daten auf der Basis einer natürlichen hierarchischen Beziehung zwischen Zeilen einer Tabelle abrufen. Die Pseudospalte `LEVEL` ermittelt, wie weit Sie eine hierarchische Baumstruktur nach unten durchlaufen haben. Mithilfe der Klausel `CONNECT BY PRIOR` können Sie die Abfragerichtung festlegen. Mit der Klausel `START WITH` geben Sie den Ausgangspunkt der Abfrage an. Um Verzweigungen der Baumstruktur auszublenden, verwenden Sie die Klauseln `WHERE` und `CONNECT BY`.

Fortgeschrittene Skripte erstellen

Ziele

Nach Ablauf dieses Anhangs haben Sie folgende Ziele erreicht:

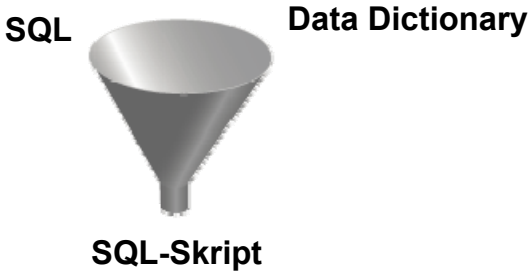
- Aufgabenstellungen beschreiben, die durch die Generierung von SQL-Skripten mit SQL gelöst werden
- Einfache SQL-Skripte erstellen
- Ausgabe in einer Datei erfassen
- Tabelleninhalt in eine Datei ausgeben
- Dynamische Prädikate generieren

ORACLE

In diesem Anhang wird beschrieben, wie Sie ein SQL-Skript erstellen, das ein SQL-Skript generiert.

SQL-Skripte mit SQL generieren

- SQL kann zum Generieren von Skripten in SQL verwendet werden.
- Das Data Dictionary:
  - ist eine Zusammenstellung von Tabellen und Views, die Datenbankinformationen enthalten
  - wird vom Oracle-Server erstellt und verwaltet



SQL ist ein leistungsfähiges Tool zum Generieren anderer SQL-Anweisungen. Meistens wird dazu eine Skriptdatei erstellt. Mit den aus SQL generierten SQL-Anweisungen können Sie:

- wiederholte Codierung vermeiden
- auf Informationen im Data Dictionary zugreifen
- Datenbankobjekte löschen oder neu erstellen
- dynamische Prädikate generieren, die Laufzeitparameter enthalten

In den Beispielen dieses Anhangs werden Informationen aus dem Data Dictionary gewählt. Das Data Dictionary ist eine Zusammenstellung von Tabellen und Views, die Informationen zur Datenbank enthalten. Diese Zusammenstellung wird vom Oracle-Server erstellt und verwaltet. Eigentümer aller Data Dictionary-Tabellen ist der Benutzer `SYS`. Zu den im Data Dictionary gespeicherten Informationen gehören die Namen von Oracle-Server-Benutzern, die den Benutzern erteilten Berechtigungen, die Namen von Datenbankobjekten, Tabellen-Constraints und Auditinformationen. Es gibt vier Kategorien von Data Dictionary Views. Jede Kategorie besitzt ein eindeutiges Präfix, das den beabsichtigten Verwendungszweck wiedergibt.

Präfix	Beschreibung
`USER_`	Enthält Details zu Objekten, die dem Benutzer gehören
`ALL_`	Enthält Details zu Objekten, für die der Benutzer Zugriffsberechtigungen besitzt und zu Objekten, die dem Benutzer gehören
`DBA_`	Enthält Details zu Benutzern mit DBA-Berechtigungen zum Zugriff auf Objekte in der Datenbank
`V$_`	Speichert Informationen zur Performance des Datenbankservers und zu Sperren; nur für den DBA verfügbar

Einfache Skripte erstellen

```
SELECT 'CREATE TABLE ' || table_name ||
       '_test ' || 'AS SELECT * FROM '
       || table_name ||' WHERE 1=2;'
       AS "Create Table Script"
FROM   user_tables;
```

	Create Table Script
1	CREATE TABLE REGIONS_test AS SELECT * FROM REGIONS WHERE 1=2;
2	CREATE TABLE LOCATIONS_test AS SELECT * FROM LOCATIONS WHERE 1=2;
3	CREATE TABLE DEPARTMENTS_test AS SELECT * FROM DEPARTMENTS WHERE 1=2;
4	CREATE TABLE JOBS_test AS SELECT * FROM JOBS WHERE 1=2;
5	CREATE TABLE EMPLOYEES_test AS SELECT * FROM EMPLOYEES WHERE 1=2;
6	CREATE TABLE JOB_HISTORY_test AS SELECT * FROM JOB_HISTORY WHERE 1=2;

ORACLE

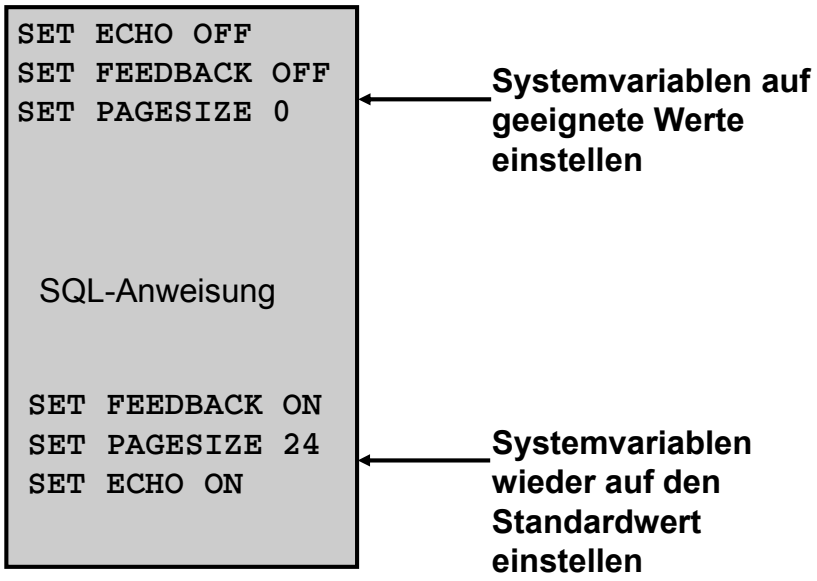
Das Beispiel auf der Folie erstellt mit `CREATE TABLE`-Anweisungen einen Bericht aus jeder Tabelle, deren Eigentümer Sie sind. Jede im Bericht erstellte `CREATE TABLE`-Anweisung enthält die Syntax, um eine Tabelle mit dem Tabellennamen und dem Suffix `_test` und der Struktur der zugehörigen bestehenden Tabelle zu erstellen. Der alte Tabellenname wird aus der Spalte `TABLE_NAME` der Data Dictionary View `USER_TABLES` abgerufen.

Im nächsten Schritt wird der Bericht erweitert, um den Prozess zu automatisieren.

Hinweis: Sie können die Data Dictionary-Tabellen abfragen, um die verschiedenen Datenbankobjekte anzuzeigen, deren Eigentümer Sie sind. Die am häufigsten abgefragten Data Dictionary Views sind:

- `USER_TABLES`: Zeigt eine Beschreibung der Tabellen an, deren Eigentümer der Benutzer ist
- `USER_OBJECTS`: Zeigt alle Objekte an, deren Eigentümer der Benutzer ist
- `USER_TAB_PRIVS_MADE`: Zeigt alle vergebenen Berechtigungen für Objekte an, deren Eigentümer der Benutzer ist
- `USER_COL_PRIVS_MADE`: Zeigt alle vergebenen Berechtigungen für Objektspalten an, deren Eigentümer der Benutzer ist

Umgebung steuern



Um die generierten SQL-Anweisungen auszuführen, speichern Sie sie in einer Datei, die anschließend ausgeführt werden kann. Außerdem müssen Sie die Bereinigung der erstellten Ausgabe planen sowie sicherstellen, dass Elemente wie Überschriften, Rückmeldungen, Kopfzeilen und so weiter unterdrückt werden. In SQL Developer können Sie diese Anweisungen in einem Skript speichern. Um den Inhalt des Feldes **Enter SQL Statement** zu speichern, klicken Sie auf das Symbol **Save**, oder verwenden Sie die Menüoption **File > Save**. Alternativ können Sie mit der rechten Maustaste in das Feld **Enter SQL Statement** klicken und im Dropdown-Menü die Option **Save File** wählen.

Hinweis: Einige SQL*Plus-Anweisungen werden nicht vom SQL Worksheet unterstützt. Eine vollständige Liste der von SQL Worksheet unterstützten und nicht unterstützten SQL*Plus-Anweisungen finden Sie in der Onlinehilfe von SQL Developer unter dem Thema *SQL*Plus Statements Supported and Not Supported in SQL Worksheet*.

Gesamtbild

```
SET ECHO OFF
SET FEEDBACK OFF
SET PAGESIZE 0

SELECT 'DROP TABLE ' || object_name || ';'
FROM    user_objects
WHERE   object_type = 'TABLE'
/

SET FEEDBACK ON
SET PAGESIZE 24
SET ECHO ON
```


Die Ausgabe des Befehls auf der Folie wird in SQL Developer in der Datei `dropem.sql` gespeichert. Um in SQL Developer die Ausgabe in einer Datei zu speichern, verwenden Sie im Fensterbereich **Script Output** die Option **Save File**. Die Datei `dropem.sql` enthält die unten stehenden Daten. Sie können die Datei jetzt in SQL Developer starten, indem Sie die Skriptdatei suchen, laden und ausführen.

	'DROPTABLE'||OBJECT_NAME||';'
1	DROP TABLE REGIONS;
2	DROP TABLE COUNTRIES;
3	DROP TABLE LOCATIONS;
4	DROP TABLE DEPARTMENTS;
5	DROP TABLE JOBS;
6	DROP TABLE EMPLOYEES;
7	DROP TABLE JOB_HISTORY;
8	DROP TABLE JOB_GRADES;

Tabelleninhalt in eine Datei ausgeben

```
SET HEADING OFF ECHO OFF FEEDBACK OFF
SET PAGESIZE 0


SELECT
 'INSERT INTO departments_test VALUES
  (' || department_id || ', ''' || department_name ||
   ''', ''' || location_id || ''');'
  AS "Insert Statements Script"
FROM   departments
/


SET PAGESIZE 24
SET HEADING ON ECHO ON FEEDBACK ON
```

ORACLE

Manchmal ist es sinnvoll, die Werte für die Tabellenzeilen im Format der Anweisung `INSERT INTO VALUES` in einer Textdatei zu speichern. Falls die Tabelle versehentlich gelöscht wird, können Sie sie durch Ausführen dieses Skriptes füllen.

Das Beispiel auf der Folie erstellt Anweisungen vom Typ `INSERT` für die Tabelle `DEPARTMENTS_TEST`. Diese Anweisungen werden mit der SQL Developer-Option **Save File** in der Datei `data.sql` gespeichert.

Die Skriptdatei `data.sql` hat folgenden Inhalt:

```
INSERT INTO departments_test VALUES
  (10, 'Administration', 1700);
INSERT INTO departments_test VALUES
  (20, 'Marketing', 1800);
INSERT INTO departments_test VALUES
  (50, 'Shipping', 1500);
INSERT INTO departments_test VALUES
  (60, 'IT', 1400);
...
```

Tabelleninhalt in eine Datei ausgeben

Quelle	Ergebnis
'''X'''	'X'
''''	'
''''||department_name||''''	'Administration'
''', '''	','
''');'	');

Die vorherige Folie enthält viele Hochkommas. Eine Gruppe aus vier Hochkommas erzeugt in der endgültigen Anweisung ein einzelnes Hochkomma. Literale Datums- und Zeichenwerte müssen in Hochkommas gesetzt werden.

Um ein Hochkomma innerhalb einer Zeichenfolge anzuzeigen, müssen Sie diesem ein weiteres Hochkomma als Präfix voranstellen. Im fünften Beispiel auf der Folie gelten die umschließenden Hochkommas für die gesamte Zeichenfolge. Das zweite Hochkomma dient als Präfix, um das dritte Hochkomma anzuzeigen. Das Ergebnis ist ein Hochkomma, gefolgt von der Klammer und dem Semikolon.

Dynamische Prädikate generieren

```
COLUMN my_col NEW_VALUE dyn_where_clause

SELECT DECODE('&&deptno', null,
DECODE ('&&hiredate', null, ' ',
'WHERE hire_date=TO_DATE('''||'&&hiredate'',''DD-MON-YYYY'')'),
DECODE ('&&hiredate', null,
'WHERE department_id = ' || '&&deptno',
'WHERE department_id = ' || '&&deptno' ||
' AND hire_date = TO_DATE('''||'&&hiredate'',''DD-MON-YYYY'')'))
AS my_col FROM dual;
```

```
SELECT last_name FROM employees &dyn_where_clause;
```

ORACLE

Das Beispiel auf der Folie generiert eine SELECT-Anweisung, die Daten aller Mitarbeiter einer Abteilung abruft, die an einem bestimmten Tag eingestellt wurden. Das Skript generiert die WHERE-Klausel dynamisch.

Hinweis: Wenn die Benutzervariable verwendet wurde, muss sie mit dem Befehl UNDEFINE gelöscht werden.

Die erste SELECT-Anweisung fordert zur Eingabe der Abteilungsnummer auf. Wenn Sie keine Abteilungsnummer eingeben, wird sie von der Funktion DECODE als Nullwert interpretiert. Danach folgt die Aufforderung zur Eingabe des Einstellungsdatums. Wenn Sie kein Einstellungsdatum eingeben, wird es von der Funktion DECODE als Nullwert interpretiert. Aus diesem Grund ist die generierte dynamische Klausel WHERE ebenfalls ein Nullwert, weshalb die zweite SELECT-Anweisung alle Zeilen aus der Tabelle EMPLOYEES abruft.

Hinweis: Die Variable NEW_V[ALUE] gibt eine Variable für einen Spaltenwert an. Sie können die Variable in TTITLE-Befehlen referenzieren. Mit NEW_VALUE können Sie Spaltenwerte oder Datumsangaben in der Kopfzeile anzeigen. Geben Sie die Spalte in einem Befehl BREAK mit der Aktion SKIP PAGE an. Der Variablenname darf kein Nummernzeichen (#) enthalten. NEW_VALUE ist für Master-Detail-Berichte nützlich, bei denen es für jede Seite einen neuen Master-Datensatz gibt.

Hinweis: Das Einstellungsdatum muss im Format DD-MON-YYYY eingegeben werden.

Die Anweisung SELECT im Beispiel kann wie folgt interpretiert werden:

```
IF   (<<deptno>> is not entered)  THEN
     IF  (<<hiredate>> is not entered)  THEN
           return empty string
     ELSE
           return the string 'WHERE hire_date =
 TO_DATE('<<hiredate>>', 'DD-MON-YYYY')'
 ELSE
         IF (<<hiredate>> is not entered)  THEN
               return the string 'WHERE department_id =
 <<deptno>> entered'
         ELSE
               return the string 'WHERE department_id =
 <<deptno>> entered
                                                AND hire_date =
 TO_DATE(' <<hiredate>>', 'DD-MON-YYYY')'
END IF
```

Die zurückgegebene Zeichenfolge wird zum Wert der Variablen DYN_WHERE_CLAUSE, die in der zweiten SELECT-Anweisung verwendet wird.

Hinweis: Verwenden Sie SQL*Plus für diese Beispiele.

Sobald das erste Beispiel auf der Folie ausgeführt ist, wird der Benutzer aufgefordert, die Werte für DEPTNO und HIREDATE einzugeben:

Geben Sie Werte für DEPTNO und HIREDATE ein: 10 und 17-SEP-2007

Für MY_COL wird folgender Wert generiert:

	MY_COL
1	WHERE department_id = 10 AND hire_date = TO_DATE('17-SEP-2007','DD-MON-YYYY')

Sobald das zweite Beispiel auf der Folie ausgeführt ist, wird folgende Ausgabe generiert:

```
LAST_NAME
-------------------------
Whalen
```

Zusammenfassung

In diesem Anhang haben Sie Folgendes gelernt:

- Einfache SQL-Skripte erstellen
- Ausgabe in einer Datei erfassen
- Tabelleninhalt in eine Datei ausgeben
- Dynamische Prädikate generieren

ORACLE

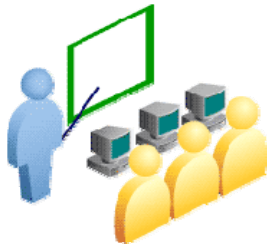
Mit SQL können Sie SQL-Skripte generieren. Mit diesen Skripten können Sie wiederholte Kodierung vermeiden, Objekte löschen und neu erstellen, das Data Dictionary für Informationen heranziehen und dynamische Prädikate erstellen, die Laufzeitparameter enthalten.

Oracle Database – Architekturkomponenten

Ziele

Nach Ablauf dieses Anhangs haben Sie folgende Ziele erreicht:

- Die wichtigsten Architekturkomponenten der Datenbank nennen
- Hintergrundprozesse beschreiben
- Memorystrukturen beschreiben
- Beziehung zwischen logischen und physischen Storage-Strukturen erläutern



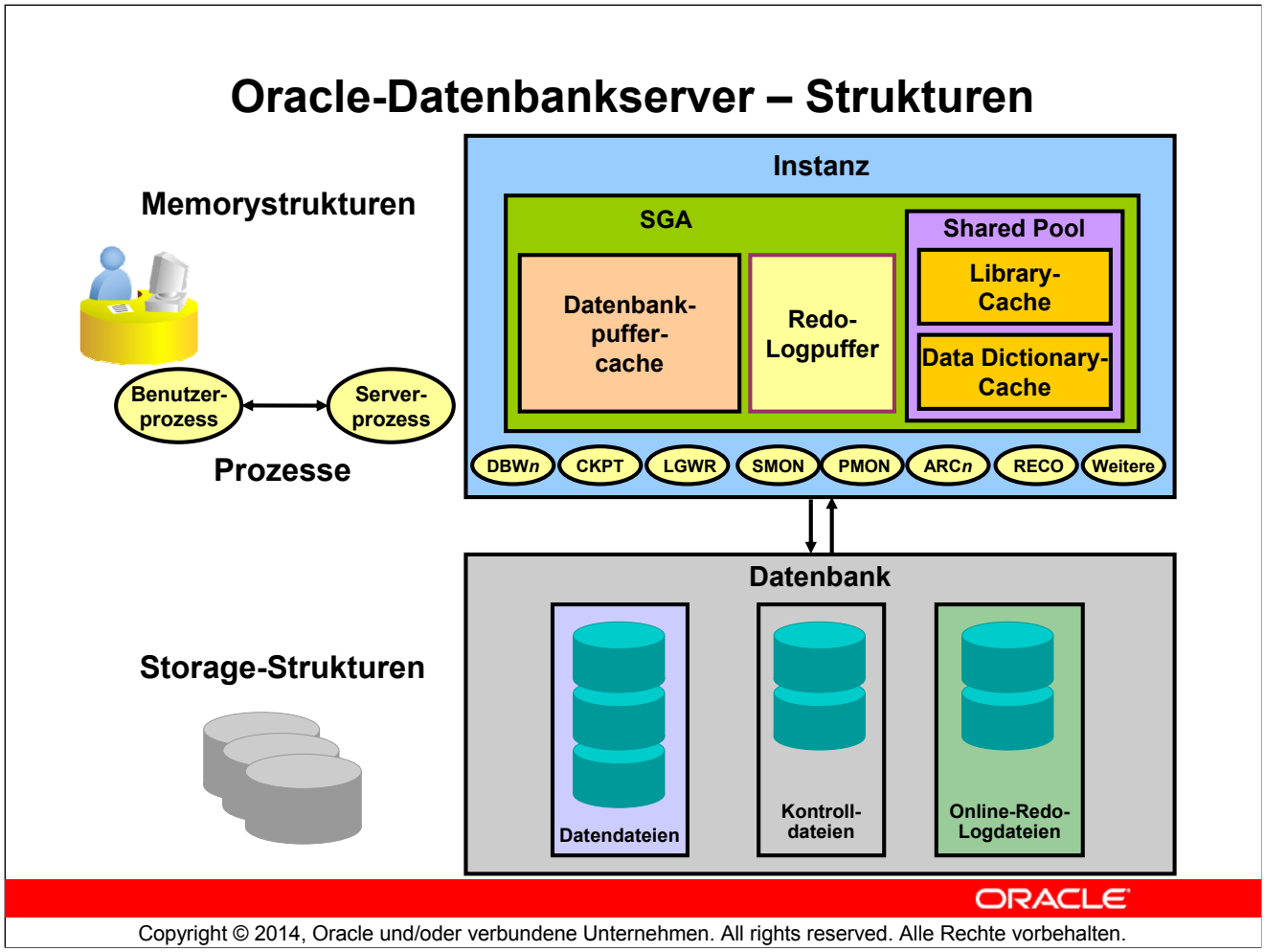
Dieser Anhang bietet einen Überblick über die Architektur von Oracle Database. Sie lernen die physischen und logischen Strukturen sowie verschiedene Komponenten von Oracle Database und die zugehörigen Funktionen kennen.

Architektur von Oracle Database – Überblick

Das Oracle Relational Database Management System (RDBMS) ist ein Datenbankmanagementsystem, das ein offenes, umfassendes und integriertes Informationsmanagement ermöglicht.

In einer Datenbank sind Daten zusammengestellt, die als Einheit behandelt werden. Der Zweck einer Datenbank besteht im Speichern und Abrufen von zusammengehörigen Daten.

Oracle Database verwaltet zuverlässig große Datenmengen in einer Mehrbenutzerumgebung und gewährleistet so, dass zahlreiche Benutzer gleichzeitig auf dieselben Daten zugreifen können. Dabei wird eine hohe Performance sichergestellt. Das Managementsystem verhindert darüber hinaus Zugriffe ohne Autorisierung und bietet effiziente Lösungen für die Wiederherstellung der Daten nach einem Ausfall.



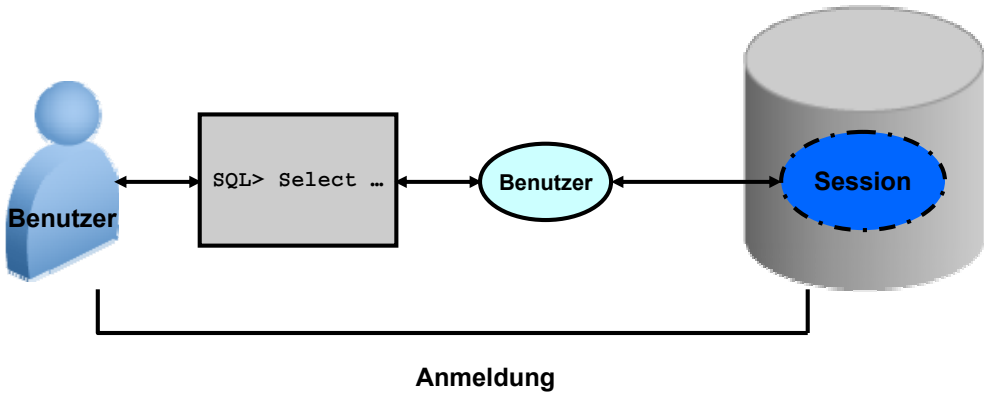
Eine Oracle-Datenbank besteht aus zwei Hauptkomponenten, der Instanz und der eigentlichen Datenbank.

- Die Instanz besteht aus der System Global Area (SGA), einer Zusammenstellung von Memorystrukturen, und den Hintergrundprozessen, die Aufgaben in der Datenbank ausführen. Bei jedem Start einer Instanz wird die SGA zugewiesen, und die Hintergrundprozesse werden gestartet.
- Die Datenbank umfasst physische und logische Strukturen. Da sich die Strukturen in getrennten Bereichen befinden, können die Daten im physischen Storage verwaltet werden, ohne dass der Zugriff auf die logischen Storage-Strukturen beeinträchtigt wird. Die physischen Storage-Strukturen umfassen:
  - Kontrolldateien, in denen die Datenbankkonfiguration gespeichert ist
  - Redo-Logdateien, die die erforderlichen Informationen für ein Datenbank-Recovery enthalten
  - Datendateien, in denen alle Daten gespeichert sind

Eine Oracle-Instanz verwendet Memorystrukturen und Prozesse zur Verwaltung und zum Zugriff auf Storage-Strukturen der Datenbank. Sämtliche Memorystrukturen befinden sich im Hauptmemory der Rechner, die den Datenbankserver bilden. Prozesse sind Jobs, die im Memory dieser Rechner ausgeführt werden. Sie sind definiert als "Steuerungsthread" oder Verfahren in einem Betriebssystem zur Ausführung einer Folge von Schritten.

Bei der Datenbank anmelden

- Verbindung: Kommunikationspfad zwischen einem Benutzerprozess und einer Datenbankinstanz
- Session: Spezifische Verbindung zwischen einem Benutzer und einer Datenbankinstanz durch einen Benutzerprozess



Damit der Benutzer auf Daten in der Datenbank zugreifen kann, muss er sich über ein Tool wie SQL*Plus bei der Datenbank anmelden. Nachdem sich der Benutzer angemeldet und somit eine Verbindung zur Datenbank hergestellt hat, wird eine Session für ihn erstellt. Verbindung und Session hängen zwar eng mit dem Benutzerprozess zusammen, haben aber völlig verschiedene Bedeutungen.

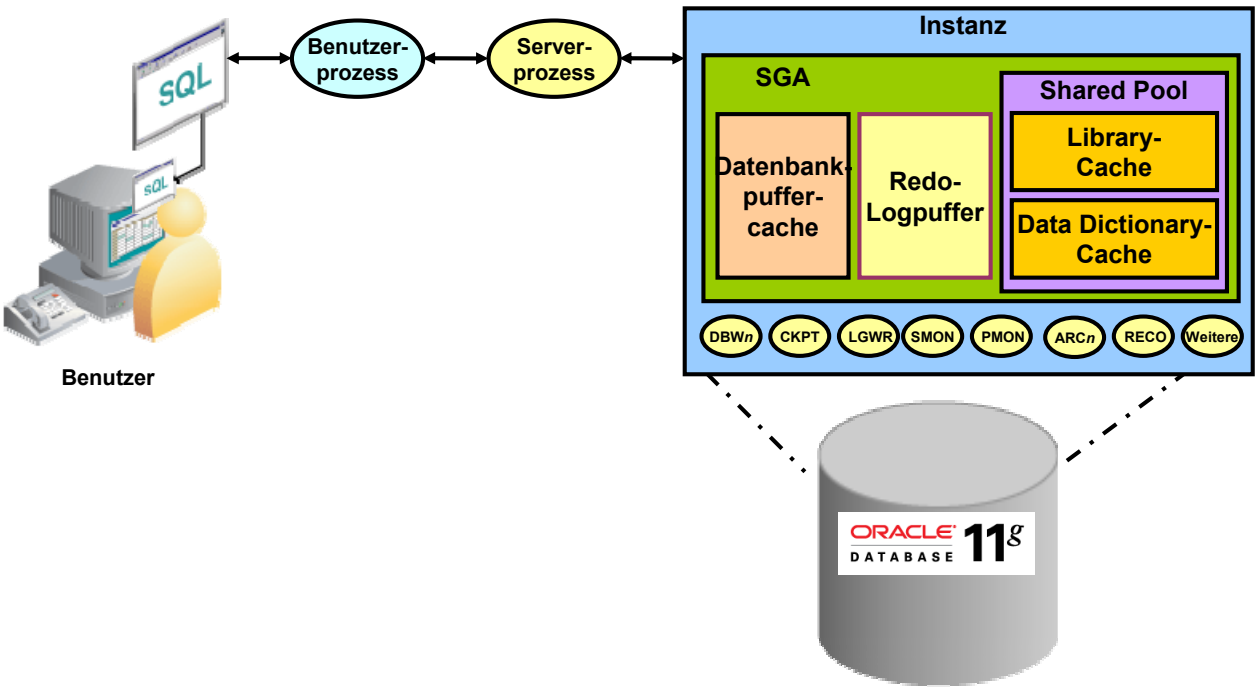
Eine Verbindung ist ein Kommunikationspfad zwischen einem Benutzerprozess und einer Oracle-Datenbankinstanz. Der Kommunikationspfad wird mithilfe von verfügbaren Kommunikations-mechanismen zwischen Prozessen oder einer Netzwerksoftware (wenn die Datenbank-anwendung und Oracle Database auf mehreren Rechnern laufen, die über ein Netzwerk kommunizieren) aufgebaut.

Eine Session stellt die aktuelle Anmeldung eines Benutzers bei der Datenbankinstanz dar. Startet ein Benutzer beispielsweise SQL*Plus, muss er einen gültigen Benutzernamen und ein gültiges Kennwort eingeben. Daraufhin wird eine Session für diesen Benutzer eingerichtet. Eine Session dauert von der Anmeldung des Benutzers bis zu dem Zeitpunkt, zu dem sich der Benutzer von der Datenbank abmeldet oder die Datenbankanwendung beendet.

Bei einer dedizierten Verbindung wird die Session durch einen permanenten dedizierten Prozess verarbeitet. Bei einer gemeinsamen Verbindung wird die Session von einem verfügbaren Serverprozess verarbeitet, der von der Middle Tier oder von der Oracle Shared Server-Architektur aus einem Pool gewählt wird.

Für einen einzelnen Oracle Database-Benutzer können mehrere Sessions erstellt werden, die parallel existieren. Der Benutzer meldet sich unter demselben Benutzernamen an, greift jedoch über verschiedene Anwendungen oder über verschiedene Aufrufe derselben Anwendung auf die Datenbank zu.

Mit einer Oracle-Datenbank interagieren

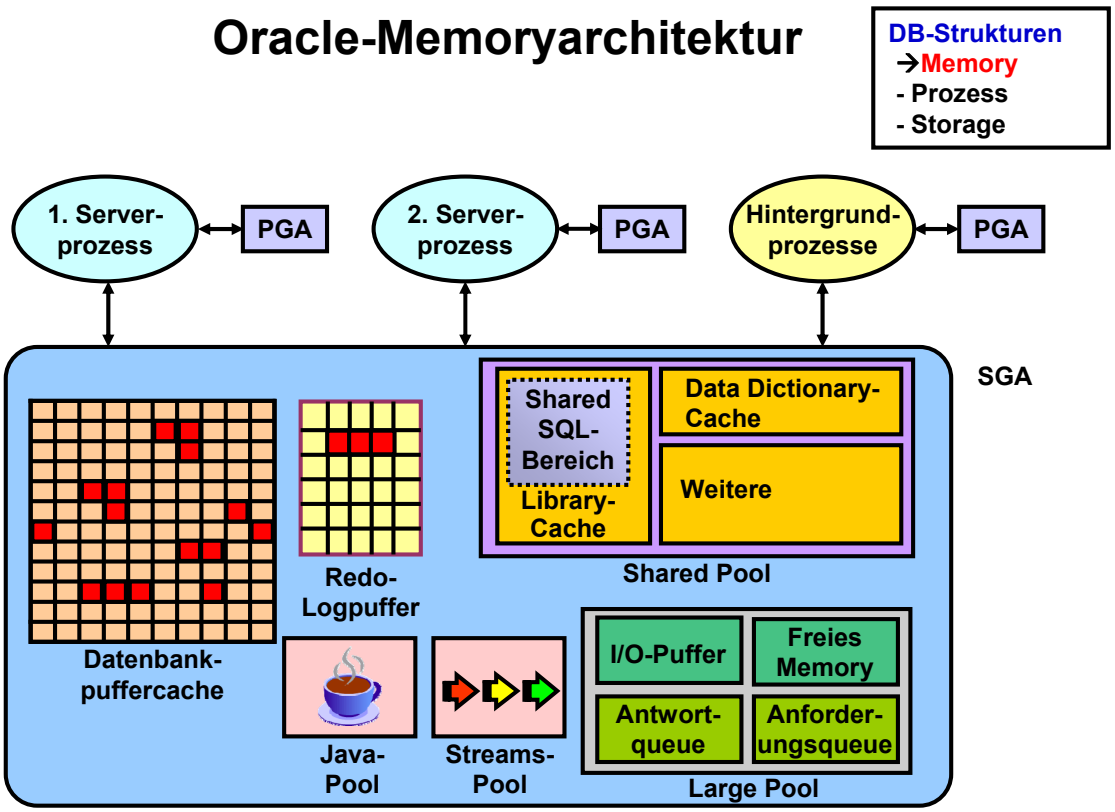


Im folgenden Beispiel werden die grundlegendsten Oracle Database-Vorgänge beschrieben. Dabei ist eine Oracle Database-Konfiguration dargestellt, in der sich der Benutzerprozess und der dazugehörige Serverprozess auf separaten Rechnern befinden, die über ein Netzwerk verbunden sind.

1. Eine Instanz wird auf dem Knotenrechner gestartet, auf dem Oracle Database installiert ist. Der Rechner wird häufig auch als Host oder Datenbankserver bezeichnet.
2. Ein Benutzer startet eine Anwendung, die einen Benutzerprozess startet. Die Anwendung versucht, eine Verbindung zum Server herzustellen. (Bei der Verbindung kann es sich um eine lokale Verbindung, eine Client/Server-Verbindung oder eine Three-Tier-Verbindung von einer Middle Tier handeln.)
3. Der Server führt einen Listener aus, der über den entsprechenden Oracle Net Services-Handler verfügt. Der Server erkennt die Verbindungsanforderung der Anwendung und erstellt für den Benutzerprozess einen dedizierten Serverprozess.
4. Der Benutzer führt eine SQL-Anweisung vom Typ DML aus und schreibt die Transaktion fest. Beispiel: Der Benutzer ändert die Adresse eines Kunden in einer Tabelle und schreibt die Änderung fest.

5. Der Serverprozess empfängt die Anweisung und prüft, ob im Shared Pool (eine SGA-Komponente) ein Shared SQL-Bereich vorhanden ist, der eine ähnliche SQL-Anweisung enthält. Wenn ein solcher Shared SQL-Bereich verfügbar ist, prüft der Serverprozess die Zugriffsberechtigungen des Benutzers auf die angeforderten Daten. Der vorhandene Shared SQL-Bereich wird zur Verarbeitung der Anweisung verwendet. Andernfalls wird der Anweisung ein neuer Shared SQL-Bereich zugewiesen, damit die Anweisung geparst und verarbeitet werden kann.
6. Der Serverprozess ruft alle erforderlichen Datenwerte aus der tatsächlichen Datendatei (in der die Tabelle gespeichert ist) oder aus den in der SGA gespeicherten Daten ab.
7. Der Serverprozess ändert die Daten in der SGA. Da die Transaktion festgeschrieben wurde, zeichnet der Log Writer-Prozess (LGWR) die Transaktion sofort in der Redo-Logdatei auf. Der Database Writer-Prozess (DBW*n*) schreibt geänderte Blöcke zu einem geeigneten Zeitpunkt permanent auf Datenträger.
8. Verläuft die Transaktion erfolgreich, sendet der Serverprozess über das Netzwerk eine Meldung an die Anwendung. Andernfalls wird eine Fehlermeldung ausgegeben.
9. Während des gesamten Vorgangs werden die anderen Hintergrundprozesse ausgeführt. Sie suchen nach Bedingungen, die ein Eingreifen erforderlich machen. Zusätzlich verwaltet der Datenbankserver die Transaktionen anderer Benutzer und verhindert Konflikte zwischen Transaktionen, die dieselben Daten anfordern.

Oracle-Memoryarchitektur



Die Oracle-Datenbank erstellt und verwendet Memorystrukturen für verschiedene Zwecke. Im Memory werden beispielsweise der ausgeführte Programmcode, die von den Benutzern gemeinsam genutzten Daten und private Datenbereiche für jeden angemeldeten Benutzer gespeichert.
Einer Instanz sind zwei grundlegende Memorystrukturen zugeordnet:

- Die System Global Area (SGA) ist eine Gruppe von Shared-Memorystrukturen, die als SGA-Komponenten bezeichnet werden. Sie enthalten Daten und Steuerinformationen für jeweils eine Oracle-Datenbankinstanz. Die SGA wird von allen Server- und Hintergrundprozessen gemeinsam genutzt. Die SGA enthält beispielsweise gecachte Datenblöcke und Shared SQL-Bereiche.
- Die Program Global Areas (PGAs) sind Memorybereiche, die Daten und Steuer-informationen für einen Server- oder Hintergrundprozess enthalten. Eine PGA ist ein nicht zur gemeinsamen Nutzung vorgesehenes Memory, das die Oracle-Datenbank beim Start eines Server- oder Hintergrundprozesses erstellt. Ausschließlich der betreffende Serverprozess hat Zugriff auf die PGA. Jeder Serverprozess und jeder Hintergrundprozess verfügt über eine eigene PGA.

Im Memorybereich der SGA sind die Daten und Steuerinformationen für die Instanz abgelegt. Die SGA enthält die folgenden Datenstrukturen:

- **Datenbank-Puffercache:** Cache für die aus der Datenbank abgerufenen Datenblöcke
- **Redo-Logpuffer:** Cache für Redo-Informationen (für das Instanz-Recovery), die noch nicht in die physischen Redo-Logdateien auf dem Datenträger geschrieben werden können
- **Shared Pool:** Cache für verschiedene Konstrukte, die Benutzer gemeinsam nutzen können
- **Large Pool:** Optionaler Bereich mit großer Memoryzuweisung für bestimmte umfassende Prozesse wie Oracle-Backup- und -Recovery-Vorgänge und I/O-Serverprozesse
- **Java-Pool:** Für sessionspezifischen Java-Code und für die Daten in der Java Virtual Machine (JVM)
- **Streams-Pool:** Mit seiner Hilfe speichert Oracle Streams Informationen, die von Capture- und Apply-Prozessen benötigt werden

Wenn Sie die Instanz mit Enterprise Manager oder SQL*Plus starten, wird die für die SGA zugewiesene Memorykapazität angezeigt.

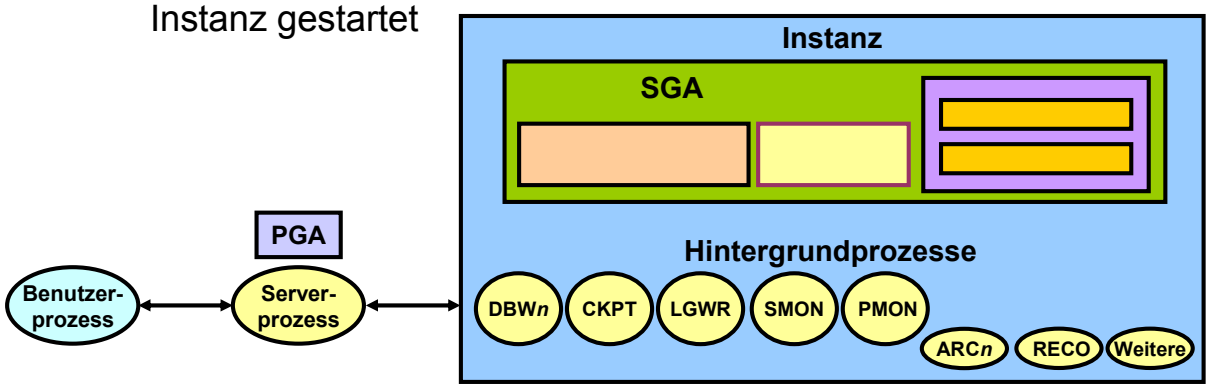
Aufgrund der dynamischen SGA-Infrastruktur können Sie die Größe des Datenbank-Puffercaches, des Shared Pools, des Large Pools, des Java-Pools und des Streams-Pools ändern, ohne die Instanz herunterzufahren.

Oracle Database verwendet Initialisierungsparameter zum Erstellen und Konfigurieren von Memorystrukturen. Beispiel: Der `SGA_TARGET`-Parameter gibt die Gesamtgröße der SGA-Komponenten an. Wenn Sie `SGA_TARGET` auf 0 einstellen, wird Automatic Shared Memory Management deaktiviert.

Prozessarchitektur

DB-Strukturen
- Memory
→ Prozess
- Storage

- Benutzerprozess:
  - Wird gestartet, wenn sich ein Datenbankbenutzer oder ein Batchprozess bei einer Oracle-Datenbank anmeldet
- Datenbankprozesse:
  - Serverprozess: Meldet sich bei der Oracle-Instanz an und wird gestartet, wenn ein Benutzer eine Session aufbaut
  - Hintergrundprozesse: Werden zusammen mit der Oracle-Instanz gestartet



ORACLE

Die Prozesse in einem Oracle-Datenbankserver können in zwei Hauptgruppen unterteilt werden:

- Benutzerprozesse, die den Code der Anwendung oder des Oracle-Tools ausführen
- Oracle-Datenbankprozesse, die den Code des Oracle-Datenbankservers ausführen. Hierzu zählen Serverprozesse und Hintergrundprozesse.

Wenn ein Benutzer ein Anwendungsprogramm oder ein Oracle-Tool wie SQL*Plus ausführt, erstellt die Oracle-Datenbank zur Ausführung der Benutzeranwendung einen *Benutzerprozess*. Außerdem erstellt Oracle Database einen *Serverprozess* , um die vom Benutzerprozess abgesetzten Befehle auszuführen. Darüber hinaus sind auf dem Oracle-Server eine Reihe von *Hintergrundprozessen* für eine Instanz verfügbar, die miteinander und mit dem Betriebssystem interagieren, um die Memorystrukturen zu verwalten, asynchrone I/O-Prozesse zum Schreiben von Daten auf Datenträger auszuführen und weitere Aufgaben zu erledigen.

Die Prozessstruktur hängt davon ab, mit welchem Betriebssystem und welchen Optionen die Oracle-Datenbank konfiguriert ist. Der Code für angemeldete Benutzer kann als dedizierter Server oder als Shared Server konfiguriert werden.

- Mit einem dedizierten Server wird die Datenbankanwendung für jeden Benutzer durch einen anderen Prozess (Benutzerprozess) ausgeführt, als dem Prozess, der den Oracle-Servercode (dedizierter Serverprozess) ausführt.
- Bei einem Shared Server ist kein dedizierter Serverprozess für jede Verbindung erforderlich. Ein Dispatcher leitet mehrere eingehende Netzwerksessionanforderungen an einen Pool von Shared Server-Prozessen. Alle Clientanforderungen werden von einem Shared Server-Prozess abgewickelt.

Serverprozesse

Die Oracle-Datenbank erstellt Serverprozesse, mit denen die Anforderungen der bei der Instanz angemeldeten Benutzerprozesse verarbeitet werden. Wenn die Anwendung und Oracle Database auf demselben Rechner ausgeführt werden, können der Benutzerprozess und der zugehörige Serverprozess gegebenenfalls zu einem einzigen Prozess zusammengefasst werden, um den Systemoverhead zu reduzieren. Wenn die Anwendung und die Oracle-Datenbank hingegen auf verschiedenen Rechnern ausgeführt werden, kommuniziert ein Benutzerprozess stets über einen separaten Serverprozess mit der Oracle-Datenbank.

Die für die Anwendungen der einzelnen Benutzer erstellten Serverprozesse können die folgenden Aufgaben übernehmen:

- Über die Anwendung abgesetzte SQL-Anweisungen parsen und ausführen
- Benötige Datenblöcke ermitteln und aus den Datendateien auf dem Datenträger in die gemeinsam genutzten Datenbankpuffer der SGA einlesen
- Ergebnisse in einer Form zurückgeben, in der die Anwendung die Daten verarbeiten kann

Hintergrundprozesse

Zur Maximierung der Performance und Bedienung zahlreicher Benutzer verwendet ein multiprozessfähiges Oracle-Datenbanksystem zusätzliche Oracle-Datenbankprozesse, die so genannten Hintergrundprozesse. Eine Oracle-Datenbankinstanz kann viele Hintergrundprozesse ausführen.

Um die Datenbankinstanz erfolgreich zu starten, sind folgende Hintergrundprozesse erforderlich:

- Database Writer (DBW*n*)
- Log-Writer (LGWR)
- Checkpoint (CKPT)
- System Monitor (SMON)
- Process Monitor (PMON)

Bei den folgenden Hintergrundprozessen handelt es sich um einige Beispiele für optionale Hintergrundprozesse, die gegebenenfalls gestartet werden können:

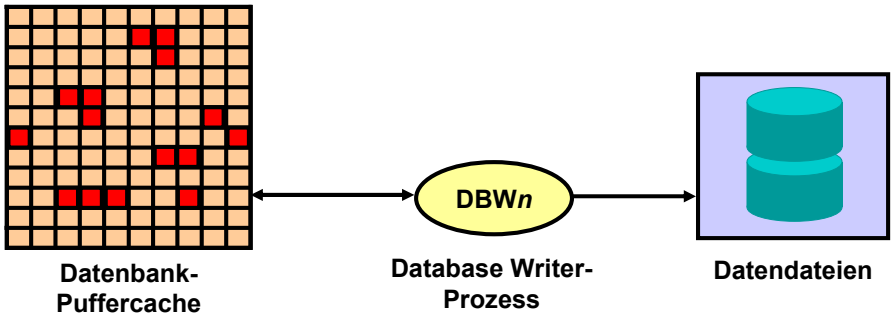
- Recoverer (RECO)
- Job Queue
- Archiver (ARC*n*)
- Queue Monitor (QMN*n*)

Andere Hintergrundprozesse finden sich in fortgeschritteneren Konfigurationen wie Real Application Clustern (RAC). Einzelheiten zu Hintergrundprozessen finden Sie in der View `V$BGPROCESS`.
Bei vielen Betriebssystemen werden Hintergrundprozesse beim Starten einer Instanz automatisch erstellt.

Database Writer

Schreibt geänderte ("Dirty") Puffer aus dem Datenbank-Puffercache auf den Datenträger:

- Asynchron während der Ausführung anderer Verarbeitungsvorgänge
- Regelmäßig, um den Checkpoint fortzuschreiben

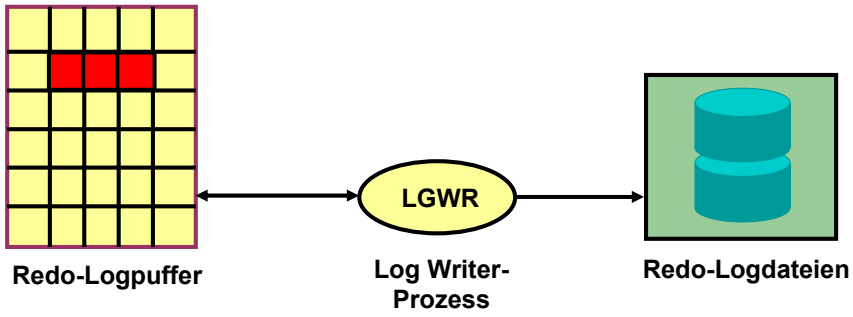


Der Database Writer-Prozess (DBW*n*) schreibt den Inhalt von Puffern in Datendateien. Geänderte Puffer aus dem Datenbank-Puffercache (so genannte "Dirty-Puffer") werden auf einen Datenträger geschrieben. Für die meisten Systeme reicht ein einzelner Database Writer-Prozess (DBW0) aus. Wenn das System jedoch große Datenmengen bearbeitet, können Sie zur Erhöhung der Schreibleistung zusätzliche Prozesse (DBW1 bis DBW9 und DBWa bis DBWz) konfigurieren. Diese zusätzlichen DBW*n*-Prozesse sind für Uniprocessor-Systeme nicht sinnvoll.

Wird ein Puffer im Datenbank-Puffercache geändert, wird er als "dirty" markiert und zur LRUW-Liste der "Dirty" Puffer hinzugefügt. Diese Liste ist nach SCNs sortiert. Sie entspricht somit der Reihenfolge der entsprechenden Redo-Daten für diese geänderten Puffer, die in den Redo-Logs gespeichert werden. Fällt die Anzahl der im Puffercache verfügbaren Puffer unter einen internen Schwellenwert, und die Serverprozesse haben Probleme, einen verfügbaren Puffer zu finden, schreibt DBW*n* "Dirty" Puffer in der Reihenfolge in die Datendateien, in der sie geändert wurden. Hierbei folgt DBWn der Reihenfolge in der LRUW-Liste.

Log Writer

- Schreibt Redo-Logpuffer in eine Redo-Logdatei auf dem Datenträger
- Der Schreibvorgang erfolgt in den folgenden Fällen:
  - Ein Prozess schreibt eine Transaktion fest.
  - Der Redo-Logpuffer ist zu einem Drittel mit Daten gefüllt.
  - Ein DBW*n*-Prozess will geänderte Puffer auf Datenträger schreiben.



Der Log Writer-Prozess (LGWR) ist für die Verwaltung des Redo-Logpuffers verantwortlich. Er schreibt die Redo-Logpuffereinträge in eine Redo-Logdatei auf dem Datenträger. LGWR schreibt alle Redo-Einträge in eine Logdatei, die seit dem letzten Schreibvorgang in den Puffer kopiert wurden.

Der Redo-Logpuffer ist ein zyklischer Puffer. Wenn der LGWR-Prozess Redo-Einträge aus dem Redo-Logpuffer in eine Redo-Logdatei auf dem Datenträger schreibt, können die so gespeicherten Einträge im Redo-Logpuffer anschließend durch neue Einträge überschrieben werden. LGWR schreibt normalerweise so schnell, dass im Puffer immer Speicherplatz für neue Einträge verfügbar ist, und zwar auch dann, wenn zahlreiche Zugriffe auf das Redo-Log erfolgen.

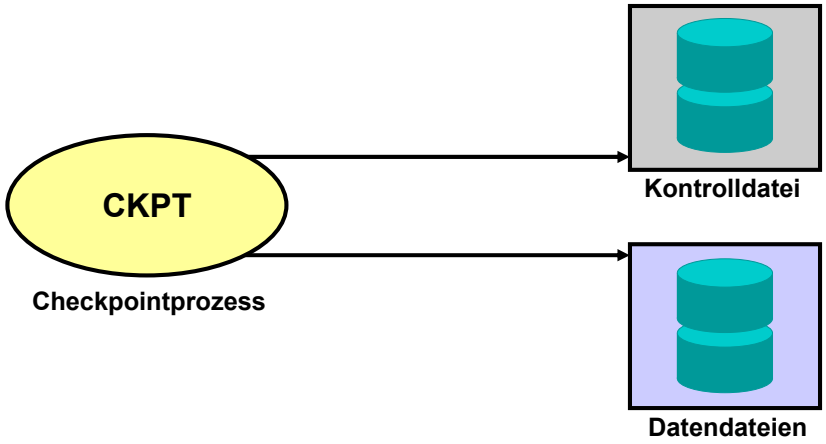
LGWR schreibt einen zusammenhängenden Teil des Puffers auf den Datenträger. Der Schreibvorgang erfolgt in den folgenden Fällen:

- Ein Benutzerprozess schreibt eine Transaktion fest.
- Der Redo-Logpuffer ist zu einem Drittel mit Daten gefüllt.
- Ein DBW*n*-Prozess will geänderte Puffer auf den Datenträger schreiben.

Checkpoint

Zeichnet Checkpoint-Informationen auf:

- In der Kontrolldatei
- Im Header jeder Datendatei



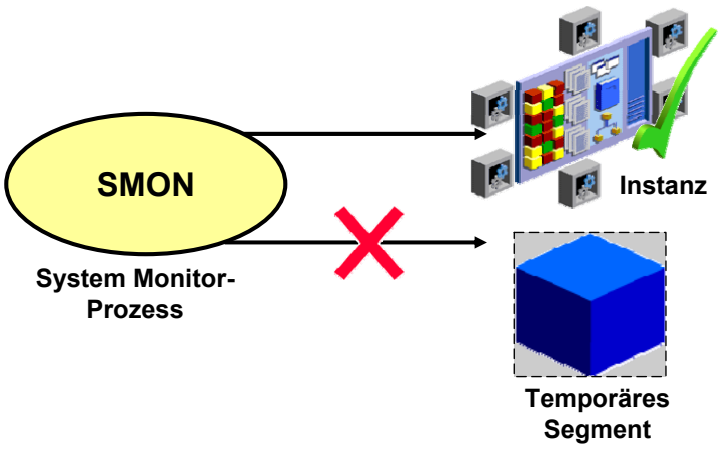
Ein Checkpoint ist eine Datenstruktur, die eine System Change Number (SCN) im Redo Thread einer Datenbank definiert. Checkpoints werden in der Kontrolldatei und im Header jeder Datendatei aufgezeichnet und sind ein wichtiges Element beim Recovery.

An den Checkpoints aktualisiert die Oracle-Datenbank die Header aller Datendateien, sodass die Details der Checkpoints aufgezeichnet werden. Dieser Vorgang wird vom CKPT-Prozess ausgeführt. Der CKPT-Prozess schreibt jedoch keine Blöcke auf den Datenträger. Diese Aufgabe wird stets von DBW*n* übernommen. Durch die in den Dateiheadern aufgezeichneten SCNs wird gewährleistet, dass alle vor der betreffenden SCN für Datenbankblöcke durchgeführten Änderungen auf Datenträger geschrieben wurden.

Die vom Monitor `SYSTEM_STATISTICS` in Oracle Enterprise Manager angezeigten statistischen DBWR-Checkpoints geben die Anzahl der abgeschlossenen Checkpoint-Anforderungen an.

System Monitor

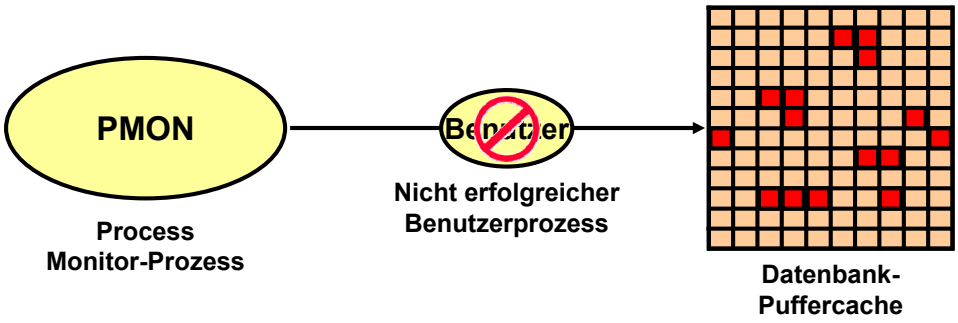
- Führt beim Hochfahren der Instanz ein Recovery durch
- Löscht nicht verwendete temporäre Segmente



Falls erforderlich, führt der System Monitor-Prozess (SMON) beim Hochfahren der Instanz ein Recovery aus. Eine weitere Aufgabe von SMON ist das Löschen temporärer Segmente, die nicht mehr verwendet werden. SMON stellt abgeschlossene Transaktionen, die beim Recovery der Instanz aufgrund von Dateilese- oder Offlinefehlern übersprungen wurden, wieder her, sobald der Tablespace oder die Datei erneut online gesetzt wird. SMON prüft regelmäßig, ob der Bereinigungsprozess erforderlich ist. SMON kann von anderen Prozessen aufgerufen werden, falls erforderlich.

Process Monitor

- Führt beim Ausfall eines Benutzerprozesses ein Prozess-Recovery durch
  - Leert den Datenbank-Puffercache
  - Gibt vom Benutzerprozess verwendete Ressourcen frei
- Überwacht Sessions auf Zeitüberschreitung inaktiver Sessions
- Registriert Datenbankservices dynamisch bei Listenern



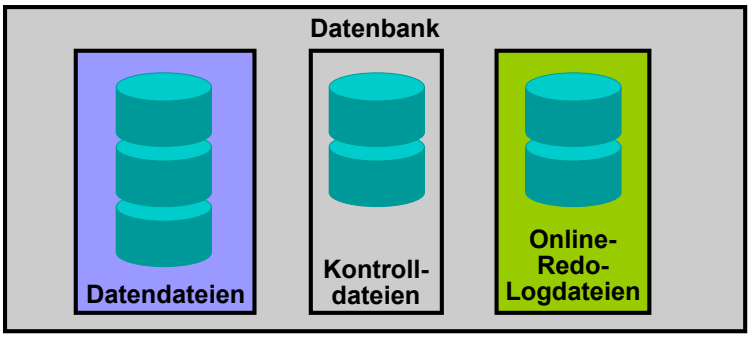
Der Process Monitor-Prozess (PMON) führt ein Prozess-Recovery durch, wenn ein Benutzerprozess nicht ausgeführt werden kann. PMON übernimmt das Löschen des Datenbank-Puffercaches und gibt vom Prozess verwendete Ressourcen frei. PMON setzt beispielsweise den Status der Tabelle für die aktive Transaktion zurück, hebt Sperren auf und entfernt die Prozess-ID aus der Liste der aktiven Prozesse.

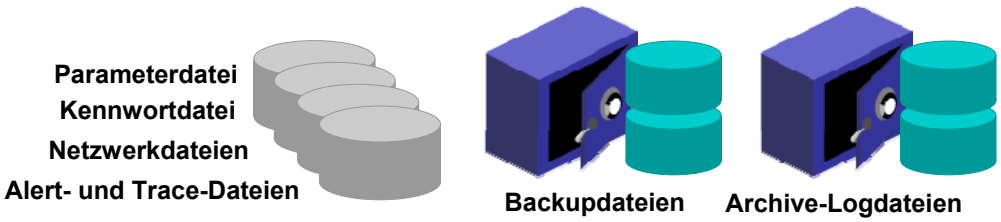
PMON prüft regelmäßig den Status von Dispatcher- und Serverprozessen und startet gestoppte Prozesse neu. (Absichtlich von der Oracle-Datenbank beendete Prozesse werden nicht neu gestartet.) Der Prozess registriert beim Network Listener auch Informationen über die Instanz- und Dispatcher-Prozesse.

Wie SMON prüft auch PMON regelmäßig, ob seine Ausführung erforderlich ist. PMON kann ebenfalls von anderen Prozessen aufgerufen werden, wenn seine Ausführung als notwendig erkannt wird.

Storage-Architektur von Oracle-Datenbanken

DB-Strukturen
- **Memory**
- **Prozess**
- → **Storage**





Die Dateien in Oracle-Datenbanken lassen sich den folgenden Kategorien zuordnen:

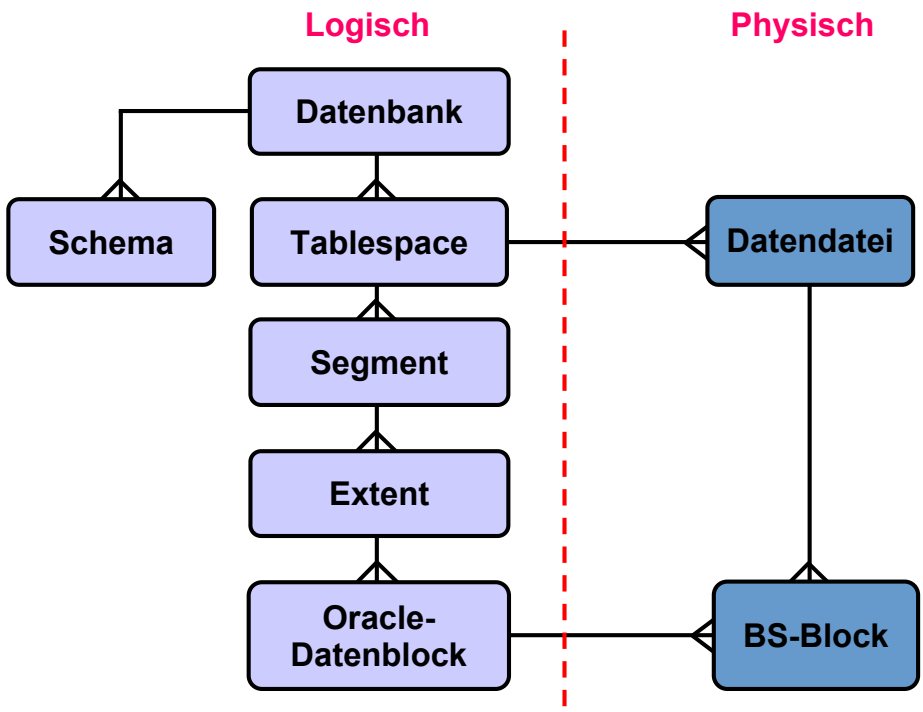
- **Kontrolldateien:** Enthalten Daten zur Datenbank selbst (das heißt, die Informationen zur physischen Datenbankstruktur). Diese Dateien sind von fundamentaler Bedeutung für die Datenbank. Ohne diese Dateien können Sie keine Datendateien öffnen, um auf die Daten in der Datenbank zuzugreifen.
- **Datendateien:** Enthalten die Benutzer- oder Anwendungsdaten der Datenbank sowie Metadaten und das Data Dictionary
- **Online-Redo-Logdateien:** Ermöglichen das Instanz-Recovery der Datenbank. Wenn der Datenbankserver ausfällt und keine Datendateien verloren gehen, kann die Datenbank mit den Informationen in diesen Dateien aus der Instanz wiederhergestellt werden.

Weitere für die erfolgreiche Ausführung der Datenbank wichtige Dateien:

- **Backupdateien:** Dienen zur Durchführung eines Datenbank-Recoverys. Eine Backupdatei wird in der Regel zurückgeschrieben, wenn die ursprüngliche Datei durch einen Mediafehler oder Benutzerfehler beschädigt oder gelöscht wurde.
- **Archive-Logdateien:** Enthalten die Historie der von der Instanz generierten Datenänderungen (Redo). Mithilfe dieser Dateien und eines Datenbankbackups können Sie eine verlorene Datendatei wiederherstellen. Archive-Logdateien ermöglichen also das Recovery zurückgeschriebener Datendateien.
- **Parameterdatei:** Enthält die Konfigurationseinstellung der Instanz beim Hochfahren
- **Kennwortdatei:** Ermöglicht `sysdba/sysoper/sysasm`, eine Remote-Verbindung mit der Datenbank herzustellen und Verwaltungsaufgaben auszuführen

- **Netzwerkdateien:** Dienen zum Starten des Datenbank-Listeners und speichern die für Benutzerverbindungen erforderlichen Informationen
- **Trace-Dateien:** Jeder Server- und Hintergrundprozess kann in eine zugehörige Trace-Datei schreiben. Erkennt ein Prozess einen internen Fehler, werden Informationen zu diesem Fehler in die entsprechende Trace-Datei geschrieben. Einige der Informationen in einer Trace-Datei sind für den Datenbankadministrator relevant, andere für den Oracle-Support.
- **Alert-Logdateien:** Hierbei handelt es sich um spezielle Trace-Einträge. Im Alert-Log einer Datenbank werden Meldungen und Fehler chronologisch protokolliert. Jede Instanz verfügt über eine Alert-Logdatei. Oracle empfiehlt eine regelmäßige Prüfung dieser Alert-Logdatei.

Logische und physische Datenbankstrukturen



Eine Oracle-Datenbank umfasst logische und physische Storage-Strukturen.

Tablespaces

Datenbanken sind in logische Storage-Einheiten unterteilt, die als Tablespaces bezeichnet werden. Sie dienen zur Gruppierung zusammenhängender logischer Strukturen. So werden beispielsweise in Tablespaces häufig alle Objekte einer Anwendung gruppiert, um bestimmte administrative Vorgänge zu vereinfachen. Sie können einen Tablespace für Anwendungsdaten und einen weiteren Tablespace für Anwendungsindizes einrichten.

Datenbanken, Tablespaces und Datendateien

Die Folie veranschaulicht die Beziehung zwischen Datenbanken, Tablespaces und Datendateien. Jede Datenbank ist logisch in einen oder mehrere Tablespaces unterteilt. Für jeden Tablespace wird explizit mindestens eine Datendatei erstellt, um die Daten aller logischen Strukturen in einem Tablespace physisch zu speichern. Handelt es sich bei dem Tablespace um einen `TEMPORARY`-Tablespace, wird anstelle einer Datendatei eine temporäre Datei erstellt.

Schemas

Ein Schema ist eine Zusammenstellung von Datenbankobjekten, deren Eigentümer ein Datenbankbenutzer ist. Schemaobjekte sind die logischen Strukturen, die sich direkt auf die Daten der Datenbank beziehen. Zu den Schemaobjekten zählen Strukturen wie Tabellen, Views, Sequenzen, Stored Procedures, Synonyme, Indizes, Cluster und Datenbanklinks. Im Allgemeinen gehören zu den Schemaobjekten alle Objekte, die eine Anwendung in der Datenbank erstellen kann.

Datenblöcke

Auf der niedrigsten Granularitätsebene werden die Daten einer Oracle-Datenbank in Datenblöcken gespeichert. Ein Datenblock entspricht einer bestimmten Bytezahl von physischem Datenbank-Speicherplatz auf dem Datenträger. Die Größe des Datenblockes wird bei der Erstellung der jeweiligen Tablespaces festgelegt. Eine Datenbank weist freien Datenbank-Speicherplatz in Oracle-Datenblöcken zu und nutzt ihn.

Extents

Die nächste Ebene eines logischen Datenbank-Speicherplatzes ist ein Extent. Bei einem Extent handelt es sich um eine bestimmte Anzahl zusammenhängender und gemeinsam zugewiesener Oracle-Datenblöcke, in denen eine bestimmte Art von Informationen gespeichert wird.

Segmente

Die nächsthöhere Ebene des logischen Datenbankspeichers über den Extents wird als Segment bezeichnet. Ein Segment ist eine Gruppe von Extents, die einer bestimmten logischen Struktur zugewiesen wurden. Zu den verschiedenen Segmenttypen zählen beispielsweise:

- **Datensegmente:** Mit Ausnahme von externen Tabellen, globalen temporären Tabellen und partitionierten Tabellen, in denen jede Tabelle mehrere Segmente enthalten kann, enthält jede nicht geclusterte und nicht indestrukturierte Tabelle ein Datensegment. Sämtliche Daten der Tabelle werden in den Extents des Datensegments gespeichert. Bei partitionierten Tabellen hat jede Partition ein Datensegment. Jeder Cluster verfügt über ein Datensegment. Die Daten der einzelnen Tabellen im Cluster werden im Datensegment des Clusters gespeichert.
- **Indexsegmente:** Jeder Index hat ein Indexsegment, in dem alle Daten gespeichert werden. Bei partitionierten Indizes hat jede Partition ein Indexsegment.
- **Undo-Segmente:** Pro Datenbankinstanz wird jeweils ein Tablespace vom Typ `UNDO` erstellt. Dieser Tablespace enthält zahlreiche Undo-Segmente, in denen *Undo*-Informationen temporär gespeichert werden. Anhand der Informationen in einem Undo-Segment können lesekonsistente Datenbankinformationen generiert und Transaktionen zurückgerollt werden, die während eines Datenbank-Recoverys nicht festgeschrieben wurden.
- **Temporäre Segmente:** Wenn für die Ausführung einer SQL-Anweisung ein temporärer Arbeitsbereich benötigt wird, erstellt die Oracle-Datenbank temporäre Segmente. Nach Ausführung der Anweisung werden die Extents des temporären Segments zur künftigen Verwendung an die Instanz zurückgegeben. Geben Sie einen Default Temporary Tablespace für jeden Benutzer oder einen Default Temporary Tablespace zur datenbankweiten Verwendung an.

Oracle Database weist Speicherplatz dynamisch zu. Wenn die vorhandenen Extents eines Segments voll sind, werden weitere Extents hinzugefügt. Da Extents nach Bedarf zugewiesen werden, sind die Extents eines Segments auf dem Datenträger möglicherweise nicht zusammenhängend.

SQL-Anweisungen verarbeiten

- Bei einer Instanz anmelden mit:
  - dem Benutzerprozess
  - dem Serverprozess
- Die verwendeten Oracle-Serverkomponenten hängen von der Art der SQL-Anweisung ab:
  - Abfragen geben Zeilen zurück.
  - DML-(Data Manipulation Language-)Anweisungen protokollieren Änderungen.
  - Commit stellt das Recovery von Transaktionen sicher.
- Einige Oracle-Serverkomponenten sind nicht an der Verarbeitung von SQL-Anweisungen beteiligt.

ORACLE

Nicht alle Komponenten einer Oracle-Instanz werden zur Verarbeitung von SQL-Anweisungen verwendet. Benutzer melden sich mithilfe von Benutzer- und Serverprozessen bei einer Oracle-Instanz an. Diese Prozesse sind nicht Bestandteil der Oracle-Instanz, werden jedoch für die Verarbeitung von SQL-Anweisungen benötigt.

Einige der Hintergrundprozesse, SGA-Strukturen und Datenbankdateien werden für die Verarbeitung von SQL-Anweisungen eingesetzt. In Abhängigkeit von der Art der SQL-Anweisung werden unterschiedliche Komponenten verwendet:

- Abfragen erfordern zusätzliche Verarbeitung, um Zeilen an den Benutzer zurückzugeben.
- DML-Anweisungen erfordern zusätzliche Verarbeitungsschritte, damit sie die an den Daten vorgenommenen Änderungen protokollieren.
- Die Commit-Verarbeitung stellt sicher, dass ein Recovery der in einer Transaktion geänderten Daten durchgeführt werden kann.

Einige erforderliche Hintergrundprozesse sind nicht direkt an der Verarbeitung von SQL-Anweisungen beteiligt, sondern werden zur Verbesserung der Performance und zur Durchführung eines Datenbank-Recoverys verwendet. Beispielsweise stellt der optionale Archiver-Hintergrundprozess (ARCARC*n*) sicher, dass das Recovery einer Produktionsdatenbank möglich ist.

Abfragen verarbeiten

- Parsen:
  - Nach identischer Anweisung suchen
  - Syntax, Objektnamen und Berechtigungen prüfen
  - Während des Parsens verwendete Objekte sperren
  - Ausführungsplan erstellen und speichern
- Ausführen: Ausgewählte Zeilen identifizieren
- Abrufen: Zeilen an den Benutzerprozess zurückgeben

ORACLE

Abfragen unterscheiden sich von anderen Typen von SQL-Anweisungen, weil sie als Ergebnis Daten zurückgeben. Während andere Anweisungen einfach nur Erfolgs- bzw. Fehlermeldungen zurückgeben, können Abfragen eine einzige Zeile oder auch Tausende Zeilen zurückgeben.

Die Abfrageverarbeitung umfasst im Wesentlichen drei Phasen:

- Parsen
- Ausführen
- Abrufen

In der *Parse*-Phase wird die SQL-Anweisung vom Benutzerprozess an den Serverprozess übergeben und eine geparste Version der SQL-Anweisung in einen Shared SQL-Bereich geladen.

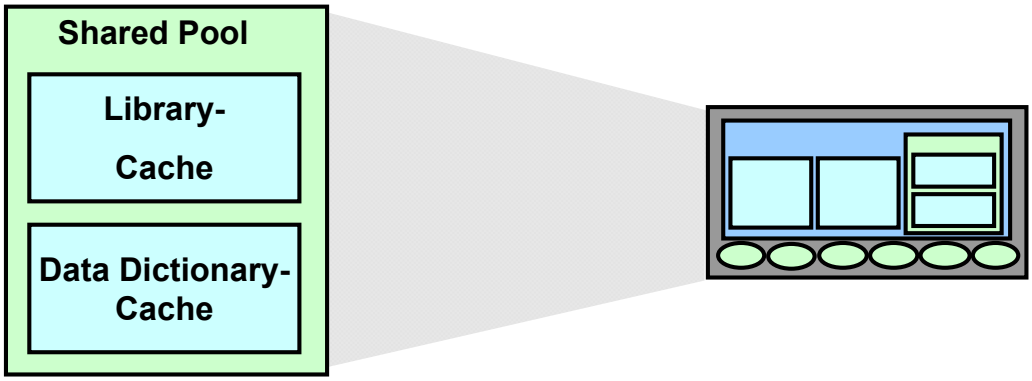
Beim Parsen führt der Serverprozess folgende Funktionen aus:

- Er sucht im Shared Pool nach einer vorhandenen Kopie der SQL-Anweisung.
- Er validiert die SQL-Anweisung durch Prüfen der Syntax.
- Er führt Suchläufe im Data Dictionary durch, um Tabellen- und Spaltendefinitionen zu validieren.

In der Ausführungsphase wird die Anweisung mithilfe des besten Optimizer-Ansatzes ausgeführt. In der Abrufphase werden die Zeilen an den Benutzer zurückgegeben.

Shared Pool

- Der Library-Cache enthält den Text der SQL-Anweisung, den geparsten Code und den Ausführungsplan.
- Der Data Dictionary-Cache enthält Definitionen und Berechtigungen für Tabellen, Spalten und andere Objekte.
- Die Größe des Shared Pools wird mit `SHARED_POOL_SIZE` festgelegt.



Während der Parse-Phase verwendet der Serverprozess den als Shared Pool bezeichneten Bereich der SGA zum Kompilieren der SQL-Anweisung. Der Shared Pool verfügt über zwei Hauptkomponenten:

- Library-Cache
- Data Dictionary-Cache

Library-Cache

Der Library-Cache speichert Informationen über die zuletzt verwendeten SQL-Anweisungen. Dazu wird eine Memorystruktur verwendet, die als Shared SQL-Bereich bezeichnet wird. Der Shared SQL-Bereich enthält:

- den Text der SQL-Anweisung
- den Parse-Baum: Eine kompilierte Version der Anweisung
- den Ausführungsplan: Die Schritte, die beim Ausführen der Anweisung durchgeführt werden müssen

Der Optimizer ist die Funktion im Oracle-Server, die den optimalen Ausführungsplan ermittelt.

Wenn eine SQL-Anweisung erneut ausgeführt wird und bereits ein Shared SQL-Bereich den Ausführungsplan für die Anweisung enthält, muss der Serverprozess die Anweisung nicht parsen. Der Library-Cache steigert die Performance von Anwendungen, die SQL-Anweisungen wiederverwenden, da er die Parse-Dauer und den Memorybedarf reduziert. Wird die SQL-Anweisung nicht wiederverwendet, wird sie schließlich als veraltet aus dem Library-Cache gelöscht.

Data Dictionary-Cache

Im Data Dictionary-Cache, der auch als Dictionary- oder Row-Cache bezeichnet wird, sind die in der Datenbank zuletzt verwendeten Definitionen zusammengestellt. Er enthält Informationen zu Datenbankdateien, Tabellen, Indizes, Spalten, Benutzern, Berechtigungen und anderen Datenbankobjekte.

Während der Parse-Phase sucht der Serverprozess im Dictionary-Cache nach Informationen, um die in der SQL-Anweisung angegebenen Objektnamen aufzulösen und die Zugriffsberechtigungen zu validieren. Falls erforderlich, bewirkt der Serverprozess, dass diese Informationen aus den Datendateien geladen werden.

Größe des Shared Pools festlegen

Die Größe des Shared Pools wird mit dem Initialisierungsparameter `SHARED_POOL_SIZE` festgelegt.

Datenbank-Puffercache

- Im Datenbank-Puffercache werden die zuletzt verwendeten Blöcke gespeichert.
- Die Puffergröße basiert auf `DB_BLOCK_SIZE`.
- Die Anzahl der Puffer wird mit `DB_BLOCK_BUFFERS` festgelegt.

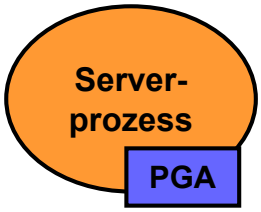
Wenn eine Abfrage verarbeitet wird, sucht der Serverprozess benötigte Blöcke im Datenbank-Puffercache. Ist ein Block nicht im Datenbank-Puffercache enthalten, liest der Serverprozess den Block aus der Datendatei und legt eine Kopie im Puffercache ab. Da nachfolgende Anforderungen den Block bereits im Memory finden können, sind unter Umständen keine physischen Lesevorgänge mehr erforderlich. Der Oracle-Server verwendet einen LRU-(Least Recently Used-)Algorithmus, um Puffer zu leeren, auf die seit einiger Zeit nicht mehr zugegriffen wurde. Dadurch wird im Puffercache Platz für neue Blöcke geschaffen.

Größe des Datenbank-Puffercaches festlegen

Die Größe jedes Puffers im Puffercache entspricht der Größe eines Oracle-Blocks und wird mit dem Parameter `DB_BLOCK_SIZE` festgelegt. Die Anzahl der Puffer entspricht dem Wert des Parameters `DB_BLOCK_BUFFERS`.

Program Global Area (PGA)

- Wird nicht gemeinsam genutzt
- Ist nur durch den Serverprozess beschreibbar
- Enthält:
  - Sortierbereich
  - Sessioninformationen
  - Cursorstatus
  - Stack-Speicherplatz



ORACLE

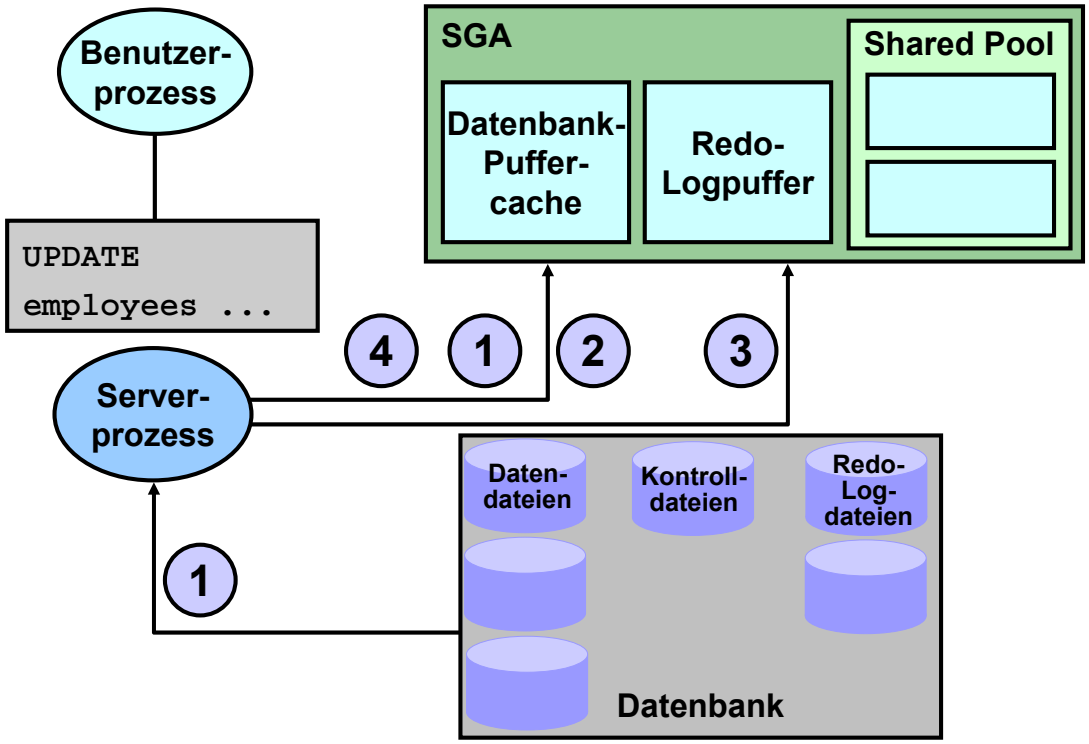
Die PGA (Program Global Area) ist ein Bereich im Memory, der Daten und Steuerinformationen zu einem Serverprozess enthält. Die PGA ist ein nicht gemeinsam genutzter Memorybereich, den Oracle beim Starten eines Serverprozesses einrichtet. Der Zugriff auf die PGA ist auf diesen Serverprozess beschränkt. Nur der Oracle-Servercode, der für diesen Serverprozess agiert, liest die PGA oder schreibt darin. Das PGA-Memory, das von jedem einer Oracle-Instanz zugeordneten Serverprozess zugewiesen wird, wird als das von der Instanz zugewiesene aggregierte PGA-Memory bezeichnet.

Bei einer dedizierten Serverkonfiguration enthält die PGA des Servers folgende Komponenten:

- **Sortierbereich:** Wird für Sortiervorgänge verwendet, die für das Verarbeiten der SQL-Anweisung erforderlich sein können
- **Sessioninformationen:** Umfassen Benutzerberechtigungen und Performancestatistiken für die Session
- **Cursorstatus:** Gibt die Verarbeitungsphase der SQL-Anweisungen an, die aktuell von der Session verwendet werden
- **Stack-Speicherplatz:** Enthält weitere Sessionvariablen

Die PGA wird zugewiesen, wenn ein Prozess erstellt wird, und freigegeben, wenn der Prozess beendet wird.

DML-Anweisungen verarbeiten



Die Verarbeitung einer DML-(Data Manipulation Language-)Anweisung erfolgt in nur zwei Phasen:

- Die Parse-Phase ist mit dem Parse-Vorgang bei der Verarbeitung von Abfragen identisch.
- Die Ausführungsphase erfordert zusätzlichen Verarbeitungsaufwand, um Daten zu ändern.

DML-Ausführungsphase

Eine DML-Anweisung wird wie folgt ausgeführt:

- Wenn sich die Daten- und Rollback-Blöcke noch nicht im Puffercache befinden, liest der Serverprozess sie aus den Datendateien in den Puffercache.
- Der Serverprozess setzt Sperren für die zu ändernden Zeilen.
- Der Serverprozess zeichnet die Änderungen, die in den Rollback- und Datenblöcken durchgeführt werden sollen, im Redo-Logpuffer auf.
- Die Rollback-Blockänderungen zeichnen die Werte der Daten vor der Änderung auf. Der Rollback-Block speichert das "Before Image" der Daten, damit die DML-Anweisungen gegebenenfalls zurückgerollt werden können.
- Die Datenblockänderungen zeichnen die neuen Werte der Daten auf.

Der Serverprozess zeichnet das "Before Image" im Rollback-Block auf und aktualisiert den Datenblock. Diese beiden Änderungen erfolgen im Datenbank-Puffercache. Geänderte Blöcke im Puffercache werden als "Dirty" Puffer gekennzeichnet (Puffer, die nicht mit den entsprechenden Blöcken auf dem Datenträger identisch sind).

Die Verarbeitung eines `DELETE`- oder `INSERT`-Befehls umfasst ähnliche Schritte. Das "Before Image" für ein `DELETE` enthält die Spaltenwerte in der gelöschten Zeile und das "Before Image" für ein `INSERT` Informationen über den Speicherort der Zeile.

Da die in den Blöcken durchgeführten Änderungen nur in Memorystrukturen aufgezeichnet und nicht sofort auf Datenträger geschrieben werden, kann es bei einem Rechnerfehler, bei dem die SGA verloren geht, auch zum Verlust dieser Änderungen kommen.

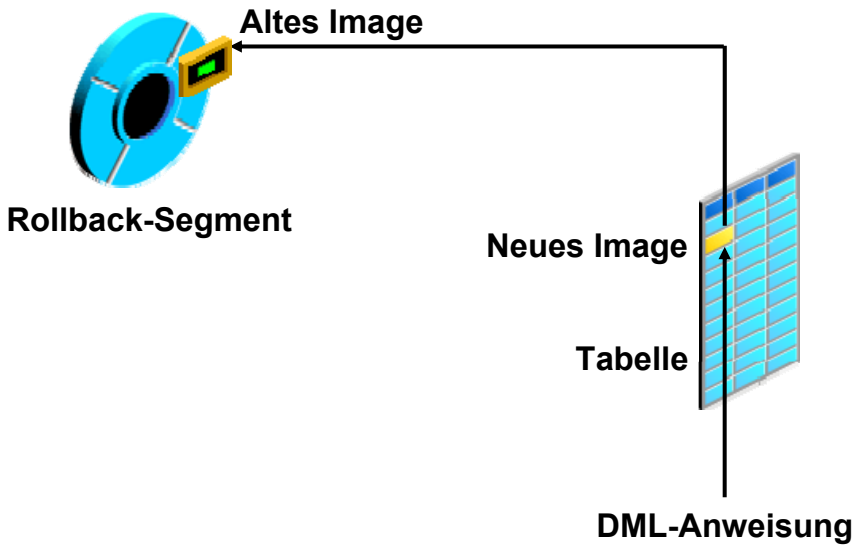
Redo-Logpuffer

- Größe wird von `LOG_BUFFER` definiert.
- Zeichnet durch die Instanz durchgeführte Änderungen auf
- Wird sequenziell verwendet
- Ist ein zyklischer Puffer

Der Serverprozess zeichnet den größten Teil der Änderungen an Datendateiblöcken im Redo-Logpuffer auf, der Teil der SGA ist. Der Redo-Logpuffer besitzt folgende Eigenschaften:

- Seine Größe in Byte wird mit dem Parameter `LOG_BUFFER` festgelegt.
- Er zeichnet den geänderten Block, die Position der Änderung und den neuen Wert in einem Redo-Eintrag auf. Der Redo-Eintrag enthält keine Angabe zum Typ des Blocks, der geändert wird. Es wird nur aufgezeichnet, welche Byte im Block geändert wurden.
- Der Redo-Logpuffer wird sequenziell verwendet. Die von einer Transaktion vorgenommenen Änderungen werden eventuell abwechselnd mit den von anderen Transaktionen durchgeführten Änderungen eingetragen.
- Der Redo-Logpuffer ist ein zyklischer Puffer, der wiederverwendet wird, nachdem er vollgeschrieben wurde. Die Wiederverwendung ist jedoch erst möglich, wenn alle alten Redo-Einträge in den Redo-Logdateien aufgezeichnet wurden.

Rollback-Segmente



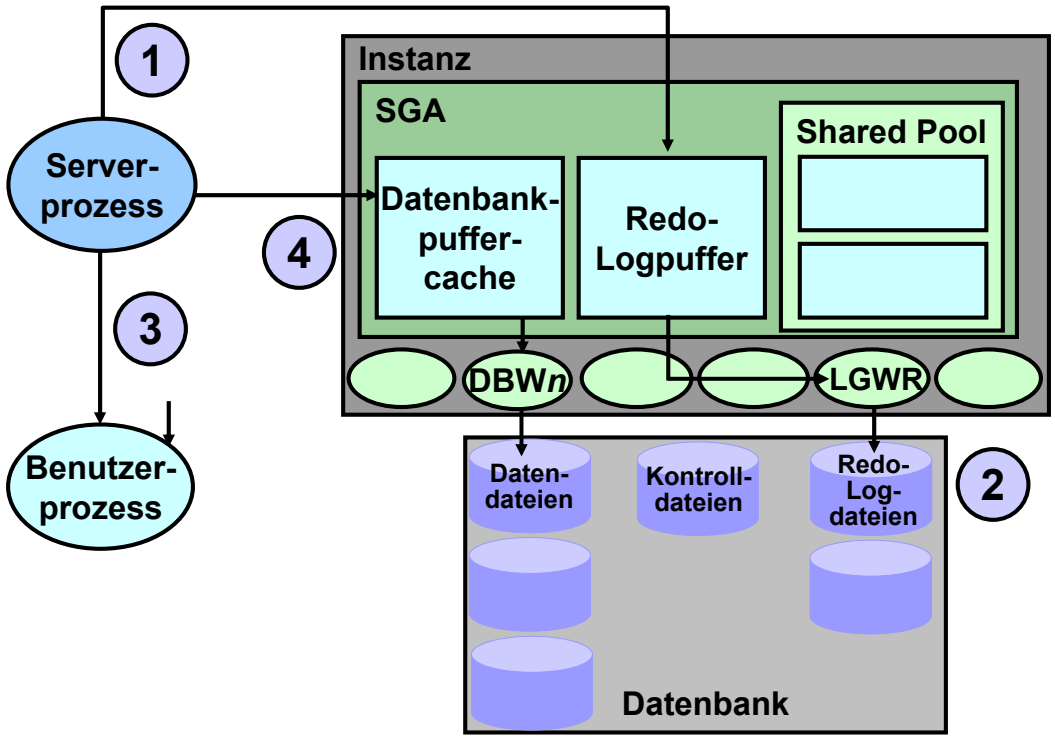
Bevor Daten geändert werden, speichert der Serverprozess zunächst den alten Datenwert in einem Rollback-Segment. Dieses "Before Image" dient dazu:

- Änderungen rückgängig zu machen, falls die Transaktion zurückgerollt wird
- Lesekonsistenz bereitzustellen, indem sichergestellt wird, dass für andere Transaktionen keine nicht festgeschriebenen Änderungen sichtbar sind, die von der DML-Anweisung ausgeführt wurden
- Datenbank bei Fehlern in einem konsistenten Zustand wiederherzustellen

Rollback-Segmente befinden sich wie Tabellen und Indizes in Datendateien. Rollback-Blöcke werden, falls erforderlich, in den Datenbank-Puffercache gelesen. Rollback-Segmente werden vom Datenbankadministrator erstellt.

Änderungen an Rollback-Segmenten werden im Redo-Logpuffer aufgezeichnet.

COMMIT-Verarbeitung



ORACLE

Der Oracle-Server verwendet einen Fast COMMIT-Mechanismus, der das Recovery festgeschriebener Änderungen bei einem Instanzfehler sicherstellt.

System Change Number

Der Oracle-Server weist Transaktionen, die festgeschrieben werden, eine SCN (System Change Number) zu. Die SCN wird in gleichmäßigen Schritten erhöht und ist in der Datenbank eindeutig. Der Oracle-Server verwendet die SCN als internen Zeitstempel, um Daten zu synchronisieren und beim Abrufen von Daten aus den Datendateien Lesekonsistenz zu gewährleisten. Mithilfe der SCN kann der Oracle-Server Konsistenzprüfungen durchführen, ohne auf Datum und Uhrzeit des Betriebssystems angewiesen zu sein.

Schritte der COMMIT-Verarbeitung

Nach dem Absetzen eines COMMIT-Befehls werden folgende Schritte ausgeführt:

1. Der Serverprozess schreibt einen Commit-Datensatz mit der zugehörigen SCN in den Redo-Logpuffer.
2. LGWR schreibt alle Redo-Logpuffereinträge bis einschließlich des Commit-Datensatzes fortlaufend in die Redo-Logdateien Danach ist gewährleistet, dass die Änderungen auch bei Instanzausfällen nicht verloren gehen.

3. Der Benutzer wird darüber informiert, dass der COMMIT-Prozess abgeschlossen ist.
4. Der Serverprozess zeichnet die Informationen auf, die angeben, dass die Transaktion abgeschlossen ist und die Ressourcensperren freigegeben werden können.

Das Wegschreiben der "Dirty" Puffer in die Datendatei wird von DBW0 unabhängig durchgeführt und kann vor oder nach dem Commit stattfinden.

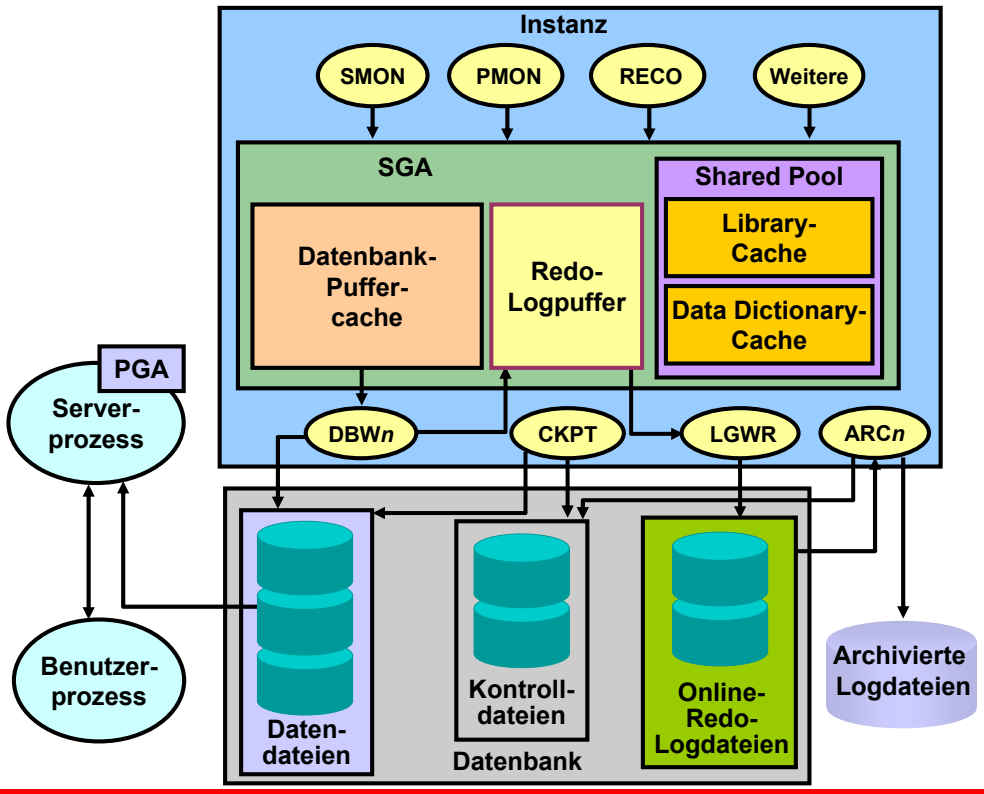
Vorteile des Fast COMMIT-Prozesses

Mit dem Fast COMMIT-Mechanismus wird ein vollständiges Recovery der Daten sichergestellt, da die Änderungen nicht in die Datendateien, sondern in den Redo-Logpuffer geschrieben werden. Vorteile von Fast Commit:

- Sequenzielle Schreibvorgänge in die Logdateien sind schneller als Schreibvorgänge in die verschiedenen Blöcke der Datendatei.
- In die Logdateien werden nur die Informationen geschrieben, die erforderlich sind, um die Änderungen aufzuzeichnen. In die Datendateien hingegen werden ganze Datenblöcke geschrieben.
- Wenn mehrere Transaktionen gleichzeitig Commit-Vorgänge anfordern, verbindet die Instanz Redo-Logaufzeichnungen nach dem Huckepackprinzip zu einem einzigen Schreibvorgang.
- Für den Fast Commit-Prozess ist ein synchroner Schreibvorgang pro Transaktion ausreichend, wenn der Redo-Logpuffer nicht besonders voll ist. Mit dem Huckepackprinzip kommt der Prozess vielleicht sogar mit weniger als einem synchronen Schreibvorgang pro Transaktion aus.
- Da die Änderungen vor dem COMMIT aus dem Redo-Logpuffer weggeschrieben werden können, wirkt sich der Umfang der Transaktion nicht auf die Dauer des konkreten COMMIT-Vorgangs aus.

Hinweis: Das Rollback einer Transaktion nicht dazu, dass LGWR Daten auf Datenträger schreibt. Der Oracle-Server rollt nicht festgeschriebene Änderungen im Rahmen eines Recoverys nach Ausfällen immer zurück. Kommt es zu einem Ausfall, nachdem ein Rollback durchgeführt und bevor die Rollback-Einträge auf dem Datenträger gespeichert wurden, ist durch den fehlenden Commit-Datensatz sichergestellt, dass die von der Transaktion vorgenommenen Änderungen zurückgerollt werden.

Architektur von Oracle Database – Überblick



Eine Oracle-Datenbank besteht aus einer Instanz und der zugehörigen Datenbank:

- Eine Instanz umfasst die SGA und die Hintergrundprozesse:
  - **SGA:** Datenbank-Puffercache, Redo-Logpuffer, Shared Pool usw.
  - **Hintergrundprozesse:** SMON, PMON, DBW*n*, CKPT, LGWR usw.
- Eine Datenbank umfasst die folgenden Storage-Strukturen:
  - **Logische:** Tablespaces, Schemas, Segmente, Extents und Oracle-Blöcke
  - **Physische:** Datendateien, Kontrolldateien, Redo-Logdateien

Wenn ein Benutzer über eine Anwendung auf die Oracle-Datenbank zugreift, übernimmt ein Serverprozess die Kommunikation mit der Instanz für den Benutzerprozess.

Zusammenfassung

In diesem Anhang haben Sie Folgendes gelernt:

- Die wichtigsten Architekturkomponenten der Datenbank nennen
- Hintergrundprozesse beschreiben
- Memorystrukturen beschreiben
- Beziehung zwischen logischen und physischen Storage-Strukturen erläutern

ORACLE

Unterstützung regulärer Ausdrücke

Ziele

Nach Ablauf dieses Anhangs haben Sie folgende Ziele erreicht:

- Vorteile regulärer Ausdrücke nennen
- Zeichenfolgen mit regulären Ausdrücken suchen, vergleichen und ersetzen

ORACLE

In diesem Anhang lernen Sie die Features zur Unterstützung regulärer Ausdrücke kennen. Die Unterstützung regulärer Ausdrücke ist in SQL und PL/SQL verfügbar.

Was sind reguläre Ausdrücke?

- Mit regulären Ausdrücke mithilfe von Standardsyntax-konventionen nach einfachen und komplexen Mustern in Zeichenfolgen suchen (und diese bearbeiten)
- Mit einer Gruppe von SQL-Funktionen und -Bedingungen in SQL und PL/SQL nach Zeichenfolgen suchen und diese bearbeiten
- Reguläre Ausdrücke mit folgenden Elementen angeben:
  - Metazeichen (Operatoren, die die Suchalgorithmen angeben)
  - Literale (die Zeichen, nach denen Sie suchen)

ORACLE

Oracle Database bietet Unterstützung für reguläre Ausdrücke. Die Implementierung folgt der im POSIX-(Portable Operating System for UNIX-)Standard des Institute of Electrical and Electronics Engineers (IEEE) für den ASCII-Datenvergleich definierten Syntax und Vergleichssemantik. Durch die Mehrsprachenfähigkeit von Oracle werden die Vergleichsfunktionen der Operatoren über den POSIX-Standard hinaus erweitert. Reguläre Ausdrücke sind eine Methode zur Beschreibung einfacher sowie komplexer Muster für die Suche und Bearbeitung.

Die Suche und Bearbeitung von Zeichenfolgen stellen einen hohen Prozentsatz der Logik in webbasierten Anwendungen dar. Die Anwendungsbereiche reichen von der einfachen Suche des Begriffs "San Francisco" in einem bestimmten Text über die komplexe Extraktion aller URL-Adressen aus einem Text bis hin zur noch komplexeren Suche nach allen Wörtern, in denen jeder zweite Buchstabe ein Vokal ist.

Wenn reguläre Ausdrücke mit systemeigenem SQL kombiniert werden, sind leistungsstarke Such- und Bearbeitungsvorgänge für die in einer Oracle-Datenbank gespeicherten Daten möglich. Mit diesem Feature können Sie Aufgabenstellungen einfach lösen, die andernfalls eine äußerst komplexe Programmierung erfordern würden.

Reguläre Ausdrücke – Vorteile

Die Verwendung regulärer Ausdrücke bei der Implementierung komplexer Vergleichslogik in der Datenbank bietet die folgenden Vorteile:

- Die Zentralisierung der Vergleichslogik in Oracle Database vermeidet aufwendige Verarbeitunge von SQL-Ergebnis-mengen in Middle-Tier-Anwendungen.
- Durch die Verwendung von serverseitigen regulären Ausdrücken zur Durchsetzung von Constraints entfällt die Kodierung der Datenvalidierungslogik auf dem Client.
- Die integrierten SQL- und PL/SQL-Funktionen und -Bedingungen für reguläre Ausdrücke machen die Bearbeitung von Zeichenfolgen leistungsstärker und einfacher im Vergleich zu vorherigen Releases von Oracle Database.

ORACLE

Reguläre Ausdrücke sind eine leistungsfähige Textverarbeitungskomponente von Programmier-sprachen wie PERL und Java. Beispiel: Ein PERL-Skript kann jede HTML-Datei in einem Verzeichnis verarbeiten, ihren Inhalt in eine skalare Variable wie eine einzelne Zeichenfolge lesen und anschließend in der Zeichenfolge mithilfe regulärer Ausdrücke nach URLs suchen. Viele Entwickler, die in PERL programmieren, loben die robuste Mustervergleichsfunktionalität dieser Sprache. Dank der Unterstützung regulärer Ausdrücke durch Oracle können Entwickler komplexe Vergleichslogik in der Datenbank implementieren. Reguläre Ausdrücke wurden in Oracle Database 10*g* eingeführt.

Funktionen und Bedingungen für reguläre Ausdrücke in SQL und PL/SQL

Name der Funktion oder Bedingung	Beschreibung
REGEXP_LIKE	Ähnelt dem Operator LIKE, führt jedoch Vergleiche der regulären Ausdrücke statt einfacher Mustervergleiche durch (Bedingung)
REGEXP_REPLACE	Sucht ein reguläres Ausdrucksmuster und ersetzt es durch eine Ersatzzeichenfolge
REGEXP_INSTR	Sucht nach einer Zeichenfolge für ein reguläres Ausdrucksmuster und gibt die Position zurück, an der die Übereinstimmung gefunden wurde
REGEXP_SUBSTR	Sucht ein reguläres Ausdrucksmuster innerhalb einer bestimmten Zeichenfolge und extrahiert die Teil-zeichenfolge, für die eine Übereinstimmung gefunden wurde
REGEXP_COUNT	Gibt die Häufigkeit des in einer Eingabezeichenfolge gefundenen Mustervergleichs zurück

ORACLE

Oracle Database stellt eine Reihe von SQL-Funktionen zur Verfügung, um Zeichenfolgen mithilfe regulärer Ausdrücke zu suchen und zu bearbeiten. Sie können diese Funktionen für Textliterale, Bindevariablen oder alle Spalten verwenden, die Daten vom Typ CHAR, NCHAR, CLOB, NCLOB, NVARCHAR2 und VARCHAR2 (aber nicht LONG) enthalten. Ein regulärer Ausdruck muss in einfachen Anführungszeichen stehen. Dies verbessert die Lesbarkeit des Codes und stellt sicher, dass der gesamte Ausdruck von der SQL-Funktion interpretiert wird.

- REGEXP_LIKE: Diese Bedingung durchsucht eine Zeichenspalte nach einem Muster. Verwenden Sie diese Bedingung in der Klausel WHERE einer Abfrage, um Zeilen zurückzugeben, die mit dem angegebenen regulären Ausdruck übereinstimmen.
- REGEXP_REPLACE: Diese Funktion durchsucht eine Zeichenspalte nach einem Muster und ersetzt jedes Vorkommen dieses Musters durch das angegebene Muster.
- REGEXP_INSTR: Diese Funktion sucht nach einer Zeichenfolge für ein bestimmtes Vorkommen eines regulären Ausdrucksmusters. Sie geben das gewünschte Vorkommen sowie den Ausgangspunkt für die Suche an. Die Funktion gibt eine ganze Zahl zurück, die die Position in der Zeichenfolge angibt, an der die Übereinstimmung gefunden wurde.
- REGEXP_SUBSTR: Diese Funktion gibt die konkrete Teilzeichenfolge zurück, die mit dem angegebenen regulären Ausdrucksmuster übereinstimmt.
- REGEXP_COUNT: Diese Funktion gibt die Häufigkeit des in einer Eingabezeichenfolge gefundenen Mustervergleichs zurück.

Was sind Metazeichen?

- Metazeichen sind Sonderzeichen, die eine spezielle Bedeutung haben, wie ein Platzhalterzeichen, ein Wiederholungszeichen, nicht übereinstimmende Zeichen oder ein Zeichenbereich.
- In Mustervergleichen lassen sich verschiedene vordefinierte Metazeichen (auch Metasymbole genannt) verwenden.
- Beispiel: Der reguläre Ausdruck `^(f|ht)tps?:$` sucht beginnend am Anfang der Zeichenfolge nach folgenden Elementen:
  - Literal `f` oder `ht`
  - Literal `t`
  - Literal `p`, optional gefolgt vom Literal `s`
  - Doppelpunktliteral ":" am Ende der Zeichenfolge

ORACLE

Der reguläre Ausdruck auf der Folie stimmt mit den Zeichenfolgen `http:`, `https:`, `ftp:` und `ftps:` überein.

Hinweis: Eine vollständige Liste der Metazeichen regulärer Ausdrücke finden Sie im *Oracle Database Advanced Application Developer's Guide* für Oracle Database 12*c*.

Metazeichen in regulären Ausdrücken

Syntax	Beschreibung
.	Stimmt mit jedem beliebigen Zeichen im unterstützten Zeichensatz außer NULL überein
+	Stimmt mit einem oder mehreren Vorkommen überein
?	Stimmt mit null oder einem Vorkommen überein
*	Stimmt mit null oder mehr Vorkommen des vorhergehenden Teilausdrucks überein
{m}	Stimmt mit genau *m* Vorkommen des vorhergehenden Ausdrucks überein
{m, }	Stimmt mit mindestens *m* Vorkommen des vorhergehenden Teilausdrucks überein
{m,n}	Stimmt mit mindestens *m*, aber nicht mehr als *n* Vorkommen des vorhergehenden Teilausdrucks überein
[…]	Stimmt mit einem beliebigen einzelnen Zeichen in der Liste in Klammern überein
|	Stimmt mit einer der Alternativen überein
(...)	Behandelt den in Klammern angeordneten Ausdruck als eine Einheit. Der Teilausdruck kann eine Zeichenfolge von Literalen oder ein komplexer Ausdruck mit Operatoren sein.

Beliebiges Zeichen, " . ": `a.b` stimmt mit den Zeichenfolgen `abb`, `acb` und `adb`, aber nicht mit `acc` überein.

Ein oder mehr Zeichen, " + ": `a+` stimmt mit den Zeichenfolgen `a`, `aa` und `aaa`, aber nicht mit `bbb` überein.

Null oder ein Zeichen, " ? ": `ab?c` stimmt mit den Zeichenfolgen `abc` und `ac`, aber nicht mit `abbc` überein.

Null oder mehr Zeichen, " * ": `ab*c` stimmt mit den Zeichenfolgen `ac`, `abc` und `abbc`, aber nicht mit `abb` überein.

Genaue Anzahl, " {m} ": `a{3}` stimmt mit der Zeichenfolge `aaa`, aber nicht mit `aa` überein.

Mindestanzahl, " {m,} ": `a{3,}` stimmt mit den Zeichenfolgen `aaa` und `aaaa`, aber nicht mit `aa` überein.

Anzahl zwischen, " {m,n} ": `a{3,5}` stimmt mit den Zeichenfolgen `aaa`, `aaaa` und `aaaaa`, aber nicht mit `aa` überein.

Liste übereinstimmender Zeichen, " […] ": `[abc]` stimmt mit dem ersten Zeichen in den Zeichenfolgen `all`, `bill` und `cold`, aber mit keinem Zeichen in `doll` überein.

Oder, " | ": `a|b` stimmt mit dem Zeichen `a` oder dem Zeichen `b` überein.

Teilausdruck, " (…) ": `(abc)?def` stimmt mit der optionalen Zeichenfolge `abc` gefolgt von `def` überein. Der Ausdruck stimmt mit `abcdefghi` und `def`, aber nicht mit `ghi` **überein.** Der Teilausdruck kann eine Zeichenfolge von Literalen oder ein komplexer Ausdruck mit Operatoren sein.

Metazeichen in regulären Ausdrücken

Syntax	Beschreibung
^	Stimmt mit dem Anfang einer Zeichenfolge überein
$	Stimmt mit dem Ende einer Zeichenfolge überein
\	Behandelt das nachfolgende Metazeichen im Ausdruck als Literal
\n	Stimmt mit dem *n*. (1-9) vorhergehenden Ausdruck vor der Gruppierung in Klammern überein. Die Klammern bewirken, dass ein Ausdruck gespeichert wird. Ein Rückverweis verweist auf ihn.
\d	Eine Ziffer
[:class:]	Stimmt mit einem beliebigen Zeichen überein, das zur angegebenen POSIX-Zeichenklasse gehört
[^:class:]	Stimmt mit einem beliebigen einzelnen Zeichen überein, das *nicht* in der Liste in Klammern enthalten ist

Zeilenanfang-/Zeilenende-Anker, " ^ " und "$": `^def` stimmt mit `def` in der Zeichenfolge `defghi`, aber nicht mit `def` in der Zeichenfolge `abcdef` überein. `def$` stimmt mit `def` in der Zeichenfolge `abcdef`, aber nicht mit `def` in der Zeichenfolge `defghi` überein.

Escape-Zeichen " \ ": `\+` sucht nach einem +. Stimmt mit dem Pluszeichen in der Zeichenfolge `abc+def`, aber nicht mit `Abcdef` überein.

Rückverweis, " \n ": `(abc|def)xy\1` stimmt mit den Zeichenfolgen `abcxyabc` und `defxydef`, aber nicht mit den Zeichenfolgen `abcxydef` und `abcxy` überein. Mit einem Rückverweis können Sie nach einer wiederholten Zeichenfolge suchen, ohne diese vorher zu kennen. Beispiel: Der Ausdruck `^(.*)\1$` stimmt mit einer Zeile überein, die aus zwei benachbarten Instanzen derselben Zeichenfolge besteht.

Ziffernzeichen, "\d": Der Ausdruck `^\[\d{3}\] \d{3}-\d{4}$` stimmt mit **[650] 555-1212**, aber nicht mit **650-555-1212** überein.

Zeichenklasse, " [:class:] ": `[[:upper:]]+` sucht nach einem oder mehreren aufeinanderfolgenden Großbuchstaben. Stimmt mit `DEF` in der Zeichenfolge `abcDEFghi`, aber nicht mit der Zeichenfolge `abcdefghi` überein.

Liste (oder Klasse) nicht übereinstimmender Zeichen, " [^...] ": `[^abc]` stimmt mit dem Zeichen `d` in der Zeichenfolge `abcdef`, aber nicht mit dem Zeichen `a`, `b` oder `c` überein.

Funktionen und Bedingungen für reguläre Ausdrücke – Syntax

```
REGEXP_LIKE    (source_char, pattern [,match_option]

REGEXP_INSTR   (source_char, pattern [, position
                [, occurrence [, return_option
                [, match_option [, subexpr]]]]])

REGEXP_SUBSTR  (source_char, pattern [, position
                [, occurrence [, match_option
                [, subexpr]]]])

REGEXP_REPLACE(source_char, pattern [,replacestr
                 [, position [, occurrence
                 [, match_option]]]])

REGEXP_COUNT  (source_char, pattern [, position
                [, occurrence [, match_option]]])
```


Die Funktionen und Bedingungen für reguläre Ausdrücke verwenden die folgende Syntax:

- `source_char`: Ein Zeichenausdruck, der als Suchwert dient
- `pattern`: Ein regulärer Ausdruck, ein Textliteral
- `occurrence`: Eine positive ganze Zahl, die anzeigt, nach welchem Mustervorkommen Oracle Server in `source_char` suchen soll. Der Standardwert lautet 1.
- `position`: Eine positive ganze Zahl, die das Zeichen in `source_char` anzeigt, bei dem der Oracle-Server die Suche beginnen soll. Der Standardwert lautet 1.
- `return_option`:
  - 0: Gibt die Position des ersten Zeichens des Vorkommens zurück (Standard)
  - 1: Gibt die Position des Zeichens zurück, das nach dem Vorkommen folgt
- `Replacestr`: Muster für die Ersatzzeichenfolge
- `match_parameter`:
  - " `c` ": Groß-/Kleinschreibung in den Übereinstimmungen beachten (Standard)
  - “ `i` ": Keine Groß-/Kleinschreibung in den Übereinstimmungen beachten
  - “ `n` ": Lässt Operator für beliebiges übereinstimmendes Zeichen zu
  - “ `m` ": Behandelt die Quellzeichenfolge wie mehrere Zeilen
- `subexpr`: Auszug des Musters in Klammern. Nähere Informationen zu Teilausdrücken erhalten Sie im weiteren Verlauf dieses Anhangs.

Einfache Suche mit der Bedingung REGEXP_LIKE durchführen

```
REGEXP_LIKE(source_char, pattern [, match_parameter ])
```

```
SELECT first_name, last_name
FROM employees
WHERE REGEXP_LIKE (first_name, '^Ste(v|ph)en$');
```

	FIRST_NAME	LAST_NAME
1	Steven	King
2	Steven	Markle
3	Stephen	Stiles

ORACLE

REGEXP_LIKE ähnelt der Bedingung LIKE mit dem einzigen Unterschied, dass REGEXP_LIKE mithilfe regulärer Ausdrücke und nicht mit einfachen Mustern wie im Falle von LIKE nach Übereinstimmungen sucht. Diese Bedingung wertet Zeichenfolgen mithilfe von Zeichen aus, die durch den Eingabezeichensatz definiert werden.

REGEXP_LIKE – Beispiel

Als Ergebnis dieser Abfrage für die Tabelle EMPLOYEES werden alle Mitarbeiter mit dem Vornamen Steven oder Stephen angezeigt. Für den verwendeten Ausdruck '^Ste(v|ph)en$ ' gilt:

- ^ gibt den Anfang des Ausdrucks an.
- $ gibt das Ende des Ausdrucks an.
- | gibt entweder/oder an.

Muster mithilfe der Funktion REGEXP_REPLACE ersetzen

```
REGEXP_REPLACE(source_char, pattern [,replacestr
[, position [, occurrence [, match_option]]]])
```

```
SELECT last_name, REGEXP_REPLACE(phone_number, '\.','-')
   AS phone
FROM employees;
```

Original

	LAST_NAME	PHONE
1	OConnell	650.507.9833
2	Grant	650.507.9844
3	Whalen	515.123.4444
4	Hartstein	515.123.5555

Teilergebnisse

	LAST_NAME	PHONE
1	OConnell	650-507-9833
2	Grant	650-507-9844
3	Whalen	515-123-4444
4	Hartstein	515-123-5555

ORACLE

Mit der Funktion `REGEXP_REPLACE` können Sie eine Telefonnummer umformatieren, um das Trennzeichen Punkt (`.`) durch einen Bindestrich (`-`) zu ersetzen. Es folgt eine Erläuterung der einzelnen Elemente, die im regulären Ausdruck im Beispiel verwendet werden:

- `phone_number` ist die Quellspalte.
- `'\.'` ist das Suchmuster.
  - Verwenden Sie einfache Anführungszeichen (`' '`), um nach dem Literalzeichen "Punkt" (`.`) zu suchen.
  - Verwenden Sie den umgekehrten Schrägstrich (`\`), um nach einem Zeichen zu suchen, das normalerweise als Metazeichen behandelt wird.
- `'-'` ist die Ersatzzeichenfolge.

Muster mithilfe der Funktion REGEXP_INSTR suchen

```
REGEXP_INSTR  (source_char, pattern [, position [,
   occurrence [, return_option [, match_option]]]])
```

```
SELECT street_address,
REGEXP_INSTR(street_address,'[[:alpha:]]') AS
   First_Alpha_Position
FROM  locations;
```

	STREET_ADDRESS	FIRST_ALPHA_POSITION
1	1297 Via Cola di Rie	6
2	93091 Calle della Testa	7
3	2017 Shinjuku-ku	6
4	9450 Kamiya-cho	6

In diesem Beispiel wird die Straßenangabe mit der Funktion `REGEXP_INSTR` durchsucht, um die Position des ersten alphabetischen Zeichens zu finden, wobei die Groß-/Kleinschreibung ignoriert wird. Beachten Sie, dass `[:<class>:]` eine Zeichenklasse angibt und nach Übereinstimmungen für ein beliebiges Zeichen in dieser Klasse sucht. `[:alpha:]` sucht nach Übereinstimmungen für ein beliebiges alphabetisches Zeichen. Die Teilergebnisse werden angezeigt.

Für den in der Abfrage verwendeten Ausdruck `'[[:alpha:]]'` gilt:

- `[` gibt den Anfang des Ausdrucks an.
- `[:alpha:]` gibt die alphabetische Zeichenklasse an.
- `]` gibt das Ende des Ausdrucks an.

Hinweis: Mit dem Zeichenklassenoperator POSIX können Sie nach einem Ausdruck in einer Liste mit Zeichen suchen, die Teil einer bestimmten Zeichenklasse vom Typ POSIX sind. Dieser Operator ermöglicht die Suche nach bestimmten Formatierungen wie Großbuchstaben oder nach Sonderzeichen wie Ziffern oder Satzzeichen. Alle POSIX-Zeichenklassen werden unterstützt. Verwenden Sie die Syntax `[:class:]`, wobei `class` der Name der POSIX-Zeichenklasse ist, nach der gesucht werden soll. Der folgende reguläre Ausdruck sucht nach einem oder mehreren aufeinanderfolgenden Zeichen in Großschreibung: `[[:upper:]]+` .

Teilzeichenfolgen mithilfe der Funktion REGEXP_SUBSTR extrahieren

```
REGEXP_SUBSTR (source_char, pattern [, position
               [, occurrence [, match_option]]])
```

```
SELECT REGEXP_SUBSTR(street_address , ' [^ ]+ ') AS Road
FROM locations;
```

	ROAD
1	Via
2	Calle
3	(null)
4	(null)
5	Jabberwocky

In diesem Beispiel werden die Straßennamen aus der Tabelle `LOCATIONS` extrahiert. Hierzu wird der Inhalt in der Spalte `STREET_ADDRESS` nach dem ersten Leerzeichen mit der Funktion `REGEXP_SUBSTR` zurückgegeben. Für den in der Abfrage verwendeten Ausdruck `' [^ ]+ '` gilt:

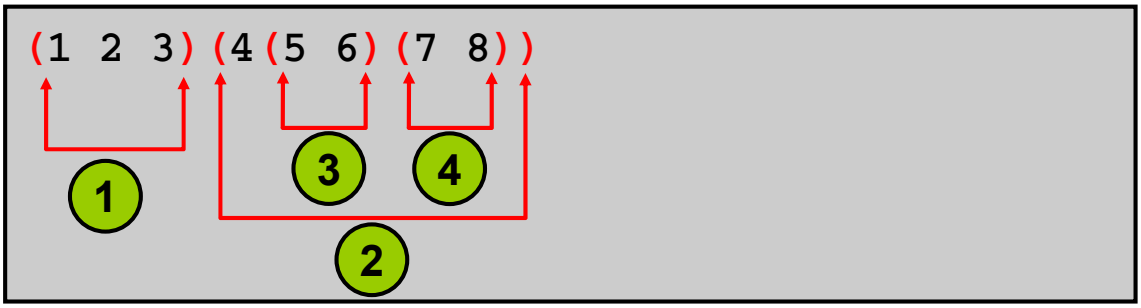
- `[` gibt den Anfang des Ausdrucks an.
- `^` gibt NOT an.
- `' '` gibt ein Leerzeichen an.
- `]` gibt das Ende des Ausdrucks an.
- `+` gibt ein oder mehrere Vorkommen an.

Teilausdrücke

Untersuchen Sie den folgenden Ausdruck:

```
(1 2 3)(4(5 6)(7 8))
```

Die Teilausdrücke lauten:



Oracle Database bietet Parameter zur Unterstützung regulärer Ausdrücke, mit denen der Zugriff auf einen Teilausdruck möglich ist. Im Beispiel auf der Folie ist eine Zeichenfolge von Ziffern dargestellt. Die Klammern geben die Teilausdrücke in der Ziffernzeichenfolge an. Wenn der Ausdruck von links nach rechts und von den äußeren Klammern zu den inneren Klammern gelesen wird, lauten die Teilausdrücke in der Ziffernzeichenfolge folgendermaßen:

1. 123
2. 45678
3. 56
4. 78

Mit den Funktionen `REGEXP_INSTR` und `REGEXP_SUBSTR` können Sie nach jedem dieser Teilausdrücke suchen.

Teilausdrücke in Verbindung mit regulären Ausdrücken

```
SELECT
  REGEXP_INSTR
(1)('0123456789',       -- source char or search value
(2)'(123)(4(56)(78))', -- regular expression patterns
(3)1,                   -- position to start searching
(4)1,                   -- occurrence
(5)0,                   -- return option
(6)'i',                 -- match option (case insensitive)
(7)1)              -- subexpression on which to search
   "Position"
FROM dual;
```

	Position
1	2

ORACLE

Die Funktionen `REGEXP_INSTR` und `REGEXP_SUBSTR` besitzen einen optionalen `SUBEXPR`-Parameter, mit dem eine bestimmte Teilzeichenfolge des auszuwertenden regulären Ausdrucks als Ziel definiert werden kann.

Im Beispiel auf der Folie soll nach dem Muster des ersten Teilausdrucks in der Liste der Teilausdrücke gesucht werden. Das dargestellte Beispiel enthält mehrere Parameter für die Funktion `REGEXP_INSTR`.

1. Die zu suchende Zeichenfolge wird angegeben.
2. Die Teilausdrücke werden angegeben. Der erste Teilausdruck lautet `123`. Der zweite Teilausdruck lautet `45678`, der dritte `56` und der vierte `78`.
3. Der dritte Parameter legt die Position fest, an der die Suche beginnen soll.
4. Der vierte Parameter gibt das Vorkommen des zu suchenden Musters an. `1` bedeutet, dass das erste Vorkommen gesucht werden soll.
5. Der fünfte Parameter ist die Rückgabeoption. Dies ist die Position des ersten Zeichens des Vorkommens. (Wenn Sie `1` angeben, wird die Position des Zeichens zurückgegeben, das nach dem Vorkommen folgt.)
6. Der sechste Parameter gibt an, ob bei der Suche Groß-/Kleinschreibung zu beachten ist.
7. Der letzte Parameter gibt an, nach welchem Teilausdruck gesucht werden soll. Im dargestellten Beispiel wird nach dem ersten Teilausdruck, d. h. `123`, gesucht.

Wieso ist der Zugriff auf den *n*. Teilausdruck wichtig?

- Realistischere Anwendung: DNA-Sequenzierung
- Sie müssen ein bestimmtes Teilmuster suchen, das in der DNA einer Maus ein Protein für die Immunität angibt.

```
SELECT
    REGEXP_INSTR('ccacctttccctccactcctcacgttctcacctgtaaagcgtccctc
    cctcatccccatgccccttaccctgcagggtagagtaggctagaaaccagagagctccaagc
    tccatctgtggagaggtgccatccttgggctgcagagagaggagaatttgccccaaagctgcc
    tgcagagcttcaccacccttagtctcacaaagccttgagttcatagcatttcttgagttttca
    ccctgcccagcaggacactgcagcacccaaagggcttcccaggagtagggttgccctcaagag
    gctcttgggtctgatggccacatcctggaattgttttcaagttgatggtcacagccctgaggc
    atgtaggggcgtggggatgcgctctgctctgctctcctctcctgaacccctgaacccctctggc
    taccccagagcacttagagccag',
                    '(gtc(tcac)(aaag))',
                     1, 1, 0, 'i',
                     1) "Position"
FROM dual;
```

	Position
1	195

ORACLE

In der Biowissenschaft müssen Sie unter Umständen die Offsets von Übereinstimmungen von Teilausdrücken aus einer DNA-Sequenz für die weitere Bearbeitung extrahieren. Beispiel: Sie müssen eine bestimmte Proteinsequenz suchen, wie den Anfangs-Offset der DNA-Sequenz, dem `gtc` vorangestellt ist, gefolgt von `tcac` und `aaag`. Diese Aufgabe können Sie mit der Funktion `REGEXP_INSTR` lösen, die die Position zurückgibt, an der eine Übereinstimmung gefunden wird.

In dem Beispiel auf der Folie wird die Position des ersten Teilausdrucks `(gtc)` zurückgegeben. `gtc` wird an Position 195 der DNA-Zeichenfolge gefunden.

Wenn Sie das Beispiel auf der Folie entsprechend ändern, um den zweiten Teilausdruck `(tcac)` zu suchen, gibt die Abfrage folgendes Ergebnis zurück. `tcac` wird an Position 198 der DNA-Zeichenfolge gefunden.

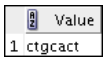
	Position
1	198

Wenn Sie das Beispiel auf der Folie entsprechend ändern, um den dritten Teilausdruck `(aaag)` zu suchen, gibt die Abfrage folgendes Ergebnis zurück. `aaag` wird an Position 202 der DNA-Zeichenfolge gefunden.

	Position
1	202

REGEXP_SUBSTR: Beispiel

```
SELECT
  REGEXP_SUBSTR
(1)('acgctgcactgca', -- source char or search value
(2) 'acg(.*)gca',    -- regular expression pattern
(3) 1,               -- position to start searching
(4) 1,               -- occurrence
(5) 'i',             -- match option (case insensitive)
(6) 1)               -- sub-expression
 "Value"
FROM dual;
```



Im Beispiel auf der Folie gilt:

1. `acgctgcactgca` ist die Quelle, die durchsucht werden soll.
2. `acg(.*)gca` ist das zu suchende Muster. Gesucht werden soll `acg` gefolgt von `gca`, wobei zwischen `acg` und `gca` Zeichen stehen können.
3. Die Suche soll beim ersten Zeichen der Quelle beginnen.
4. Das erste Vorkommen des Musters soll gesucht werden.
5. Bei Übereinstimmungen in der Quelle soll die Groß-/Kleinschreibung nicht beachtet werden.
6. Durch eine nicht negative ganze Zahl soll der *n*. Teilausdruck angegeben werden, der als Ziel zu definieren ist. Dies ist der Parameter des Teilausdrucks. In diesem Beispiel gibt `1` den ersten Teilausdruck an. Sie können einen Wert von 0 bis 9 verwenden. 0 bedeutet, dass kein Teilausdruck als Ziel definiert wird. Der Standardwert für diesen Parameter ist 0.

Funktion REGEXP_COUNT

```
REGEXP_COUNT:
                [, occurrence [, match_option]]])
```

```
SELECT REGEXP_COUNT(
    'ccacctttccctccactcctcacgttctcacctgtaaagcgtccctccctcatccccatgccccttaccctgcag
    ggtagagtaggctagaaaccagagagctccaagctccatctgtggagaggtgccatccttgggctgcagagagaggag
    aatttgccccaaagctgcctgcagagcttcaccacccttagtctcacaaagccttgagttcatagcatttcttgagtt
    ttcaccctgcccagcaggacactgcagcacccaaagggcttcccaggagtagggttgccctcaagaggctcttgggtc
    tgatggccacatcctggaattgttttcaagttgatggtcacagccctgaggcatgtaggggcgtggggatgcgctctg
    ctctgctctcctctcctgaacccctgaacccctctggctaccccagagcacttagagccag',
    'gtc') AS Count
FROM dual;
```

	COUNT
1	4

Die Funktion REGEXP_COUNT wertet Zeichenfolgen mithilfe von Zeichen aus, die durch den Eingabezeichensatz definiert werden. Sie gibt eine ganze Zahl zurück, die die Anzahl der Vorkommen des Musters angibt. Wird keine Übereinstimmung gefunden, gibt die Funktion 0 zurück.

Im Beispiel auf der Folie wird mithilfe der Funktion REGEXP_COUNT die Anzahl der Vorkommen für eine DNA-Teilzeichenfolge bestimmt.

Das folgende Beispiel zeigt, dass das Muster 123 in der Zeichenfolge 123123123123 drei Mal vorkommt. Die Suche beginnt an der zweiten Position der Zeichenfolge.

```
SELECT REGEXP_COUNT
   ('123123123123', -- source char or search value
   '123',           -- regular expression pattern
    2,              -- position where the search should start
   'i')             -- match option (case insensitive)
   As Count
FROM dual;
```

	COUNT
1	3

Reguläre Ausdrücke und CHECK-Constraints – Beispiele

```
ALTER TABLE emp8
  ADD CONSTRAINT email_addr
  CHECK(REGEXP_LIKE(email,'@')) NOVALIDATE;
```

```
INSERT INTO emp8 VALUES
  (500,'Christian','Patel','ChrisP2creme.com',
   1234567890,'12-Jan-2004','HR_REP',2000,null,102,40);
```

```
Error starting at line 1 in command:
INSERT INTO emp8 VALUES
  (500,'Christian','Patel',
  'ChrisP2creme.com', 1234567890,
   '12-Jan-2004', 'HR_REP', 2000, null, 102, 40)
Error report:
SQL Error: ORA-02290: check constraint (TEACH_B.EMAIL_ADDR) violated
02290. 00000 -  "check constraint (%s.%s) violated"
*Cause:    The values being inserted do not satisfy the named check

*Action:   do not insert values that violate the constraint.
```

ORACLE

Reguläre Ausdrücke können auch im Constraint `CHECK` verwendet werden. In diesem Beispiel wird das Constraint `CHECK` zur Spalte `EMAIL` der Tabelle `EMPLOYEES` hinzugefügt. Dadurch wird sichergestellt, dass nur Zeichenfolgen akzeptiert werden, die das Symbol "`@`" enthalten. Das Constraint wird getestet. Das Constraint `CHECK` wird verletzt, da die E-Mail-Adresse das erforderliche Symbol nicht enthält. Die Klausel `NOVALIDATE` stellt sicher, dass vorhandene Daten nicht geprüft werden.

Für das Beispiel auf der Folie wird die Tabelle `emp8` durch folgenden Code erstellt:

```
CREATE TABLE emp8 AS SELECT * FROM employees;
```

Hinweis: Das obige Beispiel wird mit der Option **Execute Statement** von SQL Developer ausgeführt. Wenn Sie die Option **Run Script** verwenden, ändert sich das Ausgabeformat.

Quiz

Mit regulären Ausdrücken in SQL und PL/SQL können Sie:

a. aufwendige Verarbeitungen von SQL-Ergebnismengen in Middle-Tier-Anwendungen vermeiden
b. Datenvalidierungslogik auf dem Client vermeiden
c. Constraints auf dem Server durchsetzen

ORACLE

Richtige Antwort: a, b, c

Zusammenfassung

In diesem Anhang haben Sie gelernt, wie Sie mit regulären Ausdrücken Zeichenfolgen suchen, diese vergleichen und ersetzen.

In dieser Lektion haben Sie die Features zur Unterstützung regulärer Ausdrücke kennengelernt. Die Unterstützung regulärer Ausdrücke ist in SQL und PL/SQL verfügbar.