Kostenloses eBook

LERNEN Oracle Database

Free unaffiliated eBook created from
Stack Overflow contributors.

#oracle

Inhaltsverzeichnis

Über

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: oracle-database

It is an unofficial and free Oracle Database ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Oracle Database.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Kapitel 1: Erste Schritte mit Oracle Database

Bemerkungen

Oracle ist ein relationales Datenbankverwaltungssystem (RDBMS), das ursprünglich von Larry Ellison, Bob Miner und Ed Oates in den späten 70er Jahren entwickelt wurde. Es sollte mit dem System R von IBM kompatibel sein.

Versionen

Ausführung	Veröffentlichungsdatum
Version 1 (nicht veröffentlicht)	1978-01-01
Oracle V2	1979-01-01
Oracle Version 3	1983-01-01
Oracle Version 4	1984-01-01
Oracle Version 5	1985-01-01
Oracle Version 6	1988-01-01
Oracle7	1992-01-01
Oracle8	1997-07-01
Oracle8i	1999-02-01
Oracle9i	2001-06-01
Oracle 10g	2003-01-01
Oracle 11g	2007-01-01
Oracle 12c	2013-01-01

Examples

Hallo Welt

```
SELECT 'Hello world!' FROM dual;
```

In der SQL- Variante von Oracle ist "Dual nur eine Convenient-Tabelle" . Ursprünglich war es

beabsichtigt , Zeilen über einen JOIN zu verdoppeln, enthält jedoch jetzt eine Zeile mit dem `DUMMY` Wert 'X'.

Hallo Welt! vom Tisch

Erstellen Sie eine einfache Tabelle

```
create table MY_table (
   what varchar2(10),
   who varchar2(10),
   mark varchar2(10)
);
```

Werte einfügen (Zielspalten können weggelassen werden, wenn Sie Werte für alle Spalten angeben)

```
insert into my_table (what, who, mark) values ('Hello', 'world', '!' );
insert into my_table values ('Bye bye', 'ponies', '?' );
insert into my_table (what) values('Hey');
```

Denken Sie daran, festzuschreiben, da Oracle *Transaktionen verwendet*

```
commit;
```

Wählen Sie Ihre Daten aus:

```
select what, who, mark from my_table where what='Hello';
```

SQL-Abfrage

Liste der Mitarbeiter, die mehr als 50000 US-Dollar verdient haben, die in diesem Jahrhundert geboren wurden. Geben Sie Namen, Geburtsdatum und Gehalt an, alphabetisch sortiert nach Namen.

```
SELECT employee_name, date_of_birth, salary
FROM   employees
WHERE  salary > 50000
   AND date_of_birth >= DATE '2000-01-01'
ORDER BY employee_name;
```

Zeigen Sie die Anzahl der Mitarbeiter in jeder Abteilung mit mindestens 5 Mitarbeitern an. Listen Sie zuerst die größten Abteilungen auf.

```
SELECT department_id, COUNT(*)
FROM   employees
GROUP BY department_id
HAVING COUNT(*) >= 5
ORDER BY COUNT(*) DESC;
```

Hallo Welt von PL / SQL

```
/* PL/SQL is a core Oracle Database technology, allowing you to build clean, secure,
   optimized APIs to SQL and business logic. */

set serveroutput on

BEGIN
   DBMS_OUTPUT.PUT_LINE ('Hello World!');
END;
```

Erste Schritte mit Oracle Database online lesen: https://riptutorial.com/de/oracle/topic/558/erste-schritte-mit-oracle-database

Kapitel 2: Abfrage auf Ebene

Bemerkungen

Die Level-Klausel ist für die Erzeugung einer N-Anzahl von Dummy-Datensätzen auf der Grundlage einer bestimmten Bedingung verantwortlich.

Examples

Generiere N Anzahl Datensätze

```
SELECT ROWNUM NO FROM DUAL CONNECT BY LEVEL <= 10
```

Wenige Verwendungen von Level Query

/ * Dies ist eine einfache Abfrage, die eine Zahlenfolge erzeugen kann. Das folgende Beispiel generiert eine Zahlenfolge aus 1..100 * /

```
select level from dual connect by level <= 100;
```

/ * Die obige Abfrage ist in verschiedenen Szenarien hilfreich, z. B. das Generieren einer Datumsfolge aus einem bestimmten Datum. Die folgende Abfrage generiert 10 aufeinander folgende Daten. * /

```
select to_date('01-01-2017','mm-dd-yyyy')+level-1 as  dates from dual connect by level <= 10;
```

01. Januar-17
02-JAN-17
03-JAN-17
04-JAN-17
05-JAN-17
06-JAN-17
07-JAN-17
08-JAN-17
09-JAN-17
10. Januar-17

Abfrage auf Ebene online lesen: https://riptutorial.com/de/oracle/topic/6548/abfrage-auf-ebene

Kapitel 3: Anonymer PL / SQL-Block

Bemerkungen

Da sie nicht benannt sind, können anonyme Blöcke nicht von anderen Programmeinheiten referenziert werden.

Examples

Ein Beispiel für einen anonymen Block

```
DECLARE
    -- declare a variable
    message varchar2(20);
BEGIN
  -- assign value to variable
  message := 'HELLO WORLD';

  -- print message to screen
  DBMS_OUTPUT.PUT_LINE(message);
END;
/
```

Anonymer PL / SQL-Block online lesen: https://riptutorial.com/de/oracle/topic/6451/anonymer-pl---sql-block

Kapitel 4: Autonome Transaktionen

Bemerkungen

Typische Anwendungsfälle für autonome Transaktionen sind.

1. Für das Erstellen jeglicher Art von Protokollierungsframework, wie das im obigen Beispiel erläuterte Fehlerprotokollierungsframework.
2. Für die Überwachung von DML-Vorgängen in Triggern in Tabellen, unabhängig vom endgültigen Status der Transaktion (COMMIT oder ROLLBACK).

Examples

Verwendung einer autonomen Transaktion zum Protokollieren von Fehlern

Das folgende Verfahren ist ein generisches Verfahren, mit dem alle Fehler in einer Anwendung in einer allgemeinen Fehlerprotokolltabelle protokolliert werden.

```
CREATE OR REPLACE PROCEDURE log_errors
(
  p_calling_program IN VARCHAR2,
  p_error_code IN INTEGER,
  p_error_description IN VARCHAR2
)
IS
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
  INSERT INTO error_log
  VALUES
  (
  p_calling_program,
  p_error_code,
  p_error_description,
  SYSDATE,
  USER
  );
  COMMIT;
END log_errors;
```

Der folgende anonyme PLSQL-Block zeigt, wie die log_errors-Prozedur aufgerufen wird.

```
BEGIN
   DELETE FROM dept WHERE deptno = 10;
EXCEPTION
   WHEN OTHERS THEN
      log_errors('Delete dept',sqlcode, sqlerrm);
      RAISE;
END;

SELECT * FROM error_log;
```

```
CALLING_PROGRAM    ERROR_CODE    ERROR_DESCRIPTION
ERROR_DATETIME         DB_USER
Delete dept        -2292         ORA-02292: integrity constraint violated - child record found
08/09/2016             APEX_PUBLIC_USER
```

Autonome Transaktionen online lesen: https://riptutorial.com/de/oracle/topic/6103/autonome-transaktionen

Kapitel 5: Datenbank-Links

Examples

Datenbanklink erstellen

```
CREATE DATABASE LINK dblink_name
CONNECT TO remote_username
IDENTIFIED BY remote_password
USING 'tns_service_name';
```

Auf die entfernte Datenbank kann dann folgendermaßen zugegriffen werden:

```
SELECT * FROM MY_TABLE@dblink_name;
```

Um eine Datenbankverknüpfungsverbindung zu testen, ohne einen der Objektnamen in der verknüpften Datenbank zu kennen, verwenden Sie die folgende Abfrage:

```
SELECT * FROM DUAL@dblink_name;
```

Um eine Domäne explizit für den verknüpften Datenbankdienst anzugeben, wird der Domänenname der `USING` Anweisung hinzugefügt. Zum Beispiel:

```
USING 'tns_service_name.WORLD'
```

Wenn kein Domänenname explizit angegeben ist, verwendet Oracle die Domäne der Datenbank, in der die Verknüpfung erstellt wird.

Oracle-Dokumentation zur Erstellung von Datenbanklinks:

- 10g: https://docs.oracle.com/cd/B19306_01/server.102/b14200/statements_5005.htm
- 11g: https://docs.oracle.com/cd/B28359_01/server.111/b28310/ds_concepts002.htm
- 12g: https://docs.oracle.com/database/121/SQLRF/statements_5006.htm#SQLRF01205

Datenbanklink erstellen

Nehmen wir an, wir haben zwei Datenbanken "ORA1" und "ORA2". Auf die Objekte von "ORA2" können wir über die Datenbank "ORA1" über eine Datenbankverbindung zugreifen.

Voraussetzungen: Zum Erstellen einer privaten Datenbankverbindung benötigen Sie ein `CREATE DATABASE LINK` Privileg. Zum Erstellen einer privaten Datenbankverbindung benötigen Sie ein `CREATE PUBLIC DATABASE LINK` Privileg.

* Oracle Net muss in beiden Instanzen vorhanden sein.

So erstellen Sie eine Datenbankverknüpfung:

Von ORA1:

```
SQL> create <public> database link ora2 connect to user1 identified by pass1 using <tns name
of ora2>;
```

Datenbanklink erstellt

Nun, da wir die DB-Verbindung eingerichtet haben, können wir dies durch Ausführen von ORA1 beweisen:

```
SQL> Select name from V$DATABASE@ORA2; -- should return ORA2
```

Sie können auch auf die DB-Objekte von "ORA2" von "ORA1" aus zugreifen, vorausgesetzt der Benutzer `user1` verfügt über das `SELECT` Privileg für diese Objekte in ORA2 (z. B. TABLE1 unten):

```
  SELECT COUNT(*) FROM TABLE1@ORA2;
```

Pre-Requistes:

- Beide Datenbanken müssen in Betrieb sein (geöffnet).
- Beide Datenbank-Listener müssen aktiv sein.
- TNS muss korrekt konfiguriert sein.
- Benutzer user1 muss in der ORA2-Datenbank vorhanden sein, das Passwort muss überprüft und überprüft werden.
- Benutzer user1 muss mindestens über das `SELECT` Privileg verfügen oder über ein anderes Zugriffsrecht auf die Objekte in ORA2 verfügen.

Datenbank-Links online lesen: https://riptutorial.com/de/oracle/topic/3859/datenbank-links

Kapitel 6: Datenpumpe

Einführung

Gehen Sie folgendermaßen vor, um einen Datenpumpenimport / -export zu erstellen:

Examples

Überwachen Sie Datapump-Jobs

Datapump-Jobs können mit überwacht werden

1. Datenwörterbuch-Ansichten:

```
   select * from dba_datapump_jobs;
   SELECT * FROM DBA_DATAPUMP_SESSIONS;
   select username,opname,target_desc,sofar,totalwork,message from V$SESSION_LONGOPS where
username = 'bkpadmin';
```

2. Datapump-Status:

- Notieren Sie den Jobnamen aus den Import- / Exportprotokollen oder dem Namen des Datenwörterbuchs und
- Führen Sie **den** Befehl zum **Anhängen aus** :
- Geben Sie den Status in der Import / Export-Eingabeaufforderung ein

```
impdp <bkpadmin>/<bkp123> attach=<SYS_IMPORT_SCHEMA_01>
Import> status
```

Drücken Sie **STRG + C** , um die Import / Export-Eingabeaufforderung zu verlassen

Schritt 3/6: Verzeichnis erstellen

```
create or replace directory DATAPUMP_REMOTE_DIR as '/oracle/scripts/expimp';
```

Schritt 7: Befehle exportieren

Befehle:

```
expdp <bkpadmin>/<bkp123>  parfile=<exp.par>
```

* Bitte ersetzen Sie die Daten in <> durch entsprechende Werte entsprechend Ihrer Umgebung. Sie können Parameter gemäß Ihren Anforderungen hinzufügen / ändern. Im obigen Beispiel werden alle übrigen Parameter in den folgenden Parameterdateien hinzugefügt: *

- Exporttyp: **Benutzer Export**
- Gesamtes Schema exportieren
- Parameterdatei-Details [say exp.par]:

```
schemas=<schema>
directory= DATAPUMP_REMOTE_DIR
dumpfile=<dbname>_<schema>.dmp
logfile=exp_<dbname>_<schema>.log
```

- Exporttyp: **Benutzer Export für großes Schema**
- Gesamtes Schema für große Datensätze exportieren: Hier werden die Export-Dump-Dateien aufgeteilt und komprimiert. Parallelität wird hier verwendet *(Hinweis: Durch das Hinzufügen von Parallelität wird die CPU-Last auf dem Server erhöht.)*
- Parameterdatei-Details [say exp.par]:

```
schemas=<schema>
directory= DATAPUMP_REMOTE_DIR
dumpfile=<dbname>_<schema>_%U.dmp
logfile=exp_<dbname>_<schema>.log
compression = all
parallel=5
```

- Exporttyp: **Tabelle Export** [Export von Tabellen]
- Parameterdatei-Details [say exp.par]:

```
tables= tname1, tname2, tname3
directory= DATAPUMP_REMOTE_DIR
dumpfile=<dbname>_<schema>.dmp
logfile=exp_<dbname>_<schema>.log
```

Schritt 9: Befehle importieren

Voraussetzung:

- Vor dem Benutzerimport empfiehlt es sich, das importierte Schema oder die Tabelle zu löschen.

Befehle:

```
impdp <bkpadmin>/<bkp123>  parfile=<imp.par>
```

* Bitte ersetzen Sie die Daten in <> durch entsprechende Werte entsprechend Ihrer Umgebung. Sie können Parameter gemäß Ihren Anforderungen hinzufügen / ändern. Im obigen Beispiel werden alle übrigen Parameter in den folgenden Parameterdateien hinzugefügt: *

- Importtyp: **Benutzerimport**
- Gesamtes Schema importieren
- Parameterdatei-Details [say imp.par]:

```
schemas=<schema>
directory= DATAPUMP_REMOTE_DIR
dumpfile=<dbname>_<schema>.dmp
logfile=imp_<dbname>_<schema>.log
```

- Importtyp: **Benutzer Import für großes Schema**
- Vollständiges Schema für große Datenmengen importieren: Hier wird Parallelität verwendet *(Hinweis: Durch Hinzufügen von Parallelität wird die CPU-Last auf dem Server erhöht.)*
- Parameterdatei-Details [say imp.par]:

```
schemas=<schema>
directory= DATAPUMP_REMOTE_DIR
dumpfile=<dbname>_<schema>_%U.dmp
logfile=imp_<dbname>_<schema>.log
parallel=5
```

- Importtyp: **Tabelle Import** [Importmenge von Tabellen]
- Parameterdatei-Details [say imp.par]:

```
tables= tname1, tname2, tname3
directory= DATAPUMP_REMOTE_DIR
dumpfile=<dbname>_<schema>.dmp
logfile=exp_<dbname>_<schema>.log
TABLE_EXISTS_ACTION= <APPEND /SKIP /TRUNCATE /REPLACE>
```

---

1. Datapump-Schritte

Quellserver [Daten exportieren]	Zielserver [Daten importieren]
1. Erstellen Sie einen Datapump-Ordner, der die Export-Dump-Dateien enthält	4. Erstellen Sie einen Datapump-Ordner, der die Import-Dump-Dateien enthält
2. Melden Sie sich beim Datenbankschema an, das den Export durchführen soll.	5. Melden Sie sich beim Datenbankschema an, das den Import durchführt.
3. Erstellen Sie ein Verzeichnis, das auf Schritt 1 zeigt.	6. Erstellen Sie ein Verzeichnis, das auf Schritt 4 zeigt.
7. Führen Sie die Exportanweisungen aus.	
8. Kopieren Sie die Sicherungskopien / SCP-Dateien auf den Zielserver.	
	9. Führen Sie Importanweisungen aus
	10. Daten überprüfen, ungültige Objekte

Quellserver [Daten exportieren]	Zielserver [Daten importieren]
	kompilieren und entsprechende Zuwendungen bereitstellen

Kopieren Sie Tabellen zwischen verschiedenen Schemata und Tabellenbereichen

```
 expdp <bkpadmin>/<bkp123> directory=DATAPUMP_REMOTE_DIR dumpfile=<customer.dmp>


 impdp <bkpadmin>/<bkp123> directory=DATAPUMP_REMOTE_DIR dumpfile=<customer.dmp>
remap_schema=<source schema>:<target schema> remap_tablespace=<source tablespace>:<target
tablespace>
```

Datenpumpe online lesen: https://riptutorial.com/de/oracle/topic/9391/datenpumpe

Kapitel 7: Datenwörterbuch

Bemerkungen

Mit den Data Dictionary-Ansichten, auch als Katalogsichten bezeichnet, können Sie den Status der Datenbank in Echtzeit überwachen:

Die Ansichten, denen `USER_` , `ALL_` und `DBA_` vorangestellt `USER_` , `ALL_` Informationen zu `DBA_` , die Ihnen gehören (`USER_`), auf die Sie `ALL_` (`ALL_`) oder auf die ein Benutzer mit SYSDBA-Berechtigung (`DBA_`) `DBA_` . Beispielsweise zeigt die Sicht `ALL_TABLES` alle Tabellen, für die Sie Berechtigungen haben.

Die `V$` -Ansichten enthalten leistungsbezogene Informationen.

Die `_PRIVS` Ansichten zeigen Berechtigungsinformationen für verschiedene Kombinationen von Benutzern, Rollen und Objekten an.

Oracle-Dokumentation: Katalogsichten / Datenwörterbuchansichten

Examples

Textquelle der gespeicherten Objekte

`USER_SOURCE` beschreibt die Textquelle der gespeicherten Objekte, die dem aktuellen Benutzer gehören. Diese Ansicht zeigt die Spalte `OWNER` .

```
select * from user_source where type='TRIGGER' and lower(text) like '%order%'
```

`ALL_SOURCE` beschreibt die Textquelle der gespeicherten Objekte, auf die der aktuelle Benutzer `ALL_SOURCE` .

```
select * from all_source where owner=:owner
```

`DBA_SOURCE` beschreibt die Textquelle aller in der Datenbank gespeicherten Objekte.

```
select * from dba_source
```

Liste aller Tabellen in Oracle abrufen

```
select owner, table_name
from all_tables
```

`ALL_TAB_COLUMNS` beschreibt die Spalten der Tabellen, Ansichten und Cluster, auf die der aktuelle Benutzer `ALL_TAB_COLUMNS` . `COLS` ist ein Synonym für `USER_TAB_COLUMNS` .

```
select *
from all_tab_columns
where table_name = :tname
```

Informationen zu Privilegien

Alle Rollen, die dem Benutzer gewährt werden.

```
select *
from dba_role_privs
where grantee= :username
```

Dem Benutzer gewährte Berechtigungen:

1. Systemprivilegien

```
select *
from dba_sys_privs
where grantee = :username
```

2. Objektzuschüsse

```
select *
from dba_tab_privs
where grantee = :username
```

Berechtigungen, die Rollen erteilt wurden.

Rollen, die anderen Rollen gewährt werden.

```
select *
from role_role_privs
where role in (select granted_role from dba_role_privs where grantee= :username)
```

1. Systemprivilegien

```
select *
from role_sys_privs
where role in (select granted_role from dba_role_privs where grantee= :username)
```

2. Objektzuschüsse

```
select *
from role_tab_privs
where role in (select granted_role from dba_role_privs where grantee= :username)
```

Oracle-Version

```
select *
from v$version
```

Beschreibt alle Objekte in der Datenbank.

```
select *
from dba_objects
```

Um alle Data Dictionary-Ansichten anzuzeigen, auf die Sie Zugriff haben

```
select * from dict
```

Datenwörterbuch online lesen: https://riptutorial.com/de/oracle/topic/7347/datenworterbuch

Kapitel 8: DUAL-Tisch

Bemerkungen

`DUAL` Tabelle `DUAL` hat eine Spalte `DUMMY` , definiert als `VARCHAR2(1)` und nur eine Zeile mit einem Wert `x` .

`DUAL` Tabelle wird automatisch im `SYS` Schema erstellt, wenn die Datenbank erstellt wird. Sie können von jedem Schema aus darauf zugreifen.

Sie können die `DUAL` Tabelle nicht ändern.

Mit der `DUAL` Tabelle können Sie eine beliebige Funktion aus der SQL-Anweisung aufrufen. Es ist nützlich, weil es nur eine Zeile hat und der Oracle Optimizer alles darüber weiß.

Examples

Das folgende Beispiel gibt das aktuelle Datum und die Uhrzeit des Betriebssystems zurück

```
select sysdate from dual
```

Das folgende Beispiel generiert Zahlen zwischen start_value und end_value

```
select :start_value + level -1 n
from dual
connect by level <= :end_value  - :start_value + 1
```

DUAL-Tisch online lesen: https://riptutorial.com/de/oracle/topic/7328/dual-tisch

Kapitel 9: Dynamisches SQL

Einführung

Mit Dynamic SQL können Sie einen SQL-Abfragecode zur Laufzeit zusammenstellen. Diese Technik hat einige Nachteile und muss sehr sorgfältig angewendet werden. Gleichzeitig können Sie komplexere Logik implementieren. PL / SQL erfordert, dass alle im Code verwendeten Objekte zum Zeitpunkt der Kompilierung vorhanden und gültig sein müssen. Aus diesem Grund können Sie DDL-Anweisungen nicht direkt in PL / SQL ausführen. Dynamisches SQL ermöglicht dies jedoch.

Bemerkungen

Einige wichtige Anmerkungen:

1. Verwenden Sie niemals Zeichenfolgenverkettung, um Werte zur Abfrage hinzuzufügen. Verwenden Sie stattdessen Parameter. Das ist falsch:

   ```
   execute immediate 'select value from my_table where id = ' ||
        id_valiable into result_variable;
   ```

   Und das ist richtig:

   ```
   execute immediate 'select value from my_table where id = :P '
       using id_valiable into result_variable;
   ```

   Dafür gibt es zwei Gründe. Der erste ist die Sicherheit. String-Verkettung ermöglicht die SQL-Injection. Wenn in der folgenden Abfrage eine Variable den Wert `1 or 1 = 1`, aktualisiert die `UPDATE` Anweisung alle Zeilen in der Tabelle:

   ```
   execute immediate 'update my_table set value = ''I have bad news for you'' where id = '
   || id;
   ```

   Der zweite Grund ist die Leistung. Oracle analysiert die Abfrage bei jeder Ausführung ohne Parameter, während die Abfrage mit dem Parameter in der Sitzung nur einmal analysiert wird.

2. Beachten Sie, dass das Datenbankmodul beim Ausführen einer DDL-Anweisung zuvor ein implizites Commit ausführt.

Examples

Wählen Sie einen Wert mit dynamischem SQL

Angenommen, ein Benutzer möchte Daten aus verschiedenen Tabellen auswählen. Eine Tabelle

wird vom Benutzer angegeben.

```
 function get_value(p_table_name varchar2, p_id number) return varchar2 is
    value varchar2(100);
  begin
    execute immediate 'select column_value from ' || p_table_name ||
                      ' where id = :P' into value using p_id;
    return value;
  end;
```

Rufen Sie diese Funktion wie gewohnt auf:

```
declare
  table_name varchar2(30) := 'my_table';
  id number := 1;
begin
  dbms_output.put_line(get_value(table_name, id));
end;
```

Tabelle zum Testen:

```
create table my_table (id number, column_value varchar2(100));
insert into my_table values (1, 'Hello, world!');
```

Werte in dynamisches SQL einfügen

Das folgende Beispiel fügt einen Wert aus dem vorherigen Beispiel in die Tabelle ein:

```
declare
  query_text varchar2(1000) := 'insert into my_table(id, column_value) values (:P_ID,
:P_VAL)';
  id number := 2;
  value varchar2(100) := 'Bonjour!';
begin
  execute immediate query_text using id, value;
end;
/
```

Aktualisieren Sie die Werte in dynamischem SQL

Lassen Sie uns die Tabelle aus dem ersten Beispiel aktualisieren:

```
declare
  query_text varchar2(1000) := 'update my_table set column_value = :P_VAL where id = :P_ID';
  id number := 2;
  value varchar2(100) := 'Bonjour le monde!';
begin
  execute immediate query_text using value, id;
end;
/
```

DDL-Anweisung ausführen

Dieser Code erstellt die Tabelle:

```
begin
  execute immediate 'create table my_table (id number, column_value varchar2(100))';
end;
/
```

Anonymen Block ausführen

Sie können eine anonyme Sperre ausführen. Dieses Beispiel zeigt auch, wie ein Wert aus dynamischem SQL zurückgegeben wird:

```
declare
  query_text varchar2(1000) := 'begin :P_OUT := cos(:P_IN); end;';
  in_value number := 0;
  out_value number;
begin
  execute immediate query_text using out out_value, in in_value;
  dbms_output.put_line('Result of anonymous block: ' || to_char(out_value));
end;
/
```

Dynamisches SQL online lesen: https://riptutorial.com/de/oracle/topic/10905/dynamisches-sql

Kapitel 10: Echte Anwendungssicherheit

Einführung

Oracle Real Application Security wurde in Oracle 12c eingeführt. Es fasst viele Sicherheitsthemen wie Benutzerrollenmodell, Zugriffskontrolle, Anwendung vs. Datenbank, Endbenutzer-Sicherheit oder Zeilen- und Spaltensicherheit zusammen

Examples

Anwendung

Um eine Anwendung mit etwas in der Datenbank zu verknüpfen, gibt es drei Hauptteile:

Anwendungsprivileg: Ein Anwendungsprivileg beschreibt Privilegien wie `SELECT` , `INSERT` , `UPDATE` , `DELETE` , ... Anwendungsprivilegien können als Aggregatprivileg zusammengefasst werden.

```
XS$PRIVILEGE(
    name=>'privilege_name'
    [, implied_priv_list=>XS$NAME_LIST('"SELECT"', '"INSERT"', '"UPDATE"', '"DELETE"')]
)

XS$PRIVILEGE_LIST(
    XS$PRIVILEGE(...),
    XS$PRIVILEGE(...),
    ...
);
```

Anwendungsbenutzer:

Benutzer der einfachen Anwendung:

```
BEGIN
    SYS.XS_PRINCIPAL.CREATE_USER('user_name');
END;
```

Anwendungsbenutzer für direkte Anmeldung:

```
BEGIN
    SYS.XS_PRINCIPAL.CREATE_USER(name => 'user_name', schema => 'schema_name');
END;

BEGIN
    SYS.XS_PRINCIPAL.SET_PASSWORD('user_name', 'password');
END;
CREATE PROFILE prof LIMIT
    PASSWORD_REUSE_TIME 1/4440
    PASSWORD_REUSE_MAX 3
    PASSWORD_VERIFY_FUNCTION Verify_Pass;
```

```
BEGIN
    SYS.XS_PRINCIPAL.SET_PROFILE('user_name', 'prof');
END;

BEGIN
    SYS.XS_PRINCIPAL.GRANT_ROLES('user_name', 'XSONNCENT');
END;
```

(wahlweise:)

```
BEGIN
    SYS.XS_PRINCIPAL.SET_VERIFIER('user_name', '6DFF060084ECE67F', XS_PRINCIPAL.XS_SHA512");
END;
```

Anwendungsrolle:

Regelmäßige Anwendungsrolle:

```
DECLARE
    st_date TIMESTAMP WITH TIME ZONE;
    ed_date TIMESTAMP WITH TIME ZONE;
BEGIN
    st_date := SYSTIMESTAMP;
    ed_date := TO_TIMESTAMP_TZ('2013-06-18 11:00:00 -5:00','YYYY-MM-DD HH:MI:SS');
    SYS.XS_PRINCIPAL.CREATE_ROLE
        (name => 'app_regular_role',
        enabled => TRUE,
        start_date => st_date,
        end_date => ed_date);
END;
```

Dynamische Anwendungsrolle: (wird dynamisch aktiviert, basierend auf dem Status der Authensierung)

```
BEGIN
    SYS.XS_PRINCIPAL.CREATE_DYNAMIC_ROLE
        (name => 'app_dynamic_role',
        duration => 40,
        scope => XS_PRINCIPAL.SESSION_SCOPE);
END;
```

Vordefinierte Anwendungsrollen:

Regulär:

- XSPUBLIC
- XSBYPASS
- XSSESSIONADMIN
- XSNAMESPACEADMIN
- XSPROVISIONER
- XSCACHEADMIN
- XSDISPATCHER

Dynamisch: (abhängig vom Authentifizierungsstatus des Anwendungsbenutzers)

- `DBMS_AUTH` : (Direktanmeldung oder andere Datenbankauthentifizierungsmethode)
- `EXTERNAL_DBMS_AUTH` : (Direktanmeldung oder andere Datenbankauthentifizierungsmethode und Benutzer ist extern)
- `DBMS_PASSWD` : (Direktanmeldung mit Passwort)
- `MIDTIER_AUTH` : (Authentifizierung durch Middle-Tier-Anwendung)
- `XSAUTHENTICATED` : (direkte oder mittelschichtige Anwendung)
- `XSSWITCH` : (Benutzer wechselt vom Proxy-Benutzer zum Anwendungsbenutzer)

Echte Anwendungssicherheit online lesen: https://riptutorial.com/de/oracle/topic/10864/echte-anwendungssicherheit

Kapitel 11: Einen Kontext erstellen

Syntax

- CREATE [ODER REPLACE] CONTEXT-Namespace USING [Schema.] Paket;
- CREATE [ODER REPLACE] CONTEXT-Namespace USING [Schema.] Paket INITIALIZED EXTERNALLY;
- CREATE [ODER REPLACE] CONTEXT-Namespace USING [Schema.] Paket INITIALIZED GLOBALLY;
- CREATE [ODER REPLACE] CONTEXT-Namespace USING [Schema.] Paket ACCESSED GLOBALLY;

Parameter

Parameter	Einzelheiten
`OR REPLACE`	Definieren Sie einen vorhandenen Kontext-Namespace neu
Namensraum	Name des Kontexts - Dies ist der Namespace für Aufrufe von `SYS_CONTEXT`
Schema	Besitzer des Pakets
Paket	Datenbankpaket, das die Kontextattribute setzt oder zurücksetzt. Hinweis: Das Datenbankpaket muss nicht vorhanden sein, um den Kontext zu erstellen.
`INITIALIZED`	Geben Sie eine andere Entität als Oracle Database an, die den Kontext festlegen kann.
`EXTERNALLY`	Erlauben Sie der OCI-Schnittstelle, den Kontext zu initialisieren.
`GLOBALLY`	Erlauben Sie dem LDAP-Verzeichnis, den Kontext beim Einrichten der Sitzung zu initialisieren.
`ACCESSED GLOBALLY`	Zugriff auf den Kontext in der gesamten Instanz zulassen - Mehrere Sitzungen können sich die Attributwerte teilen, sofern sie dieselbe Client-ID haben.

Bemerkungen

Oracle-Dokumentation (12cR1):
http://docs.oracle.com/database/121/SQLRF/statements_5003.htm

Examples

Erstellen Sie einen Kontext

```
CREATE CONTEXT my_ctx USING my_pkg;
```

Dadurch wird ein Kontext erstellt, der nur durch Routinen im Datenbankpaket `my_pkg` , z.

```
CREATE PACKAGE my_pkg AS
  PROCEDURE set_ctx;
END my_pkg;

CREATE PACKAGE BODY my_pkg AS
  PROCEDURE set_ctx IS
  BEGIN
    DBMS_SESSION.set_context('MY_CTX','THE KEY','Value');
    DBMS_SESSION.set_context('MY_CTX','ANOTHER','Bla');
  END set_ctx;
END my_pkg;
```

Wenn nun eine Sitzung dies tut:

```
my_pkg.set_ctx;
```

Es kann jetzt den Wert für den Schlüssel abrufen:

```
SELECT SYS_CONTEXT('MY_CTX','THE KEY') FROM dual;

Value
```

Einen Kontext erstellen online lesen: https://riptutorial.com/de/oracle/topic/2088/einen-kontext-erstellen

Kapitel 12: Einschränken der von einer Abfrage zurückgegebenen Zeilen (Paginierung)

Examples

Erste N Zeilen mit Zeilenbegrenzungsklausel erhalten

Die `FETCH` Klausel wurde in Oracle 12c R1 eingeführt:

```
SELECT   val
FROM     mytable
ORDER BY val DESC
FETCH FIRST 5 ROWS ONLY;
```

Ein Beispiel ohne FETCH, das auch in früheren Versionen funktioniert:

```
SELECT * FROM (
   SELECT   val
   FROM     mytable
   ORDER BY val DESC
) WHERE ROWNUM <= 5;
```

Pagination in SQL

```
SELECT val
FROM   (SELECT val, rownum AS rnum
        FROM   (SELECT val
                FROM   rownum_order_test
                ORDER BY val)
        WHERE rownum <= :upper_limit)
WHERE  rnum >= :lower_limit ;
```

Auf diese Weise können wir die Tabellendaten wie eine Web-Serch-Seite paginieren

Holen Sie sich N Anzahl Datensätze aus der Tabelle

Wir können mit rownum-Klauseln keine Zeilen aus dem Ergebnis einschränken

```
select * from
(
  select val from  mytable
) where rownum<=5
```

Wenn wir den ersten oder letzten Datensatz wünschen, möchten wir eine order by-Klausel in der inneren Abfrage, die ein auf der Reihenfolge basierendes Ergebnis liefert.

Letzte fünf Aufzeichnung:

```
select * from
(
    select val from  mytable order by val desc
) where rownum<=5
```

Die ersten fünf Platten

```
select * from
(
    select val from  mytable order by val
) where rownum<=5
```

Holen Sie sich Zeile N bis M aus vielen Zeilen (vor Oracle 12c).

Verwenden Sie die analytische Funktion row_number ():

```
with t as (
  select col1
  , col2
  , row_number() over (order by col1, col2) rn
  from table
)
select col1
, col2
from t
where rn between N and M; -- N and M are both inclusive
```

Oracle 12c kann dies mit `OFFSET` und `FETCH` einfacher `OFFSET` .

Einige Zeilen überspringen, dann einige nehmen

In Oracle 12g +

```
SELECT Id, Col1
FROM TableName
ORDER BY Id
OFFSET 20 ROWS FETCH NEXT 20 ROWS ONLY;
```

In früheren Versionen

```
SELECT Id,
   Col1
 FROM (SELECT Id,
           Col1,
           row_number() over (order by Id) RowNumber
      FROM TableName)
WHERE RowNumber BETWEEN 21 AND 40
```

Einige Zeilen aus dem Ergebnis überspringen

In Oracle 12g +

```
SELECT Id, Col1
FROM TableName
ORDER BY Id
OFFSET 5 ROWS;
```

In früheren Versionen

```
SELECT Id,
   Col1
 FROM (SELECT Id,
           Col1,
           row_number() over (order by Id) RowNumber
      FROM TableName)
WHERE RowNumber > 20
```

Einschränken der von einer Abfrage zurückgegebenen Zeilen (Paginierung) online lesen: https://riptutorial.com/de/oracle/topic/4300/einschranken-der-von-einer-abfrage-zuruckgegebenen-zeilen--paginierung-

Kapitel 13: Einschränkungen

Examples

Aktualisieren Sie Fremdschlüssel mit einem neuen Wert in Oracle

Angenommen, Sie haben eine Tabelle und möchten eine dieser primären Tabellen-ID ändern. Sie können das folgende Scrpit verwenden. primäre ID hier ist "PK_S"

```
begin
  for i in (select a.table_name, c.column_name
              from user_constraints a, user_cons_columns c
             where a.CONSTRAINT_TYPE = 'R'
               and a.R_CONSTRAINT_NAME = 'PK_S'
               and c.constraint_name = a.constraint_name) loop


         execute immediate 'update ' || i.table_name || ' set ' || i.column_name ||
                     '=to_number(''1000'' || ' ||  i.column_name || ') ';



  end loop;

end;
```

Deaktivieren Sie alle zugehörigen Fremdschlüssel in Oracle

Angenommen, Sie haben die Tabelle T1 und sie hat eine Beziehung zu vielen Tabellen. Der Name der Primärschlüssel-Einschränkung lautet "pk_t1".

```
Begin
    For I in (select table_name, constraint_name from user_constraint t where
r_constraint_name='pk_t1') loop

Execute immediate ' alter table ' || I.table_name || ' disable constraint ' ||
i.constraint_name;

   End loop;
End;
```

Einschränkungen online lesen: https://riptutorial.com/de/oracle/topic/6040/einschrankungen

Kapitel 14: Fehlerprotokollierung

Examples

Fehlerprotokollierung beim Schreiben in die Datenbank

Erstellen Sie die Oracle-Fehlerprotokolltabelle ERR $ _EXAMPLE für die vorhandene BEISPIEL-Tabelle:

```
EXECUTE DBMS_ERRLOG.CREATE_ERROR_LOG('EXAMPLE', NULL, NULL, NULL, TRUE);
```

Schreibvorgang mit SQL durchführen:

```
insert into EXAMPLE (COL1) values ('example')
LOG ERRORS INTO ERR$_EXAMPLE reject limit unlimited;
```

Fehlerprotokollierung online lesen: https://riptutorial.com/de/oracle/topic/3505/fehlerprotokollierung

Kapitel 15: Fensterfunktionen

Syntax

- Ratio_To_Report (expr) OVER (query_partition_clause)

Examples

Ratio_To_Report

Gibt das Verhältnis des aktuellen Zeilenwerts zu allen Werten innerhalb des Fensters an.

```
--Data
CREATE TABLE Employees (Name Varchar2(30), Salary Number(10));
INSERT INTO Employees Values ('Bob',2500);
INSERT INTO Employees Values ('Alice',3500);
INSERT INTO Employees Values ('Tom',2700);
INSERT INTO Employees Values ('Sue',2000);
--Query
SELECT Name, Salary, Ratio_To_Report(Salary) OVER () As Ratio
FROM Employees
ORDER BY Salary, Name, Ratio;
--Output
NAME                           SALARY      RATIO
------------------------------ ---------- ----------
Sue                                  2000 .186915888
Bob                                  2500  .23364486
Tom                                  2700 .252336449
Alice                                3500 .327102804
```

Fensterfunktionen online lesen: https://riptutorial.com/de/oracle/topic/6669/fensterfunktionen

Kapitel 16: Hierarchischer Abruf mit Oracle Database 12C

Einführung

Sie können hierarchische Abfragen verwenden, um Daten basierend auf einer natürlichen hierarchischen Beziehung zwischen Zeilen in einer Tabelle abzurufen

Examples

Verwendung von CONNECT BY Caluse

```
SELECT E.EMPLOYEE_ID,E.LAST_NAME,E.MANAGER_ID FROM HR.EMPLOYEES E
CONNECT BY PRIOR E.EMPLOYEE_ID = E.MANAGER_ID;
```

Die `CONNECT BY` Klausel zur Definition der Beziehung zwischen Mitarbeitern und Führungskräften.

Festlegen der Richtung der Abfrage von oben nach unten

```
SELECT E.LAST_NAME|| ' reports to ' ||
PRIOR E.LAST_NAME "Walk Top Down"
FROM HR.EMPLOYEES E
START WITH E.MANAGER_ID IS NULL
CONNECT BY PRIOR E.EMPLOYEE_ID = E.MANAGER_ID;
```

Hierarchischer Abruf mit Oracle Database 12C online lesen:
https://riptutorial.com/de/oracle/topic/8777/hierarchischer-abruf-mit-oracle-database-12c

Kapitel 17: Hinweise

Parameter

Parameter	Einzelheiten
Parallelitätsgrad (DOP)	Es ist die Anzahl paralleler Verbindungen / Prozesse, die Ihre Abfrage öffnen soll. Es ist normalerweise 2, 4, 8, 16 so weiter.
Tabellenname	Der Name der Tabelle, auf die der Parallelhinweis angewendet wird.

Examples

Paralleler Hinweis

Parallele Hinweise auf Anweisungsebene sind am einfachsten:

```
SELECT /*+ PARALLEL(8) */ first_name, last_name FROM employee emp;
```

Parallele Hinweise auf Objektebene geben mehr Kontrolle, sind aber anfälliger für Fehler. Entwickler vergessen häufig, den Aliasnamen anstelle des Objektnamens zu verwenden, oder vergessen, einige Objekte einzuschließen.

```
SELECT /*+ PARALLEL(emp,8) */ first_name, last_name FROM employee emp;
```

```
SELECT /*+ PARALLEL(table_alias,Degree of Parallelism) */ FROM table_name table_alias;
```

Angenommen, es dauert 100 Sekunden, bis eine Abfrage ausgeführt wird, ohne einen parallelen Hinweis zu verwenden. Wenn wir DOP für dieselbe Abfrage auf 2 ändern, *dauert* die Abfrage mit Parallelhinweis *idealerweise* 50 Sekunden. Die Verwendung von DOP als 4 dauert ebenfalls 25 Sekunden.

In der Praxis hängt die parallele Ausführung von vielen anderen Faktoren ab und skaliert nicht linear. Dies gilt insbesondere für kleine Laufzeiten, bei denen der parallele Overhead möglicherweise größer ist als die Gewinne, die auf mehreren parallelen Servern auftreten.

USE_NL

Verwenden Sie verschachtelte Schleifen.

Verwendung: `use_nl(AB)`

Dieser Hinweis fordert die Engine auf, die Tabellen A und B mit der Nested-Loop-Methode zu verknüpfen. Der Hinweis erzwingt nicht die Reihenfolge der Verknüpfung, fragt nur nach NL.

```
SELECT /*+use_nl(e d)*/ *
FROM Employees E
JOIN Departments D on E.DepartmentID = D.ID
```

APPEND TIPP

Msgstr "Verwenden Sie die Methode DIRECT PATH zum Einfügen neuer Zeilen".

Der `APPEND` Hinweis weist die Engine an, die direkte `APPEND` zu verwenden. Dies bedeutet, dass die Engine keine herkömmliche Einfügung verwendet, die Speicherstrukturen und Standardsperren verwendet, sondern die Daten direkt in den Tablespace schreibt. Erstellt immer neue Blöcke, die an das Tabellensegment angehängt werden. Dies wird schneller sein, hat aber einige Einschränkungen:

- Sie können nicht aus der Tabelle lesen, die Sie in derselben Sitzung angefügt haben, bis Sie die Transaktion übertragen oder zurücksetzen.
- Wenn in der Tabelle Auslöser definiert sind, verwendet Oracle keinen direkten Pfad (dies ist eine andere Geschichte für das Laden von sqlldr).
- Andere

Beispiel.

```
INSERT /*+append*/ INTO Employees
SELECT *
FROM Employees;
```

USE_HASH

Weist die Engine an, die Hash-Methode zum Verknüpfen von Tabellen im Argument zu verwenden.

Verwendung: `use_hash(TableA [TableB] ... [TableN])`

Wie bereits an vielen Stellen erläutert , "greift Oracle in einem HASH-Join auf eine Tabelle (normalerweise auf das kleinere der verbundenen Ergebnisse) auf und erstellt eine Hashtabelle für den Join-Schlüssel im Speicher. Dann durchsucht er die andere Tabelle im Join (normalerweise die größere) one) und prüft die Hashtabelle auf Übereinstimmungen. "

Dies ist der Nested Loops-Methode vorzuziehen, wenn die Tabellen groß sind, keine Indizes vorhanden sind usw.

Hinweis : Der Hinweis erzwingt nicht die Reihenfolge der Verknüpfung, sondern fragt nur nach der HASH JOIN-Methode.

Verwendungsbeispiel:

```
SELECT /*+use_hash(e d)*/ *
FROM Employees E
JOIN Departments D on E.DepartmentID = D.ID
```

VOLL

Der FULL-Hinweis weist Oracle an, eine vollständige Tabellensuche für eine angegebene Tabelle durchzuführen, unabhängig davon, ob ein Index verwendet werden kann.

```
create table fullTable(id) as select level from dual connect by level < 100000;
create index idx on fullTable(id);
```

Ohne Hinweise wird der Index verwendet:

```
select count(1) from fullTable f where id between 10 and 100;
-------------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |     1 |    13 |     3   (0)| 00:00:01 |
|   1 |  SORT AGGREGATE   |      |     1 |    13 |            |          |
|*  2 |   INDEX RANGE SCAN| IDX  |     2 |    26 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

FULL-Hinweis erzwingt einen vollständigen Scan:

```
select /*+ full(f) */ count(1) from fullTable f where id between 10 and 100;
-----------------------------------------------------------------------------------
| Id  | Operation          | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |           |     1 |    13 |    47   (3)| 00:00:01 |
|   1 |  SORT AGGREGATE    |           |     1 |    13 |            |          |
|*  2 |   TABLE ACCESS FULL| FULLTABLE |     2 |    26 |    47   (3)| 00:00:01 |
-----------------------------------------------------------------------------------
```

Ergebnis-Cache

Oracle (*11g und höher*) ermöglicht, dass die SQL-Abfragen im SGA zwischengespeichert und zur Verbesserung der Leistung wiederverwendet werden. Es fragt die Daten aus dem Cache ab und nicht aus der Datenbank. Die anschließende Ausführung derselben Abfrage ist schneller, da die Daten jetzt aus dem Cache gezogen werden.

```
SELECT /*+ result_cache */ number FROM main_table;
```

Ausgabe -

```
Number
------
   1
   2
   3
   4
   5
   6
   7
   8
   9
```

```
   10

Elapsed: 00:00:02.20
```

Wenn ich dieselbe Abfrage jetzt erneut ausführen, wird die Ausführungszeit reduziert, da die Daten jetzt aus dem Cache abgerufen werden, der bei der ersten Ausführung festgelegt wurde.

Ausgabe -

```
Number
------
   1
   2
   3
   4
   5
   6
   7
   8
   9
   10

Elapsed: 00:00:00.10
```

Beachten Sie, wie die verstrichene Zeit von **2,20 Sekunden** auf **0,10 Sekunden** gesunken ist.

> Ergebnis Der Cache speichert den Cache, bis die Daten in der Datenbank aktualisiert / geändert / gelöscht wurden. Bei jeder Änderung wird der Cache freigegeben.

Hinweise online lesen: https://riptutorial.com/de/oracle/topic/1490/hinweise

Kapitel 18: Indizes

Einführung

Hier werde ich verschiedene Indizes anhand von Beispielen erklären, wie der Index die Abfrageleistung erhöht, wie der Index die DML-Leistung verringert usw.

Examples

B-Tree-Index

```
CREATE INDEX ord_customer_ix ON orders (customer_id);
```

Wenn wir nichts erwähnen, erstellt oracle standardmäßig einen Index als B-Tree-Index. Aber wir sollten wissen, wann wir damit umgehen sollen. Der B-Tree-Index speichert Daten als binäres Baumformat. Wie wir wissen, ist index ein Schemaobjekt, das für jeden Wert für die indizierte Spalte eine Art Eintrag speichert. Immer wenn in diesen Spalten gesucht wird, wird im Index nach dem genauen Ort des Datensatzes gesucht, um schnell darauf zugreifen zu können. Einige Punkte zur Indizierung:

- Um nach einem Eintrag im Index zu suchen, wird eine Art binärer Suchalgorithmus verwendet.
- Wenn **die Kardinalität von Daten hoch ist, ist der B-Tree-** Index perfekt zu verwenden.
- Durch den Index wird DML langsam, da für jeden Datensatz ein Eintrag für die indizierte Spalte im Index vorhanden sein sollte.
- Wenn dies nicht notwendig ist, sollten wir es daher vermeiden, einen Index zu erstellen.

Bitmap-Index

```
CREATE BITMAP INDEX
emp_bitmap_idx
ON index_demo (gender);
```

- Der Bitmap-Index wird verwendet, wenn **die Kardinalität der Daten niedrig ist.**
- Hier hat **Gender** einen Wert mit geringer Kardinalität. Werte sind möglicherweise männlich, weiblich und andere.
- Wenn wir also während der Suche einen binären Baum für diese 3 Werte erstellen, wird dies unnötig durchlaufen.
- In Bitmap-Strukturen wird ein zweidimensionales Array mit einer Spalte für jede Zeile der indizierten Tabelle erstellt. Jede Spalte repräsentiert einen bestimmten Wert innerhalb des Bitmap-Index. Dieses zweidimensionale Array stellt jeden Wert innerhalb des Index dar, multipliziert mit der Anzahl der Zeilen in der Tabelle.
- Zum Zeitpunkt des Zeilenabrufs dekomprimiert Oracle die Bitmap in die RAM-Datenpuffer, sodass sie schnell nach übereinstimmenden Werten durchsucht werden kann. Diese

Übereinstimmungswerte werden in Form einer Row-ID-Liste an Oracle übermittelt. Diese Row-ID-Werte können direkt auf die erforderlichen Informationen zugreifen.

Funktionsbasierter Index

```
CREATE INDEX first_name_idx ON user_data (UPPER(first_name));


SELECT *
FROM   user_data
WHERE  UPPER(first_name) = 'JOHN2';
```

- Funktionsbasierter Index bedeutet, Index basierend auf einer Funktion zu erstellen.
- Wenn in search (where-Klausel) häufig eine Funktion verwendet wird, ist es besser, einen Index zu erstellen, der auf dieser Funktion basiert.
- In diesem Beispiel wird für die Suche die Funktion **Upper ()** verwendet. Daher ist es besser, einen Index mit der oberen Funktion zu erstellen.

Indizes online lesen: https://riptutorial.com/de/oracle/topic/9978/indizes

Kapitel 19: Mit Datumsangaben arbeiten

Examples

Datums-Arithmetik

Oracle unterstützt die Datentypen `DATE` (enthält die Zeit bis zur nächsten Sekunde) und `TIMESTAMP` (enthält die Zeit bis zum Bruchteil einer Sekunde), wodurch die Arithmetik (Addition und Subtraktion) nativ möglich ist. Zum Beispiel:

Um den nächsten Tag zu bekommen:

```
select to_char(sysdate + 1, 'YYYY-MM-DD') as tomorrow from dual;
```

Um den Vortag zu erhalten:

```
select to_char(sysdate - 1, 'YYYY-MM-DD') as yesterday from dual;
```

So fügen Sie dem aktuellen Datum 5 Tage hinzu:

```
select to_char(sysdate + 5, 'YYYY-MM-DD') as five_days_from_now from dual;
```

So fügen Sie dem aktuellen Datum 5 Stunden hinzu:

```
select to_char(sysdate + (5/24), 'YYYY-MM-DD HH24:MI:SS') as five_hours_from_now from dual;
```

So fügen Sie 10 Minuten zum aktuellen Datum hinzu:

```
select to_char(sysdate + (10/1440), 'YYYY-MM-DD HH24:MI:SS') as ten_mintues_from_now from
dual;
```

So fügen Sie dem aktuellen Datum 7 Sekunden hinzu:

```
select to_char(sysdate + (7/86400), 'YYYY-MM-DD HH24:MI:SS') as seven_seconds_from_now from
dual;
```

So wählen Sie Zeilen aus, für die `hire_date` vor 30 Tagen oder mehr liegt:

```
select * from emp where hire_date < sysdate - 30;
```

So wählen Sie Zeilen aus, in `last_updated` Spalte `last_updated` in der letzten Stunde befindet:

```
select * from logfile where last_updated >= sysdate - (1/24);
```

Oracle bietet auch den integrierten Datentyp `INTERVAL` der eine Zeitdauer darstellt (z. B. 1,5 Tage,

36 Stunden, 2 Monate usw.). Diese können auch mit Arithmetik mit verwendet werden `DATE` und `TIMESTAMP` Ausdrücke. Zum Beispiel:

```
select * from logfile where last_updated >= sysdate - interval '1' hour;
```

Add_months-Funktion

Syntax: `add_months(p_date, integer) return date;`

Add_months-Funktion fügt dem Datum von p_date einen Monat hinzu.

```
SELECT add_months(date'2015-01-12', 2) m FROM dual;
```

M
2015-03-12

Sie können auch subtrahieren Monate mit einem negativen `amt`

```
SELECT add_months(date'2015-01-12', -2) m FROM dual;
```

M
2014-11-12

Wenn der berechnete Monat weniger Tage als das angegebene Datum hat, wird der letzte Tag des berechneten Monats zurückgegeben.

```
SELECT to_char( add_months(date'2015-01-31', 1),'YYYY-MM-DD') m FROM dual;
```

M
2015-02-28

Mit Datumsangaben arbeiten online lesen: https://riptutorial.com/de/oracle/topic/768/mit-datumsangaben-arbeiten

Kapitel 20: Oracle Advanced Queuing (AQ)

Bemerkungen

- Verwenden Sie niemals DDL oder DML für Tabellen, die mit `dbms_aqadm.create_queue_table` erstellt wurden. Verwenden Sie nur dbms_aqadm und dbms_aq, um mit diesen Tabellen zu arbeiten. Oracle erstellt möglicherweise mehrere unterstützende Tabellen, Indizes usw., die Sie nicht kennen. Das manuelle Ausführen von DDL oder DML für die Tabelle kann zu einem Szenario führen, in dem der Oracle-Support die Tabelle und die Warteschlangen löschen und erneut erstellen muss, um die Situation zu beheben.

- Es wird dringend empfohlen, `dbms_aq.forever` für eine Wait-Option nicht zu verwenden. Dies hat in der Vergangenheit zu Problemen geführt, da Oracle möglicherweise eine übermäßige Anzahl von Worker-Jobs für die Bearbeitung der nicht benötigten Warteschlangen einplanen kann (siehe Oracle Doc ID 2001165.1).

- Es wird empfohlen, den Parameter AQ_TM_PROCESSES in Version 10.1 und höher nicht festzulegen. Vermeiden Sie insbesondere, diesen Wert auf Null zu setzen, da dadurch der QMON-Hintergrundjob deaktiviert wird, der zum Verwalten der Warteschlangen erforderlich ist. Sie können diesen Wert mit dem folgenden Befehl auf den Oracle-Standard zurücksetzen und die Datenbank neu starten. `alter system reset aq_tm_processes scope=spfile sid='*';`

Examples

Einfacher Produzent / Verbraucher

Überblick

Erstellen Sie eine Warteschlange, an die wir eine Nachricht senden können. Oracle benachrichtigt unsere gespeicherte Prozedur darüber, dass eine Nachricht in die Warteschlange aufgenommen wurde und bearbeitet werden sollte. Wir fügen auch einige Unterprogramme hinzu, die wir in einem Notfall verwenden können, um das Abmelden von Nachrichten zu verhindern, ein erneutes Löschen von Nachrichten zuzulassen und einen einfachen Batch-Job auszuführen, um alle Nachrichten durchzuarbeiten.

Diese Beispiele wurden mit Oracle Database 12c Enterprise Edition Version 12.1.0.2.0 - 64-Bit-Produktion getestet.

Warteschlange erstellen

Wir erstellen einen Nachrichtentyp, eine Warteschlangentabelle, die die Nachrichten enthalten kann, und eine Warteschlange. Nachrichten in der Warteschlange werden zuerst nach Priorität und dann nach ihrer Wartezeit aus der Warteschlange entfernt. Wenn beim Ausführen der

Nachricht etwas schief läuft und die Zurücksetzung rückgängig gemacht wird, stellt AQ die Nachricht 3600 Sekunden später für die Zurücksetzung zur Verfügung. Es wird dies 48 Mal tun, bevor es in eine Ausnahmewarteschlange verschoben wird.

```
create type message_t as object
   (
   sender  varchar2 ( 50 ),
   message varchar2 ( 512 )
   );
/
-- Type MESSAGE_T compiled
begin dbms_aqadm.create_queue_table(
     queue_table        => 'MESSAGE_Q_TBL',
     queue_payload_type => 'MESSAGE_T',
     sort_list          => 'PRIORITY,ENQ_TIME',
     multiple_consumers =>  false,
     compatible         => '10.0.0');
  end;
/
-- PL/SQL procedure successfully completed.
begin dbms_aqadm.create_queue(
     queue_name          => 'MESSAGE_Q',
     queue_table         => 'MESSAGE_Q_TBL',
     queue_type          =>  0,
     max_retries         =>  48,
     retry_delay         =>  3600,
     dependency_tracking =>  false);
  end;
/
-- PL/SQL procedure successfully completed.
```

Nun, da wir einen Platz für die Nachrichten haben, können Sie ein Paket zum Verwalten und Bearbeiten von Nachrichten in der Warteschlange erstellen.

```
create or replace package message_worker_pkg
is
   queue_name_c constant varchar2(20) := 'MESSAGE_Q';

   -- allows the workers to process messages in the queue
   procedure enable_dequeue;

   -- prevents messages from being worked but will still allow them to be created and enqueued
   procedure disable_dequeue;

   -- called only by Oracle Advanced Queueing.  Do not call anywhere else.
   procedure on_message_enqueued (context        in raw,
                                  reginfo        in sys.aq$_reg_info,
                                  descr          in sys.aq$_descriptor,
                                  payload        in raw,
                                  payloadl       in number);

   -- allows messages to be worked if we missed the notification (or a retry
   -- is pending)
   procedure work_old_messages;

end;
/
```

```
create or replace package body message_worker_pkg
is
   -- raised by Oracle when we try to dequeue but no more messages are ready to
   -- be dequeued at this moment
   no_more_messages_ex          exception;
   pragma exception_init (no_more_messages_ex,
                          -25228);

   -- allows the workers to process messages in the queue
   procedure enable_dequeue
   as
   begin
      dbms_aqadm.start_queue (queue_name => queue_name_c, dequeue => true);
   end enable_dequeue;

   -- prevents messages from being worked but will still allow them to be created and enqueued
   procedure disable_dequeue
   as
   begin
      dbms_aqadm.stop_queue (queue_name => queue_name_c, dequeue => true, enqueue => false);
   end disable_dequeue;

   procedure work_message (message_in in out nocopy message_t)
   as
   begin
      dbms_output.put_line ( message_in.sender || ' says ' || message_in.message );
   end work_message;

   -- called only by Oracle Advanced Queueing.  Do not call anywhere else.

   procedure on_message_enqueued (context        in raw,
                                  reginfo        in sys.aq$_reg_info,
                                  descr          in sys.aq$_descriptor,
                                  payload        in raw,
                                  payloadl       in number)
   as
      pragma autonomous_transaction;
      dequeue_options_l      dbms_aq.dequeue_options_t;
      message_id_l           raw (16);
      message_l              message_t;
      message_properties_l   dbms_aq.message_properties_t;
   begin
      dequeue_options_l.msgid         := descr.msg_id;
      dequeue_options_l.consumer_name := descr.consumer_name;
      dequeue_options_l.wait          := dbms_aq.no_wait;
      dbms_aq.dequeue (queue_name           => descr.queue_name,
                       dequeue_options      => dequeue_options_l,
                       message_properties   => message_properties_l,
                       payload              => message_l,
                       msgid                => message_id_l);
      work_message (message_l);
      commit;
   exception
      when no_more_messages_ex
      then
         -- it's possible work_old_messages already dequeued the message
         commit;
      when others
      then
         -- we don't need to have a raise here.  I just wanted to point out that
         -- since this will be called by AQ throwing the exception back to it
```

```
         -- will have it put the message back on the queue and retry later
         raise;
   end on_message_enqueued;

   -- allows messages to be worked if we missed the notification (or a retry
   -- is pending)
   procedure work_old_messages
   as
      pragma autonomous_transaction;
      dequeue_options_l      dbms_aq.dequeue_options_t;
      message_id_l           raw (16);
      message_l              message_t;
      message_properties_l   dbms_aq.message_properties_t;
   begin
      dequeue_options_l.wait       := dbms_aq.no_wait;
      dequeue_options_l.navigation := dbms_aq.first_message;

      while (true) loop -- way out is no_more_messages_ex
         dbms_aq.dequeue (queue_name           => queue_name_c,
                          dequeue_options      => dequeue_options_l,
                          message_properties   => message_properties_l,
                          payload              => message_l,
                          msgid                => message_id_l);
         work_message (message_l);
         commit;
      end loop;
   exception
      when no_more_messages_ex
      then
         null;
   end work_old_messages;
end;
```

Sagen Sie als Nächstes AQ, dass, wenn eine Nachricht in MESSAGE_Q in die Warteschlange eingereiht wird (und festgeschrieben wird), unsere Prozedur benachrichtigt wird, die sie zu erledigen hat. AQ startet einen Job in einer eigenen Sitzung, um dies zu erledigen.

```
begin
  dbms_aq.register (
     sys.aq$_reg_info_list (
        sys.aq$_reg_info (user || '.' || message_worker_pkg.queue_name_c,
                          dbms_aq.namespace_aq,
                          'plsql://' || user || '.message_worker_pkg.on_message_enqueued',
                          hextoraw ('FF'))),
     1);
  commit;
end;
```

Warteschlange starten und Nachricht senden

```
declare
   enqueue_options_l      dbms_aq.enqueue_options_t;
   message_properties_l   dbms_aq.message_properties_t;
   message_id_l           raw (16);
   message_l              message_t;
begin
   -- only need to do this next line ONCE
```

```
   dbms_aqadm.start_queue (queue_name => message_worker_pkg.queue_name_c, enqueue => true ,
dequeue => true);

   message_l := new message_t ( 'Jon', 'Hello, world!' );
   dbms_aq.enqueue (queue_name           => message_worker_pkg.queue_name_c,
                    enqueue_options      => enqueue_options_l,
                    message_properties   => message_properties_l,
                    payload              => message_l,
                    msgid                => message_id_l);
   commit;
end;
```

Oracle Advanced Queuing (AQ) online lesen: https://riptutorial.com/de/oracle/topic/4362/oracle-advanced-queuing--aq-

Kapitel 21: Oracle MAF

Examples

Um einen Wert von der Bindung zu erhalten

```
        ValueExpression ve = AdfmfJavaUtilities.getValueExpression(<binding>, String.class);
       String <variable_name> = (String) ve.getValue(AdfmfJavaUtilities.getELContext());
```

"Bindung" gibt hier den EL-Ausdruck an, aus dem der Wert abgerufen werden soll.

"variable_name" der Parameter, in dem der Wert aus der Bindung gespeichert werden soll

Wert auf Bindung setzen

```
        ValueExpression ve = AdfmfJavaUtilities.getValueExpression(<binding>, String.class);
        ve.setValue(AdfmfJavaUtilities.getELContext(), <value>);
```

"Bindung" gibt hier den EL-Ausdruck an, in dem der Wert gespeichert werden soll.

"value" ist der gewünschte Wert, der zur Bindung hinzugefügt werden soll

So rufen Sie eine Methode aus der Bindung auf

```
    AdfELContext adfELContext = AdfmfJavaUtilities.getAdfELContext();
    MethodExpression me;
    me = AdfmfJavaUtilities.getMethodExpression(<binding>, Object.class, new Class[] { });
    me.invoke(adfELContext, new Object[] { });
```

"Bindung" gibt den EL-Ausdruck an, von dem aus eine Methode aufgerufen werden soll

So rufen Sie eine JavaScript-Funktion auf

```
AdfmfContainerUtilities.invokeContainerJavaScriptFunction(AdfmfJavaUtilities.getFeatureId(),
<function>, new Object[] {
                                                                    });
```

"Funktion" ist die gewünschte aufzurufende js-Funktion

Oracle MAF online lesen: https://riptutorial.com/de/oracle/topic/6352/oracle-maf

Kapitel 22: Partitionierung der Tabelle

Einführung

Partitionieren ist eine Funktion zum Aufteilen von Tabellen und Indizes in kleinere Teile. Es wird verwendet, um die Leistung zu verbessern und die kleineren Teile einzeln zu verwalten. Der Partitionsschlüssel ist eine Spalte oder eine Menge von Spalten, die definiert, in welcher Partition jede Zeile gespeichert wird. Überblick über die Partitionierung in der offiziellen Oracle-Dokumentation

Bemerkungen

Partitionieren ist eine zusätzliche Option und nur für die Enterprise Edition verfügbar.

Examples

Hash-Partitionierung

Dadurch wird eine durch Hash partitionierte Tabelle erstellt, in diesem Beispiel für die Store-ID.

```
CREATE TABLE orders (
    order_nr NUMBER(15),
    user_id VARCHAR2(2),
    order_value NUMBER(15),
    store_id NUMBER(5)
)
PARTITION BY HASH(store_id) PARTITIONS 8;
```

Sie sollten eine Potenz von 2 für die Anzahl der Hash-Partitionen verwenden, um eine gleichmäßige Verteilung der Partitionsgröße zu erhalten.

Bereichsaufteilung

Dadurch wird eine nach Bereichen partitionierte Tabelle erstellt, in diesem Beispiel nach Auftragswerten.

```
CREATE TABLE orders (
    order_nr NUMBER(15),
    user_id VARCHAR2(2),
    order_value NUMBER(15),
    store_id NUMBER(5)
)
PARTITION BY RANGE(order_value) (
    PARTITION p1 VALUES LESS THAN(10),
    PARTITION p2 VALUES LESS THAN(40),
    PARTITION p3 VALUES LESS THAN(100),
    PARTITION p4 VALUES LESS THAN(MAXVALUE)
);
```

Wählen Sie vorhandene Partitionen aus

Überprüfen Sie die vorhandenen Partitionen im Schema

```
SELECT * FROM user_tab_partitions;
```

Partitionierung auflisten

Dadurch wird eine durch Listen partitionierte Tabelle erstellt, in diesem Beispiel für die Store-ID.

```
CREATE TABLE orders (
    order_nr NUMBER(15),
    user_id VARCHAR2(2),
    order_value NUMBER(15),
    store_id NUMBER(5)
)
PARTITION BY LIST(store_id) (
    PARTITION p1 VALUES (1,2,3),
    PARTITION p2 VALUES(4,5,6),
    PARTITION p3 VALUES(7,8,9),
    PARTITION p4 VALUES(10,11)
);
```

Partition ablegen

```
ALTER TABLE table_name DROP PARTITION partition_name;
```

Wählen Sie Daten von einer Partition aus

Wählen Sie Daten von einer Partition aus

```
SELECT * FROM orders PARTITION(partition_name);
```

Eine Partition abschneiden

```
ALTER TABLE table_name TRUNCATE PARTITION partition_name;
```

Benennen Sie eine Partition um

```
ALTER TABLE table_name RENAME PARTITION p3 TO p6;
```

Verschieben Sie die Partition in einen anderen Tabellenbereich

```
ALTER TABLE table_name
MOVE PARTITION partition_name TABLESPACE tablespace_name;
```

Neue Partition hinzufügen

```
ALTER TABLE table_name
ADD PARTITION new_partition VALUES LESS THAN(400);
```

Partition aufteilen

Teilt eine Partition in zwei Partitionen mit einer anderen hohen Grenze auf.

```
ALTER TABLE table_name SPLIT PARTITION old_partition
    AT (new_high_bound) INTO (PARTITION new_partition TABLESPACE new_tablespace,
    PARTITION old_partition)
```

Partitionen zusammenführen

Verbinden Sie zwei Partitionen zu einer einzigen

```
ALTER TABLE table_name
  MERGE PARTITIONS first_partition, second_partition
  INTO  PARTITION  splitted_partition TABLESPACE new_tablespace
```

Tauschen Sie eine Partition aus

Tauschen Sie eine Partition in eine nicht partitionierte Tabelle um und umgekehrt. Dies ermöglicht ein schnelles "Verschieben" von Daten zwischen den Datensegmenten (im Gegensatz zu "Einfügen ... Auswählen" oder "Erstellen einer Tabelle ... Auswahl"), da die Operation DDL ist (die Partitionsaustauschoperation ist eine Datenmenge) Wörterbuchaktualisierung ohne Verschieben der tatsächlichen Daten) und nicht DML (großer Undo / Redo-Overhead).

Grundlegende Beispiele:

1. Konvertieren Sie eine nicht partitionierte Tabelle (Tabelle "B") in eine Partition (von Tabelle "A"):

Tabelle "A" enthält keine Daten in Partition "OLD_VALUES" und Tabelle "B" enthält Daten

```
ALTER TABLE "A" EXCHANGE PARTITION "OLD_VALUES" WITH TABLE "B";
```

Ergebnis: Daten werden aus Tabelle "B" (enthält nach der Operation keine Daten) in die Partition "OLD_VALUES" verschoben.

2. Konvertieren Sie eine Partition in eine nicht partitionierte Tabelle:

Tabelle "A" enthält Daten in Partition "OLD_VALUES" und Tabelle "B" enthält keine Daten

```
ALTER TABLE "A" EXCHANGE PARTITION "OLD_VALUES" WITH TABLE "B";
```

Ergebnis: Daten werden von der Partition "OLD_VALUES" (enthält nach der Operation keine Daten) in die Tabelle "B" verschoben.

Hinweis: Für diesen Vorgang gibt es einige zusätzliche Optionen, Funktionen und Einschränkungen

Weitere Informationen finden Sie unter diesem Link ---> " https://docs.oracle.com/cd/E11882_01/server.112/e25523/part_admin002.htm#i1107555 " (Abschnitt "Partitionen austauschen").

Partitionierung der Tabelle online lesen: https://riptutorial.com/de/oracle/topic/3955/partitionierung-der-tabelle

Kapitel 23: Rekursives Unterabfrage-Factoring mit der WITH-Klausel (AKA-Ausdrücke für allgemeine Tabellen)

Bemerkungen

Rekursives Unterabfrage-Factoring ist in Oracle 11g R2 verfügbar.

Examples

Ein einfacher Integer-Generator

Abfrage :

```
WITH generator ( value ) AS (
  SELECT 1 FROM DUAL
UNION ALL
  SELECT value + 1
  FROM   generator
  WHERE  value < 10
)
SELECT value
FROM   generator;
```

Ausgabe :

```
VALUE
-----
    1
    2
    3
    4
    5
    6
    7
    8
    9
   10
```

Trennzeichen trennen

Beispieldaten :

```
CREATE TABLE table_name ( value VARCHAR2(50) );

INSERT INTO table_name ( value ) VALUES ( 'A,B,C,D,E' );
```

Abfrage :

```
WITH items ( list, item, lvl ) AS (
  SELECT value,
         REGEXP_SUBSTR( value, '[^,]+', 1, 1 ),
         1
  FROM   table_name
UNION ALL
  SELECT value,
         REGEXP_SUBSTR( value, '[^,]+', 1, lvl + 1 ),
         lvl + 1
  FROM   items
  WHERE  lvl < REGEXP_COUNT( value, '[^,]+' )
)
SELECT * FROM items;
```

Ausgabe :

```
LIST      ITEM LVL
--------- ---- ---
A,B,C,D,E    A   1
A,B,C,D,E    B   2
A,B,C,D,E    C   3
A,B,C,D,E    D   4
A,B,C,D,E    E   5
```

Rekursives Unterabfrage-Factoring mit der WITH-Klausel (AKA-Ausdrücke für allgemeine Tabellen) online lesen: https://riptutorial.com/de/oracle/topic/3506/rekursives-unterabfrage-factoring-mit-der-with-klausel--aka-ausdrucke-fur-allgemeine-tabellen-

Kapitel 24: Schlüsselwörter oder Sonderzeichen abgrenzen

Examples

Begrenzen Sie den Tabellen- oder Spaltennamen mit Sonderzeichen

Wählen Sie * aus der Adresse des Unternehmens aus.

Wählen Sie * aus "Firmas_Adresse";

Abgrenzung von Tabellen- oder Spaltennamen, der ebenfalls ein reserviertes Wort ist

Angenommen, Sie haben eine Tabelle mit dem Namen Tabelle oder möchten eine Tabelle mit einem Namen erstellen, der auch ein Schlüsselwort ist.

Wählen Sie * aus der Tabelle aus; Die obige Abfrage schlägt mit einem Syntaxfehler fehl, wobei die Abfrage wie unten ausgeführt ordnungsgemäß ausgeführt wird.

Wählen Sie * aus "table";

Schlüsselwörter oder Sonderzeichen abgrenzen online lesen:
https://riptutorial.com/de/oracle/topic/6553/schlusselworter-oder-sonderzeichen-abgrenzen

Kapitel 25: Sequenzen

Syntax

- SEQUENCE SCHEMA.SEQUENCE erstellen START MIT INTEGER | MAXVALUE INTEGER | NOMAXVALUE INTEGER | MINVALUE INTEGER | NOMINVALUE INTEGER | CYCLE INTEGER | NOCYCLE INTEGER | CACHE | NOCACHE | BESTELLUNG | NOODER}

Parameter

Parameter	Einzelheiten
Schema	Schemaname
inkrementieren um	Intervall zwischen den Zahlen
beginnen mit	erste Nummer benötigt
Maximalwert	Maximalwert für die Sequenz
Nomaxwert	Maximalwert ist voreingestellt
Minwert	Mindestwert für die Sequenz
Nennwert	Mindestwert ist voreingestellt
Zyklus	Nach Erreichen dieses Wertes auf den Start zurücksetzen
nocycle	Standard
Zwischenspeicher	Vorbelegungsgrenze
Nocache	Standard
Auftrag	Garantiere die Reihenfolge der Zahlen
keine Bestellung	Standard

Examples

Sequenz erstellen: Beispiel

Zweck

Verwenden Sie die Anweisung CREATE SEQUENCE, um eine Sequenz zu erstellen, bei der es

sich um ein Datenbankobjekt handelt, aus dem mehrere Benutzer eindeutige Ganzzahlen generieren können. Sie können Sequenzen verwenden, um automatisch Primärschlüsselwerte zu generieren.

Wenn eine Sequenznummer generiert wird, wird die Sequenz unabhängig von der Transaktion erhöht oder rückgängig gemacht. Wenn zwei Benutzer gleichzeitig dieselbe Sequenz inkrementieren, können die Sequenznummern, die jeder Benutzer erhält, Lücken aufweisen, da Sequenznummern vom anderen Benutzer generiert werden. Ein Benutzer kann niemals die von einem anderen Benutzer generierte Sequenznummer erhalten. Nachdem ein Sequenzwert von einem Benutzer generiert wurde, kann dieser Benutzer weiterhin auf diesen Wert zugreifen, unabhängig davon, ob die Sequenz von einem anderen Benutzer inkrementiert wird.

Sequenznummern werden unabhängig von Tabellen generiert, sodass dieselbe Sequenz für eine oder für mehrere Tabellen verwendet werden kann. Es kann vorkommen, dass einzelne Sequenznummern übersprungen werden, da sie in einer Transaktion generiert und verwendet wurden, die letztendlich zurückgesetzt wurde. Darüber hinaus erkennt ein einzelner Benutzer möglicherweise nicht, dass andere Benutzer aus derselben Sequenz zeichnen.

Nachdem eine Sequenz erstellt wurde, können Sie auf deren Werte in SQL-Anweisungen mit der CURRVAL-Pseudospalte zugreifen, die den aktuellen Wert der Sequenz zurückgibt, oder der NEXTVAL-Pseudospalte, die die Sequenz erhöht und den neuen Wert zurückgibt.

Voraussetzungen

Um eine Sequenz in Ihrem eigenen Schema zu erstellen, müssen Sie über das CREATE SEQUENCE-Systemprivileg verfügen.

Um eine Sequenz im Schema eines anderen Benutzers zu erstellen, müssen Sie über das Systemprivileg CREATE ANY SEQUENCE verfügen.

Erstellen einer Sequenz: Beispiel Die folgende Anweisung erstellt die Sequenz customers_seq im Beispielschema oe. Diese Reihenfolge kann verwendet werden, um Kunden-ID-Nummern anzugeben, wenn der Kundentabelle Zeilen hinzugefügt werden.

```
CREATE SEQUENCE customers_seq
START WITH     1000
INCREMENT BY   1
NOCACHE
NOCYCLE;
```

Die erste Referenz auf customers_seq.nextval gibt 1000 zurück. Die zweite Referenz gibt 1001 zurück. Jede nachfolgende Referenz gibt einen Wert zurück, der um 1 größer ist als die vorherige.

Sequenzen online lesen: https://riptutorial.com/de/oracle/topic/3709/sequenzen

Kapitel 26: Statistische Funktionen

Examples

Berechnung des Medianwertes einer Menge von Werten

Die MEDIAN-Funktion ist seit Oracle 10g eine einfach zu verwendende Aggregationsfunktion:

```
SELECT MEDIAN(SAL)
FROM EMP
```

Es gibt den Median der Werte zurück

Funktioniert auch mit `DATETIME` Werten.

> Das Ergebnis von MEDIAN wird berechnet, indem zuerst die Zeilen angeordnet werden. Mit N als Anzahl der Zeilen in der Gruppe berechnet Oracle die Zeilennummer (RN) von Interesse mit der Formel RN = (1 + (0.5 * (N-1)). Das Endergebnis der Aggregatfunktion wird mit linear berechnet Interpolation zwischen den Werten der Zeilen bei den Zeilennummern CRN = CEILING (RN) und FRN = FLOOR (RN).

Seit Oracle 9i können Sie PERCENTILE_CONT verwenden , das genauso wie die MEDIAN-Funktion mit einem Percentile-Wert von 0,5 arbeitet

```
SELECT PERCENTILE_CONT(.5) WITHIN GROUP(order by SAL)
FROM EMP
```

VARIANCE

Die Varianz misst, wie weit eine festgelegte Anzahl vom Mittelwert abweicht. Aus praktischer Sicht ist der Abstand vom Mittelwert (Mitte) quadratisch - je größer die Zahl, desto weiter der Punkt ist.

Das folgende Beispiel würde eine Abweichung der Gehaltswerte zurückgeben

```
SELECT name, salary, VARIANCE(salary) "Variance"
FROM employees
```

STDDEV

STDDEV gibt die Musterstandardabweichung von expr zurück, eine Menge von Zahlen. Sie können es sowohl als Aggregat- als auch als Analysefunktion verwenden. Sie unterscheidet sich von STDDEV_SAMP darin, dass STDDEV Null zurückgibt, wenn nur eine Zeile Eingabedaten vorhanden ist, während STDDEV_SAMP Null zurückgibt.

Oracle Database berechnet die Standardabweichung als Quadratwurzel der für die VARIANCE-Aggregatfunktion definierten Varianz.

Diese Funktion verwendet als Argument jeden numerischen Datentyp oder jeden nicht-numerischen Datentyp, der implizit in einen numerischen Datentyp konvertiert werden kann. Die Funktion gibt denselben Datentyp zurück wie der numerische Datentyp des Arguments.

Wenn Sie DISTINCT angeben, können Sie nur die query_partition_clause der analytic_clause angeben. Die order_by_clause und die windowing_clause sind nicht erlaubt.

Das folgende Beispiel gibt die Standardabweichung der Gehälter in der Beispieltabelle **hr.employees zurück** :

Wo hr ist Schema und Mitarbeiter ist ein Tabellenname.

```
SELECT STDDEV(salary) "Deviation"
FROM employees;

Deviation
----------
3909.36575
```

Die Abfrage im folgenden Beispiel gibt die kumulierte Standardabweichung der Gehälter in Abteilung 80 in der Beispieltabelle hr.employees zurück, sortiert nach hire_date:

```
SELECT last_name, salary,
STDDEV(salary) OVER (ORDER BY hire_date) "StdDev"
FROM employees
WHERE department_id = 30;

LAST_NAME                    SALARY     StdDev
------------------------- ---------- ----------
Raphaely                      11000          0
Khoo                           3100 5586.14357
Tobias                         2800  4650.0896
```

Statistische Funktionen online lesen: https://riptutorial.com/de/oracle/topic/2283/statistische-funktionen

Kapitel 27: String-Manipulation

Examples

Verkettung: Operator || oder concat () - Funktion

Das Oracle SQL und PL / SQL `||` Mit dem Operator können Sie zwei oder mehr Strings miteinander verketten.

Beispiel:

Angenommen, die folgende `customers` :

```
 id  firstname    lastname
---  -----------  ----------
  1  Thomas       Woody
```

Abfrage:

```
 SELECT firstname || ' ' || lastname || ' is in my database.' as "My Sentence"
   FROM customers;
```

Ausgabe:

```
My Sentence
---------------------------------
Thomas Woody is in my database.
```

Oracle unterstützt auch die standardmäßige SQL-Funktion `CONCAT(str1, str2)` :

Beispiel:

Abfrage:

```
 SELECT CONCAT(firstname, ' is in my database.') from customers;
```

Ausgabe:

```
Expr1
---------------------------------
Thomas is in my database.
```

OBERER, HÖHER

Mit der UPPER-Funktion können Sie alle Kleinbuchstaben in einer Zeichenfolge in Großbuchstaben konvertieren.

```
SELECT UPPER('My text 123!') AS result FROM dual;
```

Ausgabe:

```
RESULT
------------
MY TEXT 123!
```

INITCAP

Die `INITCAP` Funktion konvertiert die Groß- / Kleinschreibung einer Zeichenfolge, sodass jedes Wort mit einem Großbuchstaben beginnt und alle nachfolgenden Buchstaben in Kleinbuchstaben geschrieben werden.

```
SELECT INITCAP('HELLO mr macdonald!') AS NEW FROM dual;
```

Ausgabe

```
NEW
-------------------
Hello Mr Macdonald!
```

NIEDRIGER

LOWER konvertiert alle Großbuchstaben einer Zeichenfolge in Kleinbuchstaben.

```
SELECT LOWER('HELLO World123!') text FROM dual;
```

Ausgänge:

Text
hallo world123!

Regulären Ausdruck

Nehmen wir an, wir möchten nur Zahlen mit 2 Ziffern ersetzen: reguläre Ausdrücke finden sie mit `(\d\d)`

```
SELECT REGEXP_REPLACE ('2, 5, and 10 are numbers in this example', '(\d\d)', '#')
FROM dual;
```

Ergebnisse in:

```
'2, 5, and # are numbers in this example'
```

Wenn ich Teile des Textes austauschen möchte, verwende ich `\1` , `\2` , `\3` , um die

übereinstimmenden Zeichenfolgen aufzurufen:

```
SELECT REGEXP_REPLACE ('swap around 10 in that one ', '(.*)(\d\d )(.*)', '\3\2\1\3')
FROM dual;
```

SUBSTR

`SUBSTR` ruft einen Teil einer Zeichenfolge ab, indem die Startposition und die Anzahl der zu extrahierenden Zeichen angegeben werden

```
SELECT SUBSTR('abcdefg',2,3) FROM DUAL;
```

kehrt zurück:

```
bcd
```

Um vom Ende des Strings zu zählen, akzeptiert `SUBSTR` eine negative Zahl als zweiten Parameter, z

```
SELECT SUBSTR('abcdefg',-4,2) FROM DUAL;
```

kehrt zurück:

```
de
```

Um das letzte Zeichen in einer Zeichenfolge zu erhalten: `SUBSTR(mystring,-1,1)`

LTRIM / RTRIM

`LTRIM` und `RTRIM` entfernen Zeichen vom Anfang bzw. Ende eines Strings. Ein Satz von einem oder mehreren Zeichen kann zum Entfernen angegeben werden (Standard ist ein Leerzeichen).

Zum Beispiel,

```
select LTRIM('<===>HELLO<===>', '=<>')
      ,RTRIM('<===>HELLO<===>', '=<>')
from dual;
```

Kehrt zurück:

```
HELLO<===>
<===>HELLO
```

String-Manipulation online lesen: https://riptutorial.com/de/oracle/topic/1518/string-manipulation

Kapitel 28: Termine

Examples

Daten ohne Zeitkomponente generieren

Alle `DATE` 's haben eine Zeitkomponente; Es ist jedoch üblich, Datumsangaben zu speichern, die keine Zeitangaben enthalten müssen, wobei die Stunden / Minuten / Sekunden auf Null (dh Mitternacht) gesetzt sind.

Verwenden Sie ein ANSI `DATE` Literal (unter Verwendung des Datumsformats nach ISO 8601):

```
SELECT DATE '2000-01-01' FROM DUAL;
```

Konvertieren Sie es mit `TO_DATE()` aus einem String-Literal:

```
SELECT TO_DATE( '2001-01-01', 'YYYY-MM-DD' ) FROM DUAL;
```

(Weitere Informationen zu den Datumsformatmodellen finden Sie in der Oracle-Dokumentation.)

oder:

```
SELECT TO_DATE(
         'January 1, 2000, 00:00 A.M.',
         'Month dd, YYYY, HH12:MI A.M.',
         'NLS_DATE_LANGUAGE = American'
       )
FROM   DUAL;
```

(Wenn Sie sprachspezifische Begriffe wie Monatsnamen `nlsparam` , `TO_DATE()` es sich, den 3. `nlsparam` Parameter in die `TO_DATE()` Funktion aufzunehmen und die zu erwartende Sprache anzugeben.)

Daten mit einer Zeitkomponente generieren

Konvertieren Sie es mit `TO_DATE()` aus einem String-Literal:

```
SELECT TO_DATE( '2000-01-01 12:00:00', 'YYYY-MM-DD HH24:MI:SS' ) FROM DUAL;
```

Oder verwenden Sie ein `TIMESTAMP` Literal :

```
CREATE TABLE date_table(
  date_value DATE
);

INSERT INTO date_table ( date_value ) VALUES ( TIMESTAMP '2000-01-01 12:00:00' );
```

Oracle wird ein `TIMESTAMP` implizit in ein `DATE` `TIMESTAMP` , wenn es in einer `DATE` Spalte einer Tabelle gespeichert wird. Sie können den Wert jedoch explizit zu einem `DATE` `CAST()` :

```
SELECT CAST( TIMESTAMP '2000-01-01 12:00:00' AS DATE ) FROM DUAL;
```

Das Format eines Datums

In Oracle hat ein `DATE` Datentyp kein Format. Wenn Oracle ein `DATE` an das Client-Programm sendet (SQL / Plus, SQL / Developer, Toad, Java, Python usw.), sendet es 7 oder 8 Byte, die das Datum darstellen.

Ein `DATE` das nicht in einer Tabelle gespeichert ist (dh von `SYSDATE` erzeugt wurde und bei Verwendung des `DUMP()` "Typ 13" hat), hat 8 Byte und hat die Struktur (die Zahlen auf der rechten Seite sind die interne Darstellung von `2012-11-26 16:41:09`):

```
BYTE VALUE                          EXAMPLE
---- ------------------------------ ---------------------------------------
1    Year modulo 256                220
2    Year multiples of 256          7   (7 * 256 + 220 = 2012)
3    Month                          11
4    Day                            26
5    Hours                          16
6    Minutes                        41
7    Seconds                        9
8    Unused                         0
```

Ein `DATE` das in einer Tabelle gespeichert ist ("Typ 12" bei Verwendung des `DUMP()`), hat 7 Bytes und die Struktur (die Zahlen auf der rechten Seite sind die interne Darstellung von `2012-11-26 16:41:09`):

```
BYTE VALUE                          EXAMPLE
---- ------------------------------ ---------------------------------------
1    ( Year multiples of 100 ) + 100 120
2    ( Year modulo 100 ) + 100      112 ((120-100)*100 + (112-100) = 2012)
3    Month                          11
4    Day                            26
5    Hours + 1                      17
6    Minutes + 1                    42
7    Seconds + 1                    10
```

Wenn Sie möchten, dass das Datum ein bestimmtes Format hat, müssen Sie es in ein Format konvertieren, das ein Format hat (z. B. eine Zeichenfolge). Der SQL-Client kann dies implizit tun, oder Sie können den Wert mithilfe von `TO_CHAR( date, format_model, nls_params )` explizit in eine Zeichenfolge `TO_CHAR( date, format_model, nls_params )` .

Datumsangaben in einen String umwandeln

Verwenden Sie `TO_CHAR( date [, format_model [, nls_params]] )` :

(Hinweis: Wenn ein Format - `NLS_DATE_FORMAT` Modell nicht zur Verfügung gestellt wird , wird die `NLS_DATE_FORMAT` Session - Parameter werden als verwendet werden Standardformat - Modell , das

für jede Sitzung unterschiedlich sein , so kann nicht geltend gemacht werden , sollte es gute Praxis ist das Format - Modell immer angeben..)

```
CREATE TABLE table_name (
  date_value DATE
);

INSERT INTO table_name ( date_value ) VALUES ( DATE '2000-01-01' );
INSERT INTO table_name ( date_value ) VALUES ( TIMESTAMP '2016-07-21 08:00:00' );
INSERT INTO table_name ( date_value ) VALUES ( SYSDATE );
```

Dann:

```
SELECT TO_CHAR( date_value, 'YYYY-MM-DD' ) AS formatted_date FROM table_name;
```

Ausgänge:

```
FORMATTED_DATE
--------------
2000-01-01
2016-07-21
2016-07-21
```

Und:

```
SELECT TO_CHAR(
         date_value,
         'FMMonth d yyyy, hh12:mi:ss AM',
         'NLS_DATE_LANGUAGE = French'
       ) AS formatted_date
FROM   table_name;
```

Ausgänge:

```
FORMATTED_DATE
----------------------------
Janvier   01 2000, 12:00:00 AM
Juillet   21 2016, 08:00:00 AM
Juillet   21 2016, 19:08:31 PM
```

Festlegen des Standard-Datumsformatmodells

Wenn Oracle implizit von einem `DATE` in einen String oder umgekehrt konvertiert (oder wenn `TO_CHAR()` oder `TO_DATE()` explizit ohne `TO_DATE()` aufgerufen wird), wird der Sitzungsparameter `NLS_DATE_FORMAT` als `NLS_DATE_FORMAT` bei der Konvertierung verwendet. Wenn das Literal nicht mit dem Formatmodell übereinstimmt, wird eine Ausnahme ausgelöst.

Sie können diesen Parameter folgendermaßen überprüfen:

```
SELECT VALUE FROM NLS_SESSION_PARAMETERS WHERE PARAMETER = 'NLS_DATE_FORMAT';
```

Sie können diesen Wert in Ihrer aktuellen Sitzung einstellen, indem Sie

```
ALTER SESSION SET NLS_DATE_FORMAT = 'YYYY-MM-DD HH24:MI:SS';
```

(Hinweis: Der Wert für andere Benutzer wird dadurch nicht geändert.)

Wenn Sie sich darauf `NLS_DATE_FORMAT` , dass `NLS_DATE_FORMAT` die Formatmaske in `TO_DATE()` oder `TO_CHAR()` , sollten Sie sich nicht wundern, wenn Ihre Abfragen brechen, wenn dieser Wert jemals geändert wird.

Ändern der Anzeigedaten von SQL / Plus oder SQL Developer

Wenn Datumsangaben von SQL / Plus oder SQL Developer angezeigt werden, wird eine implizite Konvertierung in eine Zeichenfolge mit dem Standardmodell für das Datumsformat durchgeführt (siehe Beispiel für das Standardformat für das Datumsformat).

Sie können die Anzeige eines Datums ändern, indem Sie den Parameter `NLS_DATE_FORMAT` .

Datumsarithmetik - Differenz zwischen Datumsangaben in Tagen, Stunden, Minuten und / oder Sekunden

In Orakel kann die Differenz (in Tagen und / oder Brüchen davon) zwischen zwei `DATE` durch Subtraktion ermittelt werden:

```
SELECT DATE '2016-03-23' - DATE '2015-12-25' AS difference FROM DUAL;
```

Gibt die Anzahl der Tage zwischen den beiden Datumsangaben aus:

```
DIFFERENCE
----------
        89
```

Und:

```
SELECT TO_DATE( '2016-01-02 01:01:12', 'YYYY-MM-DD HH24:MI:SS' )
         - TO_DATE( '2016-01-01 00:00:00', 'YYYY-MM-DD HH24:MI:SS' )
         AS difference
FROM   DUAL
```

Gibt den Bruchteil der Tage zwischen zwei Datumsangaben aus:

```
DIFFERENCE
----------
    1.0425
```

Die Differenz in Stunden, Minuten oder Sekunden kann ermittelt werden, indem diese Zahl mit `24` , `24*60` bzw. `24*60*60` multipliziert wird.

Das vorherige Beispiel kann geändert werden, um die Tage, Stunden, Minuten und Sekunden

zwischen zwei Datumsangaben abzurufen.

```
SELECT TRUNC( difference                        ) AS days,
       TRUNC( MOD( difference * 24,       24 ) ) AS hours,
       TRUNC( MOD( difference * 24*60,    60 ) ) AS minutes,
       TRUNC( MOD( difference * 24*60*60, 60 ) ) AS seconds
FROM   (
  SELECT TO_DATE( '2016-01-02 01:01:12', 'YYYY-MM-DD HH24:MI:SS' )
         - TO_DATE( '2016-01-01 00:00:00', 'YYYY-MM-DD HH24:MI:SS' )
         AS difference
  FROM   DUAL
);
```

(Hinweis: `TRUNC()` wird anstelle von `FLOOR()` , um negative Unterschiede korrekt zu behandeln.)

Ausgänge:

```
DAYS HOURS MINUTES SECONDS
---- ----- ------- -------
   1     1       1      12
```

Das vorige Beispiel kann auch gelöst werden, indem die numerische Differenz mit `NUMTODSINTERVAL()` in ein Intervall `NUMTODSINTERVAL()` :

```
SELECT EXTRACT( DAY    FROM difference ) AS days,
       EXTRACT( HOUR   FROM difference ) AS hours,
       EXTRACT( MINUTE FROM difference ) AS minutes,
       EXTRACT( SECOND FROM difference ) AS seconds
FROM   (
  SELECT NUMTODSINTERVAL(
           TO_DATE( '2016-01-02 01:01:12', 'YYYY-MM-DD HH24:MI:SS' )
             - TO_DATE( '2016-01-01 00:00:00', 'YYYY-MM-DD HH24:MI:SS' ),
           'DAY'
         ) AS difference
  FROM   DUAL
);
```

Datumsarithmetik - Differenz zwischen Datumsangaben in Monaten oder Jahren

Die Differenz in Monaten zwischen zwei Datumsangaben kann mit `MONTHS_BETWEEN( date1, date2 )` :

```
SELECT MONTHS_BETWEEN( DATE '2016-03-10', DATE '2015-03-10' ) AS difference FROM DUAL;
```

Ausgänge:

```
DIFFERENCE
----------
        12
```

Wenn die Differenz Teilmonate umfasst, wird der Bruchteil des Monats zurückgegeben, der auf **31** Tagen pro Monat basiert:

```
SELECT MONTHS_BETWEEN( DATE '2015-02-15', DATE '2015-01-01' ) AS difference FROM DUAL;
```

Ausgänge:

```
DIFFERENCE
----------
 1.4516129
```

Da `MONTHS_BETWEEN` 31 Tage pro Monat `MONTHS_BETWEEN` , wenn es weniger Tage pro Monat gibt, kann dies zu unterschiedlichen Werten für Unterschiede führen, die die Grenzen zwischen Monaten überspannen.

Beispiel:

```
SELECT MONTHS_BETWEEN( DATE'2016-02-01', DATE'2016-02-01' - INTERVAL '1' DAY ) AS "JAN-FEB",
       MONTHS_BETWEEN( DATE'2016-03-01', DATE'2016-03-01' - INTERVAL '1' DAY ) AS "FEB-MAR",
       MONTHS_BETWEEN( DATE'2016-04-01', DATE'2016-04-01' - INTERVAL '1' DAY ) AS "MAR-APR",
       MONTHS_BETWEEN( DATE'2016-05-01', DATE'2016-05-01' - INTERVAL '1' DAY ) AS "APR-MAY"
FROM   DUAL;
```

Ausgabe:

```
JAN-FEB FEB-MAR MAR-APR APR-MAY
------- ------- ------- -------
0.03226 0.09677 0.03226 0.06452
```

Die Differenz in Jahren kann ermittelt werden, indem die Monatsdifferenz durch 12 geteilt wird.

Extrahieren Sie die Komponenten Jahr, Monat, Tag, Stunde, Minute oder Sekunde eines Datums

Die `EXTRACT( [ YEAR | MONTH | DAY ] FROM datevalue )` , Monats- oder `EXTRACT( [ YEAR | MONTH | DAY ] FROM datevalue )` eines `DATE` Datentyps können mit `EXTRACT( [ YEAR | MONTH | DAY ] FROM datevalue )`

```
SELECT EXTRACT (YEAR  FROM DATE '2016-07-25') AS YEAR,
       EXTRACT (MONTH FROM DATE '2016-07-25') AS MONTH,
       EXTRACT (DAY   FROM DATE '2016-07-25') AS DAY
FROM DUAL;
```

Ausgänge:

```
YEAR MONTH DAY
---- ----- ---
2016     7  25
```

Die Zeitkomponenten (Stunden, Minuten oder Sekunden) können gefunden werden durch:

- Verwenden Sie `CAST( datevalue AS TIMESTAMP )` , um das `DATE` in ein `TIMESTAMP` und verwenden Sie dann `EXTRACT( [ HOUR | MINUTE | SECOND ] FROM timestampvalue )` . oder
- Verwenden von `TO_CHAR( datevalue, format_model )` , um den Wert als Zeichenfolge `TO_CHAR( datevalue, format_model )` .

Zum Beispiel:

```
SELECT EXTRACT( HOUR   FROM CAST( datetime AS TIMESTAMP ) ) AS Hours,
       EXTRACT( MINUTE FROM CAST( datetime AS TIMESTAMP ) ) AS Minutes,
       EXTRACT( SECOND FROM CAST( datetime AS TIMESTAMP ) ) AS Seconds
FROM   (
  SELECT TO_DATE( '2016-01-01 09:42:01', 'YYYY-MM-DD HH24:MI:SS' ) AS datetime FROM DUAL
);
```

Ausgänge:

```
HOURS MINUTES SECONDS
----- ------- -------
    9      42       1
```

Zeitzonen und Sommerzeit

Der Datentyp `DATE` verarbeitet keine Zeitzonen oder Änderungen in der Sommerzeit.

Entweder:

- Verwenden Sie den `TIMESTAMP WITH TIME ZONE` . oder
- handhaben Sie die Änderungen in Ihrer Anwendungslogik.

Ein `DATE` kann als koordinierte Weltzeit (UTC) gespeichert und wie folgt in die aktuelle Sitzungszeitzone konvertiert werden:

```
SELECT FROM_TZ(
         CAST(
           TO_DATE( '2016-01-01 12:00:00', 'YYYY-MM-DD HH24:MI:SS' )
           AS TIMESTAMP
         ),
         'UTC'
       )
       AT LOCAL AS time
FROM   DUAL;
```

Wenn Sie `ALTER SESSION SET TIME_ZONE = '+01:00';` dann ist die Ausgabe:

```
TIME
-----------------------------------
2016-01-01 13:00:00.000000000 +01:00
```

und `ALTER SESSION SET TIME_ZONE = 'PST';` dann ist die Ausgabe:

```
TIME
```

```
-----------------------------------
2016-01-01 04:00:00.000000000 PST
```

Schaltsekunden

Oracle verarbeitet Schaltsekunden nicht . Weitere `2019397.2` Sie in meinem Oracle-Support-Hinweis `2019397.2` und `730795.1` .

Den Tag der Woche bekommen

Sie können `TO_CHAR( date_value, 'D' )` , um den Wochentag zu erhalten.

Dies ist jedoch vom Sitzungsparameter `NLS_TERRITORY` abhängig:

```
ALTER SESSION SET NLS_TERRITORY = 'AMERICA';        -- First day of week is Sunday
SELECT TO_CHAR( DATE '1970-01-01', 'D' ) FROM DUAL;
```

Ausgänge `5`

```
ALTER SESSION SET NLS_TERRITORY = 'UNITED KINGDOM'; -- First day of week is Monday
SELECT TO_CHAR( DATE '1970-01-01', 'D' ) FROM DUAL;
```

Ausgänge `4`

Um dies unabhängig von den `NLS` Einstellungen zu tun, können Sie das Datum auf Mitternacht des aktuellen Tages abschneiden (um etwaige Bruchteile von Tagen zu entfernen) und das abgeschnittene Datum vom Beginn der aktuellen Iso-Woche (die immer am Montag beginnt) abziehen:

```
SELECT TRUNC( date_value ) - TRUNC( date_value, 'IW' ) + 1 FROM DUAL
```

Termine online lesen: https://riptutorial.com/de/oracle/topic/2087/termine

Kapitel 29: Trennstrings trennen

Examples

Aufteilen von Zeichenfolgen mithilfe einer rekursiven Unterabfrage-Factoringklausel

Beispieldaten :

```
CREATE TABLE table_name ( id, list ) AS
SELECT 1, 'a,b,c,d' FROM DUAL UNION ALL -- Multiple items in the list
SELECT 2, 'e'       FROM DUAL UNION ALL -- Single item in the list
SELECT 3, NULL      FROM DUAL UNION ALL -- NULL list
SELECT 4, 'f,,g'    FROM DUAL;          -- NULL item in the list
```

Abfrage :

```
WITH bounds ( id, list, start_pos, end_pos, lvl ) AS (
  SELECT id, list, 1, INSTR( list, ',' ), 1 FROM table_name
UNION ALL
  SELECT id,
         list,
         end_pos + 1,
         INSTR( list, ',', end_pos + 1 ),
         lvl + 1
  FROM   bounds
  WHERE  end_pos > 0
)
SELECT id,
       SUBSTR(
         list,
         start_pos,
         CASE end_pos
           WHEN 0
           THEN LENGTH( list ) + 1
           ELSE end_pos
         END - start_pos
       ) AS item,
      lvl
FROM   bounds
ORDER BY id, lvl;
```

Ausgabe :

```
        ID ITEM           LVL
---------- ------- ----------
         1 a                 1
         1 b                 2
         1 c                 3
         1 d                 4
         2 e                 1
         3 (NULL)            1
         4 f                 1
```

```
         4 (NULL)           2
         4 g                3
```

Teilen von Strings mit einer PL / SQL-Funktion

PL / SQL-Funktion :

```
CREATE OR REPLACE FUNCTION split_String(
  i_str    IN  VARCHAR2,
  i_delim  IN  VARCHAR2 DEFAULT ','
) RETURN SYS.ODCIVARCHAR2LIST DETERMINISTIC
AS
  p_result       SYS.ODCIVARCHAR2LIST := SYS.ODCIVARCHAR2LIST();
  p_start        NUMBER(5) := 1;
  p_end          NUMBER(5);
  c_len CONSTANT NUMBER(5) := LENGTH( i_str );
  c_ld  CONSTANT NUMBER(5) := LENGTH( i_delim );
BEGIN
  IF c_len > 0 THEN
    p_end := INSTR( i_str, i_delim, p_start );
    WHILE p_end > 0 LOOP
      p_result.EXTEND;
      p_result( p_result.COUNT ) := SUBSTR( i_str, p_start, p_end - p_start );
      p_start := p_end + c_ld;
      p_end := INSTR( i_str, i_delim, p_start );
    END LOOP;
    IF p_start <= c_len + 1 THEN
      p_result.EXTEND;
      p_result( p_result.COUNT ) := SUBSTR( i_str, p_start, c_len - p_start + 1 );
    END IF;
  END IF;
  RETURN p_result;
END;
/
```

Beispieldaten :

```
CREATE TABLE table_name ( id, list ) AS
SELECT 1, 'a,b,c,d' FROM DUAL UNION ALL -- Multiple items in the list
SELECT 2, 'e'       FROM DUAL UNION ALL -- Single item in the list
SELECT 3, NULL      FROM DUAL UNION ALL -- NULL list
SELECT 4, 'f,,g'    FROM DUAL;          -- NULL item in the list
```

Abfrage :

```
SELECT t.id,
       v.column_value AS value,
       ROW_NUMBER() OVER ( PARTITION BY id ORDER BY ROWNUM ) AS lvl
FROM   table_name t,
       TABLE( split_String( t.list ) ) (+) v
```

Ausgabe :

```
        ID ITEM           LVL
---------- ------- ----------
```

```
         1 a                1
         1 b                2
         1 c                3
         1 d                4
         2 e                1
         3 (NULL)           1
         4 f                1
         4 (NULL)           2
         4 g                3
```

Aufteilen von Zeichenfolgen mit einem korrelierten Tabellenausdruck

Beispieldaten :

```
CREATE TABLE table_name ( id, list ) AS
SELECT 1, 'a,b,c,d' FROM DUAL UNION ALL -- Multiple items in the list
SELECT 2, 'e'       FROM DUAL UNION ALL -- Single item in the list
SELECT 3, NULL      FROM DUAL UNION ALL -- NULL list
SELECT 4, 'f,,g'    FROM DUAL;          -- NULL item in the list
```

Abfrage :

```
SELECT t.id,
       v.COLUMN_VALUE AS value,
       ROW_NUMBER() OVER ( PARTITION BY id ORDER BY ROWNUM ) AS lvl
FROM   table_name t,
       TABLE(
         CAST(
           MULTISET(
             SELECT REGEXP_SUBSTR( t.list, '([^,]*)(,|$)', 1, LEVEL, NULL, 1 )
             FROM   DUAL
             CONNECT BY LEVEL < REGEXP_COUNT( t.list, '[^,]*(,|$)' )
           )
           AS SYS.ODCIVARCHAR2LIST
         )
       ) v;
```

Ausgabe :

```
        ID ITEM           LVL
---------- ------- ----------
         1 a                1
         1 b                2
         1 c                3
         1 d                4
         2 e                1
         3 (NULL)           1
         4 f                1
         4 (NULL)           2
         4 g                3
```

Zeichenfolgen mit einer hierarchischen Abfrage teilen

Beispieldaten :

```
CREATE TABLE table_name ( id, list ) AS
SELECT 1, 'a,b,c,d' FROM DUAL UNION ALL -- Multiple items in the list
SELECT 2, 'e'       FROM DUAL UNION ALL -- Single item in the list
SELECT 3, NULL      FROM DUAL UNION ALL -- NULL list
SELECT 4, 'f,,g'    FROM DUAL;          -- NULL item in the list
```

Abfrage :

```
SELECT t.id,
       REGEXP_SUBSTR( list, '([^,]*)(,|$)', 1, LEVEL, NULL, 1 ) AS value,
       LEVEL AS lvl
FROM   table_name t
CONNECT BY
       id = PRIOR id
AND    PRIOR SYS_GUID() IS NOT NULL
AND    LEVEL < REGEXP_COUNT( list, '([^,]*)(,|$)' )
```

Ausgabe :

```
        ID ITEM           LVL
---------- ------- ----------
         1 a                1
         1 b                2
         1 c                3
         1 d                4
         2 e                1
         3 (NULL)           1
         4 f                1
         4 (NULL)           2
         4 g                3
```

Teilen von Zeichenfolgen mit XMLTable- und FLWOR-Ausdrücken

Diese Lösung verwendet die von Oracle 11 verfügbare XQuery-Funktion `ora:tokenize` .

Beispieldaten :

```
CREATE TABLE table_name ( id, list ) AS
SELECT 1, 'a,b,c,d' FROM DUAL UNION ALL -- Multiple items in the list
SELECT 2, 'e'       FROM DUAL UNION ALL -- Single item in the list
SELECT 3, NULL      FROM DUAL UNION ALL -- NULL list
SELECT 4, 'f,,g'    FROM DUAL;          -- NULL item in the list
```

Abfrage :

```
SELECT t.id,
       x.item,
       x.lvl
FROM   table_name t,
       XMLTABLE(
         'let $list := ora:tokenize(.,","),
              $cnt := count($list)
          for $val at $r in $list
          where $r < $cnt
          return $val'
```

```
       PASSING list||','
       COLUMNS
         item VARCHAR2(100) PATH '.',
         lvl FOR ORDINALITY
       ) (+) x;
```

Ausgabe :

```
        ID ITEM           LVL
---------- ------- ----------
         1 a                1
         1 b                2
         1 c                3
         1 d                4
         2 e                1
         3 (NULL)       (NULL)
         4 f                1
         4 (NULL)           2
         4 g                3
```

Strings mit CROSS APPLY aufteilen (Oracle 12c)

Beispieldaten :

```
CREATE TABLE table_name ( id, list ) AS
SELECT 1, 'a,b,c,d' FROM DUAL UNION ALL -- Multiple items in the list
SELECT 2, 'e'       FROM DUAL UNION ALL -- Single item in the list
SELECT 3, NULL      FROM DUAL UNION ALL -- NULL list
SELECT 4, 'f,,g'    FROM DUAL;          -- NULL item in the list
```

Abfrage :

```
SELECT t.id,
       REGEXP_SUBSTR( t.list, '([^,]*)($|,)', 1, l.lvl, NULL, 1 ) AS item,
       l.lvl
FROM   table_name t
       CROSS APPLY
       (
         SELECT LEVEL AS lvl
         FROM   DUAL
         CONNECT BY LEVEL <= REGEXP_COUNT( t.list, ',' ) + 1
       ) l;
```

Ausgabe :

```
        ID ITEM           LVL
---------- ------- ----------
         1 a                1
         1 b                2
         1 c                3
         1 d                4
         2 e                1
         3 (NULL)           1
         4 f                1
         4 (NULL)           2
```

```
         4 g                3
```

Aufteilen von Trennzeichen mit XMLTable

Beispieldaten :

```
CREATE TABLE table_name ( id, list ) AS
SELECT 1, 'a,b,c,d' FROM DUAL UNION ALL -- Multiple items in the list
SELECT 2, 'e'       FROM DUAL UNION ALL -- Single item in the list
SELECT 3, NULL      FROM DUAL UNION ALL -- NULL list
SELECT 4, 'f,,g'    FROM DUAL;          -- NULL item in the list
```

Abfrage :

```
SELECT t.id,
       SUBSTR( x.item.getStringVal(), 2 ) AS item,
       x.lvl
FROM   table_name t
       CROSS JOIN
       XMLTABLE(
         ( '"#' || REPLACE( t.list, ',', '","#' ) || '"' )
         COLUMNS item XMLTYPE PATH '.',
                 lvl  FOR ORDINALITY
       ) x;
```

(Hinweis: Das Zeichen `#` wird angehängt, um das Extrahieren von `NULL` Werten zu erleichtern; es wird später mit `SUBSTR( item, 2 )` . Wenn `NULL` Werte nicht erforderlich sind, können Sie die Abfrage vereinfachen und diese weglassen.)

Ausgabe :

```
        ID ITEM            LVL
---------- ------- ----------
         1 a                 1
         1 b                 2
         1 c                 3
         1 d                 4
         2 e                 1
         3 (NULL)            1
         4 f                 1
         4 (NULL)            2
         4 g                 3
```

Trennstrings trennen online lesen: https://riptutorial.com/de/oracle/topic/1968/trennstrings-trennen

Kapitel 30: Umgang mit NULL-Werten

Einführung

Eine Spalte ist NULL, wenn sie keinen Wert hat, unabhängig vom Datentyp dieser Spalte. Eine Spalte sollte niemals mit NULL verglichen werden, wenn diese Syntax `a = NULL` da das Ergebnis UNKNOWN wäre. Verwenden Sie stattdessen `a IS NULL` oder `a IS NOT NULL` Bedingung. NULL ist nicht gleich NULL. Verwenden Sie zum Vergleichen zweier Ausdrücke, bei denen Null auftreten kann, eine der unten beschriebenen Funktionen. Alle Operatoren mit Ausnahme der Verkettung geben NULL zurück, wenn einer ihrer Operanden NULL ist. Zum Beispiel ist das Ergebnis von `3 * NULL + 5` null.

Bemerkungen

NULL darf nicht in Spalten angezeigt werden, die durch einen PRIMARY KEY oder eine NOT NULL-Einschränkung eingeschränkt sind. (Ausnahme ist eine neue Einschränkung mit NOVALIDATE-Klausel.)

Examples

Spalten eines beliebigen Datentyps können NULL-Werte enthalten

```
SELECT 1 NUM_COLUMN, 'foo' VARCHAR2_COLUMN from DUAL
UNION ALL
SELECT NULL, NULL from DUAL;
```

NUM_COLUMN	VARCHAR2_COLUMN
1	foo
(Null)	(Null)

Leere Zeichenfolgen sind NULL

```
SELECT 1 a, '' b from DUAL;
```

EIN	B
1	(Null)

Operationen, die NULL enthalten, sind NULL mit Ausnahme der Verkettung

```
SELECT 3 * NULL + 5, 'Hello ' || NULL || 'world'   from DUAL;
```

3 * NULL + 5	'HALLO' || NULL || 'WELT'
(Null)	Hallo Welt

NVL zum Ersetzen des Nullwerts

```
SELECT a column_with_null, NVL(a, 'N/A') column_without_null FROM
  (SELECT NULL a FROM DUAL);
```

COLUMN_WITH_NULL	COLUMN_WITHOUT_NULL
(Null)	N / A

NVL ist nützlich, um zwei Werte zu vergleichen, die NULL enthalten können:

```
SELECT
    CASE WHEN a = b THEN 1 WHEN a <> b THEN 0 else -1 END comparison_without_nvl,
    CASE WHEN NVL(a, -1) = NVL(b, -1) THEN 1 WHEN NVL(a, -1) <> NVL(b, -1) THEN 0 else -1 END
comparison_with_nvl
  FROM
    (select null a, 3 b FROM DUAL
     UNION ALL
     SELECT NULL, NULL FROM DUAL);
```

COMPARISON_WITHOUT_NVL	COMPARISON_WITH_NVL
-1	0
-1	1

NVL2, um ein anderes Ergebnis zu erhalten, wenn ein Wert null ist oder nicht

Wenn der erste Parameter NOT NULL ist, gibt NVL2 den zweiten Parameter zurück. Andernfalls wird der dritte zurückgegeben.

```
SELECT NVL2(null, 'Foo', 'Bar'), NVL2(5, 'Foo', 'Bar') FROM DUAL;
```

NVL2 (NULL, 'FOO', 'BAR')	NVL2 (5, 'FOO', 'BAR')
Bar	Foo

COALESCE, um den ersten Nicht-NULL-Wert zurückzugeben

```
SELECT COALESCE(a, b, c, d, 5) FROM
    (SELECT NULL A, NULL b, NULL c, 4 d FROM DUAL);
```

COALESCE (A, B, C, D, 5)
4

In einigen Fällen kann die Verwendung von COALESCE mit zwei Parametern schneller sein als die Verwendung von NVL, wenn der zweite Parameter keine Konstante ist. NVL wertet immer beide Parameter aus. COALESCE stoppt beim ersten Nicht-NULL-Wert, auf den es trifft. Wenn der erste Wert nicht NULL ist, ist COALESCE schneller.

Umgang mit NULL-Werten online lesen: https://riptutorial.com/de/oracle/topic/8183/umgang-mit-null-werten

Kapitel 31: Update mit Joins

Einführung

Im Gegensatz zu weitverbreiteten Missverständnissen (einschließlich SO) erlaubt Oracle Updates über Joins. Es gibt jedoch einige (ziemlich logische) Anforderungen. Wir veranschaulichen, was nicht funktioniert und was funktioniert anhand eines einfachen Beispiels. Eine andere Möglichkeit, dies zu erreichen, ist die MERGE-Anweisung.

Examples

Beispiele: Was funktioniert und was nicht?

```
create table tgt ( id, val ) as
  select 1, 'a' from dual union all
  select 2, 'b' from dual
;

Table TGT created.

create table src ( id, val ) as
  select 1, 'x' from dual union all
  select 2, 'y' from dual
;

Table SRC created.

update
  ( select t.val as t_val, s.val as s_val
    from   tgt t inner join src s on t.id = s.id
  )
set t_val = s_val
;

SQL Error: ORA-01779: cannot modify a column which maps to a non key-preserved table
01779. 00000 -  "cannot modify a column which maps to a non key-preserved table"
*Cause:    An attempt was made to insert or update columns of a join view which
           map to a non-key-preserved table.
*Action:   Modify the underlying base tables directly.
```

Stellen Sie sich vor, was passiert, wenn wir mehr als einmal den Wert `1` in der Spalte `src.id` , mit unterschiedlichen Werten für `src.val` . Offensichtlich macht das Update keinen Sinn (in JEDER Datenbank - das ist ein logisches Problem). Nun wissen **wir** , dass es in `src.id` keine Duplikate `src.id` , aber die Oracle-Engine weiß das nicht - also beschwert sie sich. Vielleicht glauben deshalb viele Praktizierende, dass Oracle "kein Update mit Joins hat"?

Was Oracle erwartet, ist, dass `src.id` eindeutig sein sollte und dass Oracle, das, dies vorher wissen würde. Einfach zu reparieren! Beachten Sie, dass dasselbe mit zusammengesetzten Schlüsseln (für mehr als eine Spalte) funktioniert, wenn der Abgleich für das Update mehr als eine Spalte verwenden muss. In der Praxis ist `src.id` möglicherweise PK und `tgt.id` kann FK sein, die

auf diese PK verweist, aber dies ist für Updates mit Join nicht relevant. was relevant ist die eindeutige Einschränkung.

```
alter table src add constraint src_uc unique (id);

Table SRC altered.

update
  ( select t.val as t_val, s.val as s_val
    from   tgt t inner join src s on t.id = s.id
  )
set t_val = s_val
;

2 rows updated.

select * from tgt;

ID  VAL
--  ---
 1  x
 2  y
```

Das gleiche Ergebnis könnte mit einer MERGE-Anweisung (die einen eigenen Dokumentationsartikel verdient) erzielt werden, und ich persönlich bevorzuge MERGE in diesen Fällen, aber der Grund ist nicht, dass "Oracle keine Updates mit Verknüpfungen vornimmt". Wie dieses Beispiel zeigt, *tut* Oracle - Updates tun mit beitritt.

Update mit Joins online lesen: https://riptutorial.com/de/oracle/topic/8061/update-mit-joins

Kapitel 32: VERBINDUNGEN

Examples

CROSS JOIN

Ein `CROSS JOIN` führt einen Join zwischen zwei Tabellen aus, der keine explizite Join-Klausel verwendet, und führt zum kartesischen Produkt von zwei Tabellen. Ein kartesisches Produkt bedeutet, dass jede Zeile einer Tabelle mit jeder Zeile der zweiten Tabelle im Join kombiniert wird. Wenn `TABLEA` beispielsweise 20 Zeilen und `TABLEB` 20 Zeilen hat, wäre das Ergebnis `20*20 = 400` Ausgabezeilen.

Beispiel:

```
SELECT *
FROM TABLEA CROSS JOIN TABLEB;
```

Dies kann auch geschrieben werden als:

```
SELECT *
FROM TABLEA, TABLEB;
```

Hier ist ein Beispiel für Cross Join in SQL zwischen zwei Tabellen:

Probentabelle **:** TABLEA

```
+-------+---------+
| VALUE |   NAME  |
+-------+---------+
|   1   |   ONE   |
|   2   |   TWO   |
+-------+---------+
```

Beispieltabelle: TABLEB

```
+-------+--------+
| VALUE |  NAME  |
+-------+--------+
|   3   |  THREE |
|   4   |  FOUR  |
+-------+--------+
```

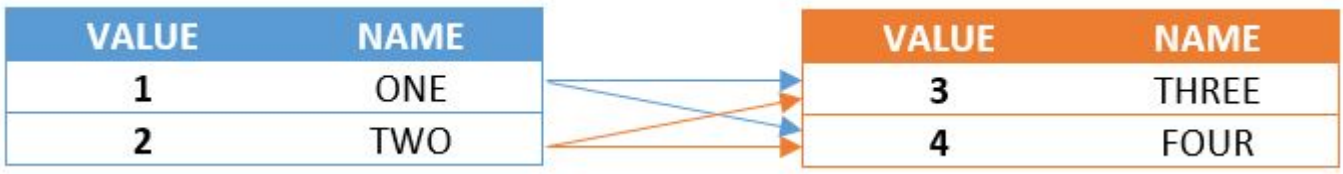
Wenn Sie nun die Abfrage ausführen:

```
SELECT *
FROM TABLEA CROSS JOIN TABLEB;
```

Ausgabe:

```
+-------+--------+-------+--------+
| VALUE |  NAME  | VALUE |  NAME  |
+-------+--------+-------+--------+
|   1   |   ONE  |   3   |  THREE |
|   1   |   ONE  |   4   |  FOUR  |
|   2   |   TWO  |   3   |  THREE |
|   2   |   TWO  |   4   |  FOUR  |
+-------+--------+-------+--------+
```

So erfolgt das Cross-Joining zwischen zwei Tabellen:



Weitere Informationen zu Cross Join: Oracle-Dokumentation

INNER JOIN

Ein INNER JOIN ist eine JOIN-Operation, mit der Sie eine explizite Join-Klausel angeben können.

Syntax

TableExpression [INNER] JOIN TableExpression {ON booleanExpression | USING-Klausel}

Sie können die Join-Klausel angeben, indem Sie ON mit einem booleschen Ausdruck angeben.

Der Geltungsbereich von Ausdrücken in der ON-Klausel umfasst die aktuellen Tabellen und alle Tabellen in äußeren Abfrageblöcken für das aktuelle SELECT. Im folgenden Beispiel verweist die ON-Klausel auf die aktuellen Tabellen:

```
-- Join the EMP_ACT and EMPLOYEE tables
-- select all the columns from the EMP_ACT table and
-- add the employee's surname (LASTNAME) from the EMPLOYEE table
-- to each row of the result
SELECT SAMP.EMP_ACT.*, LASTNAME
 FROM SAMP.EMP_ACT JOIN SAMP.EMPLOYEE
 ON EMP_ACT.EMPNO = EMPLOYEE.EMPNO
-- Join the EMPLOYEE and DEPARTMENT tables,
-- select the employee number (EMPNO),
-- employee surname (LASTNAME),
-- department number (WORKDEPT in the EMPLOYEE table and DEPTNO in the
-- DEPARTMENT table)
-- and department name (DEPTNAME)
-- of all employees who were born (BIRTHDATE) earlier than 1930.
SELECT EMPNO, LASTNAME, WORKDEPT, DEPTNAME
 FROM SAMP.EMPLOYEE JOIN SAMP.DEPARTMENT
 ON WORKDEPT = DEPTNO
 AND YEAR(BIRTHDATE) < 1930

-- Another example of "generating" new data values,
-- using a query which selects from a VALUES clause (which is an
-- alternate form of a fullselect).
-- This query shows how a table can be derived called "X"
```

```
-- having 2 columns "R1" and "R2" and 1 row of data
SELECT *
FROM (VALUES (3, 4), (1, 5), (2, 6))
AS VALUESTABLE1(C1, C2)
JOIN (VALUES (3, 2), (1, 2),
(0, 3)) AS VALUESTABLE2(c1, c2)
ON VALUESTABLE1.c1 = VALUESTABLE2.c1
-- This results in:
-- C1          |C2          |C1          |2
-- -----------------------------------------------
-- 3           |4           |3           |2
-- 1           |5           |1           |2


-- List every department with the employee number and
-- last name of the manager

SELECT DEPTNO, DEPTNAME, EMPNO, LASTNAME
FROM DEPARTMENT INNER JOIN EMPLOYEE
ON MGRNO = EMPNO

-- List every employee number and last name
-- with the employee number and last name of their manager
SELECT E.EMPNO, E.LASTNAME, M.EMPNO, M.LASTNAME
FROM EMPLOYEE E INNER JOIN
DEPARTMENT INNER JOIN EMPLOYEE M
    ON MGRNO = M.EMPNO
    ON E.WORKDEPT = DEPTNO
```

LINKE ÄUSSERE VERBINDUNG

Ein `LEFT OUTER JOIN` führt eine Verknüpfung zwischen zwei Tabellen aus, die eine explizite Verknüpfungsklausel erfordert, aber nicht übereinstimmende Zeilen nicht aus der ersten Tabelle ausschließt.

Beispiel:

```
SELECT
      ENAME,
      DNAME,
      EMP.DEPTNO,
      DEPT.DEPTNO
FROM
      SCOTT.EMP LEFT OUTER JOIN SCOTT.DEPT
      ON EMP.DEPTNO = DEPT.DEPTNO;
```

Obwohl die empfohlene Methode die ANSI-Syntax ist, ist es wahrscheinlich, dass sie häufig auf eine ältere Syntax stößt. Die Verwendung von `(+)` innerhalb einer Bedingung bestimmt, welche Seite der Gleichung als *äußere betrachtet wird* .

```
SELECT
      ENAME,
      DNAME,
      EMP.DEPTNO,
      DEPT.DEPTNO
FROM
```

```
      SCOTT.EMP,
      SCOTT.DEPT
WHERE
      EMP.DEPTNO = DEPT.DEPTNO(+);
```

Hier ist ein Beispiel für Left Outer Join zwischen zwei Tabellen:

Beispieltabelle: MITARBEITER

```
+-----------+---------+
|    NAME   |  DEPTNO |
+-----------+---------+
|     A     |    2    |
|     B     |    1    |
|     C     |    3    |
|     D     |    2    |
|     E     |    1    |
|     F     |    1    |
|     G     |    4    |
|     H     |    4    |
+-----------+---------+
```

Beispieltabelle: DEPT

```
+---------+--------------+
|  DEPTNO |   DEPTNAME   |
+---------+--------------+
|    1    |  ACCOUNTING  |
|    2    |    FINANCE   |
|    5    |   MARKETING  |
|    6    |      HR      |
+---------+--------------+
```

Wenn Sie nun die Abfrage ausführen:

```
SELECT
      *
FROM
      EMPLOYEE LEFT OUTER JOIN DEPT
      ON EMPLOYEE.DEPTNO = DEPT.DEPTNO;
```

Ausgabe:

```
+-----------+---------+---------+--------------+
|    NAME   |  DEPTNO |  DEPTNO |   DEPTNAME   |
+-----------+---------+---------+--------------+
|     F     |    1    |    1    |  ACCOUNTING  |
|     E     |    1    |    1    |  ACCOUNTING  |
|     B     |    1    |    1    |  ACCOUNTING  |
|     D     |    2    |    2    |    FINANCE   |
|     A     |    2    |    2    |    FINANCE   |
|     C     |    3    |         |              |
|     H     |    4    |         |              |
|     G     |    4    |         |              |
+-----------+---------+---------+--------------+
```

RECHTE AUSSEN VERBINDEN

Ein `RIGHT OUTER JOIN` führt eine Verknüpfung zwischen zwei Tabellen aus, für die eine explizite Verknüpfungsklausel erforderlich ist, jedoch nicht übereinstimmende Zeilen nicht aus der zweiten Tabelle ausgeschlossen werden.

Beispiel:

```
SELECT
      ENAME,
      DNAME,
      EMP.DEPTNO,
      DEPT.DEPTNO
FROM
      SCOTT.EMP RIGHT OUTER JOIN SCOTT.DEPT
      ON EMP.DEPTNO = DEPT.DEPTNO;
```

Da die nicht `SCOTT.DEPT` Zeilen von `SCOTT.DEPT` enthalten sind, aber nicht `SCOTT.EMP` Zeilen von `SCOTT.EMP` nicht, entspricht das obige der folgenden Anweisung, die `LEFT OUTER JOIN` .

```
SELECT
      ENAME,
      DNAME,
      EMP.DEPTNO,
      DEPT.DEPTNO
FROM
      SCOTT.DEPT RIGHT OUTER JOIN SCOTT.EMP
      ON DEPT.DEPTNO = EMP.DEPTNO;
```

Hier ist ein Beispiel für den Right Outer Join zwischen zwei Tabellen:

Beispieltabelle: MITARBEITER

```
+-----------+---------+
|    NAME   |  DEPTNO |
+-----------+---------+
|     A     |    2    |
|     B     |    1    |
|     C     |    3    |
|     D     |    2    |
|     E     |    1    |
|     F     |    1    |
|     G     |    4    |
|     H     |    4    |
+-----------+---------+
```

Beispieltabelle: DEPT

```
+---------+--------------+
|  DEPTNO |   DEPTNAME   |
+---------+--------------+
|    1    |  ACCOUNTING  |
|    2    |    FINANCE   |
|    5    |   MARKETING  |
```

```
|    6    |      HR      |
+---------+--------------+
```

Wenn Sie nun die Abfrage ausführen:

```
SELECT
      *
FROM
      EMPLOYEE RIGHT OUTER JOIN DEPT
      ON EMPLOYEE.DEPTNO = DEPT.DEPTNO;
```

Ausgabe:

```
+-----------+---------+---------+--------------+
|    NAME   |  DEPTNO |  DEPTNO |   DEPTNAME   |
+-----------+---------+---------+--------------+
|     A     |    2    |    2    |    FINANCE   |
|     B     |    1    |    1    |  ACCOUNTING  |
|     D     |    2    |    2    |    FINANCE   |
|     E     |    1    |    1    |  ACCOUNTING  |
|     F     |    1    |    1    |  ACCOUNTING  |
|           |         |    5    |   MARKETING  |
|           |         |    6    |      HR      |
+-----------+---------+---------+--------------+
```

Die Oracle-Syntax (+) entspricht der Abfrage:

```
SELECT *
FROM EMPLOYEE, DEPT
WHERE EMPLOYEE.DEPTNO(+) = DEPT.DEPTNO;
```

VOLL AUSSEN MITGLIED

Ein `FULL OUTER JOIN` führt eine Verknüpfung zwischen zwei Tabellen aus, die eine explizite Verknüpfungsklausel erfordert, aber nicht übereinstimmende Zeilen in beiden Tabellen nicht ausschließt. Mit anderen Worten, es werden alle Zeilen in jeder Tabelle zurückgegeben.

Beispiel:

```
SELECT
      *
FROM
      EMPLOYEE FULL OUTER JOIN DEPT
      ON EMPLOYEE.DEPTNO = DEPT.DEPTNO;
```

Hier ist ein Beispiel für einen vollständigen äußeren Join zwischen zwei Tabellen:

Beispieltabelle: MITARBEITER

```
+-----------+---------+
|    NAME   |  DEPTNO |
+-----------+---------+
```

```
|     A     |    2    |
|     B     |    1    |
|     C     |    3    |
|     D     |    2    |
|     E     |    1    |
|     F     |    1    |
|     G     |    4    |
|     H     |    4    |
+-----------+---------+
```

Beispieltabelle: DEPT

```
+---------+--------------+
|  DEPTNO |   DEPTNAME   |
+---------+--------------+
|    1    |  ACCOUNTING  |
|    2    |    FINANCE   |
|    5    |   MARKETING  |
|    6    |      HR      |
+---------+--------------+
```

Wenn Sie nun die Abfrage ausführen:

```
SELECT
      *
FROM
      EMPLOYEE FULL OUTER JOIN DEPT
      ON EMPLOYEE.DEPTNO = DEPT.DEPTNO;
```

Ausgabe

```
+-----------+---------+---------+--------------+
|    NAME   |  DEPTNO |  DEPTNO |   DEPTNAME   |
+-----------+---------+---------+--------------+
|     A     |    2    |    2    |    FINANCE   |
|     B     |    1    |    1    |  ACCOUNTING  |
|     C     |    3    |         |              |
|     D     |    2    |    2    |    FINANCE   |
|     E     |    1    |    1    |  ACCOUNTING  |
|     F     |    1    |    1    |  ACCOUNTING  |
|     G     |    4    |         |              |
|     H     |    4    |         |              |
|           |         |    6    |      HR      |
|           |         |    5    |   MARKETING  |
+-----------+---------+---------+--------------+
```

Hier wurden die Spalten, die nicht übereinstimmen, auf NULL gehalten.

ANTIJOIN

Ein Antijoin gibt Zeilen von der linken Seite des Prädikats zurück, für die es keine entsprechenden Zeilen auf der rechten Seite des Prädikats gibt. Es gibt Zeilen zurück, die nicht mit der Unterabfrage auf der rechten Seite übereinstimmen (NOT IN).

```
SELECT * FROM employees
   WHERE department_id NOT IN
   (SELECT department_id FROM departments
       WHERE location_id = 1700)
   ORDER BY last_name;
```

Hier ist ein Beispiel für Anti-Join zwischen zwei Tabellen:

Beispieltabelle: MITARBEITER

```
+-----------+---------+
|    NAME   |  DEPTNO |
+-----------+---------+
|     A     |    2    |
|     B     |    1    |
|     C     |    3    |
|     D     |    2    |
|     E     |    1    |
|     F     |    1    |
|     G     |    4    |
|     H     |    4    |
+-----------+---------+
```

Beispieltabelle: DEPT

```
+---------+--------------+
|  DEPTNO |   DEPTNAME   |
+---------+--------------+
|    1    |  ACCOUNTING  |
|    2    |    FINANCE   |
|    5    |   MARKETING  |
|    6    |      HR      |
+---------+--------------+
```

Wenn Sie nun die Abfrage ausführen:

```
SELECT
      *
FROM
      EMPLOYEE WHERE DEPTNO NOT IN
      (SELECT DEPTNO FROM DEPT);
```

Ausgabe:

```
+-----------+---------+
|    NAME   |  DEPTNO |
+-----------+---------+
|     C     |    3    |
|     H     |    4    |
|     G     |    4    |
+-----------+---------+
```

Die Ausgabe zeigt, dass nur die Zeilen der EMPLOYEE-Tabelle, deren DEPTNO nicht in der DEPT-Tabelle vorhanden war.

SEMIJOIN

Eine Semijoin-Abfrage kann beispielsweise dazu verwendet werden, alle Abteilungen mit mindestens einem Mitarbeiter zu finden, dessen Gehalt 2500 übersteigt.

```
SELECT * FROM departments
   WHERE EXISTS
   (SELECT 1 FROM employees
       WHERE departments.department_id = employees.department_id
       AND employees.salary > 2500)
   ORDER BY department_name;
```

Dies ist effizienter als die vollständige Join-Alternative, da inneres Beitreten von Mitarbeitern dann eine Klausel enthält, in der angegeben wird, dass das Gehalt mehr als 2500 betragen muss und dieselbe Abteilung mehrmals wiederkommen könnte. Wenn die Feuerwehr über `n` Mitarbeiter verfügt, die alle das Gehalt 3000 haben, `select * from departments, employees` mit den erforderlichen IDs aus und unsere where-Klausel würde die Feuerwehr `n` mal zurückgeben.

BEITRETEN

Die `JOIN` Operation führt eine Verknüpfung zwischen zwei Tabellen aus, wobei alle nicht übereinstimmenden Zeilen aus der ersten Tabelle ausgeschlossen werden. Ab Oracle 9i entspricht der `JOIN` in seiner Funktion dem `INNER JOIN` . Dieser Vorgang erfordert eine explizite Join-Klausel, im Gegensatz zu den Operatoren `CROSS JOIN` und `NATURAL JOIN` .

Beispiel:

```
select t1.*,
       t2.DeptId
  from table_1 t1
  join table_2 t2 on t2.DeptNo = t1.DeptNo
```

Oracle-Dokumentation:

- 10 g
- 11g
- 12g

NATÜRLICHER JOIN

NATURAL JOIN erfordert keine explizite Join-Bedingung. Es wird ein Feld erstellt, das auf allen Feldern mit demselben Namen in den verbundenen Tabellen basiert.

```
create table tab1(id number,  descr varchar2(100));
create table tab2(id number,  descr varchar2(100));
insert into tab1 values(1, 'one');
insert into tab1 values(2, 'two');
insert into tab1 values(3, 'three');
insert into tab2 values(1, 'ONE');
insert into tab2 values(3, 'three');
```

Der Join wird für die Felder ID und DESCR durchgeführt, die für beide Tabellen gelten:

```
SQL> select *
  2  from tab1
  3        natural join
  4       tab2;

        ID DESCR
---------- ----------
         3 three
```

Spalten mit unterschiedlichen Namen werden in der JOIN-Bedingung nicht verwendet:

```
SQL> select *
  2  from (select id as id, descr as descr1 from tab1)
  3       natural join
  4       (select id as id, descr as descr2 from tab2);

        ID DESCR1     DESCR2
---------- ---------- ----------
         1 one        ONE
         3 three      three
```

Wenn die verknüpften Tabellen keine gemeinsamen Spalten haben, wird ein JOIN ohne Bedingungen ausgeführt:

```
SQL> select *
  2  from (select id as id1, descr as descr1 from tab1)
  3       natural join
  4       (select id as id2, descr as descr2 from tab2);

       ID1 DESCR1            ID2 DESCR2
---------- ---------- ---------- ----------
         1 one                 1 ONE
         2 two                 1 ONE
         3 three               1 ONE
         1 one                 3 three
         2 two                 3 three
         3 three               3 three
```

VERBINDUNGEN online lesen: https://riptutorial.com/de/oracle/topic/4192/verbindungen

Kapitel 33: Verschiedene Möglichkeiten, Datensätze zu aktualisieren

Syntax

- UPDATE-Tabellenname [[AS] Korrelationsname] SET Spaltenname = Wert [, Spaltenname = Wert}] * [WHERE-Klausel]
- UPDATE-Tabellenname SET Spaltenname = Wert [, Spaltenname = Wert] * WHERE CURRENT OF

Examples

Aktualisieren Sie die Syntax mit einem Beispiel

Normales Update

```
UPDATE
      TESTTABLE
SET
      TEST_COLUMN= 'Testvalue',TEST_COLUMN2= 123
WHERE
      EXISTS
          (SELECT MASTERTABLE.TESTTABLE_ID
           FROM MASTERTABLE
           WHERE ID_NUMBER=11);
```

Update über Inline-Ansicht

Verwenden der Inline-Ansicht (falls von Oracle als aktualisierbar erachtet)

Hinweis : Wenn Sie einen Fehler erhalten, der nicht durch Schlüssel erhalten bleibt, fügen Sie einen Index hinzu, um ihn aufzulösen, damit er aktualisierbar ist

```
UPDATE
      (SELECT
            TESTTABLE.TEST_COLUMN AS OLD,
            'Testvalue' AS NEW
       FROM
                TESTTABLE
            INNER JOIN
                MASTERTABLE
            ON TESTTABLE.TESTTABLE_ID = MASTERTABLE.TESTTABLE_ID
       WHERE ID_NUMBER=11) T
SET
      T.OLD        = T.NEW;
```

Aktualisieren Sie mit Merge

Merge verwenden

```
MERGE INTO
      TESTTABLE
USING
      (SELECT
            T1.ROWID AS RID,
            T2.TESTTABLE_ID
       FROM
                TESTTABLE T1
            INNER JOIN
                MASTERTABLE T2
            ON TESTTABLE.TESTTABLE_ID = MASTERTABLE.TESTTABLE_ID
       WHERE ID_NUMBER=11)
ON
      ( ROWID = RID )
WHEN MATCHED
THEN
    UPDATE SET TEST_COLUMN= 'Testvalue';
```

Mit Beispieldaten zusammenführen

```
drop table table01;
drop table table02;

create table table01 (
       code int,
       name varchar(50),
       old int
);

create table table02 (
       code int,
       name varchar(50),
       old int
);

truncate table table01;
insert into table01 values (1, 'A', 10);
insert into table01 values (9, 'B', 12);
insert into table01 values (3, 'C', 14);
insert into table01 values (4, 'D', 16);
insert into table01 values (5, 'E', 18);

truncate table table02;
insert into table02 values (1, 'AA', null);
insert into table02 values (2, 'BB', 123);
insert into table02 values (3, 'CC', null);
insert into table02 values (4, 'DD', null);
insert into table02 values (5, 'EE', null);

select * from table01 a order by 2;
select * from table02 a order by 2;

--

merge into table02 a using (
      select b.code, b.old from table01 b
) c on (
```

```
  a.code = c.code
)
when matched then update set a.old = c.old
;

--

select a.*, b.* from table01 a
inner join table02 b on a.code = b.codetable01;

select * from table01 a
where
       exists
       (
         select 'x' from table02 b where a.code = b.codetable01
       );

select * from table01 a where a.code in (select b.codetable01 from table02 b);

--

select * from table01 a
where
       not exists
       (
         select 'x' from table02 b where a.code = b.codetable01
       );

select * from table01 a where a.code not in (select b.codetable01 from table02 b);
```

Verschiedene Möglichkeiten, Datensätze zu aktualisieren online lesen: https://riptutorial.com/de/oracle/topic/4193/verschiedene-moglichkeiten--datensatze-zu-aktualisieren

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit Oracle Database	Community, J. Chomel, Jeffrey Kemp, Jon Ericson, Kevin Montrose, Mark Stewart, Sanjay Radadiya, Steven Feuerstein, tonirush
2	Abfrage auf Ebene	Sanjay Radadiya, TechEnthusiast
3	Anonymer PL / SQL-Block	Jon Heller, Skynet, Zohar Elkayam
4	Autonome Transaktionen	phonetic_man
5	Datenbank-Links	carlosb, Daniel Langemann, g00dy, kasi
6	Datenpumpe	Vidya Thotangare
7	Datenwörterbuch	Mark Stewart, Pancho, Slava Babin
8	DUAL-Tisch	Slava Babin
9	Dynamisches SQL	Dmitry
10	Echte Anwendungssicherheit	Ben H
11	Einen Kontext erstellen	Jeffrey Kemp
12	Einschränken der von einer Abfrage zurückgegebenen Zeilen (Paginierung)	Ahmed Mohamed, Martin Schapendonk, Matas Vaitkevicius, Sanjay Radadiya, tonirush, trincot
13	Einschränkungen	SSD
14	Fehlerprotokollierung	zygimantus
15	Fensterfunktionen	Leigh Riffel
16	Hierarchischer Abruf mit Oracle Database 12C	Muntasir, Vahid

17	Hinweise	Aleksej, Florin Ghita, Jon Heller, Mark Stewart, Pirate X
18	Indizes	smshafiqulislam
19	Mit Datumsangaben arbeiten	David Aldridge, Florin Ghita, Jeffrey Kemp, Mark Stewart, tonirush, zygimantus
20	Oracle Advanced Queuing (AQ)	Jon Theriault
21	Oracle MAF	Anand Raj
22	Partitionierung der Tabelle	BobC, carlosb, ivanzg, JeromeFr, Kamil Islamov, Stephen Leppik, tonirush
23	Rekursives Unterabfrage-Factoring mit der WITH-Klausel (AKA-Ausdrücke für allgemeine Tabellen)	B Samedi, MT0
24	Schlüsselwörter oder Sonderzeichen abgrenzen	dev
25	Sequenzen	Pranav Shah, SriniV
26	Statistische Funktionen	Evgeniy K., Matas Vaitkevicius, ppeterka, Pranav Shah
27	String-Manipulation	carlosb, Eric B., Florin Ghita, Francesco Serra, J. Chomel, J.Hudler, Jeffrey Kemp, Mark Stewart, SriniV, Thunder, walen, zhliu03
28	Termine	carlosb, MT0, Roman, tonirush
29	Trennstrings trennen	Arkadiusz Łukasiewicz, MT0
30	Umgang mit NULL-Werten	Dalex, JeromeFr
31	Update mit Joins	mathguy
32	VERBINDUNGEN	Aleksej, B Samedi, Bakhtiar Hasan, Daniel Langemann, Erkan Haspulat, Pranav Shah, Robin James, SriniV, Sumner Evans
33	Verschiedene Möglichkeiten, Datensätze zu	nimour pristou, Nogueira Jr, SriniV

	aktualisieren	