

Cornel Brücher • Frank Jüdes
Mario Haupt • Uwe Küchler • Andreas Takano

Oracle Survival Guide

Das *Schweizer Messer* für
Oracle-Entwickler und -Supporter

Die wichtigsten Informationen zum
Überleben in Oracle-Projekten

Leicht verständliche und kurze
Erklärungen, nachschlagefähiger
Aufbau aller Fachgebiete,
praxiserprobte Beispiele

SQL

SQL (Structured Query Language) ist die standardisierte Datenzugriffssprache für relationale Datenbanken. SQL ist keine Programmiersprache, da sich keine Programmabläufe formulieren lassen. Die aktuelle Version erlaubt zwar schon ausgefeilte Entscheidungsstrukturen mit CASE oder DECODE, aber Schleifen und Befehlsfolgen existieren nicht. (Diese Anforderungen werden von PL/SQL erfüllt.)

SQL unterteilt sich in DDL (Data Definition Language) und DML (Data Manipulation Language). In DDL werden die Datenbankobjekte und -strukturen definiert (zum Beispiel Tabellen angelegt). Mit DML werden die Daten eingefügt (**INSERT**), geändert (**UPDATE**), abgefragt (**SELECT**) und gelöscht (**DELETE**).

Das Einfügen von Datensätzen mit einem **INSERT**-Statement geschieht meistens satzweise. **SELECT**-, **UPDATE**- und **DELETE**-Statements arbeiten mit Datenmengen. In der **WHERE**-Klausel der **SELECT**-, **UPDATE**- und **DELETE**-Statements werden die Eigenschaften der Datensätze (»Records«) in der Treffermenge angegeben. Dazu gehört auch das Arbeiten mit Vereinigungsmengen, Schnittmengen und Mengensubtraktion. Die Datensätze eines **SELECT**-Statements entsprechen nicht zwangsweise den Datensätzen einer Tabelle in der Datenbank, sondern können durch Verknüpfungen (»Joins«) aus Datensätzen verschiedener Tabellen, die durch sogenannte Relationen verbunden sind, zusammengesetzt sein. Die Spalten einer **SELECT**-Ergebnismenge entsprechen auch nicht zwangsweise den Spalten der abgefragten Tabellen. Es kann sich um Ergebnisspalten von Berechnungen oder der Anwendung von SQL-Funktionen handeln. Ein

```
SELECT 1+1 FROM dual;
```

liefert unabhängig von den Spalten der Tabelle (dual) das Ergebnis 2 zurück, für jede Zeile in der Tabelle einmal.

Mit einem **SELECT**-Statement formulieren Sie nicht, was ein Programm tun soll, sondern Sie beschreiben nur das Ergebnis, das Sie sehen wollen, und benennen die Strukturen, in denen die gewünschten Daten zu finden sind.

1.1 Data Manipulation Language (DML)

DML-Befehle dienen dem Einfügen, Abfragen und Verändern der Daten in den Tabellen einer Datenbank.

1.1.1 Einfache Abfrage ohne Bedingung (SELECT)

```
-- Das ist eine Kommentarzeile  
SELECT spalte1  
      , spalte2  
FROM   tabelle;
```

1.1.2 Abfragen mit WHERE-Bedingungen

Nur komplette Tabellen auszugeben ist auf Dauer ziemlich langweilig, mit einer **WHERE**-Bedingung lässt sich die Ausgabe des **SELECT**-Befehls einschränken:

```
-- Das ist eine Kommentarzeile  
SELECT spalte1  
      , spalte2  
FROM   tabelle  
WHERE  (spalte = wert);
```

Einzelbedingungen

Die Verwendung von Klammern rund um **WHERE**-Bedingungen ist aus Übersichtsgründen sehr zu empfehlen, vor allem wenn Bedingungen kombiniert und damit richtig komplex werden. Hier nur ein Überblick über die angebotenen Möglichkeiten:

Bedingung	Erläuterung
wert1 = wert2	wert1 ist gleich wert2
wert1 < wert2	wert1 ist kleiner als wert2
wert1 > wert2	wert1 ist größer als wert2
wert1 <= wert2	wert1 ist kleiner als oder gleich wert2
wert1 >= wert2	wert1 ist größer als oder gleich wert2
wert1 <> wert2	wert1 ist ungleich wert2
wert1 != wert2	wert1 ist ungleich wert2
wert BETWEEN wert1 AND wert2	wert ist im Bereich zwischen wert1 und wert2 (einschließlich), Bedingung entspricht: wert >= wert1 AND wert <= wert2

Tabelle 1.1: WHERE-Bedingungen

Bedingung	Erläuterung
wert NOT BETWEEN wert1 AND wert2	wert ist nicht im angegebenen Bereich, also entweder kleiner als wert1 oder größer als wert2 entspricht: (wert < wert1) OR (wert > wert2)
wert IN (wert1,wert2,...,wertn)	wert ist in der angegebenen Werteliste
wert1 NOT IN (wert1,wert2,...,wertx)	wert ist nicht in der angegebenen Werteliste
spalte1 LIKE '%wert1%'	spalte1 enthält das Textmuster wert1 an beliebiger Position
spalte1 LIKE 'wert1%'	spalte1 enthält das Textmuster wert1 am Anfang
spalte1 LIKE '%wert1'	spalte1 enthält das Textmuster wert1 am Ende
spalte1 NOT LIKE '%wert1%'	spalte1 enthält nicht das Textmuster wert1
spalte1 LIKE 'AB_DE'	spalte1 entspricht AB gefolgt von genau einem beliebigen Zeichen gefolgt von DE
wert1 IS NULL	wert1 ist leer
wert1 IS NOT NULL	wert1 ist nicht leer
REGEXP_LIKE(spalte1, 'RegExp', 'quellstring')	spalte1 enthält das mit RegExp beschriebene Textmuster; mehr zu regulären Ausdrücken steht im Kapitel 15 <i>Tipps & Tricks</i>

Tabelle 1.1: WHERE-Bedingungen (Forts.)

Escaping von Sonderzeichen bei LIKE

Für die Suche mit LIKE gibt es, wie oben beispielhaft angegeben, zwei Sonderzeichen:

- "%" für beliebig viele beliebige Zeichen
- "_" für genau ein beliebiges Zeichen

Das ist genau dann unpraktisch, wenn das Suchmuster ein oder beide Sonderzeichen enthalten soll. Für diesen Zweck gibt es einen "Escaping"-Mechanismus. Man gibt nach dem LIKE-Suchmuster das Escaping-Zeichen an, und benutzt es im Suchstring. Das Escaping-Zeichen sorgt dafür, dass genau ein folgendes Zeichen nicht als Sonderzeichen interpretiert wird.

- SELECT * FROM user_tables
- WHERE table_name LIKE '%_TMP_%';

Dieser SELECT bringt alle Tabellennamen, die ein "_TMP_" enthalten, aber auch alle, die nur "TMP" enthalten, da die Unterstriche als jeweils ein beliebiges Zeichen angesehen werden.

- SELECT * FROM user_tables
- WHERE table_name LIKE '%_TMP_%' ESCAPE '\';

Dieser SELECT bringt alle Tabellennamen, die ein "_TMP_" enthalten, aber nicht alle anderen, die nur "TMP" enthalten, da die Unterstriche jetzt mit zur gesuchten Zeichenkette gehören.

Verneinung von Bedingungen

Grundsätzlich kann jede Bedingung (und jede Kombination von Bedingungen) durch ein vorangestelltes **NOT** verneint werden. Klammern müssen nicht immer gesetzt werden, sind aber für die Übersichtlichkeit sehr förderlich. Das **NOT** kann bei einigen Bedingungen auch an anderer Stelle gesetzt werden (siehe Tabelle 1.1).

```
... WHERE NOT job LIKE 'SALES%'  
entspricht:  
... WHERE job NOT LIKE 'SALES%'  
Nicht möglich ist:  
... WHERE dept NOT = 30  
Richtig wäre:  
... WHERE NOT (dept = 30)  
oder:  
... WHERE dept != 30  
oder auch:  
... WHERE dept <> 30
```

Verknüpfung von Bedingungen

Mehrere **WHERE**-Bedingungen können mit **AND** sowie **OR** verknüpft werden. **AND** ist hochwertiger als **OR**, die **AND**-Verknüpfungen werden also zuerst berücksichtigt. Für die Übersichtlichkeit empfiehlt es sich auch hier, immer Klammern zu verwenden:

```
Bedingung1 AND Bedingung2 OR Bedingung3
```

entspricht:

```
(Bedingung1 AND Bedingung2) OR Bedingung3
```

1.1.3 Abfrage mit Verknüpfung über zwei Tabellen (JOIN)

```
SELECT a.spalte1  
      , b.spalte2  
FROM   tabelle1 a -- Buchstabe a als Tabellen-Alias  
      , tabelle2 b -- Buchstabe b als Tabellen-Alias  
WHERE  a.key_spalte = b.key_spalte;
```

Über die **WHERE**-Bedingung werden die zusammengehörigen Datensätze zweier Tabellen verknüpft. Ohne diese Bedingung würde jede Zeile von `tabelle1` mit jeder Zeile von `tabelle2` kombiniert (sogenanntes »kartesisches Produkt«).

Zukunftssicherer und einfacher zu lesen ist die neuere ANSI-Syntax von 1992, deren Verwendung wir ausdrücklich empfehlen. Das obige Beispiel sieht in der neuen Syntax so aus:

```
SELECT a.spalte1
,      b.spalte2
FROM   tabelle1 a -- Buchstabe a als Tabellen-Alias
INNER
JOIN   tabelle2 b -- Buchstabe b als Tabellen-Alias
ON     a.foreign_key_spalte = b.primary_key_spalte;
```

Das Schlüsselwort **INNER** ist immer optional, wir empfehlen jedoch, in eigenen SQL-Statements immer **INNER** zu verwenden. (Na, beim Lesen dieses Satzes gestolpert? – Absicht!)

1.1.4 Kartesisches Produkt über zwei Tabellen (neue ANSI-Syntax)

Bei Verwendung der neuen Syntax ist es unmöglich, versehentlich ein kartesisches Produkt zu erzeugen, da dieses explizit als **CROSS JOIN** angefordert werden muss:

```
SELECT a.spalte1
,      b.spalte2
FROM   tabelle1 a -- Buchstabe a als Tabellen-Alias
CROSS
JOIN   tabelle2 b -- Buchstabe b als Tabellen-Alias;
```

Hierbei werden alle Datensätze von `tabelle1` mit allen Datensätzen von `tabelle2` verknüpft, eben »Kreuzprodukte« gebildet. Übrigens: Durch das Bilden von Kreuzprodukten auf größeren Tabellen schafft man sich viele Freunde unter den DBAs ...

1.1.5 Offene Verknüpfung über zwei Tabellen (OUTER JOIN)

```
SELECT a.spalte1
,      b.spalte2
FROM   tabelle1 a -- Buchstabe a als Tabellen-Alias
,      tabelle2 b -- Buchstabe b als Tabellen-Alias
WHERE  a.spalte99 = b.spalte99(+);
```

Es werden alle Zeilen aus `tabelle1` (Alias `a`) angezeigt und die dazu passenden Einträge aus `tabelle2` (Alias `b`). Wenn kein zu `tabelle1` passender Eintrag in `tabelle2` gefunden wird, wird das Feld `b.spalte2` leer angezeigt (Inhalt `NULL`). Mit dieser Form eines Joins, genannt **OUTER JOIN**, werden auch die Zeilen angezeigt, zu denen in der verknüpften (auf Denglisch: »gejointen«) Tabelle keine passenden Einträge existieren.

In der neuen ANSI-Syntax sieht das obige Beispiel so aus:

```
SELECT a.spalte1
,      b.spalte2
FROM   tabelle1 a -- Buchstabe a als Tabellen-Alias
LEFT OUTER
JOIN   tabelle2 b -- Buchstabe b als Tabellen-Alias
ON     a.spalte99 = b.spalte99;
```

Außer dem **LEFT Outer Join** gibt es noch den **RIGHT Outer Join**, der sich aber nur durch die Reihenfolge der Tabellen unterscheidet. Jeder **RIGHT OUTER JOIN** kann in einen **LEFT OUTER JOIN** überführt werden, empfehlenswert ist es, sich für eine Join-Art zu entscheiden, also entweder rechts oder links.

Zusätzlich gibt es in der ANSI-Syntax auch noch den **FULL OUTER JOIN**:

```
SELECT a.spalte1
,      b.spalte2
FROM   tabelle1 a -- Buchstabe a als Tabellen-Alias
FULL OUTER
JOIN   tabelle2 b -- Buchstabe b als Tabellen-Alias
ON     a.foreign_key_spalte = b.primary_key_spalte;
```

Hierbei werden zusätzlich zum »normalen« Outer Join auch alle Zeilen der »Gegenrichtung« zurückgeliefert, in denen es für einen Schlüssel in der verknüpften (»gejointen«) Tabelle keine Entsprechung in der verknüpfenden (auch genannt: treibenden) Tabelle gibt. Ein **FULL OUTER JOIN** entspricht einem **LEFT OUTER JOIN**, der über **UNION** mit einem **RIGHT OUTER JOIN** kombiniert ist (siehe Abschnitt *Abfragemengen verknüpfen (UNION/MINUS)* weiter hinten in diesem Kapitel).

1.1.6 Geschachtelte Abfrage (Subquery/Unterabfrage)

```
SELECT a.spalte1
FROM   tabelle1 a -- Buchstabe a als Tabellen-Alias
WHERE  a.spalte99 IN (SELECT spalte2
                     FROM   tabelle2
                     );
```

Es werden alle Zeilen aus `tabelle1` (Alias `a`) angezeigt, deren Werte in `tabelle2.spalte2` existieren. Ergibt die Unterabfrage keinen Treffer, wird insgesamt kein Treffer zurückgeliefert.

Bei der geschachtelten Abfrage kann auch mit anderen Operatoren gearbeitet werden. Beispiel:

```
SELECT a.spalte1
FROM tabelle1 a -- Buchstabe a als Tabellen-Alias
WHERE a.spalte1 = (SELECT spalte2
                  FROM tabelle2
                  WHERE spalte3 = wert
                  );
```

Diese Abfrage funktioniert nur, wenn die Unterabfrage *genau einen* Wert zurückliefert. Wenn die Unterabfrage keinen oder mehrere Werte zurückliefert, führt das zu einem Fehler. Sinnvoll ist die Verwendung der Unterabfrage also eher, wenn man auch mehrere Elemente erwartet und man den **IN**-Operator verwendet.

```
SELECT a.spalte1
FROM tabelle1 a -- Buchstabe a als Tabellen-Alias
WHERE a.spalte1 IN (SELECT spalte2
                  FROM tabelle2
                  WHERE spalte3 = wert);
```

Subqueries können an vier Stellen im **SELECT**-Statement eingesetzt werden: In der **WHERE**-Bedingung zur Ergebniseinschränkung, in der **FROM**-Klausel zur Bildung der Tabelle, in den Spaltenangaben und schließlich auch noch in der **ORDER BY**-Klausel zum Sortieren. Bei Einsatz als Ergebnisspalte gilt, wie auch bei Einsatz mit `=` in der **WHERE**-Bedingung, dass ein Mehrfachtreffer zu einem Fehler führt.

Beispiel für den Einsatz von Subqueries:

```
-- Alle selektieren, die den gleichen Job wie Jones haben
-- Variante1: Geschachtelte Selects
SELECT empno
,      ename
,      job
,      (SELECT dname FROM dept WHERE deptno = e.deptno) dname
FROM   (SELECT empno, ename, job, deptno FROM emp) e
WHERE  job = (SELECT job FROM emp WHERE ename = 'JONES');
```

Die Subqueries sind eine bequeme Lösung, aber man kann das gleiche Ergebnis auch mit Joins erreichen. Faustregel: Unterabfragen verwenden zum Einschränken der Ergebnismenge Joins, um Daten auszugeben.

```
-- Alle selektieren, die den gleichen Job wie Jones haben
-- Variante2: Joins
SELECT e1.empno
,      e1.ename
,      e1.job
,      d.dname
FROM   emp e1
,      emp e2
,      dept d
WHERE  e1.deptno = d.deptno
AND    e1.job = e2.job
AND    e2.ename = 'JONES';
```

1.1.7 Abfrage mit Gruppierung (GROUP BY HAVING)

```
SELECT gruppenfunktion(spalte1)
,      spalte2
FROM   tabelle
WHERE  spalte = wert
GROUP BY spalte2
HAVING gruppenfunktion(spalte) > wert;
```

Die Bedingungen hinter **WHERE** beziehen sich immer auf einzelne Zeilen (bevor sie gruppiert werden), die Bedingungen hinter **HAVING** auf das Ergebnis von Gruppenfunktionen (nach der Gruppierung). Häufig benutzte Gruppenfunktionen:

Funktion	Erläuterung
avg(spalte)	Berechnet den Durchschnittswert über eine Spalte
count(spalte)	Zählt die Anzahl der nicht leeren Zellen einer Spalte
max(spalte)	Ermittelt den Maximalwert einer Spalte
min(spalte)	Ermittelt den Minimalwert einer Spalte
sum(spalte)	Berechnet die Summe über eine Spalte

Tabelle 1.2: Häufig verwendete Gruppenfunktionen

Die vollständige Liste der Gruppenfunktionen finden Sie in der Oracle SQL Reference unter Functions > SQL Functions > Aggregate Functions.

1.1.8 Abfrage mit Satzsperrung (FOR UPDATE)

```
SELECT spalte1
,      spalte2
FROM   tabelle
WHERE  spalte = wert
FOR UPDATE;
```

Durch die **FOR UPDATE**-Klausel werden die selektierten Datensätze gesperrt, sodass andere Benutzer diese Datensätze weder sperren noch ändern können, bis Sie Ihre Transaktion durch **COMMIT** oder **ROLLBACK** beendet haben. Wenn Sie einen **SELECT** mit **FOR UPDATE**-Klausel über mehrere gejointe Tabellen hinweg ausführen, werden die betroffenen Datensätze in allen beteiligten Tabellen gesperrt.

```
SELECT a.spalte1
,      b.spalte2
FROM   tabelle1 a -- Buchstabe a als Tabellen-Alias
,      tabelle2 b -- Buchstabe b als Tabellen-Alias
WHERE  a.foreign_key_spalte = b.primary_key_spalte
AND    a.spalte = wert
FOR UPDATE; -- Sperre in beiden Tabellen
```

Wenn Sie die Satzsperrung auf eine Tabelle beschränken wollen, dann muss eine Spalte dieser Tabelle in der **FOR UPDATE**-Klausel angegeben werden.

```
SELECT a.spalte1
,      b.spalte2
FROM   tabelle1 a -- Buchstabe a als Tabellen-Alias
,      tabelle2 b -- Buchstabe b als Tabellen-Alias
WHERE  a.foreign_key_spalte = b.primary_key_spalte
AND    a.spalte = wert
FOR UPDATE OF a.irgendeinespalte; --Sperre nur in Tabelle a;
```

Die Angabe der Spalte dient nur syntaktischen Zwecken, die Auswahl der Spalte ist irrelevant, da Oracle immer nur komplette Zeilen sperrt.

1.1.9 Hierarchische Abfrage (CONNECT BY PRIOR)

```
SELECT level -- Hierarchische Ebene des Datensatzes
,      spalte1
,      spalte2
,      spalte3
```

```
FROM tabelle
CONNECT BY PRIOR spalte1 = spalte3
START WITH spalte3 IS NULL; -- Rootelement festlegen
```

Der Inhalt von Spalte3 entspricht der Spalte1 im übergeordneten Datensatz.

CONNECT BY PRIOR spalte1 = spalte3 heißt also: »Verbindung durch spalte1 des vorrangigen (prior) Records mit spalte3 im aktuellen Record«. Das Rootelement hat keinen übergeordneten Record, also ist dort spalte3 mit **NULL** belegt. Die Startbedingung kann allerdings beliebig festgelegt werden, zum Beispiel:

```
START WITH position = 'BOSS';
```

Eine nützliche SQL-Funktion zur Anzeige des Pfades zu einem Element der hierarchischen Struktur ist **SYS_CONNECT_BY_PATH**:

```
SELECT SYS_CONNECT_BY_PATH(ename, '/') "Path"
FROM emp
CONNECT BY (PRIOR empno = mgr)
START WITH mgr is null;
```

Das Ergebnis sieht in SQL*Plus dann so aus:

```
SQL> SELECT SYS_CONNECT_BY_PATH(ename, '/') "Path"
2 FROM emp
3 CONNECT BY (prior empno = mgr)
4 START WITH mgr is null;
```

```
Path
-----
/KING
/KING/JONES
/KING/JONES/SCOTT
/KING/JONES/SCOTT/ADAMS
/KING/JONES/FORD
/KING/JONES/FORD/SMITH
/KING/BLAKE
/KING/BLAKE/ALLEN
/KING/BLAKE/WARD
/KING/BLAKE/MARTIN
/KING/BLAKE/TURNER
/KING/BLAKE/JAMES
/KING/CLARK
/KING/CLARK/MILLER
```