NULL

Was Sie schon immer über NICHTS wissen wollten…

Uwe Küchler, Senior Consultant

OPITZ CONSULTING Deutschland GmbH

(NL Bad Homburg v.d.H.)

E-Mail: uwe.kuechler@opitz-consulting.com

Web: www.opitz-consulting.com

DOAG-Konferenz Nürnberg, 21.11.2013

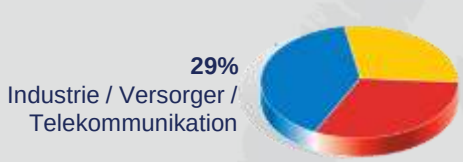
Mission

Wir entwickeln gemeinsam mit allen Branchen Lösungen, die dazu führen, dass sich diese Organisationen besser entwickeln als ihr Wettbewerb.

Unsere Dienstleistung erfolgt partnerschaftlich und ist auf eine langjährige Zusammenarbeit angelegt.

Märkte

- Branchenübergreifend
- Über 600 Kunden



Leistungsangebot

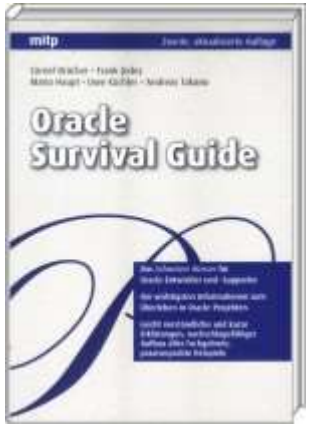
- Business IT Alignment
- Business Information Management
- Business Process Management
- Anwendungsentwicklung
- SOA und System-Integration
- IT-Infrastruktur-Management

Eckdaten

- Gründung 1990
- 400 Mitarbeiter
- 9 Standorte

Zur Person

- Generation C=64
- Seit über 25 Jahren in der IT tätig
- 1997-2000 bei Oracle Deutschland
- Seither durchgehend Oracle-Berater, im DBA- und Entwicklungs-Umfeld, Tutor
- Seit 09/2013 bei OPITZ CONSULTING
- Buch- und Blogautor (oraculix.de)
- Performance als „Steckenpferd“



Agenda

1. **Historie und Hintergründe**
2. **Umgang mit NULLs**
3. **Performance**
4. **Mythen und Missverständnisse**
5. **Fragen + Antworten**

1 Historie und Hintergründe

“Tertium non datur.”

Satz vom ausgeschlossenen Dritten

Historie und Hintergründe: Eine kurze Geschichte von Nichts

- **Aristoteles (4. Jhdt. v. Chr.)**
  - Begründer der antiken Logik
  - Formuliert erstmalig Satz vom ausgeschlossenen Dritten
  - Formuliert aber auch Problem mit zukünftigen, noch unbekannten Werten
  - Seeschlacht-Beispiel in „Περὶ ἑρμηνείας“, 9. Kapitel

- **Jan Łukasiewicz**
  - Formalisierte 1920 die ternäre Logik („L_3“)
  - Bezog sich dabei auf Aristoteles‘ Seeschlacht-Beispiel

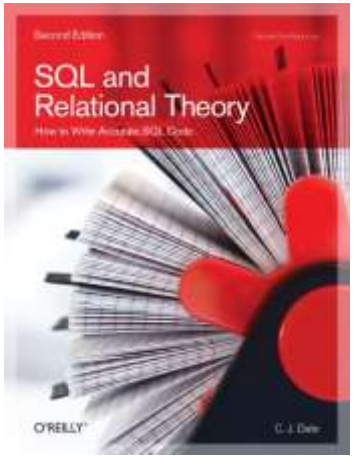
- **Moderne Logik:**
  - Weist Satz vom ausgeschlossenen Dritten zurück
  - Dritte Möglichkeit: Wahrheitswert einer Aussage ist ***unbekannt***
  - Gegenentwurf: Schwache Negation: Was nicht als wahr bewiesen werden kann, gilt zunächst als falsch.

Historie und Hintergründe: Eine kurze Geschichte von Nichts (2)

- **E. F. Codd (1923-2003)**
  - 1969-1970: "A Relational Model of Data for Large Shared Data Banks“
    - NULL als “Platzhalter” (nicht als Wert) ohne Datentyp
  - 1980‘er: Veröffentlichung von 13 Regeln (0-12), die ein RDBMS ausmachen („Codds Zwölf Gebote“, „Codd‘s Law“, …)
  - **Regel 3**: Ein DBMS muss Null-Werte zur systematischen Darstellung fehlender und unanwendbarer Information unterstützen, unabhängig von Datentypen.
  - Im späteren Modell 1990 zwei Arten von NULLs (applicable/inapplicable):
    - “A-Marker”: Der *Wert* ist unbekannt.
    - “I-Marker”: Das *Attribut* existiert nicht.
- **Erste kommerzielle Umsetzung: SEQUEL von IBM**
  - Codd arbeitete für IBM, sein “Alpha” wurde aber nicht verwendet.
  - SQL entstand aus SEQUEL heraus.
- **Implementierung von SQL kennt nur *ein* NULL.**

Historie und Hintergründe: Eine kurze Geschichte von Nichts (3)

- **Chris J. Date, Hugh Darwen, et al.**
  - Date war Geschäftspartner von Codd, aber
  - Gegner der Einführung der NULL im relationalen Modell.
  - Debatte darüber bis heute nicht verstummt
  - >>**SQL's "Nulls" Are A Disaster**<< ("The Third Manifesto" 2000, "Relational Database Writings" von Date und Darwen 1985-1997)
- **Hauptkritikpunkt an SQL: Ternäre statt binäre Logik**
  - Die Implementierung des neueren Modells nach Codd würde sogar eine vierwertige Logik bedeuten.



Historie und Hintergründe: Dreiwertige (ternäre) Logik

- **Jan Łukasiewicz**
  - Formalisierte 1920 die ternäre Logik ("L_3")

A und B			
A	B		
	f	u	w
f	f	f	f
u	f	u	u
w	f	u	w

A oder B			
A	B		
	f	u	w
f	f	u	w
u	u	u	w
w	w	w	w

nicht A	
A	nicht A
f	w
u	u
w	f

Historie und Hintergründe: Dreiwertige Logik in SQL

- **Deduktions- und Beweisregeln**
  - Łukasiewicz et al. hatten Ihre Logiksysteme damit ausgestattet
  - Problemfall „Implikation“: Mit 3VL mehrere Varianten
- **SQL kennt solche Regeln nicht**
  - ➔ kein formal logisches System!
  - ➔ Lookup-Tabellen statt Wahrheitstabellen

IMPLIES	TRUE	FALSE
TRUE	TRUE	FALSE
FALSE	TRUE	TRUE

(A IMPLIES B) = NOT (A AND NOT B)

IMPLIES-1	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	TRUE	TRUE	**UNKNOWN**
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

= (NOT(A) OR B)

IMPLIES-2	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	TRUE	**TRUE**
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN

(nach De Morganschem Gesetz)

Historie und Hintergründe: Fazit

- **Umsetzung ternärer Logik in SQL ist lückenhaft**
- **Hersteller handhaben NULLs unterschiedlich**
- **Semantik von NULL liegt daher im Auge des Betrachters**
  - Ist es ein Default-Wert? (APEX!)
  - Ist der Wert momentan unbekannt?
  - Ist das Attribut hier nicht anwendbar?
  - …
- **Genau dies führt immer wieder zur Verwirrung**
  - Daher: Entweder vermeiden, z.B. durch Normalisierung/Dekomposition
  - Oder: Klar dokumentieren, welche Bedeutung NULLs im DB-Modell haben.

2 Umgang mit NULLs

Toon Koppelaars
@ToonKoppelaars

Folgen

This is a first for me. Looking at following join-condition:
WHERE ((p.code = c.code) or (p.code is null and c.code is null))
WTF?

Toon Koppelaars @ToonKoppelaars 26 Sep
@ReneJanV Might be. But the NULL's get cartesianal join. The others get normal join. Probably a bug me thinks. And ofcourse awfull db-design
Details

Rene Jan Veldwijk @ReneJanV
@ToonKoppelaars Cartesian product have 2 unique indexes on 2 composite other columns are mandatory. #oracle
Details

Jonathan Lewis @JLOracle
@ToonKoppelaars Toon, I know that n anathema - but Oracle even invented type of thing: jonathanlewis.wordpress ...
Details

AllYourDatabase @AllYourDataBase 26 Sep
@Mo_Ltd @MladenPrajdic @ToonKoppelaars @DBAKevlar if p.code was null and c.code was 0, ifnull(p.code,0) = ifnull(c.code,0)...bad result
Details

Mo van Praag @Mo_Ltd 26 Sep
@AllYourDataBase @MladenPrajdic @ToonKoppelaars @DBAKevlar choose a unique value for ifnull, ie one that doesnt exist in ur data
Details

AllYourDatabase @AllYourDataBase 26 Sep
@Mo_Ltd @MladenPrajdic @ToonKoppelaars @DBAKevlar and a frustrating future bug is born :)
Details

Umgang mit NULLs: Wo erzeugt SQL NULLs?

```
> -- NULL kann in SQL einen Datentyp haben:
> select CAST (NULL AS NUMBER) from dual
CAST(NULLASNUMBER)
------------------

> -- Rechenoperationen mit NULL ergeben stets NULL:
> select 0 / NULL from dual
    0/NULL
----------

> -- Kein Fehler "Division by zero":
> select NULL / 0 from dual
    NULL/0
----------
```

Umgang mit NULLs: Wo erzeugt SQL NULLs? (2)

```
create table S ( SNO integer, STADT varchar2(30) );
create table T ( TNO integer, STADT varchar2(30) );
insert into S values( 1, 'Nürnberg' );
insert into T values( 1, NULL );
commit;
```

S	
sno	stadt
1	Nürnberg

T	
tno	stadt
1	

Umgang mit NULLs: Wo erzeugt SQL NULLs?

```
> select SUM( sno ) from S where STADT='Nürnberg'
  SUM(SNO)
----------
         1
> -- SUM() kann NULL liefern, COUNT() nicht!
> select SUM( tno ) from T where STADT='Nürnberg'
  SUM(TNO)
----------

> select COUNT(*) from T
  COUNT(*)
----------
         1
> -- COUNT(*) != COUNT( spalte )!
> select COUNT( STADT ) from T
COUNT(STADT)
------------
           0
```

Umgang mit NULLs: Wo erzeugt SQL NULLs?

```
> -- Leerstrings entsprechen NULL in Oracle:
> select CASE WHEN '' IS NULL then 'NULL' ELSE 'Leerstring' END leer
from dual
LEER
----------
NULL

> -- Können durch Typkonvertierung wieder String werden:
> select NULL || 0 from dual
NULL||0
-------
0
```

Umgang mit NULLs: Wo erzeugt SQL NULLs?

```
> -- Fehlender ELSE-Zweig kann in NULL resultieren:
> select CASE WHEN S.STADT='Fürth' THEN 1 END AS a from S
         A
----------

> -- Auch wenn Spalte NULL ist, liefert CASE eine Zeile:
> select CASE WHEN T.STADT='Nürnberg' THEN 1 END AS a from T
         A
----------

> -- ELSE-Zweig vermeidet NULL-Ergebnis. Ist das aber valide?
> select CASE WHEN T.STADT='Nürnberg' THEN 1 ELSE 0 END AS a from T
         A
----------
         0
```

Umgang mit NULLs: NVL, NVL2, COALESCE, DECODE, CASE

- **NVL() und NVL2()**
  - Oracle-spezifische Funktionen
  - Vermutlich immer noch die beliebteste Methode zum Vergleichen und Ersetzen von NULLs
- **Nachteile:**
  - Kann nur genau einen Ausdruck mit einem anderen ersetzen; muss ansonsten kaskadiert werden
  - Der zweite Ausdruck wird immer ausgewertet, auch dann, wenn der erste Ausdruck bereits NOT NULL ist.

Umgang mit NULLs: NVL, NVL2, COALESCE, DECODE, CASE

- **COALESCE**
  - Gibt es seit Oracle 9i
  - ermöglicht die Auswertung von mehr als zwei Ausdrücken; der erste, der NOT NULL ist, wird zurückgeliefert.
  - COALESCE ist ANSI-SQL-konform
- **„Short Circuit”:**
  - Funktion wird beendet, sobald der erste Nicht-NULL-Ausdruck gefunden wurde.
  - Verhält sich also wie ein CASE … WHEN
  - Entscheidender Geschwindigkeitsvorteil!
- **DECODE**
  - Verhält sich wie CASE (short circuit)
  - Betrachtet zwei NULLs als äquivalent (ohne IS NULL)

Umgang mit NULLs: NVL, NVL2, COALESCE, DECODE, CASE

```
create or replace function waddema
return number
as
begin
  dbms_lock.sleep(5);
  return 1 ;
end;
/
> set timing on
> select NVL( 5, waddema ) from dual
NVL(5,WADDEMA)
--------------
             5

Elapsed: 00:00:05.130

> select COALESCE( 5, waddema ) from dual
COALESCE(5,WADDEMA)
-------------------
                  5

Elapsed: 00:00:00.042
```

Umgang mit NULLs: Sortierung

```
> -- NULLs werden als niedrigster Wert interpretiert:
> select owner, table_name, num_rows
  from all_tables
 order by num_rows desc
OWNER         TABLE_NAME                       NUM_ROWS
------------ ------------------------------ ----------
APEX_030200   WWV_FLOW_TEMP_TABLE
SYS           PSTUBTBL
MDSYS         SDO_WFS_LOCAL_TXNS
MDSYS         SDO_TOPO_DATA$
...
```

Umgang mit NULLs: Sortierung

```
> -- Abhilfe mit NULLS FIRST/LAST
> select owner, table_name, num_rows
  from all_tables
 order by num_rows desc NULLS LAST
```

OWNER	TABLE_NAME	NUM_ROWS
MDSYS	SDO_COORD_OP_PARAM_VALS	9964
MDSYS	SDO_CS_SRS	4476
MDSYS	SDO_COORD_REF_SYS	4476

…

Umgang mit NULLs: IN-Listen

```
SQL> SELECT count(*)
  2    FROM all_tables
  3   WHERE owner NOT IN ( USER, 'SYSTEM' );

 COUNT(*)
----------
      2106

-- Nur eine NULL in der Liste vernichtet das Ergebnis:
SQL> SELECT count(*)
  2    FROM all_tables
  3   WHERE owner NOT IN ( NULL, USER, 'SYSTEM' );

 COUNT(*)
----------
         0
```

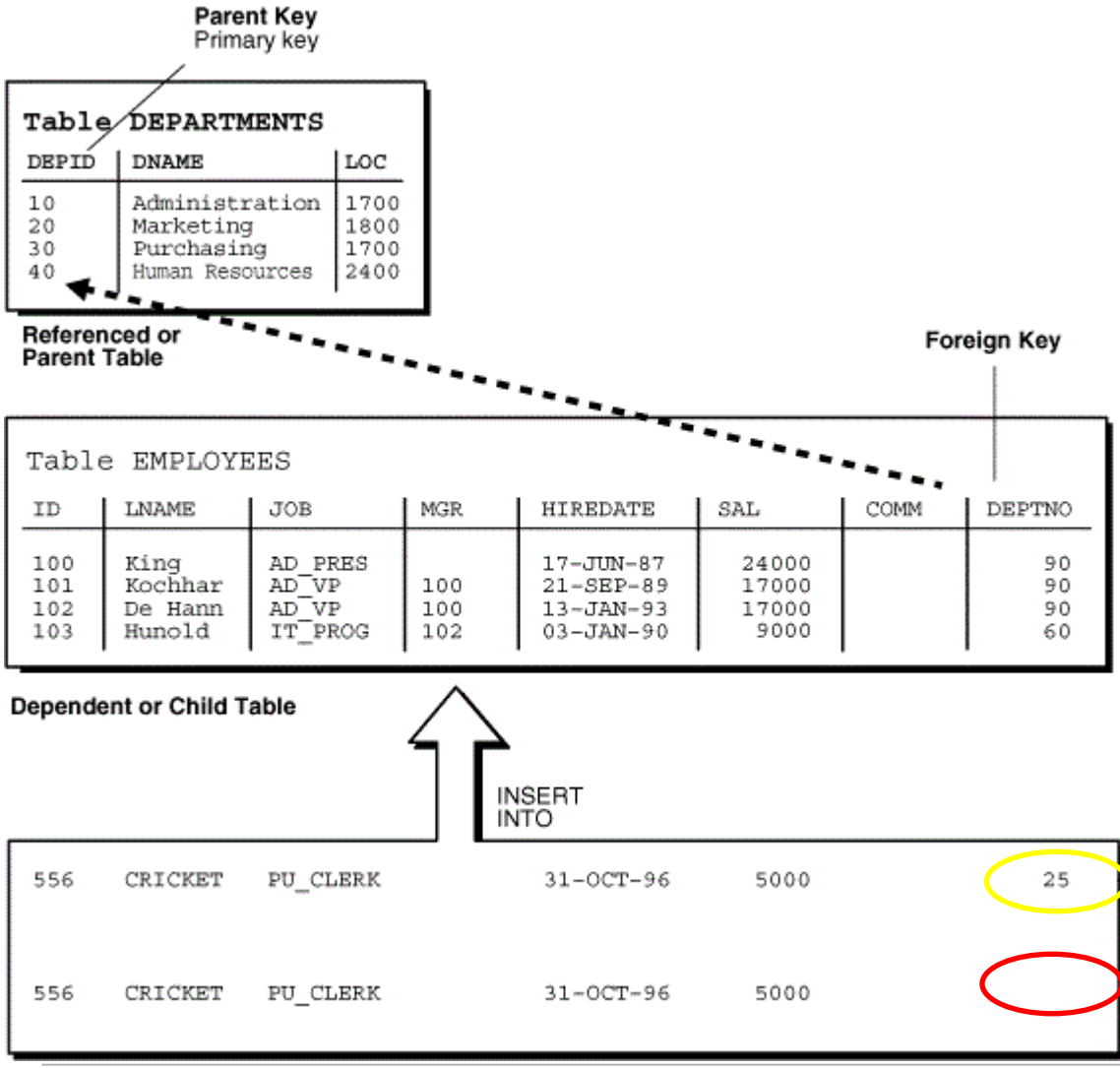
Umgang mit NULLs: Foreign Keys

- **Der ANSI/ISO SQL92-Standard kennt mehrere Matching-Regeln für Fremdschlüssel (match none/partial/full)**
  - Full: FKs dürfen *keine* NULL enthalten, auch nicht in einzelnen Spalten
  - Partial: FKs dürfen in *allen* Spalten NULL enthalten; oder in Teilen, wobei es dann eine Entsprechung in der Elterntabelle geben muss.
  - None: FKs dürfen *in Teilen* NULL enthalten; es muss *keine Entsprechung* in der Elterntabelle geben.
- **Oracle-Standard ist die Regel MATCH NONE**
- **➔ Kind-Einträge ohne Eltern-Eintrag möglich!**
  - MATCH FULL kann mit Constraints nachgebaut werden
  - MATCH PARTIAL erfordert dagegen Trigger

```
-- Bsp. für FK mit Spalten a, b, c:
CHECK ((a IS NULL AND b IS NULL AND c IS NULL) OR
       (a IS NOT NULL AND b IS NOT NULL AND c IS NOT NULL))
```

Umgang mit NULLs: Foreign Keys

Quelle: Oracle-Doku 11.2



3 NULL und Performance

NULL und Performance: Use The Constraint, Luke!

```
CREATE TABLE demo
(
  id    NUMBER NOT NULL
, name  VARCHAR2(50)
, typ   VARCHAR2(1));

INSERT INTO demo VALUES( 1, 'Asterix', 'a' );
INSERT INTO demo VALUES( 2, 'Oraculix', 'b' );

SET AUTOTRACE TRACEONLY

SELECT count(*) FROM demo WHERE name IS NULL;

Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
          7  consistent gets
          0  physical reads
```

NULL und Performance: Use The Constraint, Luke!

- **Die vorausgegangene Abfrage auf eine Spalte ohne NOT NULL-Constraint führte zu einem Table Scan, also logischem I/O (LIO).**

- **Fragen wir nun eine Spalte mit Constraint ab:**

NULL und Performance: Use The Constraint, Luke!

```
SELECT count(*) FROM demo WHERE id IS NULL;
-----------------------------------------------------
| Id  | Operation           | Name | Rows  | Bytes |
-----------------------------------------------------
|   0 | SELECT STATEMENT    |      |     1 |    13 |
|   1 |  SORT AGGREGATE     |      |     1 |    13 |
|*  2 |   FILTER            |      |       |       |
|   3 |    TABLE ACCESS FULL| DEMO |     2 |    26 |
-----------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter(NULL IS NOT NULL)

Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
          0  consistent gets
          0  physical reads
```

NULL und Performance: Use The Constraint, Luke!

- **Die Abfrage wurde nun ohne LIO durchgeführt.**
- **Die etwas eigenartige Filteroperation “NULL IS NOT NULL” zeigt, daß der Optimizer die Möglichkeit zur Vereinfachung der Abfrage wahrgenommen hat.**

- **In Kombination mit Fremdschlüsseln (s.o.)**
  - Werden Schlüsselfelder zu Pflichtfeldern erklärt
  - Ist sichergestellt, dass es eine – und nur eine – Entsprechung in der Eltern-Tabelle gibt
  - Können Joins zur Elterntabelle weggelassen werden, wenn mit NULLs in Schlüsselspalten gesucht wird („Table Elimination“)

NULL und Performance: Use The Index, Luke!

- **NULLs in letzten Spalten eines Indizes werden nicht gespeichert.**
  - Index über nur eine Spalte: kein Index-Eintrag, wenn NULL
- **NULLs können genutzt werden, um schnelle Zugriffe über Indizes zu erreichen**
  - Bsp. Status-Spalte: Wenige Werte für „eingegangen“, sonst NULL
  - Schneller Zugriff auf „eingegangen“ über Index
- **NULLs können genutzt werden, um selektive Zugriffe über Indizes zu erreichen**
  - Index auf virtuelle Spalte mit Filterung
  - Function-Based Index mit selbst definierter Filter-Funktion

NULL und Performance: Use The Index, Luke!

```
alter table emp add( topverdiener AS
  (CASE WHEN sal >= 3000 THEN sal ELSE NULL END) );
create index topverdiener_ix on emp( topverdiener );
exec dbms_stats.gather_table_stats( 'scott', 'emp' )

EXPLAIN PLAN FOR
select * from emp
 where topverdiener IS NOT NULL;
select * from table( dbms_xplan.display( format => 'BASIC') );

-----------------------------------------------------
| Id  | Operation                    | Name           |
-----------------------------------------------------
|   0 | SELECT STATEMENT             |                |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP             |
|   2 |   INDEX FULL SCAN            | TOPVERDIENER_IX |
-----------------------------------------------------
```

4 Mythen und Missverständnisse

SQL-Operationen mit einer NULL als Operand ergeben immer NULL!

```
select decode(null,null,1,2) from dual;
```

Aggregatfunktionen, CONCAT, count(), …

sys_op_map_nonnull

NULLs werden NIE in Indizes gespeichert!

```
create index MGR_TO_IX on EMP ( MGR, EMPNO );
-- Spalte MGR darf NULL sein. Verhalten?
```

Wenn ich NVL/COALESCE/… benutze, wird kein Index mehr verwendet!

Das kann widerlegt werden:

Mythen und Missverständnisse: NVL, DECODE, COALESCE und Indizes

```
var m number
exec :m := 7566
EXPLAIN PLAN FOR
select * from emp where mgr = NVL( :m, 0 );
--select * from emp where mgr = COALESCE( :m, 0 );
--select * from emp where mgr = DECODE( :m, NULL, 0, :m );
select * from table( dbms_xplan.display() );

-------------------------------------------------------------------------------------
| Id  | Operation                    | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |            |     2 |    76 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP        |     2 |    76 |     2   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN           | EMP_MGR_IX |     2 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("MGR"=NVL(:M,0))
```

Fazit

- **Idealerweise werden NULLs vermieden**
  - NOT NULL in der Spaltendeklaration verwenden
  - Vertikale + horizontale Dekomposition
  - Dennoch auf Fälle achten, in denen SQL NULLs liefern kann:
    - Outer Joins
    - Aggregate auf leere Mengen
    - CASE ohne ELSE
    - Nicht initialisierte Binds
    - …
- **Wenn NULLs geplant eingesetzt werden**
  - Einheitliche Betrachtungsweise festlegen und dokumentieren (unbekannter, nicht zutreffender oder Standard-Wert)
  - Fremdschlüssel-Spalten möglichst NOT NULL setzen
  - COALESCE und CASE anstelle NVL und DECODE verwenden

Literatur

- **Geschichte und Hintergründe**
  - Wikipedia: http://en.wikipedia.org/wiki/Null_(SQL)
  - Techopedia zu Codd's Law: http://www.techopedia.com/definition/1170/codds-rules
  - Wikipedia zu E.F. Codd: http://en.wikipedia.org/wiki/Edgar_F._Codd
  - Codd, E.F., The Relational Model for Database Management (Version 2 ed.). Addison Wesley 1990. ISBN 0-201-14192-2.
  - Chris J. Date, Hugh Darwen: The Third Manifesto. http://www.thethirdmanifesto.com/
  - C. J. Date: SQL and Relational Theory; O'Reilly 2009, ISBN-13: 978-1449316402
  - Aristoteles' Seeschlacht-Beispiel: http://de.wikipedia.org/wiki/De_interpretatione#Kapitel_9:_Aussagen_.C3.BCber_die_Zukunft_.289._Kapitel.29
  - https://www.simple-talk.com/sql/learn-sql-server/sql-and-the-snare-of-three-valued-logic/
  - Fabian Pascal: Null Confusion. http://www.dbdebunk.com/2012/07/null-confusion.html
  - https://de.wikipedia.org/wiki/Closed_world_assumption
- **Umgang mit NULLs**
  - C.J. Date: SQL and Relational Theory, O'Reilly 2012, ISBN-13: 978-1449316402
  - NULL oder NOT NULL: Das ist hier die Frage! http://www.oracle.com/webfolder/technetwork/de/community/apex/tipps/sqlnull/index.html
  - http://oraculix.wordpress.com/2012/02/28/nvl-st-du-noch-oder-coalesce-t-du-schon/
  - http://asktom.oracle.com/pls/apex/f?p=100:11:0::::P11_QUESTION_ID:524526200346472289
  - http://hoopercharles.wordpress.com/2012/02/28/repeat-after-me-null-values-are-not-stored-in-indexes/
  - http://www.oracledba.co.uk/tips/sys_op_map_nonnull.htm
  - ANSI/ISO SQL92-Standard: http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt
  - Foreign Keys: http://docs.oracle.com/cd/E11882_01/appdev.112/e25518/adfns_constraints.htm#ADFNS273
- **NULLs und Performance**
  - http://richardfoote.wordpress.com/2008/01/30/how-are-nulls-actually-indexed-fascination/
  - http://hoopercharles.wordpress.com/2012/02/28/repeat-after-me-null-values-are-not-stored-in-indexes/
  - http://structureddata.org/2008/05/22/null-aware-anti-join/