Oracle Indexing Primer

Data Warehousing and Data Mining

Patrick Schäfer

Berlin, 18. Dezember 2017

patrick.schaefer@hu-berlin.de

Vorlesung: https://hu.berlin/vl_dwhdm17
Übung: https://hu.berlin/ue_dwhdm17

Agenda

- Indexerstellung mit ORACLE
  - Entscheiden ob und welcher Index sinnvoll für eine gegebene Query ist
  - Ausführungsplan erstellen und lesen
  - CREATE INDEX Statement in Oracle schreiben
- Subquery Factoring
- Hierarchische Anfragen

„The most important thing to remember is that for the database to use an index, the column(s) must be in your query! So if a column never appears in your queries, it's not worth indexing.“

https://blogs.oracle.com/sql/how-to-create-and-use-indexes-in-oracle-database

Vorab

- Welche Anfragen gibt es?
- Welche Spalten werden abgefragt in
  - WHERE
  - ORDER BY
  - GROUP BY
  - JOINS
  - Funktionen (Aggregate)
- Welche Spalten werden zusammen abgefragt
- Wie oft werden Updates/Deletes/Inserts vorgenommen
  - Indexe müssen bei jedem Update aktualisiert werden!

https://www.toptal.com/sql/sql-database-tuning-for-developers

Use-Case Mitarbeiterverzeichnis

- Tabelle mit Angestellten des Instituts

```
People (MAIL, TITLE, GIVEN_NAME, LAST_NAME,
        DEPARTMENT,ROOM,PHONE,WEBSITE)
```

	MAIL	TITLE	GIVEN_NAME	LAST_NAME	DEPARTMENT	ROOM	PHONE	WEBSITE
1	dagmar.woerister@geo.hu-berlin.de	(null)	Dagmar	Wörister	Geomatik	Raum 2.227	(030)2093-6905	http://www.geographie.
2	sylvia.zinke-friedrich@geo.hu-berlin.de	(null)	Sylvia	Zinke-Friedrich	Klimageographie	Raum 1.223	(030)2093-6807	http://www.geographie.
3	anja.koerle@geo.hu-berlin.de	(null)	Anja	Körle	Physisch-Geographisches Labor	Raum 0.203	(030)2093-6838	(null)
4	olof.krueger@hu-berlin.de	(null)	Olof	Krüger	(null)	(null)	(null)	http://www.iri-thesys.
5	karoline.kucharzyk@geo.hu-berlin.de	M. Ed.	Karoline	Kucharzyk	(null)	Raum 2.208	(030)2093-6882	http://www.geographie.
6	elmar.kulke@geo.hu-berlin.de	Prof. Dr.	Elmar	Kulke	Wirtschaftsgeographie	Raum 5.101	(030)2093-6814	http://www.geographie.
7	tobias.kuemmerle@hu-berlin.de	Prof. Dr. rer. nat.	Tobias	Kümmerle	Biogeographie	Raum 2.106	(030)2093-9372	http://www.geographie.
8	tobia.lakes@geo.hu-berlin.de	Prof. Dr.	Tobia	Lakes	Angewandte Geoinformationsverarbeitung	Raum 2.220	(030)2093-6873	http://www.geographie.
9	nlanfer@t-online.de	Prof. Dr.	Norbert	Lanfer	Klimatologie und Vegetationsgeographie	Raum 1.215	(030)2093-9437	(null)
10	moritz.langer@geo.hu-berlin.de	(null)	Moritz	Langer	Klimageographie	Raum 1.220	(030)2093-6880	(null)
11	jan.lentschke@geo.hu-berlin.de	Dr. rer. nat.	Jan	Lentschke	Geomorphologie / Bodengeographie / Quart...	Raum 1.209	(030)2093-6839	http://www.geographie.
12	barbara.lenz@geo.hu-berlin.de	Prof. Dr. rer. nat.	Barbara	Lenz	Verkehrsgeographie (S)	Raum 5.110	(030)2093-6815	http://www.geographie.
13	barbara.lenz@dlr.de	Prof. Dr.	Barbara	Lenz	(null)	Raum 2.107	(030)2093-6803	(null)
14	christian.levers@geo.hu-berlin.de	Dr.	Christian	Levers	Biogeographie	Raum 2.101	(030)2093-9341	http://www.geographie.
15	lewe@igb-berlin.de	PD Dr. Dipl.-Ing.	Jörg	Lewandowski	Klimageographie	(null)	(030)64181-668	http://www.igb-berlin.
16	sebastian.linden@hu-berlin.de	Dr.	Sebastian	Linden	(null)	(null)	(null)	http://www.iri-thesys.
17	david.loibl@geo.hu-berlin.de	Dr.	David	Loibl	Klimageographie	Raum 1.216	(030)2093-6824	(null)
18	wolfgang.lucht@geo.hu-berlin.de	Prof. Dr.	Wolfgang	Lucht	Alexander von Humboldt Chair in Sustaina...	Raum 1.215	(030)2093-9454	http://www.geographie.
19	macchile@hu-berlin.de	Dr.	Leandro	Macchi	Biogeographie	Raum 2.105	(030)2093-5394	(null)
20	ulrike.mackrodt@geo.hu-berlin.de	Dipl.-Geogr.	Ulrike	Mackrodt	Kultur- und Sozialgeographie	Raum 3.107	(030)2093-6842	http://www.geographie.
21	erasmus.geographie@geo.hu-berlin.de	PD Dr. rer. nat.	Mohsen	Makki	Geomorphologie / Bodengeographie / Quart...	Raum 0.202	(030)2093-6895	http://www.geographie.

Beispielanfrage

- Suche einen bestimmten Eintrag:

```
SELECT m.* FROM People m WHERE m.LAST_NAME = 'Leser';
```

- Falls kein Index existiert, müssen alle Einträge (ROWs) der Tabelle durchsucht werden:

```
for each row P in People do
    if P.LAST_NAME = ‘LESER’
            then print P;
    end if;
end for
```

Abschätzung der Laufzeit

- Durchschnittliche Längen der Einträge der Tabelle in Bytes:
  `mail 30, title 9, given_name 7,  last_name 7, department 30, room 10, phone 14, website 50`

- Insgesamt ~165Byte (+Header):
  - 157 Byte für Daten und 8 Byte für Längen

- Bei 1 Millionen Einträgen:
  - $10^6 * \frac{165}{1024*1024} Byte = 157\ MB$
  - Bei 100 MB/s brauchen wir ~ 1,5 Sekunden für einen sequentiellen Scan.

Index vs. Full-Table Scan

- Trade-off von Index-Strukturen:
  - Anfragen werden meist schneller, aber Löschen und Einfügen wird langsamer
  - Wenn (zu) viele Index-Strukturen existieren, benötigt der Query-Optimizer mehr Zeit den Ausführungsplan zu bestimmen
- Selektivität:
  Ein Index ist sinnvoll, falls nur ein kleiner Teil der Einträge (Rows) selektiert wird
- Typischerweise sind das <4% der Einträge
- Andernfalls ist ein full-table scan meistens günstiger
  - Für sehr kleine Tabellen ist ein full-table scan fast immer besser

Ausführungsplan (EXPLAIN PLAN)

- Zeigt die Schritte zur Ausführung einer SQL-Query
- Reihenfolge und Art der Operatoren wird vom Optimizer bestimmt
- Wird typischerweise in Tabellenform dargestellt
-
```
EXPLAIN PLAN for
  SELECT m.* FROM People m WHERE m.LAST_NAME = 'Leser';

SELECT PLAN_TABLE_OUTPUT FROM TABLE(DBMS_XPLAN.DISPLAY());
```

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |        |     1 |   144 |     7   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| PEOPLE |     1 |   144 |     7   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

- Cost: Internes Maß (Abschätzung) für CPU + Speicher + I/O
- Rows: Abschätzung für Ergebnisgröße
- Wie sieht ein guter Plan aus? Möglichst niedrige Kosten

Index Erstellung

- Index wird automatisch für Primärschlüssel (PRIMARY KEY) angelegt
- Anderer Index muss selbst angelegt werden:

```
CREATE INDEX idx_last_name ON People(LAST_NAME)
```

- Transparent für den Anwender: Der Query-Optimizer entscheidet eigenständig, ob er den Index nutzt

```
EXPLAIN PLAN for
  SELECT m.* FROM People m WHERE m.LAST_NAME = 'Leser';
SELECT PLAN_TABLE_OUTPUT FROM TABLE(DBMS_XPLAN.DISPLAY());
```

```
|-------------------------------------------------------------------------------------------
| Id  | Operation                             | Name          | Rows  | Bytes | Cost (%CPU)| Time
|-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                      |               |     1 |   144 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| PEOPLE        |     1 |   144 |     2   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                    | IDX_LAST_NAME |     1 |       |     1   (0)| 00:00:01 |
|-------------------------------------------------------------------------------------------
```

Punktanfrage (WHERE-Klausel)

- Punktanfragen mit hoher Selektivität werden schneller, wenn ein Index verwendet werden kann
- Indexstruktur speichert inverses Mapping von indexierten Werten (Spalten) auf die Einträge (Zeilen)

- Beispiele:

```
SELECT m.* FROM People m WHERE m.LAST_NAME = 'Leser';

SELECT m.* FROM People m
 WHERE m.GIVEN_NAME = 'Patrick' AND m.ROOM = 'Raum 4.404';
```

Punktanfrage (WHERE-Klausel)

- Es können mehrere Indexe für die gleiche Tabelle angelegt werden

```
SELECT m.* FROM People m
 WHERE m.GIVEN_NAME = 'Patrick' AND m.ROOM = 'Raum 4.404';
```

- Index für GIVEN_NAME und für ROOM anlegen

```
CREATE INDEX idx_given_name ON People(GIVEN_NAME)
CREATE INDEX idx_room ON People(ROOM)
```

- Der Optimizer entscheidet, wie die Anfrage ausgeführt wird
  - Nur einen Index oder beide (Schnittmenge der ROWs) nutzen?

```
-------------------------------------------------------------------------------------------
| Id  | Operation                           | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |                |     1 |   144 |     3   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS BY INDEX ROWID BATCHED| PEOPLE         |     1 |   144 |     3   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                  | IDX_GIVEN_NAME |     2 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
```

Zusammengesetzter Index

- Es kann ein Index über mehrere Attribute angelegt werden

```
SELECT m.* FROM People m
 WHERE m.GIVEN_NAME = 'Patrick' AND m.ROOM = 'Raum 4.404';
```

- Index auf GIVEN_NAME und auf ROOM anlegen

```
CREATE INDEX idx_given_name_room ON People (GIVEN_NAME, ROOM);
```

- Eine Anfrage auf beiden Attributen wird somit schneller beantwortet

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                           | Name                 | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |                      |     1 |   144 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| PEOPLE               |     1 |   144 |     2   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                  | IDX_GIVEN_NAME_ROOM  |     1 |       |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------
```

Zusammengesetzter Index (oversized)

- Ein Index für mehrere Attribute (A,B) kann auch für Anfragen auf A verwendet werden (...aber nicht, wenn nur B angegeben wird)

```
SELECT m.* FROM People m
 WHERE m.GIVEN_NAME = 'Patrick';
```

```
-------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name                | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |                     |     2 |   288 |     4   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| PEOPLE              |     2 |   288 |     4   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                  | IDX_GIVEN_NAME_ROOM |     2 |       |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------
```

- Aber: Ein Index nur auf A hat eine geringere Höhe und ist damit schneller

Bereichsanfragen (WHERE-Klausel)

- Ein Index kann auch für Bereichsanfragen genutzt werden:
  `SELECT m.* FROM People m WHERE m.LAST_NAME LIKE 'P%';`
- Intern kann das zB umgeschrieben werden zu:
  `SELECT m.* FROM People m WHERE m.LAST_NAME>='P' AND m.LAST_NAME<'Q';`

```
-------------------------------------------------------------------------------------------
| Id  | Operation                           | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
|-----------------------------------------------------------------------------------------|
|   0 | SELECT STATEMENT                    |               |     4 |   576 |     5   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| PEOPLE        |     4 |   576 |     5   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                  | IDX_LAST_NAME |     4 |       |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
```

- Funktioniert das auch mit dem Suffix?
  `SELECT m.* FROM PEOPLE m WHERE m.LAST_NAME LIKE ‘%r';`

Joinanfragen

- Ein Index kann für Joins genutzt werden:

```
SELECT p.given_name, p.last_name, p.mail, p.phone
  FROM People p, People p2
 WHERE p.PHONE = p2.PHONE AND p.MAIL != p2.MAIL;
```

- Index auf GIVEN_NAME und auf MAIL anlegen

```
CREATE INDEX idx_phone_mail on People(phone, mail);
```

```
-------------------------------------------------------------------------------
| Id  | Operation             | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |                |   332 | 36188 |    10   (0)| 00:00:01 |
|*  1 |  HASH JOIN            |                |   332 | 36188 |    10   (0)| 00:00:01 |
|   2 |   INDEX FAST FULL SCAN| IDX_PHONE_MAIL |   438 | 20148 |     3   (0)| 00:00:01 |
|   3 |   TABLE ACCESS FULL   | PEOPLE         |   438 | 27594 |     7   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

p.PHONE=p2.PHONE
p.MAIL!=p2.MAIL

Index Only Anfragen

- Anfragen die nur Attribute des Index nutzen, können effizient ausgeführt werden, da sie nicht auf die Daten zugreifen müssen

```
SELECT count(DISTINCT room) AS anzahl FROM People
 WHERE room is NOT NULL ORDER BY room;
```

- Index auf ROOM anlegen

```
CREATE INDEX idx_room on People(room);
```

```
-------------------------------------------------------------------------------
| Id  | Operation             | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |          |     1 |    11 |     2   (0)| 00:00:01 |
|   1 |  SORT GROUP BY        |          |     1 |    11 |            |          |
|*  2 |   INDEX FAST FULL SCAN| IDX_ROOM |   417 |  4587 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

Index auf Null-Werte

- Standardmäßig werden NULL-Werte in ORACLE im B-Baum nicht indexiert
  - Dadurch wird der Index deutlich kleiner
- Das kann die Nutzung eines Index verhindern

```
CREATE INDEX idx_room ON People(ROOM);
```

a)
```
SELECT room FROM People WHERE room IS NOT NULL;
```
```
-------------------------------------------------------------------------------
|*  1 |  INDEX FAST FULL SCAN| IDX_ROOM |   417 |  4587 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

b)
```
SELECT room FROM people WHERE room IS NULL;
```
```
---------------------------------------------------------------------------
|*  1 |  TABLE ACCESS FULL   | PEOPLE |    21 |   231 |     7   (0)| 00:00:01 |
---------------------------------------------------------------------------
```

- Aber seit Oracle 11g ist es möglich, NULL Werte zu indexieren:

```
CREATE INDEX idx_room ON People(ROOM, 1);
```
```
-------------------------------------------------------------------------------
|*  1 |  INDEX RANGE SCAN| IDX_ROOM_NULL |    21 |   231 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

Oracle CREATE INDEX

- Oracle unterstützt die folgenden Index-Typen:
  - *Normaler* Index (B-Baum)
    - Standard, O(log n)
    - NULL-Werte werden nicht indexiert
    - Gut für Primary Key, sehr selektive und zusammengesetzte Indexe
  - Bitmap Index
    - Sinnvoll, wenn geringe Kardinalität (Anzahl unterschiedlicher Werte) vorhanden ist
    - komprimiert/kleiner
    - NULL-Werte werden immer indexiert
  - Function-based (Berechneter) Index
    - Sinnvoll, wenn Aggregate in der Anfrage genutzt werden
  - (Hash-Partitioned Index)
    - Vorsortierung/Filterung der Partitionen für schnelleren I/O
  - (Bitmapped Join Index)
    - Materialisierung eines Joins
  - (Compressed Index)

B-Baum Index

- `CREATE INDEX index_name ON table_name (column1, column2, ... column_n);`

- Beispiele:

```
CREATE INDEX idx_lastname
   ON People (last_name);
CREATE INDEX idx_pname
   ON People (last_name, given_name);

SELECT last_name, first_name
   FROM People
  WHERE last_name=‚Leser‘ AND given_name=‚Ulf‘;
```

Bitmap Index

- `CREATE BITMAP INDEX index_name ON table_name (column1, column2, ... column_n);`

- Beispiele:

```
CREATE BITMAP INDEX idx_lastname
   ON People (Bundesland);

SELECT * FROM People WHERE bundesland='Berlin'
```

Function-Based (Berechneter) Index

- ```
  CREATE INDEX index_name ON
      table_name (function1, function2, ... function_n)
  ```

- Beispiel:

```
CREATE INDEX idx_lname ON People (UPPER(last_name));

SELECT UPPER(last_name), first_name
   FROM People WHERE UPPER(last_name) IS NOT NULL
   ORDER BY UPPER(last_name);
```
```

Hash-Partitioned Global Index

- ```
  CREATE INDEX index_name ON table_name (column1)
      GLOBAL PARTITION BY HASH (column1) PARTITIONS number;
  ```

- Beispiele:

  ```
  CREATE INDEX cust_last_name_ix ON People(last_name)
     GLOBAL PARTITION BY HASH (last_name) PARTITIONS 4;
  ```
  ```

Index löschen

- Syntax:
  ```
  DROP INDEX index_name;
  ```

- Mögliches Vorgehen:
  - Index anlegen
  - Prüfen, ob er genutzt wird und die Kosten reduziert
  - Sonst Löschen

- Alle Indexe anzeigen:
  ```
  SELECT * FROM USER_INDEXES;
  ```

Subquery Factoring 1/2

- Angenommen wir haben eine verschachtelte Query:

```
SELECT ...
    FROM (SELECT ...) subquery,
WHERE ...
```

- Diese kann strukturiert werden als

```
WITH subquery AS (
    SELECT ...
)
SELECT ... FROM subquery sq ...
```

- Komplizierte Anfragen können in mehrere SQL-Statements zerlegt werden
- subquery kann dann mehrfach referenziert/verwendet werden

Subquery Factoring 2/2

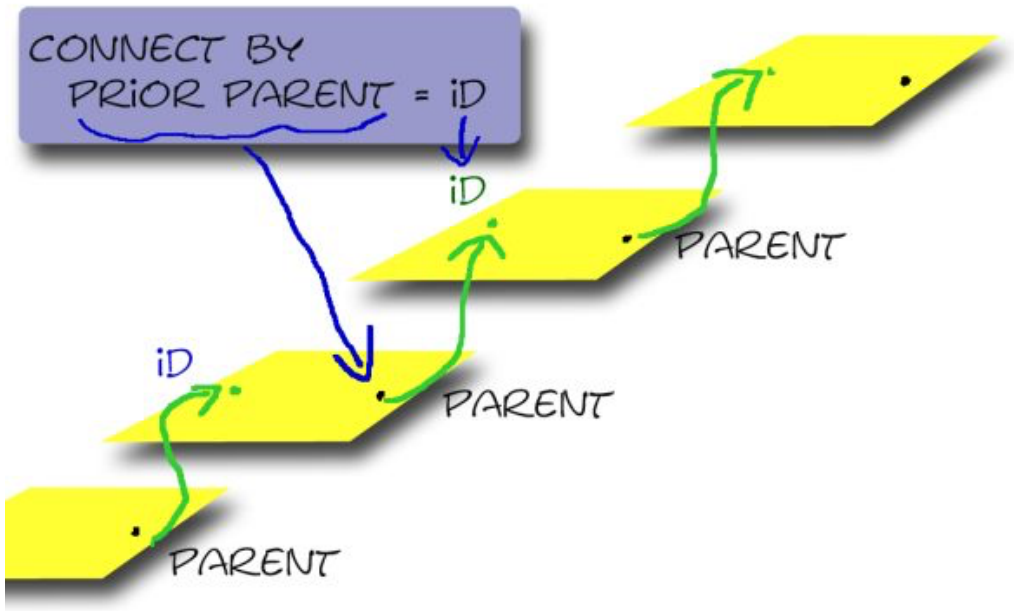
- Oder eine verschachtelte Anfrage:

```
SELECT ...
  FROM (SELECT ...
     FROM (SELECT ... ) subquery sq
   )  another_subquery  sq,
WHERE ...
```

- Wird zu:

```
WITH
    subquery AS ( SELECT ... ) ,
    another_subquery AS ( SELECT ... FROM subquery)
SELECT ... FROM another_subquery sq...
```

Hierarchische Anfragen



http://kenntwas.de/2011/datenbanken/oracle/sql_plsql/oracle-connect-by-hierarchische-queries/

- Vater- / Kindrelationen.
  - id ist der Primärschlüssel der Tabelle
  - parent ist ein Fremdschlüssel auf die Vaterkomponente

- Beispiel:

```
CREATE TABLE People (
  id VARCHAR2(100) PRIMARY KEY,

  ...

  parent VARCHAR2(100),

  CONSTRAINT
   fk_vorgesetzt FOREIGN KEY (parent)
         REFERENCES People(id)
);
```

Hierarchische Anfragen (CONNECT-BY)

```
SELECT …
   FROM <tabelle>
  WHERE  ...
  START WITH <Spaltenname1 = Konstante>
  CONNECT BY PRIOR <Spaltenname1> =
<Spaltenname2>
  ORDER SIBLINGS BY <Spaltenname1>
```

- Die Hierarchie muss innerhalb der gleichen Tabelle gebildet werden (kein Join)
- PRIOR-Spalte ist die untergeordnete Spalte

SYS_CONNECT_BY_PATH

- `SYS_CONNECT_BY_PATH(column,char)`

- Liefert den Pfad, den die Rekursion durchlaufen hat
- Nur nutzbar in hierarchischen Anfragen
- Das Trennzeichen darf nicht in den Daten vorkommen
- Z.B.:

```
SELECT SYS_CONNECT_BY_PATH(last_name, '/') FROM ...
```

- Liefert:

```
/Kochhar/Kochhar/Greenberg
/Kochhar/Greenberg/Faviet
/Kochhar/Greenberg/Chen
```

Hierarchiestufe: LEVEL

- LEVEL enthält das Rekursionslevel und ist nur in Hierarchischen Anfragen gültig
- Kann für unterschiedliche Zwecke (z.B. Sortierung) verwenden werden
- Die höchste Hierarchiestufe (Wurzel) hat LEVEL = 1

```
SELECT ..., LEVEL
  FROM ...
  CONNECT BY PRIOR ...
  START WITH ...;
```

PRIOR: Top-Down, Bottom-Up

- Top-Down:
  prior id = parent

ID	PARENT	PATH	LEVEL
824	576	/55/246/576/824	4
986	744	/183/485/744/986	4
986	744	/485/744/986	3
650	246	/55/246/650	3
670	246	/55/246/670	3
781	246	/55/246/781	3
470	246	/55/246/470	3
576	246	/55/246/576	3
824	576	/246/576/824	3
644	246	/55/246/644	3
…			
576	246	/55/246/576	2

- Bottom-Up:
  id = prior parent

ID	PARENT	PATH	LEVEL
55	NULL	/824/576/246/55	4
183	NULL	/986/744/485/183	4
55	NULL	/576/246/55	3
169	NULL	/620/371/169	3
169	NULL	/609/371/169	3
169	NULL	/581/371/169	3
80	NULL	/579/278/80	3
55	NULL	/644/246/55	3
55	NULL	/470/246/55	3
158	NULL	/622/447/158	3
…			
246	55	/824/576	2

Weiterführende Links

- https://docs.oracle.com/en/database/oracle/oracle-database/12.2/sqlrf/CREATE-INDEX.html
- https://docs.oracle.com/en/database/oracle/oracle-database/12.2/tgsql/query-execution-plans.html
- https://docs.oracle.com/en/database/oracle/oracle-database/12.2/tgsql/guidelines-for-indexes-and-table-clusters.html
- https://docs.oracle.com/en/database/oracle/oracle-database/12.2/sqlrf/Hierarchical-Queries.html
- https://blogs.oracle.com/sql/how-to-create-and-use-indexes-in-oracle-database
- http://use-the-index-luke.com/sql/table-of-contents
- https://www.toptal.com/sql/sql-database-tuning-for-developers