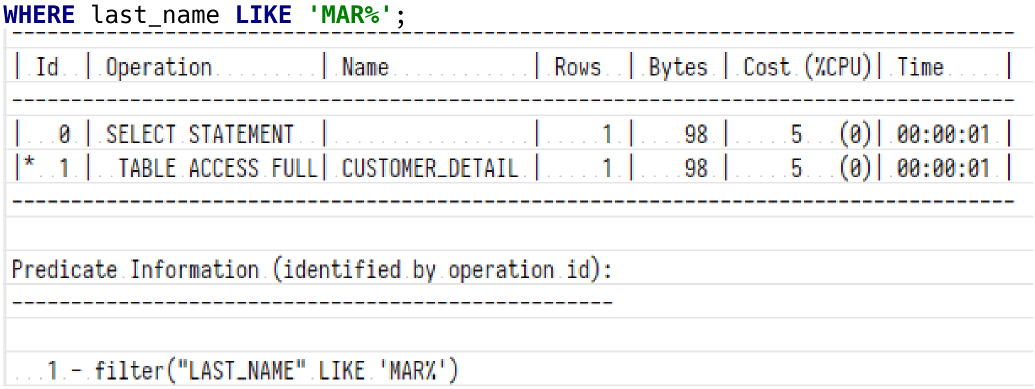
Ausarbeitung UE11

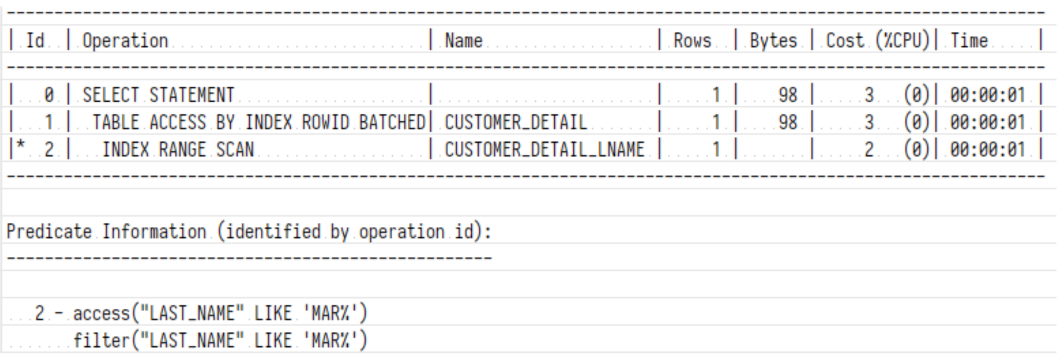
1. Indizes

```
-- 1.1
CREATE TABLE customer_detail AS
SELECT customer_id,
       first_name,
       last_name,
       email,
       phone,
       district,
       postal_code,
       city,
       country
FROM customer
       INNER JOIN address USING (address_id)
       INNER JOIN city USING (city_id)
       INNER JOIN country USING (country_id);

-- 1.2
SELECT *
FROM customer_detail
WHERE last_name LIKE 'MAR%';
```



```
-------------------------------------------------------------------------------------
| Id  | Operation         | Name            | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |                 |     1 |    98 |     5   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| CUSTOMER_DETAIL |     1 |    98 |     5   (0)| 00:00:01 |
-------------------------------------------------------------------------------------

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

   1 - filter("LAST_NAME" LIKE 'MAR%')
```

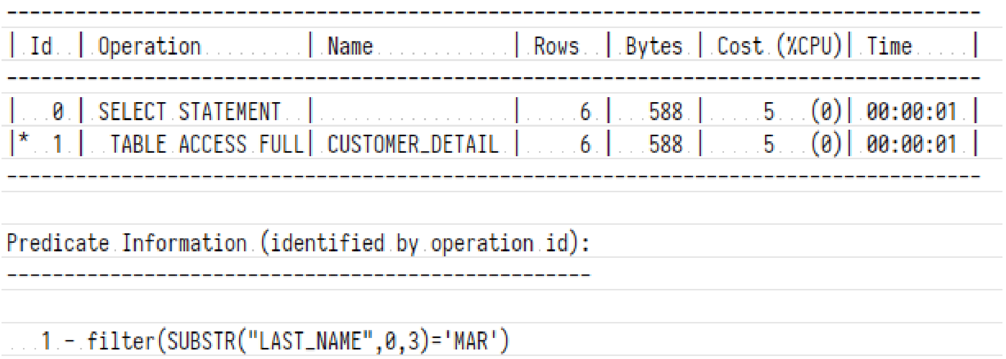
```
-- 1.3
CREATE INDEX customer_detail_lname ON customer_detail (last_name);

SELECT *
FROM customer_detail
WHERE last_name LIKE 'MAR%';
```



```
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name                  | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |                       |     1 |    98 |     3   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| CUSTOMER_DETAIL       |     1 |    98 |     3   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                  | CUSTOMER_DETAIL_LNAME |     1 |       |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------------------

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

   2 - access("LAST_NAME" LIKE 'MAR%')
       filter("LAST_NAME" LIKE 'MAR%')
```

Illustration 1: Ausführungsplan berücksichtigt erstellten Index!

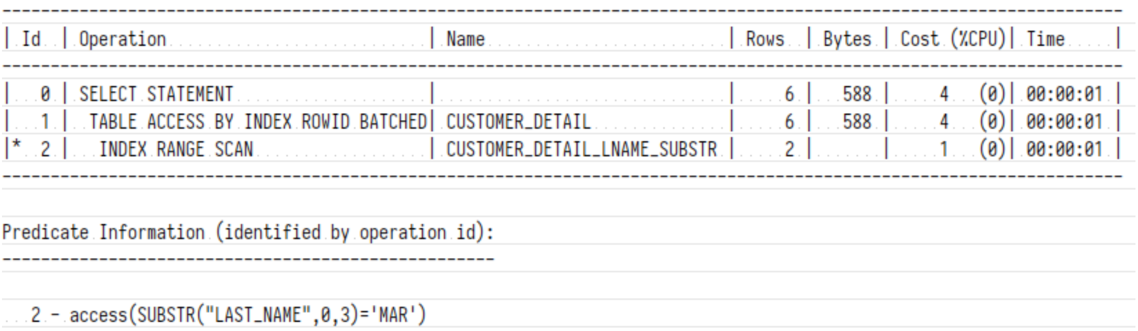
```
--1.4
SELECT *
FROM customer_detail
WHERE SUBSTR(last_name, 0, 3) = 'MAR';
```



```
--------------------------------------------------------------------------------------
| Id  | Operation         | Name            | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |                 |     6 |   588 |     5   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| CUSTOMER_DETAIL |     6 |   588 |     5   (0)| 00:00:01 |
--------------------------------------------------------------------------------------

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

   1 - filter(SUBSTR("LAST_NAME",0,3)='MAR')
```

Illustration 2: SUBSTR benötigt trotz Index einen full table access!

```
--1.5
CREATE INDEX customer_detail_lname_substr ON customer_detail (SUBSTR(last_name, 0, 3));

SELECT *
FROM customer_detail
WHERE SUBSTR(last_name, 0, 3) = 'MAR';
```

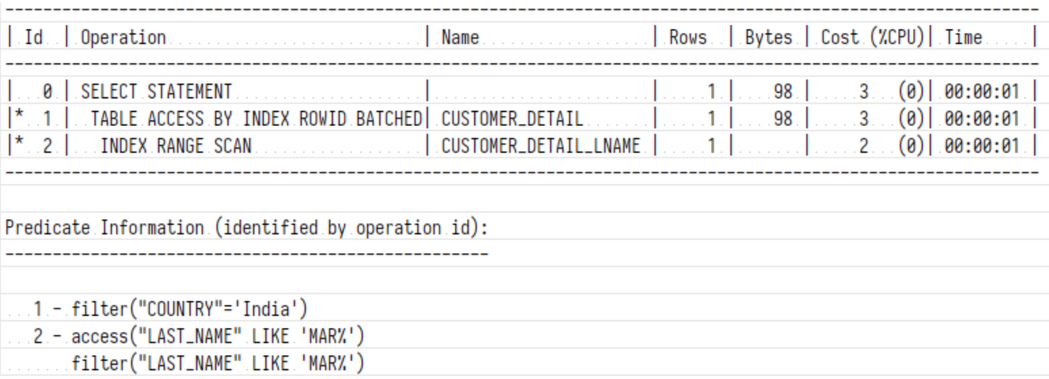


```
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name                         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |                              |     6 |   588 |     4   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| CUSTOMER_DETAIL              |     6 |   588 |     4   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                  | CUSTOMER_DETAIL_LNAME_SUBSTR |     2 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------------------------

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

   2 - access(SUBSTR("LAST_NAME",0,3)='MAR')
```

Illustration 3: Neuer function-based Index wird zur Optimierung herangezogen.

2. Optimizer Hints

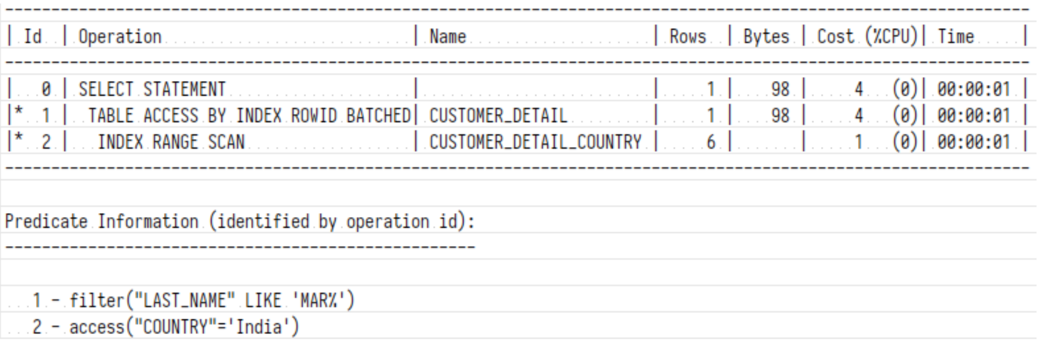
```
-- 2.1
CREATE INDEX customer_detail_country ON customer_detail (country);

SELECT *
FROM customer_detail
WHERE country = 'India'
  AND last_name LIKE 'MAR%';
```



```
-------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name                  | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |                       |     1 |    98 |     3   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS BY INDEX ROWID BATCHED| CUSTOMER_DETAIL       |     1 |    98 |     3   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                  | CUSTOMER_DETAIL_LNAME |     1 |       |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

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

   1 - filter("COUNTRY"='India')
   2 - access("LAST_NAME" LIKE 'MAR%')
       filter("LAST_NAME" LIKE 'MAR%')
```

Illustration 4: Lt. Ausführungsplan wird der neu erstellte Index nicht verwendet.

```
-- 2.2
SELECT /*+ INDEX (customer_detail customer_detail_country)*/ *
FROM customer_detail
WHERE country = 'India'
  AND last_name LIKE 'MAR%';
```



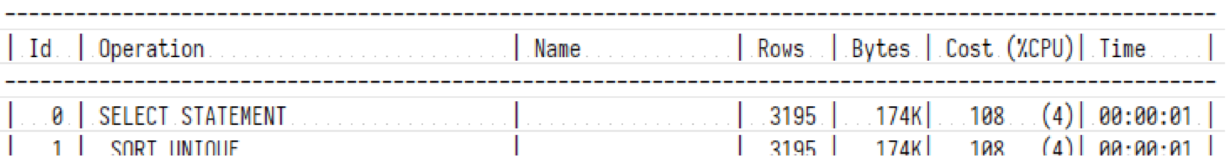
```
-------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name                    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |                         |     1 |    98 |     4   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS BY INDEX ROWID BATCHED| CUSTOMER_DETAIL         |     1 |    98 |     4   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                  | CUSTOMER_DETAIL_COUNTRY |     6 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

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

   1 - filter("LAST_NAME" LIKE 'MAR%')
   2 - access("COUNTRY"='India')
```

Illustration 5: Mit explizitem Hinweis verwendet der Query optimizer den neuen Index!

3. Optimierung von SQL-Statements

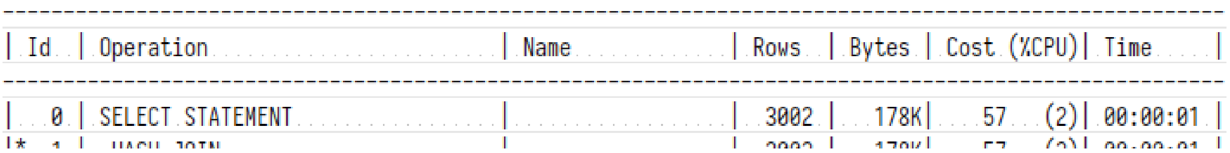
Anmerkung: Gleichheit der Ergebnistupelmengen der jeweils unoptimierten und optimierten Versionen werden nur in der anliegenden SQL Datei getestet und in diesem Dokument nicht angeführt.

```
-- 3.1
SELECT customer_id, first_name, last_name
FROM customer
       INNER JOIN rental USING (customer_id)
       INNER JOIN inventory USING (inventory_id)
       INNER JOIN film USING (film_id)
       INNER JOIN film_category USING (film_id)
       INNER JOIN category USING (category_id)
WHERE (name IN ('Comedy', 'Family', 'Children') AND length < 100)
   OR name IN ('Classics', 'Animation');
```



```
----------------------------------------------------------------------------------------
| Id  | Operation                          | Name               | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |                    |  3195 |  174K|   108   (4)| 00:00:01 |
|   1 |  SORT UNIQUE                       |                    |  3195 |  174K|   108   (4)| 00:00:01 |
```

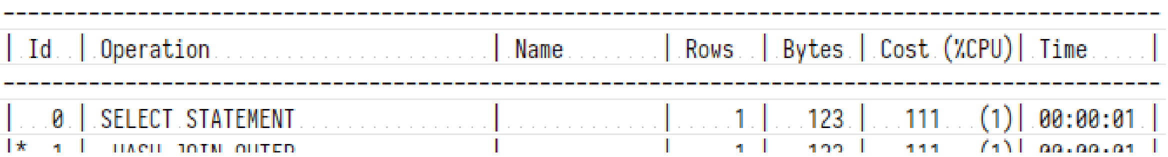
Illustration 6: Abfrage 3.1 ohne Optimierung



```
----------------------------------------------------------------------------------------
| Id  | Operation                          | Name               | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |                    |  3002 |  178K|    57   (2)| 00:00:01 |
|*  1 |  HASH JOIN                         |                    |  3002 |  178K|    57   (2)| 00:00:01 |
```

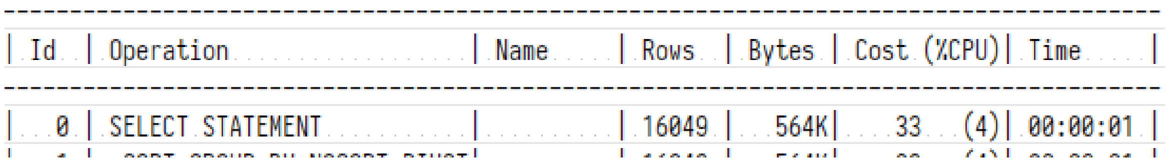
Illustration 7: Abfrage 3.1 mit Optimierung

```
-- 3.2
SELECT *
FROM (
       SELECT COALESCE(to_char(payment_date, 'yy'), 'total') AS year,
              customer_id,
              first_name,
              last_name,
              SUM(amount)                                    AS umsatz
       FROM customer
              INNER JOIN payment USING (customer_id)
       GROUP BY GROUPING SETS (
         (to_char(payment_date, 'yy'), customer_id, first_name, last_name),
         (customer_id, first_name, last_name)
         )
     )
       PIVOT (
              AVG(umsatz)
              FOR year
              IN ('13' AS umsatz13,
                '14' AS umsatz14,
                '15' AS umsatz15,
                'total' AS umsatzgesamt)
    );
```



```
--------------------------------------------------------------------------------
| Id  | Operation                        | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                 |              |     1 |   123 |   111   (1)| 00:00:01 |
|*  1 |  HASH JOIN OUTER                 |              |     1 |   123 |   111   (1)| 00:00:01 |
```

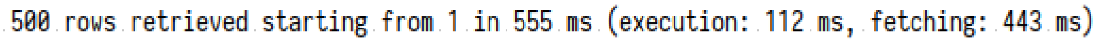
Illustration 8: Abfrage 3.2 ohne Optimierung



```
--------------------------------------------------------------------------------
| Id  | Operation                    | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |          | 16049 |  564K|    33   (4)| 00:00:01 |
|   1 |  SORT GROUP BY NOSORT PIVOT  |          | 16049 |  564K|    33   (4)| 00:00:01 |
```

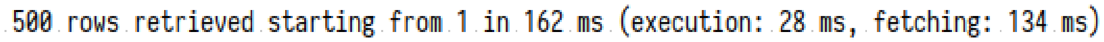
Illustration 9: Abfrage 3.2 mit Optimierung

```
-- 3.3
SELECT film_id,
       title,
       RANK() OVER (
         PARTITION BY category_id
         ORDER BY length DESC) - 1 AS longerfilmsincategory
FROM film
       INNER JOIN film_category USING (film_id)
ORDER BY film_id;
```



```
500 rows retrieved starting from 1 in 555 ms (execution: 112 ms, fetching: 443 ms)
```

Illustration 10: Abfrage 3.3 ohne Optimierung



```
500 rows retrieved starting from 1 in 162 ms (execution: 28 ms, fetching: 134 ms)
```

Illustration 11: Abfrage 3.3 mit Optimierung