

Query from HW 4 Problem 4

```
EXPLAIN ANALYZE

SELECT
  COUNT(1)

FROM
  posts
  JOIN followers ON posts.author = followers.following_handle
  JOIN users ON followers.follower_handle = users.handle

WHERE
  users.last_name = 'Anderson'
  AND users.first_name = 'Abigail';

PLAN
SELECT
  COUNT(*)
FROM "PUBLIC"."USERS"
/* PUBLIC.USERS.tableScan */
/* WHERE (USERS.LAST_NAME = 'Anderson')
   AND (USERS.FIRST_NAME = 'Abigail')
*/
/* scanCount: 10001 */
INNER JOIN "PUBLIC"."FOLLOWERS"
/* PUBLIC.PRIMARY_KEY_D: FOLLOWER_HANDLE = USERS.HANDLE */
ON 1=1
/* WHERE FOLLOWERS.FOLLOWER_HANDLE = USERS.HANDLE
*/
/* scanCount: 6 */
INNER JOIN "PUBLIC"."POSTS"
/* PUBLIC.POSTS_AUTHOR_IDX: AUTHOR = FOLLOWERS.FOLLOWING_HANDLE */
ON 1=1
/* scanCount: 210 */
WHERE (("USERS"."LAST_NAME" = 'Anderson')
 AND ("USERS"."FIRST_NAME" = 'Abigail'))
 AND ("FOLLOWERS"."FOLLOWER_HANDLE" = "USERS"."HANDLE")
 AND ("POSTS"."AUTHOR" = "FOLLOWERS"."FOLLOWING_HANDLE")
(1 row, 12 ms)
```

Given that the table sizes are:

posts: 995086, followers: 995040, users: 10000

Hence the expected join sequence will be:

Users -> Followers -> Posts

Which the EXPLAIN ANALYZE proves to be correctly implemented

HW 5 : Query 1 - Single Table

```
EXPLAIN ANALYZE

SELECT
  *

FROM
  customers;

PLAN
SELECT
  "PUBLIC"."CUSTOMERS"."CUSTOMER_ID",
  "PUBLIC"."CUSTOMERS"."NAME",
  "PUBLIC"."CUSTOMERS"."COUNTRY"
FROM "PUBLIC"."CUSTOMERS"
/* PUBLIC.CUSTOMERS.tableScan */
/* scanCount: 11 */
(1 row, 0 ms)
```

Nothing much here, simply doing a table scan through customers.

HW 5 : Query 2 - Just Two Tables

```
EXPLAIN ANALYZE

SELECT
  name,
  COUNT(order_id)

FROM
  customers
  JOIN orders ON customers.customer_id = orders.customer_id

GROUP BY
  customers.name;
```

PLAN

```
SELECT
  "NAME",
  COUNT("ORDER_ID")
FROM "PUBLIC"."CUSTOMERS"
/* PUBLIC.CUSTOMERS.tableScan */
/* scanCount: 11 */
INNER JOIN "PUBLIC"."ORDERS"
/* PUBLIC.CONSTRAINT_INDEX_8: CUSTOMER_ID = CUSTOMERS.CUSTOMER_ID */
ON 1=1
/* scanCount: 210 */
WHERE "CUSTOMERS"."CUSTOMER_ID" = "ORDERS"."CUSTOMER_ID"
GROUP BY "CUSTOMERS"."NAME"
(1 row, 2 ms)
```

Two tables joining together has no sequence issues, hence the query simply executes by order

HW 5: Query 3 - Three Tables

```
EXPLAIN ANALYZE

SELECT
  orders.order_id,
  products.name,
  order_details.quantity

FROM
  order_details
  JOIN products ON order_details.product_id = products.product_id
  JOIN orders ON order_details.order_id = orders.order_id;
```

PLAN

```
SELECT
  "ORDERS"."ORDER_ID",
  "PRODUCTS"."NAME",
  "ORDER_DETAILS"."QUANTITY"
FROM "PUBLIC"."PRODUCTS"
/* PUBLIC.PRODUCTS.tableScan */
/* scanCount: 51 */
INNER JOIN "PUBLIC"."ORDER_DETAILS"
/* PUBLIC.CONSTRAINT_INDEX_800: PRODUCT_ID = PRODUCTS.PRODUCT_ID */
ON 1=1
/* WHERE ORDER_DETAILS.PRODUCT_ID = PRODUCTS.PRODUCT_ID */
/* scanCount: 550 */
INNER JOIN "PUBLIC"."ORDERS"
/* PUBLIC.PRIMARY_KEY_8: ORDER_ID = ORDER_DETAILS.ORDER_ID */
ON 1=1
/* scanCount: 1000 */
WHERE ("ORDER_DETAILS"."ORDER_ID" = "ORDERS"."ORDER_ID")
AND ("ORDER_DETAILS"."PRODUCT_ID" = "PRODUCTS"."PRODUCT_ID")
(1 row, 5 ms)
```

Three tables with given sizes:

products: 50 rows, orders: 200 rows, order_details: 500 rows

The expected join order is:

products -> order_details -> orders

Which matches with the EXPLAIN ANALYZE above.

HW 5 : Query 4 - Four Tables

```
SELECT
  customers.name,
  orders.order_id,
  products.name,
  order_details.quantity

FROM
  customers
  JOIN orders ON customers.customer_id = orders.customer_id
  JOIN order_details ON orders.order_id = order_details.order_id
  JOIN products ON order_details.product_id = products.product_id;
```

PLAN

```
SELECT
  "CUSTOMERS"."NAME",
  "ORDERS"."ORDER_ID",
  "PRODUCTS"."NAME",
  "ORDER_DETAILS"."QUANTITY"
FROM "PUBLIC"."CUSTOMERS"
/* PUBLIC.CUSTOMERS.tableScan */
/* scanCount: 11 */
INNER JOIN "PUBLIC"."ORDERS"
/* PUBLIC.CONSTRAINT_INDEX_8: CUSTOMER_ID = CUSTOMERS.CUSTOMER_ID */
ON 1=1
/* WHERE CUSTOMERS.CUSTOMER_ID = ORDERS.CUSTOMER_ID
*/
/* scanCount: 210 */
INNER JOIN "PUBLIC"."ORDER_DETAILS"
/* PUBLIC.CONSTRAINT_INDEX_80: ORDER_ID = ORDERS.ORDER_ID */
ON 1=1
/* WHERE ORDERS.ORDER_ID = ORDER_DETAILS.ORDER_ID
*/
/* scanCount: 700 */
INNER JOIN "PUBLIC"."PRODUCTS"
/* PUBLIC.PRIMARY_KEY_F: PRODUCT_ID = ORDER_DETAILS.PRODUCT_ID */
ON 1=1
/* scanCount: 1000 */
WHERE ("ORDER_DETAILS"."PRODUCT_ID" = "PRODUCTS"."PRODUCT_ID")
AND ("ORDERS"."ORDER_ID" = "ORDER_DETAILS"."ORDER_ID")
AND ("CUSTOMERS"."CUSTOMER_ID" = "ORDERS"."CUSTOMER_ID")
(1 row, 3 ms)
```

The involved tables here are:

customers: 10 rows, products: 50 rows, orders: 200 rows, order_details: 500 rows

The expected join order is:

customers -> orders -> order_details -> products

Shown in the EXPLAIN ANALYZE, the plan reads the smallest table, then joins them in relations.

HW 5 : Query 5 - Five Tables

```

JOIN orders ON order_details.order_id = orders.order_id
JOIN customers ON orders.customer_id = customers.customer_id;
PLAN
SELECT
  "ORDER_DETAILS"."ORDER_DETAIL_ID",
  "ORDERS"."ORDER_ID",
  "CUSTOMERS"."NAME",
  "PRODUCTS"."NAME",
  "SUPPLIERS"."NAME",
  "ORDER_DETAILS"."QUANTITY"
FROM "PUBLIC"."CUSTOMERS"
/* PUBLIC.CUSTOMERS.tableScan */
/* scanCount: 11 */
INNER JOIN "PUBLIC"."ORDERS"
/* PUBLIC.CONSTRAINT_INDEX_8: CUSTOMER_ID = CUSTOMERS.CUSTOMER_ID */
ON 1=1
/* WHERE ORDERS.CUSTOMER_ID = CUSTOMERS.CUSTOMER_ID
*/
/* scanCount: 210 */
INNER JOIN "PUBLIC"."ORDER_DETAILS"
/* PUBLIC.CONSTRAINT_INDEX_80: ORDER_ID = ORDERS.ORDER_ID */
ON 1=1
/* WHERE ORDER_DETAILS.ORDER_ID = ORDERS.ORDER_ID
*/
/* scanCount: 700 */
INNER JOIN "PUBLIC"."PRODUCTS"
/* PUBLIC.PRIMARY_KEY_F: PRODUCT_ID = ORDER_DETAILS.PRODUCT_ID */
ON 1=1
/* WHERE ORDER_DETAILS.PRODUCT_ID = PRODUCTS.PRODUCT_ID
*/
/* scanCount: 1000 */
INNER JOIN "PUBLIC"."SUPPLIERS"
/* PUBLIC.PRIMARY_KEY_A: SUPPLIER_ID = PRODUCTS.SUPPLIER_ID */
ON 1=1
/* scanCount: 1000 */
WHERE ("ORDERS"."CUSTOMER_ID" = "CUSTOMERS"."CUSTOMER_ID")
AND ("ORDER_DETAILS"."ORDER_ID" = "ORDERS"."ORDER_ID")
AND ("PRODUCTS"."SUPPLIER_ID" = "SUPPLIERS"."SUPPLIER_ID")
AND ("ORDER_DETAILS"."PRODUCT_ID" = "PRODUCTS"."PRODUCT_ID")
(1 row, 3 ms)

```

The involved tables here are:

- customers: 10 rows, suppliers: 15 rows, products: 50 rows, orders: 200 rows, order_details: 500 rows.

The expected join order is:

customers -> orders -> order_details -> products -> suppliers

Hence also shown in the screenshot.

The Code simply uses `getFullConditions()` to get all the Join Conditions entered to the Filter, then uses `getRowCountApproximation()` to return the number of rows, with a recursive match of `getTableName()` on left and right.

HW 5 : Query 6 - Four Tables, More Options

```
PLAN
SELECT
  "ORDERS"."ORDER_ID",
  "ORDER_PAYMENTS"."AMOUNT",
  "ORDER_DETAILS"."QUANTITY",
  "PRODUCTS"."NAME"
FROM "PUBLIC"."PRODUCTS"
/* PUBLIC.PRODUCTS.tableScan */
/* scanCount: 51 */
INNER JOIN "PUBLIC"."ORDER_DETAILS"
/* PUBLIC.CONSTRAINT_INDEX_800: PRODUCT_ID = PRODUCTS.PRODUCT_ID */
ON 1=1
/* WHERE ORDER_DETAILS.PRODUCT_ID = PRODUCTS.PRODUCT_ID
*/
/* scanCount: 550 */
INNER JOIN "PUBLIC"."ORDER_PAYMENTS"
/* PUBLIC.CONSTRAINT_INDEX_C: ORDER_ID = ORDER_DETAILS.ORDER_ID */
ON 1=1
/* WHERE ORDER_DETAILS.ORDER_ID = ORDER_PAYMENTS.ORDER_ID
*/
/* scanCount: 900 */
INNER JOIN "PUBLIC"."ORDERS"
/* PUBLIC.PRIMARY_KEY_8: ORDER_ID = ORDER_DETAILS.ORDER_ID
AND ORDER_ID = ORDER_PAYMENTS.ORDER_ID
*/
ON 1=1
/* scanCount: 800 */
WHERE ("ORDER_DETAILS"."PRODUCT_ID" = "PRODUCTS"."PRODUCT_ID")
AND ("ORDER_DETAILS"."ORDER_ID" = "ORDER_PAYMENTS"."ORDER_ID")
AND ("ORDERS"."ORDER_ID" = "ORDER_DETAILS"."ORDER_ID")
AND ("ORDER_PAYMENTS"."ORDER_ID" = "ORDERS"."ORDER_ID")
(1 row, 3 ms)
```

This question is done to check whether the code allows recursive joins to happen. It does not according to the screenshot.

Our rule based optimizer is still fairly limited. Can you think of a query in which it would perform a fairly catastrophic join order?

This approach is simply the greedy algorithm where we choose the smallest table and joins the next worthy and smallest table. This means that joining lots of small things first could happen, without considering large joins in the end.

Query example:

```
SELECT *
FROM tiny1
JOIN bridge ON tiny1.id = bridge.t1_id
JOIN tiny2 ON tiny2.id = bridge.t2_id
JOIN large1 ON bridge.l1_id = large1.id
JOIN large2 ON bridge.l2_id = large2.id
```

tiny1=10, tiny2=20, large1=100,000, large2=100,000, bridge=1000

The query will pick tiny1->bridge->tiny2->large1->large2, which does not eliminate most rows in earlier operations (Against basic principles)

Our rule based optimizer is still fairly limited. If you were to improve it, what additional rules would you include?

Instead of always picking the smallest table, prefer tables that connect to more remaining tables. Also delay large-large joins unless filtered or necessary for connectivity. Also add index logic.