# SALES Pre-Computation

# Original Query

SELECT cate.category\_id

,cust.customer\_id

,sum(quantity)

,sum(price)

FROM (

SELECT category\_id

,sum(price) AS dollar\_value

FROM sales.category NATURAL

INNER JOIN sales.product NATURAL

INNER JOIN sales.sale

GROUP BY category\_id

ORDER BY dollar\_value DESC limit 10

) AS cate

,(

SELECT customer\_id

,sum(price) AS dollar\_value

FROM sales.sale

GROUP BY customer\_id

ORDER BY dollar\_value DESC limit 10

) AS cust

,sales.sale s

,sales.product p

WHERE p.category\_id = cate.category\_id

AND s.customer\_id = cust.customer\_id

AND s.product\_id = p.product\_id

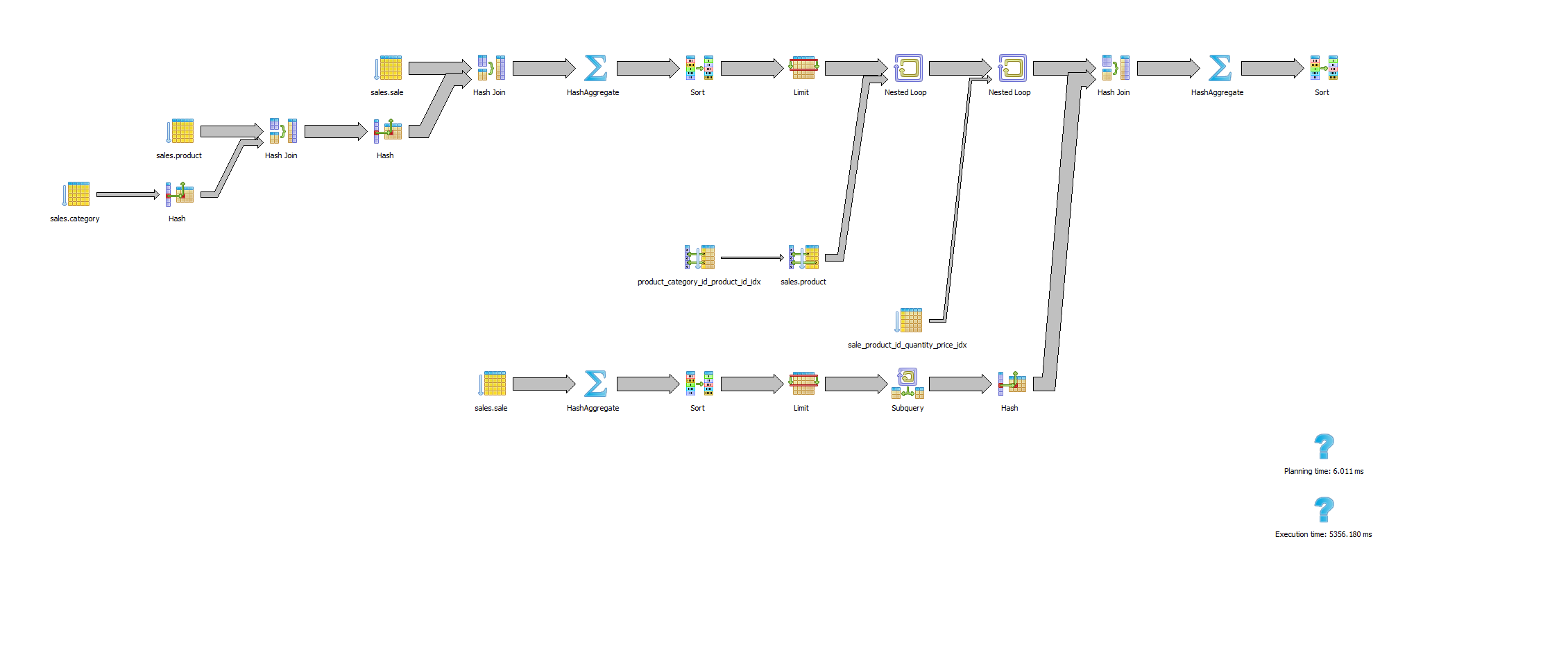
GROUP BY cate.category\_id

,cust.customer\_id

ORDER BY cate.category\_id

|  |  |  |
| --- | --- | --- |
| Query Cost | Execution Time | Table IO |
| 206047.84..206048.04 | 6148.910 ms | 45667 |

### Query Plan:



## New Query using Pre-Computed Table

WITH cate AS (

SELECT category\_id

FROM sales.PRE\_Category\_Sale

ORDER BY dollar\_value DESC limit 10

)

,cust as (

SELECT customer\_id

FROM sales.PRE\_Customer\_Sale

ORDER BY dollar\_value DESC limit 10

)

SELECT cate.category\_id

,cust.customer\_id

,sum(quantity)

,sum(price)

FROM cate

,cust

,sales.sale s

,sales.product p

WHERE p.category\_id = cate.category\_id

AND s.customer\_id = cust.customer\_id

AND s.product\_id = p.product\_id

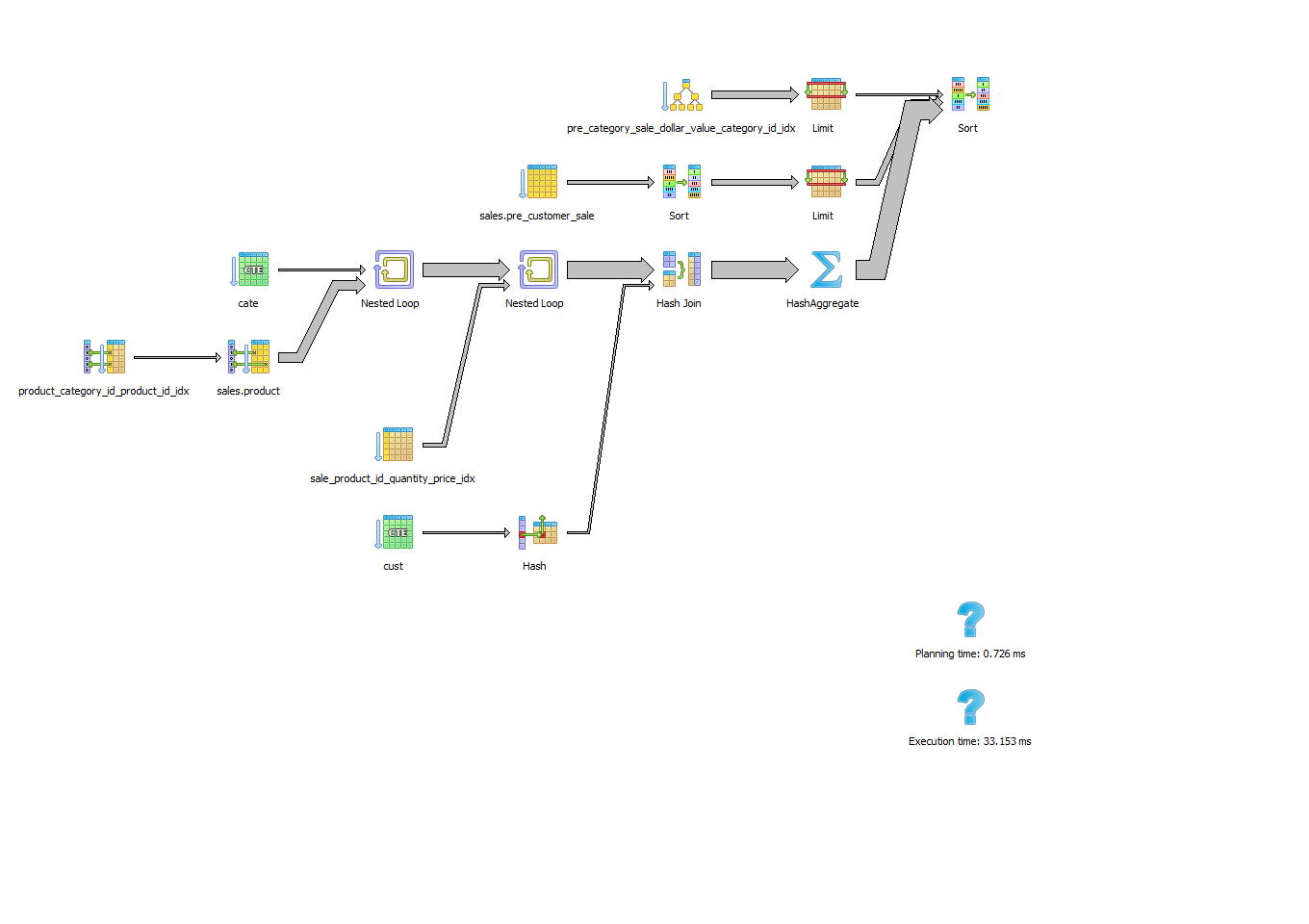
GROUP BY cate.category\_id

,cust.customer\_id

ORDER BY cate.category\_id

|  |  |  |
| --- | --- | --- |
| Query Cost | Execution Time | Table IO |
| 27749.81..27750.01 | 49.866 ms | 9288 |

### Query Plan:



## Pre-Computation

Two pre-computed tables were created, one for each subquery: sales.PRE\_Category\_Sale, sales.PRE\_Customer\_Sale.

DROP TABLE IF EXISTS sales.PRE\_Category\_Sale ;

create table sales.PRE\_Category\_Sale as

SELECT category\_id

,sum(price) AS dollar\_value

FROM sales.category NATURAL

INNER JOIN sales.product NATURAL

INNER JOIN sales.sale

GROUP BY category\_id;

DROP TABLE IF EXISTS sales.PRE\_Customer\_Sale;

create table sales.PRE\_Customer\_Sale as

SELECT customer\_id

,sum(price) AS dollar\_value

FROM sales.sale

GROUP BY customer\_id;

CREATE or replace FUNCTION sales.FN\_PRE\_Category\_Sale() RETURNS trigger AS $FN\_PRE\_Category\_Sale$

BEGIN

UPDATE sales.PRE\_Category\_Sale as pre

SET dollar\_value = pre.dollar\_value + NEW.price

FROM sales.product as p

WHERE pre.category\_id = p.category\_Id

AND p.product\_id = NEW.product\_id;

IF NOT FOUND THEN

--no category\_id updated

INSERT INTO sales.PRE\_Category\_Sale (category\_id, dollar\_value)

SELECT p.category\_id, NEW.price

FROM sales.product p

WHERE p.product\_id = NEW.product\_id;

END IF;

RETURN NEW;

END;

$FN\_PRE\_Category\_Sale$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS trigger\_Sale\_Category ON sales.sale ;

CREATE TRIGGER trigger\_Sale\_Category

BEFORE INSERT ON sales.sale

FOR EACH ROW EXECUTE PROCEDURE sales.FN\_PRE\_Category\_Sale();

CREATE or replace FUNCTION sales.FN\_PRE\_Customer\_Sale() RETURNS trigger AS $FN\_PRE\_Customer\_Sale$

BEGIN

UPDATE sales.PRE\_Customer\_Sale

SET dollar\_value = dollar\_value + NEW.price

WHERE customer\_id = NEW.customer\_id;

IF NOT FOUND THEN

--no category\_id updated

INSERT INTO sales.PRE\_Customer\_Sale (customer\_id, dollar\_value)

SELECT NEW.customer\_id, NEW.price;

END IF;

RETURN NEW;

END;

$FN\_PRE\_Customer\_Sale$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS trigger\_Sale\_Customer ON sales.sale ;

CREATE TRIGGER trigger\_Sale\_Customer

BEFORE INSERT ON sales.sale

FOR EACH ROW EXECUTE PROCEDURE sales.FN\_PRE\_Customer\_Sale();

|  |  |
| --- | --- |
|  | Maintenance Cost |
| Update sales.Category\_Sale | 0.70..157.24 |
| Update sales.Customer\_Sale | 0.00..1.10 |
| Total Cost | 0.70..158.34 |

## Indexes

The following indexes were used by the trigger to improve maintenance cost or to improve the cost of the final query.

### pre\_category\_sale\_dollar\_value\_category\_id\_idx

CREATE INDEX pre\_category\_sale\_dollar\_value\_category\_id\_idx

ON sales.pre\_category\_sale

USING btree

(dollar\_value DESC, category\_id);

### product\_category\_id\_product\_id\_idx

CREATE INDEX product\_category\_id\_product\_id\_idx

ON sales.product

USING btree

(category\_id, product\_id);

### product\_product\_id\_category\_id\_idx

CREATE INDEX product\_product\_id\_category\_id\_idx

ON sales.product

USING btree

(product\_id, category\_id);

### sale\_product\_id\_quantity\_price\_idx

CREATE INDEX sale\_product\_id\_quantity\_price\_idx

ON sales.sale

USING btree

(product\_id, quantity, price);

# Conclusion

|  |  |  |  |
| --- | --- | --- | --- |
|  | Query Cost | Execution Time | Table IO |
| Original Query | 206047.84..206048.04 | 6148.910 ms | 45667 |
| Using Pre-compute | 27749.81..27750.01 | 49.866 ms | 9288 |
| Difference | (178298.03..178298.03) | (6099.044) ms | (36379) |

Maintenance Cost for n=1 (1 new sale): 0.70..158.34

Given the maintenance execution cost of 158.34, using the pre-computation described would be optimal for values of n between 1 and 1,301.