**Case 1 - Small Database**

Overall running time: 203 ms

1. "SELECT name FROM categories WHERE id = " + filter"
   * 1 ms
2. "SELECT COUNT(\*) as num FROM (SELECT name FROM products INNER JOIN orders ON products.id = orders.product\_id GROUP BY name) p"
   * Before index: 12 ms
   * Using index on orders.product\_id: 10 ms (experimented, not useful)
3. "SELECT COUNT(\*) as num FROM (SELECT name FROM products INNER JOIN orders ON products.id = orders.product\_id WHERE products.category\_id = " + filterString + " GROUP BY name) p"
   * Before index: 5 ms
   * Using index on orders.product\_id: 5 ms (experimented, not useful)
   * Using index on products.category\_id: sure to be useful since we want to match the category\_id with the filtered id, using index will find faster.
4. "SELECT COUNT(\*) as num FROM (SELECT users.name as name FROM users INNER JOIN orders ON users.id = orders.user\_id GROUP BY users.name) p"
   * Before index: 13 ms
   * Using index on orders.user\_id: 13 ms (experimented, not useful)
5. "SELECT COUNT(\*) as num FROM (SELECT users.name as name FROM users INNER JOIN orders ON users.id = orders.user\_id INNER JOIN products ON products.category\_id = " + filterString + " GROUP BY users.name) c"
   * Before index: 58 ms
   * Using index on orders.user\_id: 57 ms (experimented, not useful)
   * Using index on products.category\_id: sure to be useful since we want to match the category\_id with the filtered id, using index will find faster.
6. "SELECT COUNT(\*) as num FROM (SELECT state FROM users INNER JOIN orders ON users.id = orders.user\_id GROUP BY state) p"
   * Before index: 12 ms
   * Using index on orders.user\_id: 11 ms (experimented, not useful)
7. "SELECT COUNT(\*) as num FROM (SELECT state FROM users INNER JOIN orders ON users.id = orders.user\_id INNER JOIN products ON products.category\_id = " + filterString + " GROUP BY state) s"
   * Before index: 60 ms
   * Using index on orders.user\_id: 59 ms (experimented, not useful)
   * Using index on products.category\_id: sure to be useful since we want to match the category\_id with the filtered id, using index will find faster.
8. "WITH col\_header(product\_id, totalsales) AS (SELECT product\_id, SUM(orders.price) AS totalsales FROM orders GROUP BY product\_id) SELECT products.id AS id, products.name AS name, col\_header.totalsales AS totalsales FROM products INNER JOIN col\_header ON products.id = col\_header.product\_id " + orderString + "LIMIT 10 OFFSET " + session.getAttribute("colNum")"
   * Before index: 13 ms
9. "WITH col\_header(product\_id, totalsales) AS (SELECT product\_id, SUM(orders.price) AS totalsales FROM products INNER JOIN orders on orders.product\_id = products.id WHERE products.category\_id = " + filterString + " GROUP BY product\_id) SELECT products.id AS id, products.name AS name, col\_header.totalsales AS totalsales FROM products INNER JOIN col\_header ON products.id = col\_header.product\_id" + orderString + "LIMIT 10 OFFSET " + session.getAttribute("colNum")"
   * Before index: 5 ms
   * Using index on orders.product\_id: 5 ms (experimented, not useful)
   * Using index on products.category\_id: sure to be useful since we want to match the category\_id with the filtered id, using index will find faster.
10. "WITH row\_header(id, name, totalsales) AS (SELECT users.id AS id, users.name AS name, SUM(orders.price) AS totalsales FROM users INNER JOIN orders ON users.id = orders.user\_id GROUP BY users.id) SELECT DISTINCT LEFT(users.name, 10) AS name, users.id AS id, row\_header.totalsales AS totalsales FROM users INNER JOIN row\_header ON row\_header.name = users.name" + orderString + "LIMIT 20 OFFSET " + session.getAttribute("rowNum")"
    * Before index: 18 ms
    * Using index on orders.user\_id: 18 ms (experimented, not useful)
    * Using index on users.name: 18 ms (experimented, not useful)
11. "WITH row\_header(id, name, totalsales) AS (SELECT users.id AS id, users.name AS name, SUM(orders.price) AS totalsales FROM users INNER JOIN orders ON users.id = orders.user\_id INNER JOIN products ON orders.product\_id = products.id WHERE products.category\_id = " + filterString + " GROUP BY users.id) SELECT DISTINCT LEFT(users.name, 10) AS name, users.id AS id, row\_header.totalsales AS totalsales FROM users INNER JOIN row\_header ON row\_header.name = users.name" + orderString + "LIMIT 20 OFFSET " + session.getAttribute("rowNum")"
    * Before index: 8 ms
    * Using index on orders.user\_id: 8 ms (experimented, not useful)
    * Using index on orders.product\_id: 7 ms (experimented, not useful)
    * Using index on users.name: 7 ms (experimented, not useful)
    * Using index on products.category\_id: sure to be useful since we want to match the category\_id with the filtered id, using index will find faster.
12. "WITH row\_header(state, totalsales) AS (SELECT users.state AS state, SUM(orders.price) AS totalsales FROM users, orders WHERE users.id = orders.user\_id GROUP BY users.state ORDER BY totalsales DESC) SELECT DISTINCT LEFT(users.state, 10) AS state, users.id AS id, row\_header.totalsales AS totalsales FROM users INNER JOIN row\_header ON row\_header.state = users.state" + orderString2 + "LIMIT 20 OFFSET " + session.getAttribute("rowNum")"
    * Before index: 20 ms
    * Using index on orders.user\_id: 20 ms (experimented, not useful)
    * Using index on users.state: 20 ms (experimented, not useful)
13. "WITH row\_header(state, totalsales) AS (SELECT users.state AS state, SUM(orders.price) AS totalsales FROM users INNER JOIN orders ON users.id = orders.user\_id INNER JOIN products ON orders.product\_id = products.id WHERE products.category\_id = " + filterString + " GROUP BY users.state ORDER BY totalsales DESC) SELECT DISTINCT LEFT(users.state, 10) AS state, users.id AS id, row\_header.totalsales AS totalsales FROM users INNER JOIN row\_header ON row\_header.state = users.state" + orderString2 + "LIMIT 20 OFFSET " + session.getAttribute("rowNum")"
    * Before index: 11 ms
    * Using index on orders.user\_id: 10 ms (experimented, not useful)
    * Using index on orders.product\_id: 11 ms (experimented, not useful)
    * Using index on users.name: 11 ms (experimented, not useful)
    * Using index on users.state: 10 ms (experimented, not useful)
    * Using index on products.category\_id: sure to be useful since we want to match the category\_id with the filtered id, using index will find faster.
14. "SELECT SUM(orders.price) AS totalprices FROM orders WHERE orders.product\_id = " + rs2.getString("id") + " AND orders.user\_id = " + rs3.getString("id") + " GROUP BY orders.product\_id, orders.user\_id"
    * Before index: 3 ms
    * Using index on orders.user\_id: 3 ms (experimented, not useful)
    * Using index on orders.product\_id: 3 ms (experimented, not useful)
15. "SELECT SUM(orders.price) AS totalprices FROM orders INNER JOIN users ON users.id = orders.user\_id INNER JOIN products on products.id = orders.product\_id WHERE orders.product\_id = '" + rs2.getString("id") + "' AND users.state = '" + rs3.getString("state") + "' GROUP BY orders.product\_id, users.state"
    * Before index: 4 ms
    * Using index on orders.user\_id: 3 ms (experimented, not useful)
    * Using index on orders.product\_id: 4 ms (experimented, not useful)
    * Using index on users.state: 4 ms (experimented, not useful

**Case 2 - Large Database**

Overall running time: 990.9 s

1. "SELECT name FROM categories WHERE id = " + filter"
   * 1 ms
2. "SELECT COUNT(\*) as num FROM (SELECT name FROM products INNER JOIN orders ON products.id = orders.product\_id GROUP BY name) p"
   * Before index: 9.6 s
   * Using index on orders.product\_id: 9.6 s (experimented, not useful)
3. "SELECT COUNT(\*) as num FROM (SELECT name FROM products INNER JOIN orders ON products.id = orders.product\_id WHERE products.category\_id = " + filterString + " GROUP BY name) p"
   * Before index: 2.8 s
   * Using index on orders.product\_id: sure to be useful since we want to match the orders.product\_id with the products.id and there are many orders (over 10 millions), using index will find faster.
   * Using index on products.category\_id: 2.8 s (experimented, not useful)
4. "SELECT COUNT(\*) as num FROM (SELECT users.name as name FROM users INNER JOIN orders ON users.id = orders.user\_id GROUP BY users.name) c"
   * Before index: 8.9 s
   * Using index on orders.user\_id: 8.9 s (experimented, not useful)
5. "SELECT COUNT(\*) as num FROM (SELECT users.name as name FROM users INNER JOIN orders ON users.id = orders.user\_id INNER JOIN products ON products.category\_id = " + filterString + " GROUP BY users.name) c"
   * Before index: 497.2 s
   * Using index on orders.user\_id: 497.2 s (experimented, not useful)
   * Using index on products.category\_id: 497.2 s (experimented, not useful)
6. "SELECT COUNT(\*) as num FROM (SELECT state FROM users INNER JOIN orders ON users.id = orders.user\_id GROUP BY state) s"
   * Before index: 8.6 s
   * Using index on orders.user\_id: 8.3 ms (experimented, not useful)
7. "SELECT COUNT(\*) as num FROM (SELECT state FROM users INNER JOIN orders ON users.id = orders.user\_id INNER JOIN products ON products.category\_id = " + filterString + " GROUP BY state) s"
   * Before index: 459.5 s
   * Using index on orders.user\_id: 459.5 s (experimented, not useful)
   * Using index on products.category\_id: 459.5 s (experimented, not useful)
8. "WITH col\_header(product\_id, totalsales) AS (SELECT product\_id, SUM(orders.price) AS totalsales FROM orders GROUP BY product\_id) SELECT products.id AS id, products.name AS name, col\_header.totalsales AS totalsales FROM products INNER JOIN col\_header ON products.id = col\_header.product\_id " + orderString + "LIMIT 10 OFFSET " + session.getAttribute("colNum")"
   * Before index: 5.9 s
9. "WITH col\_header(product\_id, totalsales) AS (SELECT product\_id, SUM(orders.price) AS totalsales FROM products INNER JOIN orders on orders.product\_id = products.id WHERE products.category\_id = " + filterString + " GROUP BY product\_id) SELECT products.id AS id, products.name AS name, col\_header.totalsales AS totalsales FROM products INNER JOIN col\_header ON products.id = col\_header.product\_id" + orderString + "LIMIT 10 OFFSET " + session.getAttribute("colNum")"
   * Before index: 2.9 s
   * Using index on orders.product\_id: sure to be useful since we want to match the orders.product\_id with the products.id and there are many orders (over 10 millions), using index will find faster.
   * Using index on products.category\_id: 2.9 s (experimented, not useful)
10. "WITH row\_header(id, name, totalsales) AS (SELECT users.id AS id, users.name AS name, SUM(orders.price) AS totalsales FROM users INNER JOIN orders ON users.id = orders.user\_id GROUP BY users.id) SELECT DISTINCT LEFT(users.name, 10) AS name, users.id AS id, row\_header.totalsales AS totalsales FROM users INNER JOIN row\_header ON row\_header.name = users.name" + orderString + "LIMIT 20 OFFSET " + session.getAttribute("rowNum")"
    * Before index: 10.3 s
    * Using index on orders.user\_id: 10.3 s (experimented, not useful)
    * Using index on users.name: 10.3 s (experimented, not useful)
11. "WITH row\_header(id, name, totalsales) AS (SELECT users.id AS id, users.name AS name, SUM(orders.price) AS totalsales FROM users INNER JOIN orders ON users.id = orders.user\_id INNER JOIN products ON orders.product\_id = products.id WHERE products.category\_id = " + filterString + " GROUP BY users.id) SELECT DISTINCT LEFT(users.name, 10) AS name, users.id AS id, row\_header.totalsales AS totalsales FROM users INNER JOIN row\_header ON row\_header.name = users.name" + orderString + "LIMIT 20 OFFSET " + session.getAttribute("rowNum")"
    * Before index: 2.9 s
    * Using index on orders.user\_id: 2.9 s (experimented, not useful)
    * Using index on orders.product\_id: sure to be useful since we want to match the orders.product\_id with the products.id and there are many orders (over 10 millions), using index will find faster.
    * Using index on users.name: 2.9 s (experimented, not useful)
    * Using index on products.category\_id: 2.9 s (experimented, not useful)
12. "WITH row\_header(state, totalsales) AS (SELECT users.state AS state, SUM(orders.price) AS totalsales FROM users, orders WHERE users.id = orders.user\_id GROUP BY users.state ORDER BY totalsales DESC) SELECT DISTINCT LEFT(users.state, 10) AS state, users.id AS id, row\_header.totalsales AS totalsales FROM users INNER JOIN row\_header ON row\_header.state = users.state" + orderString2 + "LIMIT 20 OFFSET " + session.getAttribute("rowNum")"
    * Before index: 10.4 s
    * Using index on orders.user\_id: 10.4 s (experimented, not useful)
    * Using index on users.state: 10.4 s (experimented, not useful)
13. "WITH row\_header(state, totalsales) AS (SELECT users.state AS state, SUM(orders.price) AS totalsales FROM users INNER JOIN orders ON users.id = orders.user\_id INNER JOIN products ON orders.product\_id = products.id WHERE products.category\_id = " + filterString + " GROUP BY users.state ORDER BY totalsales DESC) SELECT DISTINCT LEFT(users.state, 10) AS state, users.id AS id, row\_header.totalsales AS totalsales FROM users INNER JOIN row\_header ON row\_header.state = users.state" + orderString2 + "LIMIT 20 OFFSET " + session.getAttribute("rowNum")"
    * Before index: 2.9 s
    * Using index on orders.user\_id: 2.9 s (experimented, not useful)
    * Using index on orders.product\_id: sure to be useful since we want to match the orders.product\_id with the products.id and there are many orders (over 10 millions), using index will find faster.
    * Using index on users.name: 2.9 s (experimented, not useful)
    * Using index on users.state: 2.9 s (experimented, not useful)
    * Using index on products.category\_id: 2.9 s (experimented, not useful)
14. "SELECT SUM(orders.price) AS totalprices FROM orders WHERE orders.product\_id = " + rs2.getString("id") + " AND orders.user\_id = " + rs3.getString("id") + " GROUP BY orders.product\_id, orders.user\_id"
    * Before index: 2.3 s
    * Using index on orders.user\_id: 22 ms (experimented, useful)
    * Using index on orders.product\_id: sure to be useful since we want to match the orders.product\_id with the products.id and there are many orders (over 10 millions), using index will find faster.
15. "SELECT SUM(orders.price) AS totalprices FROM orders INNER JOIN users ON users.id = orders.user\_id INNER JOIN products on products.id = orders.product\_id WHERE orders.product\_id = '" + rs2.getString("id") + "' AND users.state = '" + rs3.getString("state") + "' GROUP BY orders.product\_id, users.state"
    * Before index: 2.3 s
    * Using index on orders.user\_id: 2.3 s (experimented, not useful)
    * Using index on orders.product\_id: sure to be useful since we want to match the orders.product\_id with the products.id and there are many orders (over 10 millions), using index will find faster.
    * Using index on users.state: 2.3 s (experimented, not useful