SQL useful queries

Here are some useful SQL queries:

1. Basic Structure

```
SELECT (DISTINCT table1.A),
    table2.B,
    SUM(CASE table1.H = 1 THEN table1.C ELSE 0 END) AS sum_C,
    AVG(table1.D) AS avg_D
FROM table_name AS table1
LEFT JOIN table_name2 AS table2
ON table1.A = table2.A (AND table1.E = table2.E)
WHERE table1.F LIKE "%app%"
      AND DATE(table1.G) > CAST("2023-12-01" AS DATE)
GROUP BY 1(table2.B)
(HAVING SUM(table1.C) >0)
ORDER BY 1(table2.B) DESC(ASC)
(LIMIT 5);
```

2. Table manipulation

Function	**MySQL**	**HiveSQL**
Creating a database	CREATE DATABASE db_name;	CREATE DATABASE db_name;
Describing the format of a table	DESCRIBE table;	DESCRIBE (FORMATTED|EXTENDED) table;
Drop a table	DROP TABLE table_name;	DROP TABLE table_name;
Dropping a database	DROP DATABASE db_name;	DROP DATABASE db_name (CASCADE);
Listing databases	SHOW databases;	SHOW databases;
Listing tables in a database	SHOW tables;	SHOW tables;
Selecting a database	USE database;	USE database;

Deleting the content from the table	DELETE FROM db_name.table_name	DELETE FROM db_name.table_name

3. Time functions (Presto)

```
-- String to Date
SELECT DATE('2023-07-09 12:59:01');
SELECT CAST('2023-07-09' AS DATE);

-- Time calculation
SELECT CAST(CURRENT_DATE - INTERVAL '1' DAY AS VARCHAR);
SELECT DATE_ADD('DAY', 4, CURRENT_DATE);
SELECT DATE_DIFF('DAY', CAST('2023-02-01' AS TIMESTAMP), CAST('2
SELECT DATE_TRUNC('QUARTER', CURRENT_DATE);

-- Date to Time
SELECT TO_UNIXTIME(TIMESTAMP '2023-07-09 12:59:01');
SELECT TO_UNIXTIME(CAST('2023-07-09 12:59:01' AS TIMESTAMP));
SELECT FROM_UNIXTIME(1688878741);

-- Current date
SELECT NOW();
SELECT CURRENT_DATE;
SELECT CURRENT_TIME;

-- Set a longer time duration
SET SESSION query_max_execution_time='90m';
```

4. Time functions (Vertica)

```
day >= CURRENT_DATE - INTERVAL '7 day'
DATEDIFF(month, '01-31-2005'::date, '09-30-2005'::date)
DATEDIFF(day, INTERVAL '26 days', INTERVAL '1 month ')
CAST(a.run_datetime AS DATE)
```

5. Window functions

```
-- window function
avg(field 1) over(partition by country ORDER BY date RANGE BETW
avg(field 2) over(partition by country ORDER BY date rows 6 pre
avg(field 3) over()
LEAD(field 4) OVER (PARTITION BY country, day ORDER BY order_ti
COALESCE(SUM(revenue_percentage) OVER (PARTITION BY hour ORDER
```

6. Methods to improve the query efficiency

- Reduce the number of subquery, avoid querying the same table twice
- Simply the query logics and avoid using computation expensive functions (e.g. count(distinct id))
- Use the partition when the database support, to reduce the volume of data retrieved
- If the database is Hadoop based, instead of exporting the raw data and analyzing with PySpark, one can as well embed the calculation logic within SQL query and only export the results to simply
- Use With to wrap up the subqueries to increase readability

7. Common Table Expression

```
With assisted_query(
SELECT A,
            B,
            SUM(C) AS sum_C
FROM tablename
WHERE D = 'ok'
GROUP BY A,B)

-- Main query
SELECT *
FROM assisted_query
LIMIT 5
```

8. Check data relationship

```
--  1 to 1:
    SELECT COUNT(*) AS total_rows,
           COUNT(DISTINCT A) AS distinct_a,
           COUNT(DISTINCT B) AS distinct_b
    FROM your_table;

--  1 to N:
    SELECT A, COUNT(DISTINCT B)
    FROM your_table
    GROUP BY A
    HAVING COUNT(DISTINCT B) > 1;

--  N to 1:
    SELECT B, COUNT(DISTINCT A)
    FROM your_table
    GROUP BY B
    HAVING COUNT(DISTINCT A) > 1;
```

9. Arrays and lists

```
SELECT unnested_tables.db_database,
      unnested_tables.db_table,
      count(*) AS access_frequency,
      DATE_DIFF('DAY', CAST(MAX(a.day) AS TIMESTAMP), CAST('2023
FROM db_name.table_name AS a
CROSS JOIN UNNEST(a.array_column) AS unnested_field (db_table, c
WHERE a.day>='2022-09-01' AND a.day <= '2023-08-31'
GROUP BY 1,2
ORDER BY 3 DESC
```