1. WHERE 1=1

When you code has different WHERE condi
having a WHERE 1=1 simplifies the logic.

This is because 1=1 is always true and does
the query's actual result, but it allows you to
easily append additional conditions **without
special handling for the first condition.**

It improves the overall readability of the code

```
SELECT * FROM users WHERE 1=1
AND age > 25
AND city = 'New York'
AND gender = 'Female'
```

2. QUALIFY

It can be used to filter the results of a query b
the result of a window function.

You don’t need nested queries making the c
easier to read.

It is **similar to the HAVING**, but instead of fi
after an aggregation, QUALIFY filters after th
application of window functions.

```
SELECT
    id,
    salesperson,
    amount,
    ROW_NUMBER() OVER (PARTITION BY salesperson ORD
DESC) AS rank
FROM sales
QUALIFY rank = 1;
```

3. ROW_NUMBE

Incredibly useful when it comes to cleaning d

ROW_NUMBER ( ) can be used to identifyin
removing duplicates, and detecting gaps in d

It can be used to select a SINGLE row based
conditions such as latest record, highest/low
etc.

```
WITH RankedOrders AS (
    SELECT
        customer_id,
        order_id,
        ROW_NUMBER() OVER (PARTITION BY customer_id ORD
_date DESC) AS rn
    FROM orders
)
SELECT *
FROM RankedOrders
WHERE rn = 1;
```

4. EXCLUDE <C

This not standard in SQL but is a feature fou
some SQL dialects, such as BigQuery.

It allows you to easily select all columns from
except one or more specified columns.

This improves readability and reduces repet

```
SELECT col1, col2, col3, col5 -- manual
excluding col4
FROM table;
```

```
SELECT * EXCLUDE col
FROM table;
```

5. EXISTS

Employed when you want to check for the ex
records in a related table or subquery. Helps
you do your EDA.

It returns a TRUE if the subquery returns at l
row.

It is **more performant** than using IN or JOIN

```
SELECT department_name
FROM departments d
WHERE EXISTS (
  SELECT 1
  FROM employees e
  WHERE e.department_id = d.departmen
);
```

6. COALESCE

This function handles NULL values gracefull

COALESCE allows you to provide a fallback
when encountering NULL.

Rather than using a complex CASE stateme
multiple IFNULL/ISNULL functions, COALES
provides a cleaner syntax.

You can use COALESCE to choose the first
NULL value across multiple columns.

```
SELECT COALESCE(home_phone, mobile_phone, of
AS contact_number
FROM contacts;
```

7. TEMP TABLE

Temp tables allow you to break the query int
more manageable parts.

Trying to fit everything into a single, massive
statement with multiple WITH statements ca
your query too complex.

This way you can also avoid repeated calcul
re-use queries.

Also it helps you understand, debug, and op
each part of the query independently.

8. SYSCAT / SYSIN

Helps you obtain metadata on the underlying database platform that you are using.

Querying **syscat** or **sysinfo** to find out what tables, columns, etc are available.

For example, you can query SYS.COLUMNS details about all the columns in a particular ta

SYS.KEYS and SYS.CONSTRAINTS can be get info about primary keys, foreign keys, an constraints applied to tables.

```
SELECT * FROM SYS.COLUMNS
WHERE TABLE_NAME = 'employees'
```

9. LAG / LEAD

Extremely useful for performing operations t
require accessing data from previous or sub
rows.

If you are building a KPI dashboard and wan
calculate **month-over-month** or **year-over-**
then this syntax makes the calculation a lot e

```
SELECT
    Month,
    Product,
    Sales,
    LAG(Sales, 1) OVER (PARTITION BY Product ORDER
AS PreviousMonthSales,
    Sales - LAG(Sales, 1) OVER (PARTITION BY Produ
Month) AS SalesDifference
FROM Sales;
```