# [What are the Best Practices in Writing Queries?](#_rszi56bzs0nv)

## Optimizing Query Computation:

When evaluating the computation that is required by a query, consider the amount of work that is required. How much CPU time is required? Are you using functions like JavaScript user-defined functions that require additional CPU resources?

The following best practices provide guidance on controlling query computation.

### Avoid repeatedly transforming data via SQL queries:

**Best practice:** If you are using SQL to perform ETL operations, avoid situations where you are repeatedly transforming the same data.

### Avoid JavaScript user-defined functions:

**Best practice:** Avoid using JavaScript user-defined functions. Use native UDFs instead.

### Use approximate aggregation functions:

**Best practice:** If your use case supports it, use an approximate aggregation function.

### Order query operations to maximize performance:

**Best practice:** Use ORDER BY only in the outermost query or within window clauses (analytic functions). Push complex operations to the end of the query.

### Optimize your join patterns:

**Best practice:** For queries that join data from multiple tables, optimize your join patterns. Start with the largest table.

### Prune partitioned queries:

**Best practice:** When querying a [time-partitioned table](https://cloud.google.com/bigquery/docs/querying-partitioned-tables), use the \_PARTITIONTIME pseudo column to filter the partitions.

# Link for optimizing query computation:

(<https://cloud.google.com/bigquery/docs/best-practices-performance-compute>)

## Managing Query Outputs

When evaluating your output data, consider the number of bytes written by your query. How many bytes are written for your result set? Are you properly limiting the amount of data written? Are you repeatedly writing the same data? The amount of data written by a query impacts query performance (I/O). If you are writing results to a permanent (destination) table, the amount of data written also has a cost.

The following best practices provide guidance on controlling your output data.

### Avoid repeated joins and subqueries

**Best practice:** Avoid repeatedly joining the same tables and using the same subqueries.

### Carefully consider materializing large result sets

**Best practice:** Carefully consider [materializing large result sets](https://cloud.google.com/bigquery/docs/writing-results#large-results) to a destination table. Writing large result sets has performance and cost impacts.

### Use a LIMIT clause with large sorts

**Best practice:** If you are sorting a very large number of values, use a LIMIT clause.

# 

# Link for managing Query o/p

(<https://cloud.google.com/bigquery/docs/best-practices-performance-output>)

## Avoiding SQL Anti-Patterns

The following best practices provide guidance on avoiding query anti-patterns that impact performance in BigQuery.

### Self-joins

**Best practice:** Avoid self-joins. Use a [window function](https://cloud.google.com/bigquery/docs/reference/standard-sql/functions-and-operators#analytic-functions) instead.

### Data skew

**Best practice:** If your query processes keys that are heavily skewed to a few values, filter your data as early as possible.

#### i) Unbalanced joins

Data skew can also appear when you use JOIN clauses. Because BigQuery shuffles data on each side of the join, all data with the same join key goes to the same shard. This shuffling can overload the slot.

### Cross joins (Cartesian product)

**Best practice:** Avoid joins that generate more outputs than inputs. When a CROSS JOIN is required, pre-aggregate your data.

### DML statements that update or insert single rows

**Best practice:** Avoid point-specific [DML](https://cloud.google.com/bigquery/docs/reference/standard-sql/data-manipulation-language) statements (updating or inserting 1 row at a time). Batch your updates and inserts.

# Link for Avoiding SQL Anti-Patterns

(<https://cloud.google.com/bigquery/docs/best-practices-performance-patterns>)