SOQL Performance Best Practices for Large Salesforce Orgs

CHEATSHEET

Stop guessing at fixes. Learn how the query optimizer decides between fast index scans and slow table scans.

Your Apex batch worked fine at 50k records, but at 5M it throws QUERY_TIMEOUT. Stop guessing fixes and master how the Salesforce query optimizer actually selects indexes.

1 The 10% Rule: How the Optimizer Thinks

The Salesforce query optimizer decides whether to use an index based on selectivity thresholds. If your query returns too many rows, indexes are completely bypassed.

- Index Scan: Fast read of a small subset of records using indexed fields.
- Full Table Scan: Slow, unindexed scan of the entire table that causes query timeouts.
- Selectivity Threshold: Typically filters matching less than 10% of total records.
- Large Object Limit: Filters must return fewer than 333,000 records on massive objects.

2 How to Write Truly Selective Filters

To guarantee the Salesforce query optimizer uses your indexes, you must adhere to strict query design constraints in your WHERE clauses.

- Filter on indexed fields first (Id, Name, OwnerId, CreatedDate, External IDs).
- Avoid functions on filter fields (e.g., CALENDAR_YEAR() blocks index usage).
- Use explicit date ranges instead of formula functions on date fields.
- Do not use negative operators like !=, NOT IN, or EXCLUDES on large datasets.
- Use LIMIT clauses defensively to control result set sizes.

3 The Anti-Pattern: Non-Selective SOQL

Using negative operators and unindexed status fields prevents the query optimizer from leveraging standard indexes, forcing a full table scan.

APEX

```
// Problematic — Status is not indexed, returns a large
percentage of rows
List opps = [
    SELECT Id, Name, Amount
    FROM Opportunity
    WHERE StageName != 'Closed Won'
    AND OwnerId != null
];
```