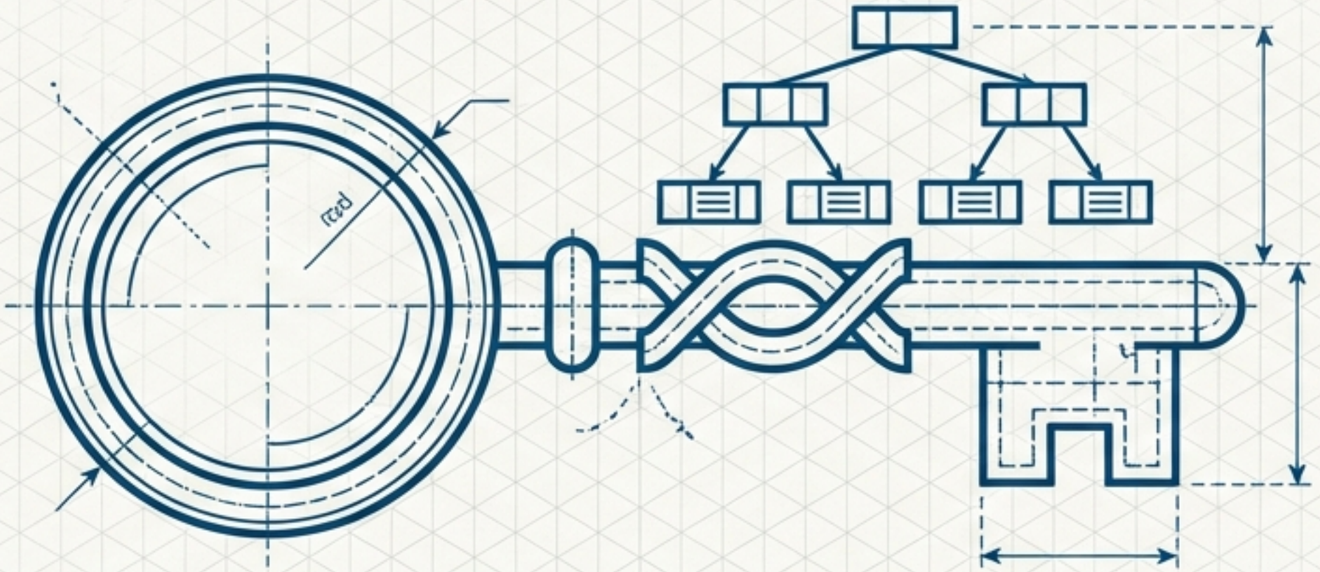


Unlocking High-Performance Search

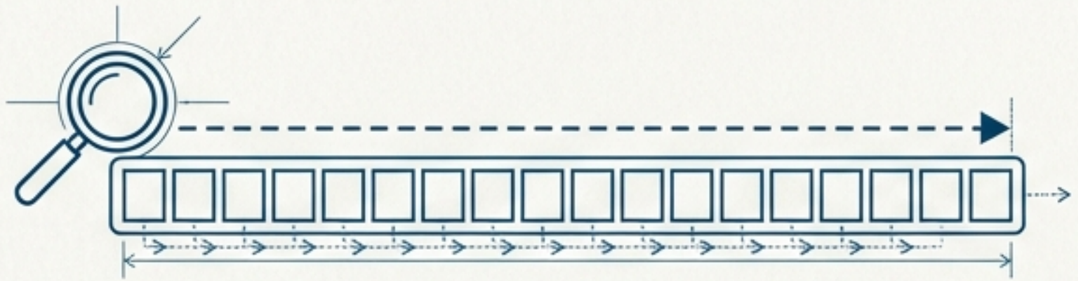
A Practical Guide to MySQL Full-Text Search in Natural Language Mode

Technical Guide / Vol. 42

Moving Beyond the Limitations of `LIKE`

The Problem

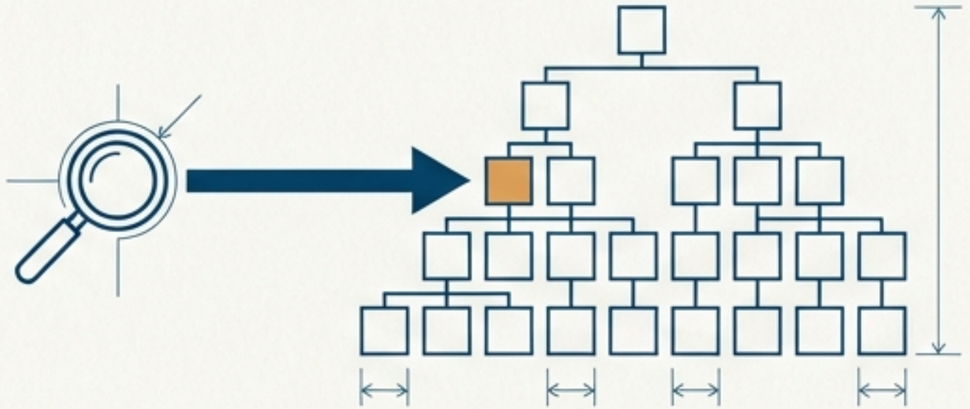
`LIKE '%term%'`



- Performs a full table scan, checking every row.
- Cannot rank results by relevance.
- Struggles with large text datasets.
- Leads to slow, inefficient query performance.

The Solution

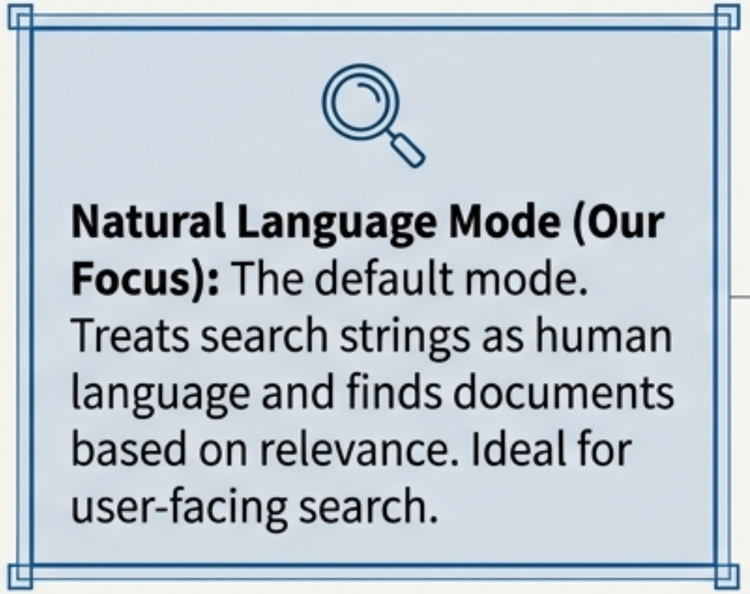
`FULLTEXT` Search



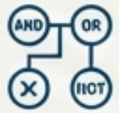
- Uses a specialized index for near-instant lookups.
- Automatically ranks results by relevance.
- Designed for optimal performance at scale.
- The integrated solution for text-heavy applications.

The Core of MySQL Full-Text Search

MySQL FTS is a built-in search capability for efficient, sophisticated text searching directly within the database engine.



Natural Language Mode (Our Focus): The default mode. Treats search strings as human language and finds documents based on relevance. Ideal for user-facing search.



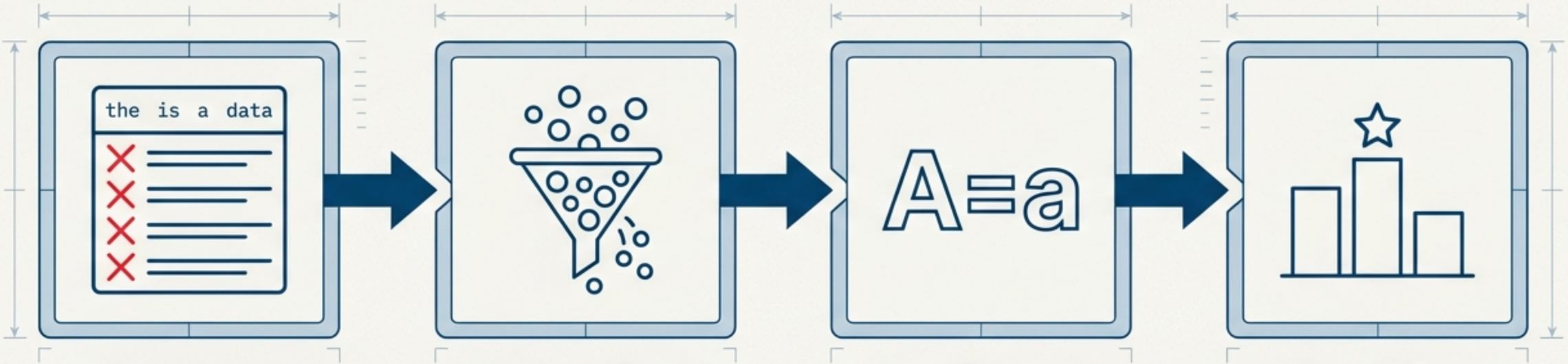
Boolean Mode: Allows for complex queries using operators like `+` (must include), `-` (must not include), and `*` (wildcard).

Query Expansion Mode: A two-pass search that broadens results by including terms found in the most relevant rows from the initial search.

****Key Takeaway*:** Natural Language mode is the ideal choice for user-facing search features where relevance is paramount.

How Natural Language Mode Interprets Your Query

The search string is treated as a natural phrase, and the engine automatically performs these steps:



Removes Stopwords

Common words like 'the', 'and', 'is' are automatically filtered out to improve search quality.

Ignores Short Words

Discards words shorter than the configured minimum length (e.g., 3 characters for InnoDB).

Performs Case-Insensitive Matching

'Search' and 'search' are treated as the same term.

Ranks by Relevance

Returns results sorted by a calculated relevance score, with the best matches appearing first.

Deconstructing Relevance: The TF-IDF Algorithm

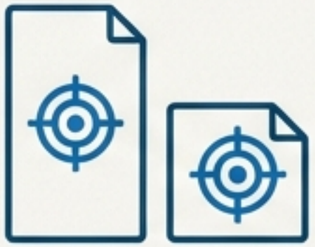
MySQL calculates a relevance score for each document based on these four key factors:

Term Frequency (TF)

How often does the search term appear in a document? *More occurrences = higher relevance.*

Inverse Document Frequency (IDF)

How rare is the term across all documents? *Rarer terms = higher relevance.*



Document Length

How long is the document? *Matches in shorter documents = higher relevance.*

Word Proximity

How close are the search terms to each other? *Terms appearing closer together = higher relevance.*

```sql
SELECT
    title,
    MATCH(title, content) AGAINST('search term') AS relevance
FROM articles
ORDER BY relevance DESC;
```

Creating Your First `FULLTEXT` Index

```
ALTER TABLE your_table ADD FULLTEXT(column1, column2);
```

Supported Column Types

- `CHAR`
- `VARCHAR`
- All `TEXT` variants (TINYTEXT, MEDIUMTEXT, LONGTEXT)

Storage Engine Support

- **InnoDB**: Supported in MySQL 5.6 and later. (Recommended)
- **MyISAM**: Supported in all versions.

Pro Tip

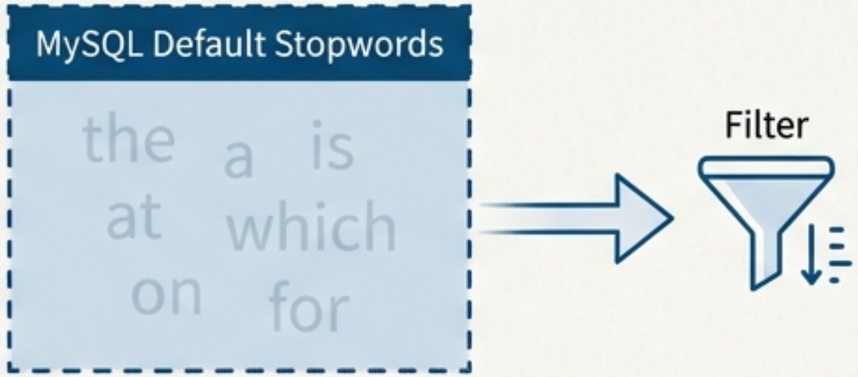
Create targeted indexes on only the columns you intend to search. Avoid including unnecessary columns to minimize index size and maintenance overhead.

Fine-Tuning the Search Engine: Key System Variables

Parameter	Default Value	Description
`innodb_ft_min_token_size`	3 (InnoDB)	The minimum length of a word to be indexed for InnoDB tables.
`ft_min_word_len`	4 (MyISAM)	The minimum length of a word to be indexed for MyISAM tables.
`innodb_ft_max_token_size`	84 (InnoDB)	The maximum length of a word to be indexed for InnoDB tables.
`ft_max_word_len`	84 (MyISAM)	The maximum length of a word to be indexed for MyISAM tables.
`ft_stopword_file`	(built-in list)	Path to a custom file containing words to ignore during indexing.
`innodb_ft_enable_stopword`	ON	Enables or disables stopword filtering for InnoDB tables.

Managing the Vocabulary with Custom Stopword Lists

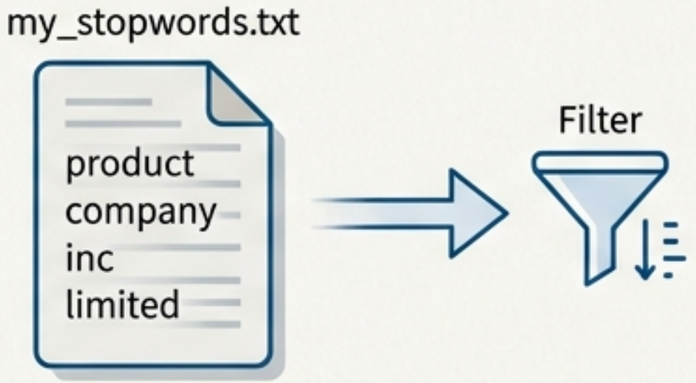
Built-in English Stopwords



MySQL automatically ignores a built-in list of over 500 common English words (e.g., 'the', 'is', 'at', 'which').

Impact: These words are excluded from the index and from search queries, improving performance and relevance.

Creating a Custom List



You can override the default list by creating a simple text file with one stopword per line and pointing the `ft_stopword_file` system variable to it.

Use Case: Essential for domain-specific searches where common words (e.g., 'product', 'company') are not meaningful search terms.

Understanding the Rules of the Road: Key Limitations

The 50% Threshold Rule: A word that appears in more than 50% of the rows is considered too common and is effectively ignored in searches.

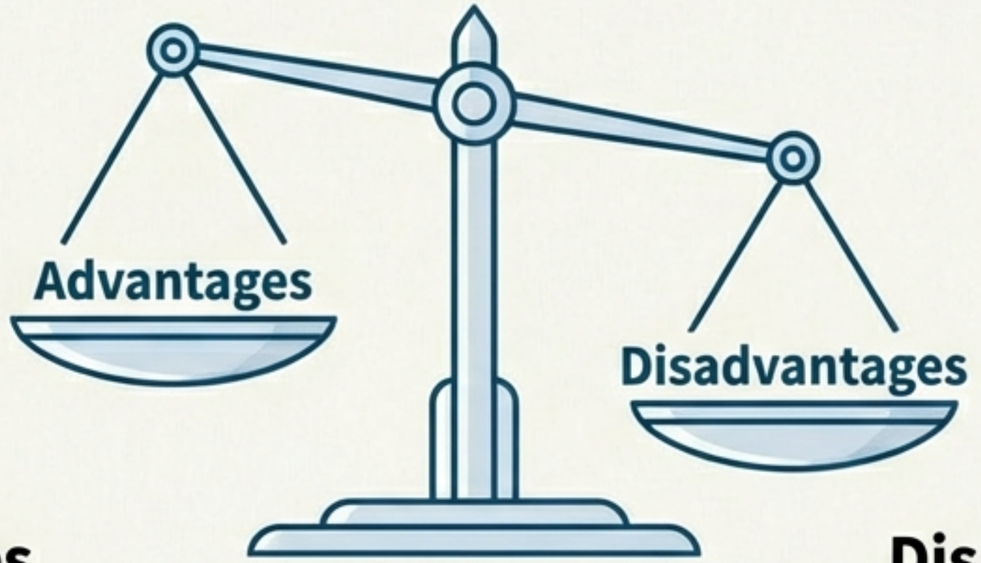
Minimum Row Requirement: FTS indexing and searching requires the table to have at least 3 rows.

No Partial Matching: A search for ‘test’ will *not* match ‘testing’.

Wildcards (*’) are only available in Boolean Mode.

Whole Word Matching: Searches respect word boundaries. A search for ‘cat’ will *not* match ‘catalog’.

The Performance Equation: Advantages vs. Overheads



Advantages

- **Speed**: Significantly faster than `LIKE '%term%'` on large text datasets.
- **Efficiency**: Indexed lookups provide logarithmic time complexity (O(log n)).
- **Built-in Ranking**: Relevance scoring is integrated, eliminating complex application-side logic.

Disadvantages

- **Write Overhead**: Index maintenance adds overhead to `INSERT` and `UPDATE` operations.
- **Disk Space**: `FULLTEXT` indexes require additional disk storage.
- **Rebuild Time**: Rebuilding an index on a very large table can be a time-intensive operation.

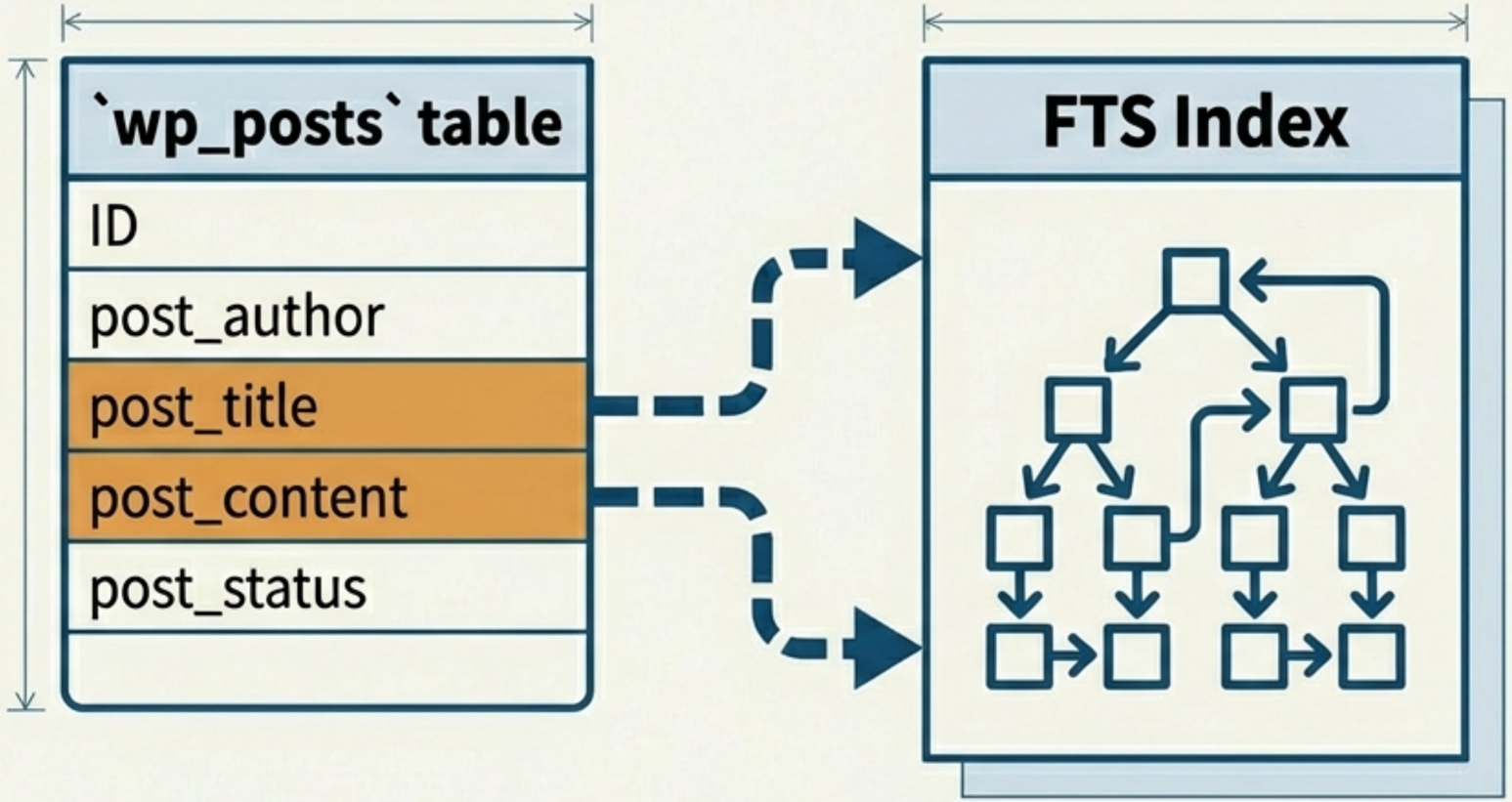
Case Study: Supercharging WordPress Search

The Problem

By default, WordPress search uses inefficient `LIKE` queries on the `wp_posts` table. On content-heavy sites with thousands of posts, this leads to slow search times and irrelevant results.

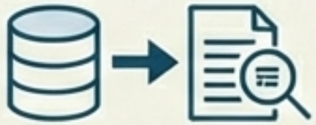
The Goal

Replace the default `LIKE` search with a high-performance `FULLTEXT` search to provide fast, relevance-ranked results.



The Implementation Blueprint for WordPress

1. Add the `FULLTEXT` Index



Create the index on the target columns in the WordPress database.

```
ALTER TABLE wp_posts ADD FULLTEXT (post_title, post_content);
```

2. Create a Custom Search Function

Intercept the default search query using a filter in your theme's `functions.php` file. The key is to modify the `WHERE` clause to use `MATCH() ... AGAINST()`.

3. Integrate Relevance Scoring

Add the relevance score to the `SELECT` list and use it in the `ORDER BY` clause to ensure the best resullts appear first.

```
// In functions.php, modifying the main WordPress query
function my_custom_search_query( $query ) {
    if ( !is_admin() && $query->is_main_query() &&
         $query->is_search() ) {

        // Add relevance score to SELECT
        $query->set('posts_fields', '*, MATCH(post_title,
             post_content) AGAINST(...) as relevance');

        // Modify WHERE clause to use FTS
        $query->set('posts_where', " AND MATCH(post_title,
             post_content) AGAINST(...) ");
        // Order by the new relevance score
        $query->set('orderby', 'relevance');
        $query->set('order', 'DESC');
    }
}
add_action('pre_get_posts', 'my_custom_search_query');
```

WordPress FTS: Advanced Tactics & Considerations

Advanced Tactic: Weighted Search

Concept: Give more importance to matches in the post title than in the content. This simple technique can dramatically improve perceived relevance.

```
SELECT
    ...,
    (MATCH(post_title) AGAINST(...) * 2)
    +
    (MATCH(post_content) AGAINST(...))
    AS relevance
FROM wp_posts
...
```

Critical Performance Considerations for WordPress

Object Caching

Use Redis or Memcached to cache search results and reduce database load on popular queries.

Filter by Post Type

Narrow your search to specific post types (e.g., `'post'`, `'page'`) for better performance.

Filter by Post Status

Always include `WHERE post_status = 'publish'` in your queries to avoid searching drafts and revisions.

Index Maintenance

Be aware that frequent post updates will trigger index updates. Ensure your server can handle the I/O load.

Choosing the Right Search Mode for the Job

Feature	Natural Language	Boolean Mode	Query Expansion
Relevance Scoring	**Yes**	Optional	Yes
Operators (+, −, ∗)	No	**Yes**	No
Auto Stopword Filter	**Yes**	Yes	Yes
Auto Query Refinement	No	No	**Yes**

Use Natural Language Mode When...

- You need general text search with relevance ranking.
- You are building user-facing search bars.
- Query simplicity and intuitive results are key.

Consider Other Modes When...

- You require exact phrase matching with operators.
- You need wildcard searches (`search*`).
- You are building complex, rule-based search systems for technical users.

From Theory to Mastery: Your FTS Checklist

- **Embrace the Index:** FTS is a powerful, indexed alternative to slow `LIKE` queries for any significant text-based searching.
- **Relevance is Your Ally:** Understand the factors of TF-IDF (frequency, rarity, length) to reason about and debug your search results.
- **Configure with Intent:** Adjust `innodb_ft_min_token_size` and custom stopwords to match the specific needs of your application's data.
- **Know the Boundaries:** Be mindful of the 50% rule and the lack of partial matching to avoid unexpected search behavior.

By mastering MySQL FTS, you can build faster, smarter, and more intuitive search experiences directly within your database.