Supabase Fuzzy Search: Detailed Documentation

This document covers two custom RPC functions built on Supabase/Postgres to enable fast, fuzzy, typo-tolerant searching over a large dataset (~400K rows) and type-ahead suggestions. It explains the motivation, schema challenges, implementation details, and usage for each function.

---

1. Background & Schema Challenges

- **Original Problem**: The client-side Fuse.js approach worked for small static datasets but cannot scale to 400,000+ rows—pulling all data into the browser is impossible.
- **Initial Supabase Attempt**: Chaining `.or().ilike()` filters on nine columns returned too many rows, was unindexed, and became slow at scale.
- **Schema Issues**:
    - Column names like `2021_noc` began with digits, requiring quoting.
    - Numeric-type columns (e.g., `noc_priority`) could not be directly trigram-indexed.
    - Separate fields needed to be combined or handled individually for fuzzy matching.

Goal: Offload fuzzy search and suggestion logic to Postgres with trigram indexing, returning only small result sets to the client.

---

2. Full-Table Substring Search RPC (`rpc_search_hot_leads`)

2.1 Purpose

Provide a **complete**, case-insensitive substring search across all rows and all searchable fields, returning every matching row where any column contains the query text.

2.2 Objectives

- **Substring matching**: find exact sequences of characters via `ILIKE '%term%'`.
- **Simplicity**: client invokes `.rpc('rpc_search_hot_leads', { term })` to retrieve all matches.
- **Broad coverage**: search across all specified columns without fuzzy/typo logic.

2.3 Updated Implementation Steps

Define (or replace) the RPC function using `ILIKE` on each field:

```
 CREATE OR REPLACE FUNCTION rpc_search_hot_leads(term text)
  RETURNS TABLE (
    date_of_job_posting text,
    state               text,
    city                text,
    email               text,
    noc_priority        text,
    occupation_title    text,
    "2021_noc"          text,
    job_location        text,
    operating_name      text
  )
LANGUAGE sql
STABLE
AS $func$
  SELECT
    date_of_job_posting::text AS date_of_job_posting,
    state,
    city,
    email,
    noc_priority::text        AS noc_priority,
    occupation_title,
    "2021_noc",
    job_location,
    operating_name
  FROM hot_leads
  WHERE
    date_of_job_posting::text ILIKE '%' || term || '%'
    OR state                  ILIKE '%' || term || '%'
    OR city                   ILIKE '%' || term || '%'
```

```
   OR email                ILIKE '%' || term || '%'
   OR noc_priority::text   ILIKE '%' || term || '%'
   OR occupation_title     ILIKE '%' || term || '%'
   OR "2021_noc"           ILIKE '%' || term || '%'
   OR job_location         ILIKE '%' || term || '%'
   OR operating_name       ILIKE '%' || term || '%';
$func$;
```

1.

Client usage remains unchanged:

```
 const { data, error } = await supabase
  .rpc('rpc_search_hot_leads', { term: query });
```

2.

2.4 Reasoning

- **Exact substring behavior**: users see only rows containing their typed query verbatim.
- **No fuzzy logic**: avoids unintended matches due to typos or partial similarity.
- **Index considerations**: leading `%` prevents B-tree index use; add column-specific GIN–trgm indexes later if performance degrades.

---

3. Deduped Type-Ahead Suggestions RPC (`rpc_search_hot_leads_suggestions`)

(`rpc_search_hot_leads_suggestions`)

3.1 Purpose

Offer **up to 20** unique, fuzzy-matched suggestions from **any** searchable column for a type-ahead UI.

3.2 Objectives

- **Speed**: sub-200ms response on 400K rows.

- **Relevance**: top matches by similarity score.
- **Deduplication**: no duplicate suggestion values across columns.
- **Minimal payload**: small result set (default 20 items).

3.3 Implementation Steps

Create per-column GIN–trgm indexes (if not already present):

```
 CREATE INDEX idx_hot_leads_state_trgm
  ON hot_leads USING gin (state gin_trgm_ops);
... (repeat for each field, casting date/numeric to text)
```

1.

Define a branching CTE function with per-column index scans, branch limits, and a final dedupe:

```
 CREATE OR REPLACE FUNCTION rpc_search_hot_leads_suggestions(
  term    text,
  p_limit int DEFAULT 20,
  branch_lim int DEFAULT 100
) RETURNS TABLE (suggestion text, column_name text)
LANGUAGE sql STABLE AS $func$
WITH b_state AS (... state % term ORDER BY similarity DESC LIMIT branch_lim),
     b_city  AS (...),
     ...
     candidates AS (
       SELECT * FROM b_state
       UNION ALL ...
     )
SELECT suggestion, column_name
FROM candidates
GROUP BY suggestion, column_name
ORDER BY MAX(sim) DESC
LIMIT p_limit;
$func$;
```

2.

Client usage:

```
 const { data: suggestions, error } = await supabase
```

```
.rpc('rpc_search_hot_leads_suggestions', {
  term: query,
  p_limit: 20,
  branch_lim: 100,
});
```

3.
4. **UI integration**:
   - Debounce input (200–300ms)
   - Render dropdown of `{ suggestion, column_name }`
   - On click, fill input and clear suggestions.

3.4 Reasoning

- **Indexed `%` operator**: each branch uses its trigram index for fast pre-filtering.
- **Branch limits**: bounding each index scan to `branch_lim` rows (e.g. 100) yields small candidate sets (~9×100 rows).
- **Deferred dedupe**: `GROUP BY` across ~900 rows is trivial compared to scanning 400K.

3.5 Performance Tips

- Increase `branch_lim` if suggestions are too narrow, or decrease for extra speed.
- Adjust similarity threshold (`%` uses default ~0.30) with `set_limit(...)` in a PL/pgSQL variant if needed.

---

4. Summary

1. **`rpc_search_hot_leads`**: full substring or fuzzy search across all rows, depending on implementation, using either `ILIKE` or trigram `%` + `similarity()` on a concatenated `search_all` column.

2. **`rpc_search_hot_leads_suggestions`**: deduped, capped, multi–column type-ahead suggestions via indexed branches and a final union + grouping.

---

5. Generalizing to Other Tables

You can apply the same pattern to any table in your database. Follow this step-by-step guide to implement both full-search and suggestion RPCs for a new table.

5.1 Identify Your Table & Columns

- **Table name**: e.g. `my_table`.
- **Searchable fields**: list each column you want users to search or get suggestions from, e.g. `col_a`, `col_b`, `col_c`, etc.
- **Data types**: note which columns are non-text (dates, numbers) that need casting to `text`.

5.2 Decide Search Mode

- **Substring search**: use `ILIKE '%term%'` if you want exact substring matches.
- **Fuzzy search**: use `pg_trgm` (`%` operator + `similarity()`) if you need typo tolerance.

5.3 Prep Your Schema

Enable pg_trgm (only for fuzzy):

```
 CREATE EXTENSION IF NOT EXISTS pg_trgm;
```

1.

(Fuzzy-only) Add a `search_all` column:

```
 ALTER TABLE my_table
ADD COLUMN search_all text GENERATED ALWAYS AS (
  coalesce(col_a::text, '') || ' ' || coalesce(col_b, '') || ' ' || ...
) STORED;
```

2.
3. **Create indexes**:
   - **Substring**: optional GIN–trgm indexes on each column to speed up leading-`%` searches.
   - **Fuzzy search_all**: one GIN–trgm index on `search_all`.
   - **Suggestions**: per-column GIN–trgm indexes on each searchable field (cast to `text` where needed).

5.4 Define Your RPC Functions

5.4.1 Full-Search RPC

- **Name**: `rpc_search_<table>`
- **Parameters**: `term text` (+ optional pagination args)
- **Returns**: same columns you want to display.
- **SQL**:
  - **Substring**: OR-chain `col ILIKE '%term%'` conditions.
  - **Fuzzy**: `WHERE search_all % term ORDER BY similarity(search_all, term) DESC`.

```
CREATE OR REPLACE FUNCTION rpc_search_my_table(term text)
  RETURNS TABLE (col_a text, col_b text, col_c text, ...)
LANGUAGE sql STABLE AS $func$
  SELECT col_a::text, col_b, col_c, ...
  FROM my_table
  -- substring example:
  WHERE col_a::text ILIKE '%' || term || '%'
    OR col_b           ILIKE '%' || term || '%'
    OR col_c           ILIKE '%' || term || '%';
$func$;
```

5.4.2 Suggestions RPC

- **Name**: `rpc_search_<table>_suggestions`
- **Parameters**: `term text, p_limit int DEFAULT 20, branch_lim int DEFAULT 100`
- **Returns**: `suggestion text, column_name text`
- **SQL**: a CTE with one branch per column using `col % term ORDER BY similarity() DESC LIMIT branch_lim`, then `UNION ALL`, `GROUP BY suggestion, column_name`, final `ORDER BY MAX(sim) DESC LIMIT p_limit`.

```
CREATE OR REPLACE FUNCTION rpc_search_my_table_suggestions(
  term text, p_limit int DEFAULT 20, branch_lim int DEFAULT 100
)
RETURNS TABLE (suggestion text, column_name text)
LANGUAGE sql STABLE AS $func$
  WITH b_col_a AS (
    SELECT col_a AS suggestion, 'col_a' AS column_name,
           similarity(col_a, term) AS sim
      FROM my_table
     WHERE col_a % term
     ORDER BY sim DESC
     LIMIT branch_lim
  ),
  -- repeat for each column...
  candidates AS (
    SELECT * FROM b_col_a
    UNION ALL SELECT * FROM b_col_b
    -- etc...
  )
  SELECT suggestion, column_name
  FROM candidates
  GROUP BY suggestion, column_name
  ORDER BY MAX(sim) DESC
  LIMIT p_limit;
$func$;
```

5.5 Test Your RPCs

In SQL editor:

```
 SELECT * FROM rpc_search_my_table('foo') LIMIT 10;
SELECT * FROM rpc_search_my_table_suggestions('foo', 20, 100);
```

-
- Verify performance and result correctness.

5.6 Client Integration

Full-search on submit:

```
 const { data, error } = await supabase.rpc('rpc_search_my_table', { term: query });
```

-

Suggestions on keystroke (debounced):

```
 const { data: suggestions } = await supabase
  .rpc('rpc_search_my_table_suggestions', { term: query, p_limit: 20, branch_lim: 100 });
```

-
- Render results or dropdown as needed.

With these steps, you can replicate the pattern for any table—whether you need simple substring matching or advanced fuzzy search with deduped suggestions.