Love this kind of question. Let’s walk through it slowly and cleanly.

I’ll first show a **cleaned-up version** of your Prolog (just fixing syntax), then explain:

- what it *means* in English
- why those conditions matter for a non-clustered index (NCI)
- how Prolog “thinks” with this rule
- what you’d need in the background for it to actually work

---

1. A cleaned-up version of the rule

Prolog is picky about syntax, so this is the same idea, just made valid:

```
/*
Author: Eugene Asahara

Description:
A non-clustered index is a good idea for a table if:
  - the table has more than 10,000 rows, and
  - the index column is not highly dominated by a single value.
*/

nci_table_scan(good, Table) :-
   rows_gt(Table, 10000),
   \+ has_dominant_value(Table, Column).
```

Key fixes (just so you know):

- Table instead of table – uppercase means **variable** in Prolog.
- 10000 instead of 10,000 – numbers can’t have commas.
- \+ instead of /+ – \+ is Prolog’s “not” (negation as failure).
- Column instead of column – again, uppercase for a variable.

Now let’s explain this from the ground up.

2. What does this rule *mean* in English?

Read the :- as **“if”** and commas as **“and”**.

```
nci_table_scan(good, Table) :-
  rows_gt(Table, 10000),
  \+ has_dominant_value(Table, Column).
```

English:

“For some table Table, a non-clustered index (NCI) is **good** if:

- that table has more than 10,000 rows, **and**
- there is **no** dominant value in the index column.”

More conversationally:

- If the table is **big enough** and
- the index column data is **not horribly skewed**
- then an NCI is probably a good idea to reduce table scans.

So this is a **rule of thumb** encoded in Prolog.

3. Quick background: table scans, NCIs, and dominant values

To understand the *why*, here’s the intuition in database terms.

3.1. Table scan vs index seek (very high level)

When you run a query, SQL Server can:

- Do a **table scan** – read **every row** in the table.
- Do an **index seek** – jump into a **sorted structure** and only read relevant rows.

A **non-clustered index (NCI)** is that sorted structure on one or more columns.

- For large tables, if your query is selective (returns a small subset of rows), an NCI can be a *huge* win.
- For tiny tables or low-selectivity predicates, a table scan can be just as fast or faster.

3.2. Why “more than 10,000 rows”?

This is a heuristic baked into your rule:

rows_gt(Table, 10000)

Meaning:

“Only consider an NCI *worth it* if the table has more than 10,000 rows.”

Why?

- For small tables, scanning the entire thing is cheap.
- The **cost of maintaining** the index on every INSERT/UPDATE/DELETE may outweigh the query benefit.
- Past some size (10k is just a convenient threshold), the benefit of being able to seek becomes more convincing.

So this part is encoding:

“If the table is too small, an NCI is usually not worth it.”

3.3. Why “no dominant value”?

\+ has_dominant_value(Table, Column).

This says:

“There should **not** be a dominant value in the index column.”

“Dominant value” means:

- One value appears in **most** of the rows.
- Example: Status = 'Active' in 98% of rows.

What’s the problem?

- If almost **all rows** have the same value in the first index key column, then an index lookup on that value retrieves most of the table anyway.
- That’s a **low-selectivity** column – the index doesn’t help narrow things down.
- SQL Server’s optimizer will say, “If I’m going to read 90% of the table anyway, I might as well just **scan**.”

So:

- The NCI **isn’t used much**, because it doesn’t filter well.
- But you still pay to **maintain** that index on every write.

That’s why your rule says:

“Good NCI = big table **and** not dominated by one major value.”

---

4. How Prolog executes this rule

Prolog doesn’t “run” like a normal imperative program.
It answers questions (queries) using facts and rules.

4.1. The rule is a definition, not an action

Your rule:

```
nci_table_scan(good, Table) :-
   rows_gt(Table, 10000),
   \+ has_dominant_value(Table, Column).
```

...is a **logical statement** saying:

nci_table_scan(good, Table) is true **if** the conditions are true.

That’s it. No loops, no mutation.

4.2. The query you’d run

Example query:

```
?- nci_table_scan(good, orders).
```

This asks:

“Is it true that an NCI is good for the table orders?”

Prolog will try to prove this by checking the body:

1. rows_gt(orders, 10000) – do we know the orders table has > 10,000 rows?
2. \+ has_dominant_value(orders, Column) – can we *fail* to prove that the index column has a dominant value?

If both are satisfied, the query succeeds, and Prolog answers:

```
true.
```

If not, you get:

```
false.
```

4.3. Where do rows_gt/2 and has_dominant_value/2 come from?

You would define these as **facts** (or as rules derived from a CSV):

```
% Example facts

rows_gt(orders, 10000).

has_dominant_value(customers, status).   % 'status' is super skewed
```

Then:

```
?- nci_table_scan(good, orders).
```

- rows_gt(orders, 10000) → found (true)
- \+ has_dominant_value(orders, Column)
  - Prolog tries to prove has_dominant_value(orders, Column)
  - Fails (no such fact)
  - So \+ ... succeeds
    ⇒ Answer: true.

But:

```
?- nci_table_scan(good, customers).
```

- rows_gt(customers, 10000) → suppose true
- \+ has_dominant_value(customers, Column)
  - Now has_dominant_value(customers, status) **does exist**
  - So Prolog can prove it
  - Therefore \+ has_dominant_value(customers, Column) fails
    ⇒ Answer: false.

So **Prolog is using your rule as a filter**:

- Tables that are large and **not** skewed → "good" for NCI.
- Large tables with skew → "not good" per this heuristic.

---

5. What each piece "is" in Prolog terms

Let's annotate your rule line by line:

```
nci_table_scan(good, Table) :-

   rows_gt(Table, 10000),
  \+ has_dominant_value(Table, Column).
```

- nci_table_scan/2 – a **predicate** with 2 arguments (arity 2): quality, table name.
- good – a **constant atom** describing the outcome (you could also have bad, neutral, etc.).
- Table – a **variable**, representing some table.
- :- – means **“if”** (rule). “Head if Body.”
- rows_gt(Table, 10000) – another predicate; you’ll assert or derive this.
- \+ – negation as failure (Prolog’s “not”). “It is not provable that…”
- has_dominant_value(Table, Column) – a predicate meaning “the distribution of values on the index column for this table is badly skewed”.

The commas:

A,

B,

C.

mean **A AND B AND C must all be true**.

---

6. Where this fits in your bigger story

Conceptually, this rule is:

- a **captured DBA heuristic**
- written in a **logic language** (Prolog)
- that you can later:
  - **generate from CSV** (your “construct Prolog from metadata” function),
  - **graph visually** in your KG / semantic network diagrams,
  - and use as an example of Prolog expressing something that’s **messy in a flowchart** or **painful in RDF/SWRL**.

It’s a wonderful example because:

- The intuition is easy: “Big table + non-skewed column ⇒ index is good.”

- The Prolog rule is **short and readable**, but the underlying database behavior is quite nuanced.

If you'd like, next step could be:

- I write a tiny Prolog "knowledge base" with sample tables and show actual queries and results.
- Or we build the **CSV → Prolog** generator for rows_gt/2 and has_dominant_value/2 in your pipeline.