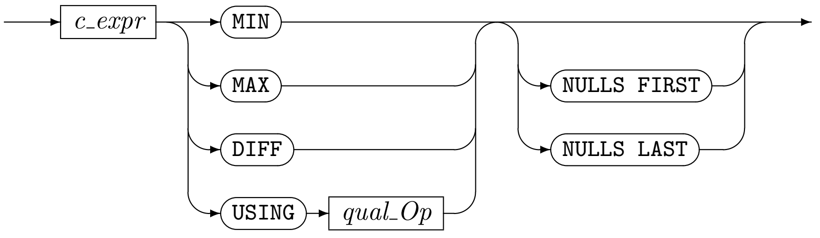
In the *target_list* of the *select_clause* and in many other places *a_expr* is used. This is the concept of a general expression and the heart of the expression syntax. In certain places *b_expr* is used, which is a restricted subset to avoid shift/reduce conflicts.

c_expr contains all the productions that are common to *a_expr* and *b_expr*. Hence *c_expr* is factored out to eliminate redundant coding, but in our case we use it directly to avoid shift/reduce conflicts. Please note that `'('` *a_expr* `')'` is a *c_expr*, so an unrestricted expression can always be used by surrounding it with parenthesis. Furthermore *c_expr* contains the most common case, a column reference (*columnref*), anyway and also function calls can be used without an extra pair of parenthesis.

skyline_of_expr



The syntax we have described so far deals only with specifying the skyline query itself and the ultimate goal of our work is building a skyline query optimizer to automatically generate a good query plan w.r.t. I/O, time, and memory consumption and decide all operational aspects of the query evaluation. It is well known that the cost estimation of the skyline queries is a non-trivial task [Chaudhuri et al., 2006] since the performance of a skyline query is sensitive to a number of parameters [Godfrey et al., 2007].

To be able to study different query plans, different physical operators, and various options for the physical operators we introduced a set of options. This options list is preluded by the keyword `WITH`.

Except for the order of the tuples in the output all of the following options do not have an impact on the result of the skyline computation. Using either BNL or SFS will change the order of tuples in the output, as SFS does a sort prior to skyline computation. Furthermore the size of the tuple window and the tuple window policy have an impact on the order of tuples. And even the elimination filter and its tuple window size have an (indirect) impact on the output order, as a different amount of potential skyline tuples reaches the final skyline computation node, although the elimination filter does not change the relative order. See detailed discussion on relative tuple order preserving property in section 4.7.2. To enforce a specific order, if desired, sort the query result by means of an appropriate `ORDER BY`-clause. We did so in regression testing, we ordered on the unique `ID` column, no matter what method and what options used, the results are the same. If not this indicates a flaw in the implementation.

The options fall into three groups: elimination filter (`EF`), physical operator (`BNL`, `SFS`, etc.), and access path selection (`NOINDEX`) and associated options if applicable.

When the skyline option `EF` is present the query planner will include an elimination filter (EF) in the query plan. If the *efwindowoptions* are omitted, the query planner uses the defaults and plans the elimination filter with an 8 KB tuple window and the tuple window policy `APPEND`. The 8 KB tuple window size stems from PostgreSQL page size `BLCKSZ`, which is 8 KB on most platforms.

The next group of options is used to specify the main algorithm used for skyline computation. Currently the work horses of our skyline operator implementation are the BNL and the SFS algorithm. BNL is the default in the query optimizer. If a suitable index access path (i.e. index) is available SFS is selected. This default procedure can be overruled by specifying `BNL` or `SFS` as an option. Furthermore the size and the tuple placement policy for