HASH JOIN pseudo-code *(high-level, simplified perspective)*

```
HASH JOIN
 CHILD_ROW_SOURCE_1  ← driving/build row source, or “left” input   alias: $r_1$  columns: $(c_1, c_2, \ldots, c_n)$
 CHILD_ROW_SOURCE_2  ← probe row source, or “right” input           alias: $r_2$  columns: $(c_1, c_2, \ldots, c_m)$
```

with join conditions as follows:

$$\left.\begin{array}{ll} & r_1.c_{h_1} = r_2.c_{j_1} \\ \text{and} & r_1.c_{h_2} = r_2.c_{j_2} \\ \ldots & \\ \text{and} & r_1.c_{h_k} = r_2.c_{j_k} \end{array}\right\} \textit{equality conditions}$$

$$\left.\begin{array}{ll} \text{and} & \textit{expression}(\, r_1.c_{h_{k+1}}, \ldots,\, r_1.c_{h_p} \\ & \quad ,\, r_2.c_{j_{k+1}}, \ldots,\, r_2.c_{j_q} \,) \end{array}\right\} \textit{non-equality conditions}$$

```
Start CHILD_ROW_SOURCE_1
For each row $r_1 = (c_1, c_2, \ldots, c_n)$ from CHILD_ROW_SOURCE_1 Loop   -- build loop
    insert $r_1$ into the hash table using $(r_1.c_{h_1}, \ldots, r_1.c_{h_k})$ as the hash key
End loop  -- CHILD_ROW_SOURCE_1 has been fully processed
If CHILD_ROW_SOURCE_1 returned at least 1 row Then
    Start CHILD_ROW_SOURCE_2
    For each row $r_2 = (c_1, c_2, \ldots, c_m)$ from CHILD_ROW_SOURCE_2 Loop   -- probe loop
        For each row $r_1$ matching $(r_2.c_{j_1}, \ldots, r_2.c_{j_k})$ in the hash table /* access conditions */ Loop
            /* evaluate non-equality conditions: filter conditions */
            If $\textit{expression}(r_1.c_{h_{k+1}}, \ldots, r_1.c_{h_p}, r_2.c_{j_{k+1}}, \ldots, r_2.c_{j_q})$ is true Then
                Yield the combined row $rj = (r_1.c_1, \ldots, r_1.c_n, r_2.c_1, \ldots, r_2.c_m)$ to the parent operation (*)
            End If
        End Loop
    End Loop
End If
```

(*) Actually, only projected columns are passed to the parent operation

Key points:

- CHILD_ROW_SOURCE_1 and _2 are started only once (per start of the parent), and processed independently, in turn
- The hash table (in workarea) is built from CHILD_ROW_SOURCE_1: rows from CHILD_ROW_SOURCE_2 are not buffered (*iff* the hash join can be processed fully in memory)
- The hash key is formed of equi-joined columns; non-equality join conditions are always used as *filter* conditions, and evaluated by *iterating* on rows matching the probe key in the hash table—if there are too many such rows, a lot of CPU time could go into that
- The optimizer may swap join inputs, depending on (estimated) memory requirements of using either as the build row source