# Semantic Analysis

★ Symbol Table

- Structure

- Global scope: stores table names loaded with "load", names created by "filter", "map", "aggregate", global variables, and built-in functions (avg, sum, count)
- Block scope: stores "row" identifier in "for row in table {...}", temporary column names assigned inside blocks (e.g. bonus = salary * 0.10) "where" expression variables, and function-call local names

- Global scope example using our example program

| | Name | Kind | Type | Additional Info |
|---|---|---|---|---|
| 1 | employees | table | table {name: string, salary: number, dept: string, id: number} | Loaded from CSV |
| 2 | high_salary | table | table {name: string, salary: number, dept: string, id: number} | From filter |
| 3 | bonus_calc | table | table {name: string, salary: number, dept: string, id: number, bonus: number} | From map |
| 4 | stats | table | table {avg_salary: number, avg_bonus: number} | From aggregate |
| 5 | avg | function | (number → number) | Built-in |
| 6 | Sum | function | (number → number) | Built-in |
| 7 | count | function | (any → number) | Built-in |

- Block scope using "filter" block

| | Name | Kind | Type | Additional Info |
|---|---|---|---|---|
| 1 | name | column | string | from "employees" |
| 2 | salary | column | number | from "employees" |
| 3 | dept | column | string | from "employees" |
| 4 | id | column | number | from "employees" |

Result: entry 2 in global symbol table

Table Construction Rules

- Table Loading:
  <load_stmt> ::= "load" IDENTIFIER "from" STRING_LITERAL
  action: insert(IDENTIFIER.name, table(schema_from_CSV(STRING_LITERAL)))
- Filter:
  <filter_stmt> ::= "filter" IDENTIFIER <block>
  action: insert(IDENTIFIER.name, table(input_table.schema))
  No modifications to schema
- Map:
  <map_stmt> ::= "map" IDENTIFIER "on" IDENTIFIER <block>
  action: insert(IDENTIFIER.name, table(<block>.schema))
  Inside map block:
  <assign_stmt> ::= IDENTIFIER "=" <expr>
  action: insert_or_update(IDENTIFIER.name, <expr>.type)
  Each <assign_stmt> adds a column to the output schema
- Aggregate:
  <aggregate_stmt> ::= "aggregate" IDENTIFIER "on" IDENTIFIER <block>
  action: insert(IDENTIFIER.name, table(<block>.results))

★ Type Checking Rules

- Arithmetic

| | |
|---|---|
| $T(E1 + E2) = number$ | if $T(E1) = number$ and $T(E2) = number$ |
| $T(E1 - E2) = number$ | if $T(E1) = number$ and $T(E2) = number$ |
| $T(E1 * E2) = number$ | if $T(E1) = number$ and $T(E2) = number$ |
| $T(E1 / E2) = number$ | if $T(E1) = number$ and $T(E2) = number$ |

- Relational Operators

| | |
|---|---|
| $T(E1 > E2) = bool$ | if $T(E1) = number$ and $T(E2) = number$ |
| $T(E1 < E2) = bool$ | if $T(E1) = number$ and $T(E2) = number$ |
| $T(E1 >= E2) = bool$ | if $T(E1) = number$ and $T(E2) = number$ |
| $T(E1 <= E2) = bool$ | if $T(E1) = number$ and $T(E2) = number$ |
| $T(E1 == E2) = bool$ | if $T(E1) = T(E2)$ |

- Dot Access

  <primary> ::= $IDENTIFIER_1$ "." $IDENTIFIER_2$

  {primary.type := schema_of ($IDENTIFIER_1$) . field_type ($IDENTIFIER_2$)}

- Assignment

  <assign_stmt> ::= IDENTIFIER "=" <expr>

  {IDENTIFIER.type := <expr>.type}

- Aggregate Functions

  avg (number) → number

  sum (number) → number

  count (any) → number

- Print : All types valid.