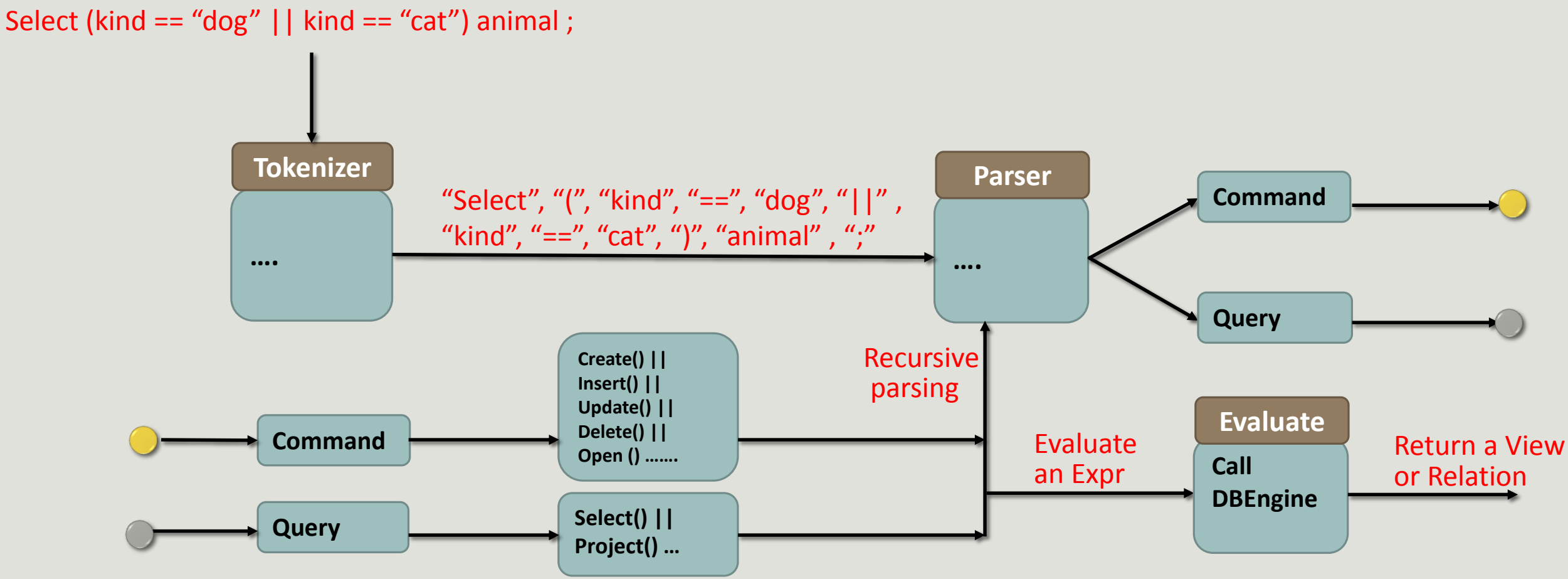
CSCE 315

TEACHING ASSISTANT: JAY CHEN JAYCHEN@CSE.TAMU.EDU

Project 1: **Database Management System**

- Design Document
- DB Engine
- Lexical Parser
- DB Engine and Lexical Parser Integration
- DB Application and integration

Recursive Descent Parser



Recursive Descent Parser

TokenList:

“Select” “(” “kind” “==” “dog” “||” “kind” “==” “cat” “)” “animal” “;”

Grammers:

query::=

command::=

selection::=

projection::=

….

Process the token list from left to right. (Keep a pointer)

Create one function to handle each grammar.

- Follow the grammar rule to “consume” the items in token list one by one. (Advance the pointer upon a successful match)

Grammars are defined recursively, so one grammar function may call many other functions.

Each grammar function returns an error and terminates the parser if any token fails to match the grammar

Parser example

Query ::= relationname <- expr ;
Command ::= (open-cmd | clos-cmd | save-cmd…….);
Insert-cmd ::=….
Relationname ::= identifier
identifier ::= alpha { (alpha | digit) }

animals		
Spot	Dog	10
Snoopy	Dog	3
Tweety	Bird	1
Joe	Cat	4

Some Parser Function Examples:

- par_line():
  - Match the first item in the token list and determine weather this is a command or a query.
  - Call functions par_command() or par_query();
  - After either par_command() or par_query() returns, make sure the line ends properly with ";" token
- par_command():
  - Match the token list to see if the current token matches any command. It there is a match, call that command's grammar function.
- par_query():
  - Match the token list to see if the current token matches any query. It there is a match, call that query's grammar function. Return if there is no match

Parser example

INSERT INTO relationname VALUES FROM (literal { , literal });

- Suppose the par_query() match a "**Insertion-cmd**" and call the par_insert() grammar function
- par_insert():
  - try to match the token list with the **Insertion** grammar.
  - call other grammar functions par_relationName() and par_literals()
  - (Later on, you also need to call dbEngine from here to actually inset a tuple)
- par_relationName():
  - try to match the current pointer in the token list with the **RelationName** grammar.
  - extract a relationName (a String) from the current token list
- par_literals():
  - match literal grammar and extract one or several String

Parser example

select (condition) atomicexpr;
- Select (kind == “dog”) project (kind, years) animal ;

- Suppose the par_query() match a “**select-query**” and calls the par_select() grammar function
- par_select():
  - try to match the token list with the **Selection** grammar.
  - call other grammar functions par_condition() and par_atomicexpr().
  - (Later on, you also need to call dbEngine here to actually find tuples and return a View)
- par_condition():
  - try to match the current pointer in the token list with the **Condition** grammar.
  - call par_conjunction()
- par_conjunction():
  - try to match the current pointer in the token list with the **Conjunction** grammar
  - Call par_comparison()

Parser example

select (condition) atomicexpr;
atomicexpr::= relationname| (expr)
expr ::= atomicexpr | selection | projection | renaming …

- Select (kind == “dog”) (project (kind, years) animal);

- Suppose the par_query() match a “**select-query**” and calls the par_select() grammar function
- par_automicexpr():
  - try to match the current pointer in the token list with the **atomic-expr** grammar.
  - If an Expr is identified, call par_expr() grammar function
- par_expr():
  - try to match the current pointer in the token list with the **expr** grammar.
  - Check the current token and identify a proper query (e.g., if project-query is identified, call par_project() grammer function).

par_projection():

- try to match the current pointer in the token list with the **projection** grammar
- Call grammar function par_attribute_list
- (later on, return a View after the evaluation)