AREL

for **computer scientists**
and **software writers**

AREL

for **computer scientists**
and **software writers**

HI, I'M
JOSHUA CODY

@JPCODY

USER EXPERIENCE ENGINEER
@ ACADEMICWORKS

AWS
CHEF
RAILS
REDIS
POSTGRES
CASSANDRA
ELASTICSEARCH

MAKING A BETTER WORLD

I. WHAT'S AN AREL?

II. WHEN SHOULD I USE IT?

III. HOW DOES IT WORK?

IV. BUT ISN'T THE CODE MESSY?

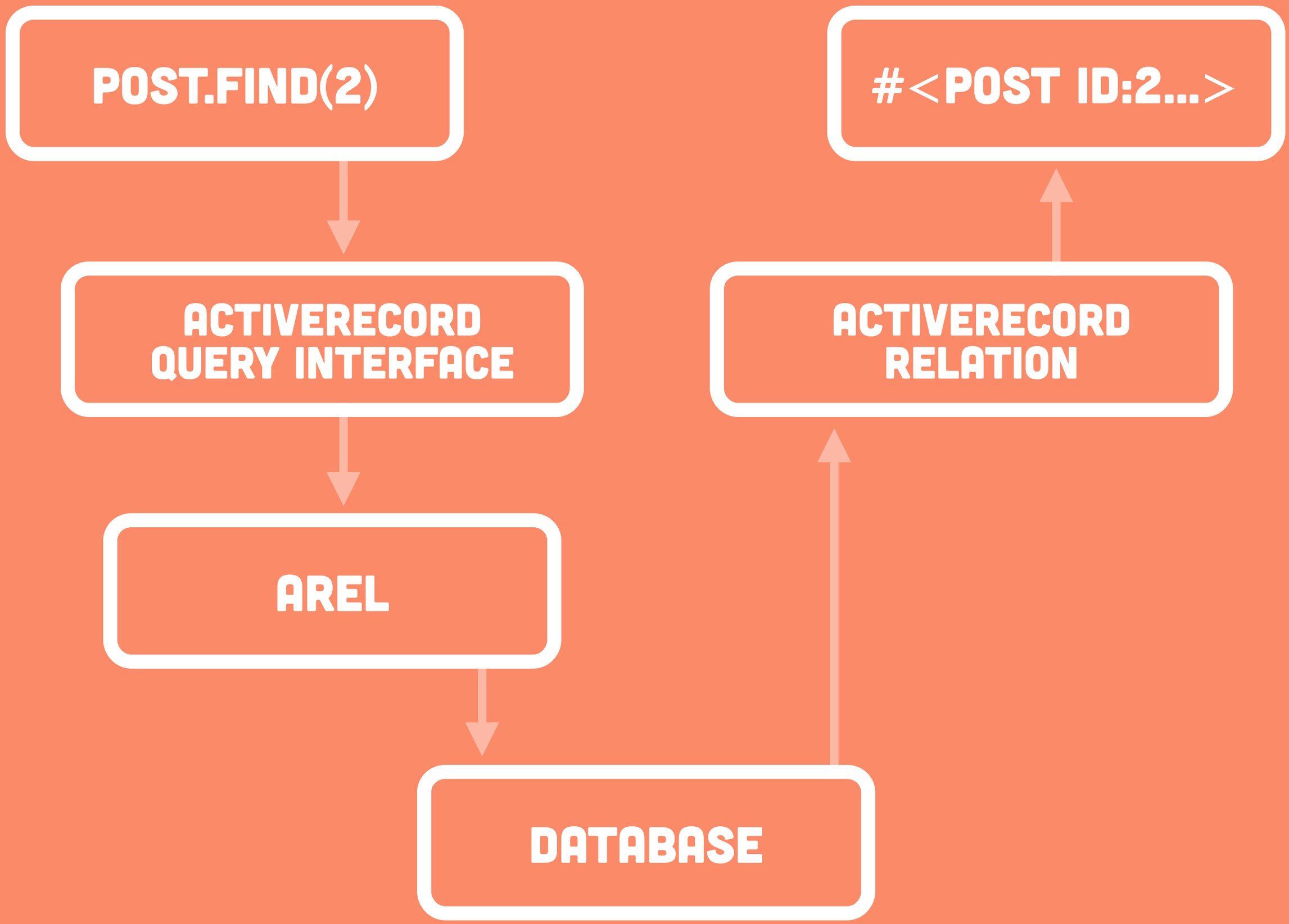
AREL (A RELATIONAL ALGEBRA) IS AN ABSTRACT SYNTAX TREE MANAGER FOR SQL, IMPLEMENTED WITH THE VISITOR PATTERN.

AREL ~~(A RELATIONAL ALGEBRA) IS AN ABSTRACT SYNTAX TREE MANAGER FOR SQL, IMPLEMENTED WITH THE VISITOR PATTERN.~~

PROGRAMATICALLY GENERATES SQL.

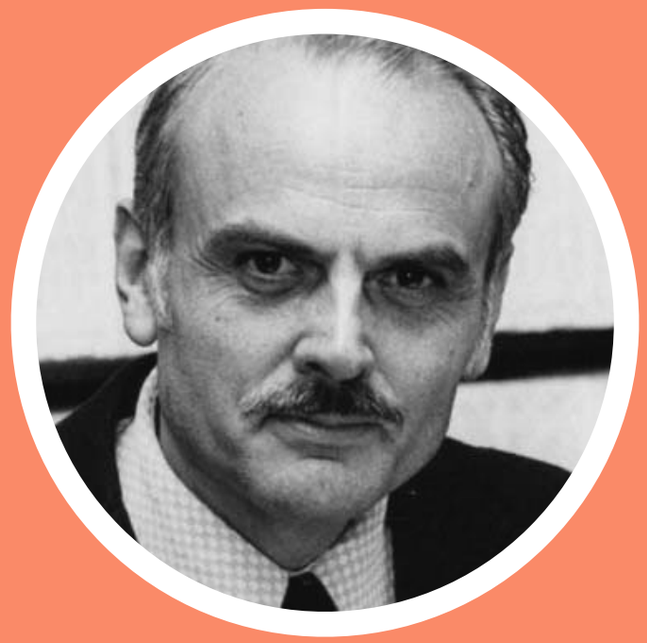
AREL (A RELATIONAL ALGEBRA) IS NOT ACTIVERECORD::RELATION

WE'LL COME BACK TO THIS.


POST.FIND(2)
ACTIVERECORD
QUERY INTERFACE
AREL
DATABASE
ACTIVERECORD
RELATION
#<POST ID:2...>

AREL (A RELATIONAL ALGEBRA) IS AN ABSTRACT SYNTAX TREE MANAGER FOR SQL, IMPLEMENTED WITH THE VISITOR PATTERN.

HI, HE'S
EDGAR FRANK CODD

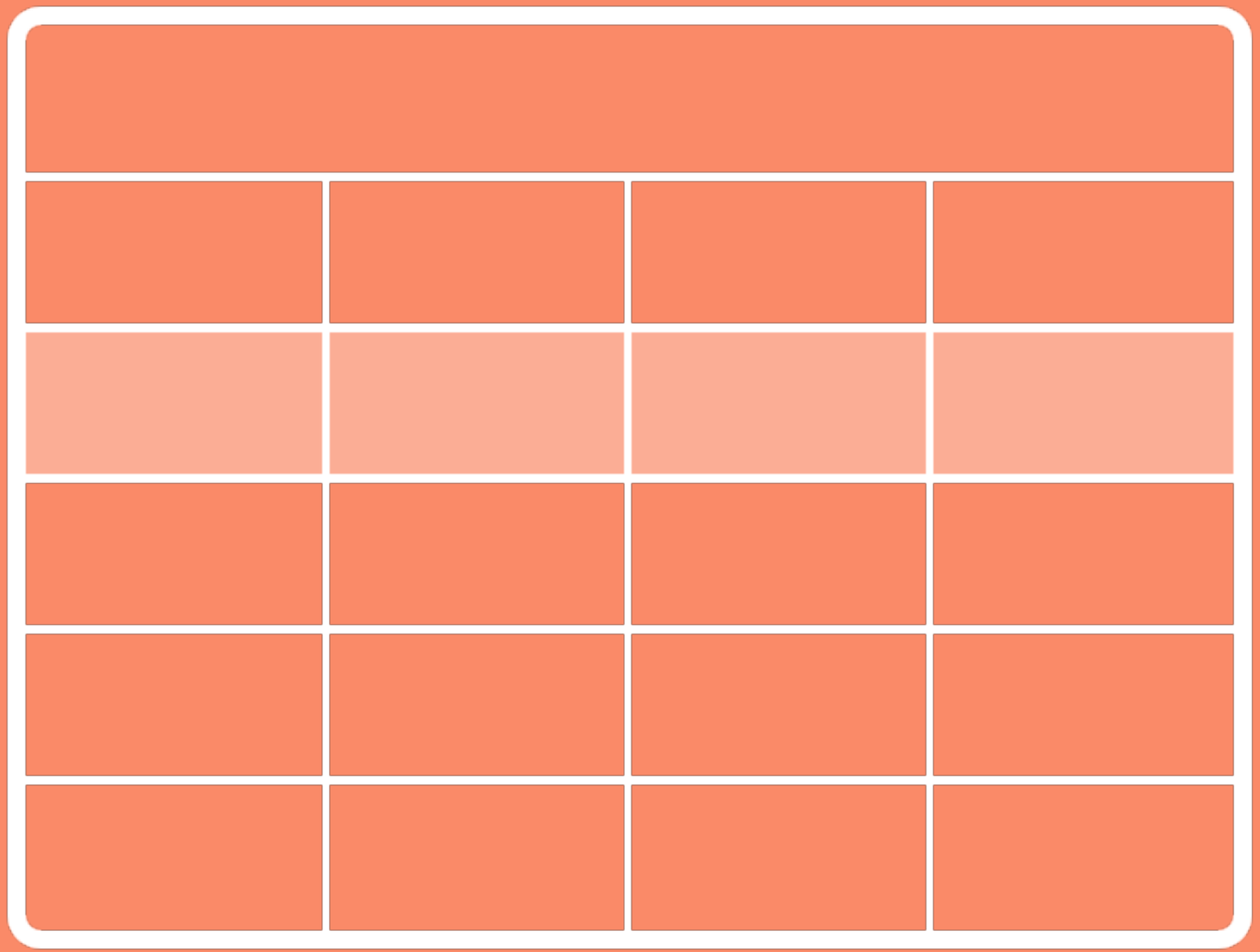


A RELATIONAL MODEL OF DATA FOR LARGE SHARED DATA BANKS - 1970

“THIS PAPER IS CONCERNED WITH THE APPLICATION OF ELEMENTARY RELATION THEORY TO SYSTEMS WHICH PROVIDE SHARED ACCESS TO LARGE BANKS OF FORMATTED DATA.”

PARAPHRASE:

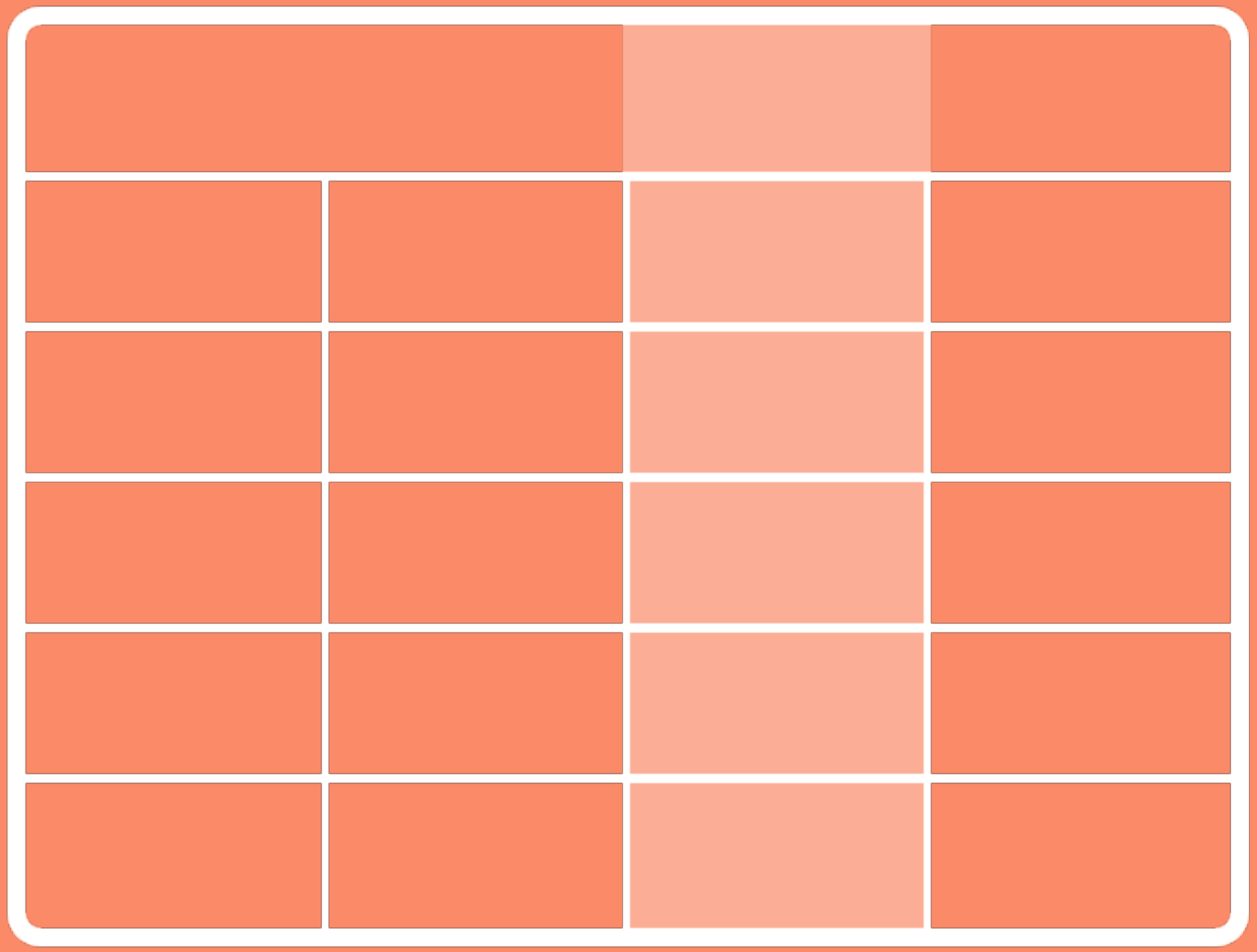
THIS PAPER APPLIES SET ALGEBRA TO THIS NEW THING I'M INVENTING: RELATIONAL DATABASES.



ELEMENTS (ROWS)

```
[[a,  b,  c,  d],
 [e,  f,  g,  h],
 [i,  j,  k,  l],
 [m,  n,  o,  p],
 [q,  r,  s,  t]]
```

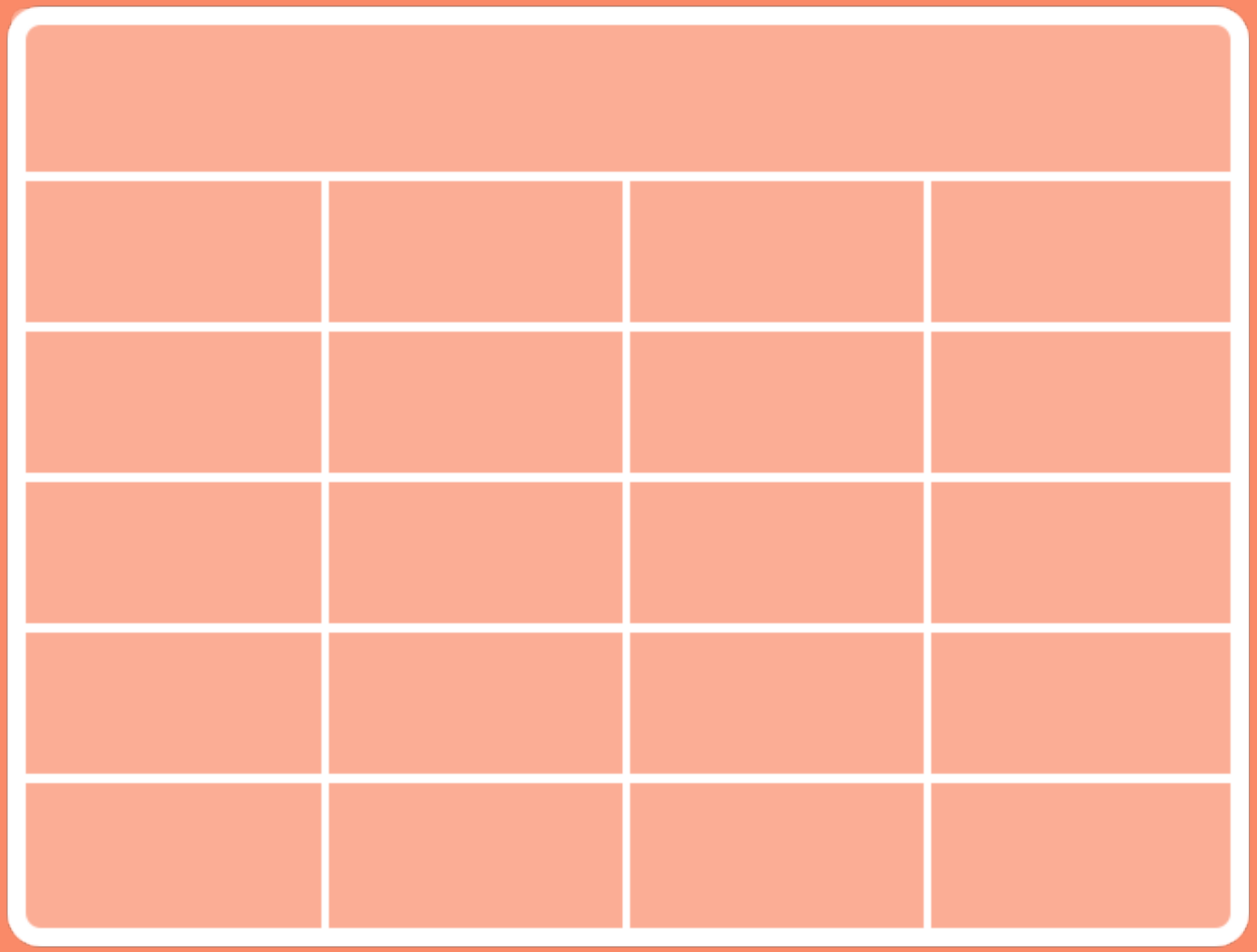
ELEMENTS (ROWS)



DOMAINS (COLUMNS)

[[a,	b,	c,	d],
[e,	f,	g,	h],
[i,	j,	k,	l],
[m,	n,	o,	p],
[q,	r,	s,	t]]

DOMAINS (COLUMNS)



RELATIONS (TABLES)

[[a,	b,	c,	d],
[e,	f,	g,	h],
[i,	j,	k,	l],
[m,	n,	o,	p],
[q,	r,	s,	t]]

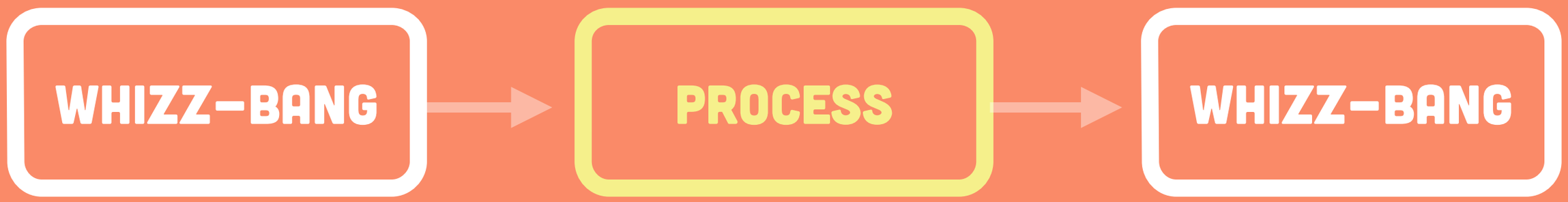
RELATIONS (TABLES)

SELECTION ...WHERE

PROJECTION SELECT *.ID

JOIN JOIN

CLOSURE UNDER COMPOSITION (CHAINING)



WE'LL COME BACK TO THIS.

WHAT'S AN AREL?

AN IMPLEMENTATION OF RELATIONAL ALGEBRA, USED FOR PROGRAMATICALLY GENERATING SQL QUERIES.

I. WHAT'S AN AREL?

II. WHEN SHOULD I USE IT?

III. HOW DOES IT WORK?

IV. BUT ISN'T THE CODE MESSY?

```
Bean.where(roasted_at: Date.today)
```

```
Bean.where("roasted_at >= ?", 3.days.ago)
```

SIMPLE QUERIES

*

(TIME + BUSINESS)

=

COMPLEX QUERIES

```
Bean.
  joins(roaster: :city).
  joins("JOIN ratings bean_ratings
           ON bean_ratings.type='Bean'
          AND bean_ratings.target_id=beans.id").
  joins("JOIN ratings roaster_ratings
           ON roaster_ratings.type='Roaster'
          AND roaster_ratings.target_id=roasters.id").
  where("roasted_at >= ?", 3.days.ago).
  where("roasters.city_id != ?", 5).
  where("flavors ILIKE ? OR flavors IS NULL", "%sweet%").
  having("avg(bean_ratings.value)    >= ? OR
          avg(roaster_ratings.value) >= ? OR
          count(bean_ratings.id)       = ? OR
          count(roaster_ratings.id)    = ?", 90, 90, 0, 0).
  group("beans.id", "roasters.name").
  order("roasters.name DESC")
```

```
    SELECT "beans".*
      FROM "beans"
INNER JOIN "roasters"
        ON "roasters"."id" = "beans"."roaster_id"
INNER JOIN "cities"
        ON "cities"."id" = "roasters"."city_id"
INNER JOIN ratings bean_ratings
        ON bean_ratings.type='Bean'
        AND bean_ratings.target_id=beans.id
INNER JOIN ratings roaster_ratings
        ON roaster_ratings.type='Roaster'
        AND roaster_ratings.target_id=roasters.id
   WHERE (roasted_at >= '2014-07-24 03:55:56.285048')
     AND (roasters.city_id != 5)
     AND (flavors ILIKE '%sweet%' OR flavors IS NULL)
GROUP BY beans.id, roasters.name
  HAVING    avg(bean_ratings.value) >= 90
        OR avg(roaster_ratings.value) >= 90
        OR count(bean_ratings.id) = 0
        OR count(roaster_ratings.id) = 0
ORDER BY roasters.name DESC
```

SQL

UNIVERSAL

ACTIVERECORD

SIMPLE

AREL

POWERFUL

SQL

MISMATCHED MODEL

ACTIVERECORD

LEAKY ABSTRACTION

AREL

YAQL

```
arel = Bean.
  arel_table.project(Bean.arel_table[Arel.star]).
  join(Roaster.arel_table).on(Roaster.arel_table[:id].eq(Bean.arel_table[:roaster_id])).
  join(City.arel_table).on(City.arel_table[:id].eq(Roaster.arel_table[:city_id])).
  join(Rating.arel_table.alias("bean_ratings")).on(
    Arel::Table.new(:bean_ratings)[:type].eq("Bean"),
    Arel::Table.new(:bean_ratings)[:target_id].eq(Bean.arel_table[:id])
  ).
  join(Rating.arel_table.alias("roaster_ratings")).on(
    Arel::Table.new(:roaster_ratings)[:type].eq("Roaster"),
    Arel::Table.new(:roaster_ratings)[:target_id].eq(Roaster.arel_table[:id])
  ).
  where(Bean.arel_table[:roasted_at].gteq(3.days.ago)).
  where(Roaster.arel_table[:city_id].not_eq(5)).
  where(
    Bean.arel_table[:flavors].matches("%sweet%").or(
    Bean.arel_table[:flavors].eq(nil))
  ).
  having(
    Arel::Table.new(:bean_ratings)[:value].average.gteq(90).or(
    Arel::Table.new(:roaster_ratings)[:value].average.gteq(90).or(
    Arel::Table.new(:bean_ratings)[:id].count.eq(0).or(
    Arel::Table.new(:roaster_ratings)[:id].count.eq(0))))
  ).
  group(
    Bean.arel_table[:id],
    Roaster.arel_table[:name]
  ).
  order(Roaster.arel_table[:name].desc).to_sql

Bean.find_by_sql(arel)
```

WHEN SHOULD I USE IT?

WHEN YOUR BESPOKE QUERIES ARE GETTING TACKY, YOU'RE GETTING REPETITIVE, OR YOU NEED MORE COMPOSABILITY.

I. WHAT'S AN AREL?

II. WHEN SHOULD I USE IT?

III. HOW DOES IT WORK?

IV. BUT ISN'T THE CODE MESSY?

BY EXPOSING METHODS
TO BUILD AN
ABSTRACT SYNTAX TREE,
THEN USING
THE VISITOR PATTERN
TO GENERATE SQL STRINGS.

PREDICATIONS

IN MATCHES LT GT_EQ IN_ANY MATCHES_ALL
LT_EQ_ALL NOT_EQ_ANY NOT_IN EQ

AGGREGATES

COUNT SUM MAXIMUM AVERAGE MINIMUM

MANAGEMENT

DESC ASC AS SKIP LIMIT ON GROUP HAVING TAKE
DISTINCT UNION ORDER OR AND

MORE

NAMED FUNCTIONS · COMMON TABLE EXPRESSIONS
INLINE MATH · OUTER JOINS

ABSTRACT SYNTAX TREES

```
total = (a + b) < 5 ? (a + b) : 5
```

CODE STRING

↓

TOKENIZATION

PARSING

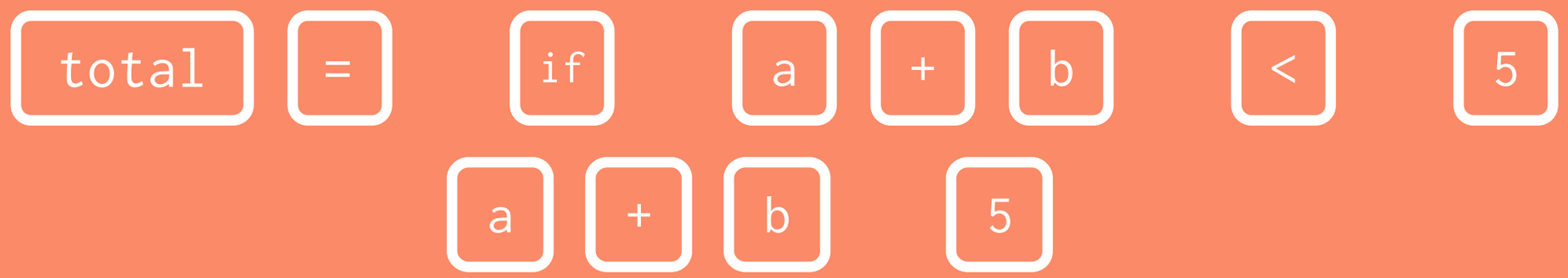
AST CONSTRUCTION

COMPILATION

↓

MACHINE CODE

TOKENIZATION



CODE STRING

↓

TOKENIZATION

PARSING

AST CONSTRUCTION

COMPILATION

↓

MACHINE CODE

CODE STRING

↓

TOKENIZATION

PARSING

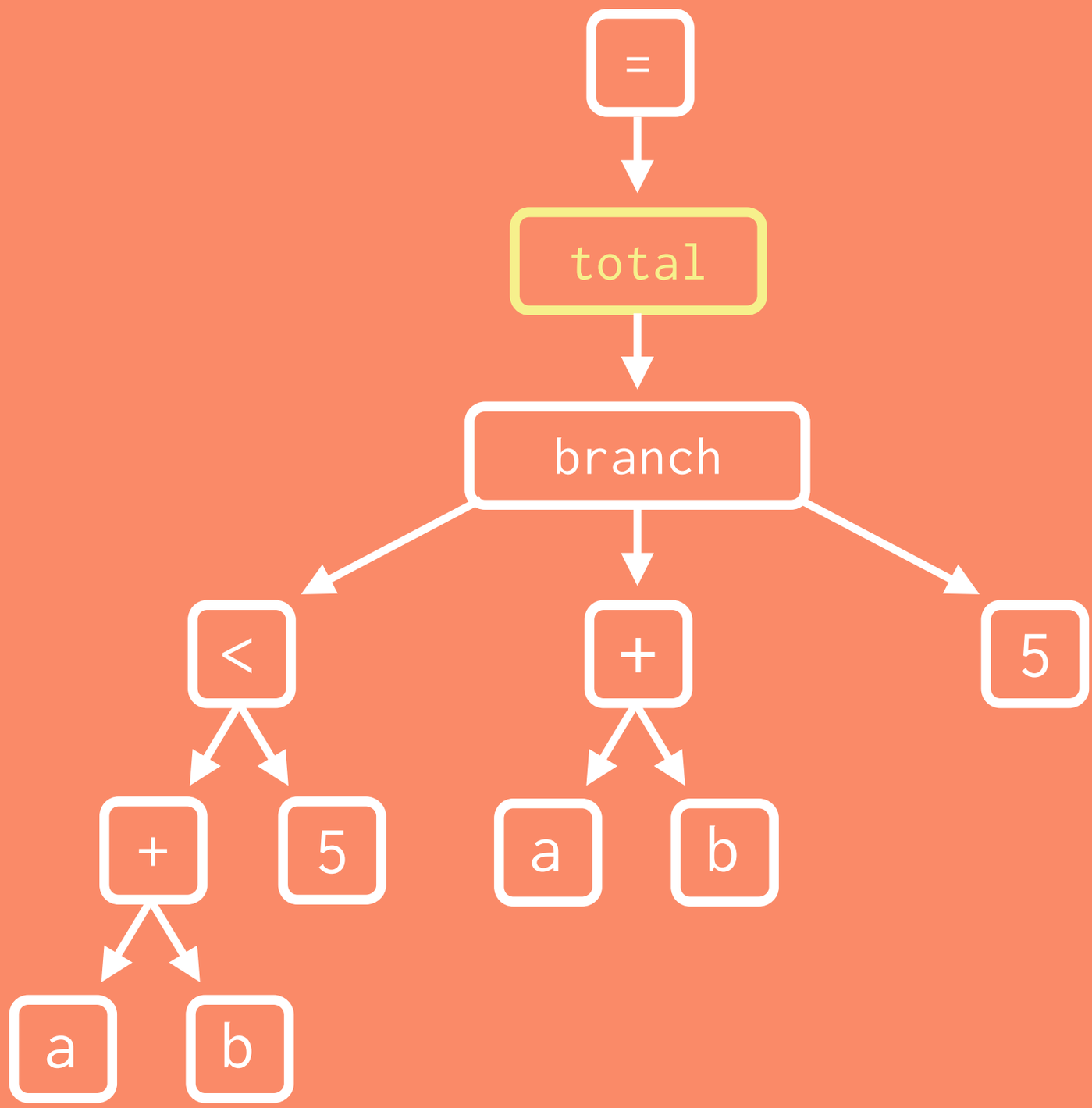
AST CONSTRUCTION

COMPILATION

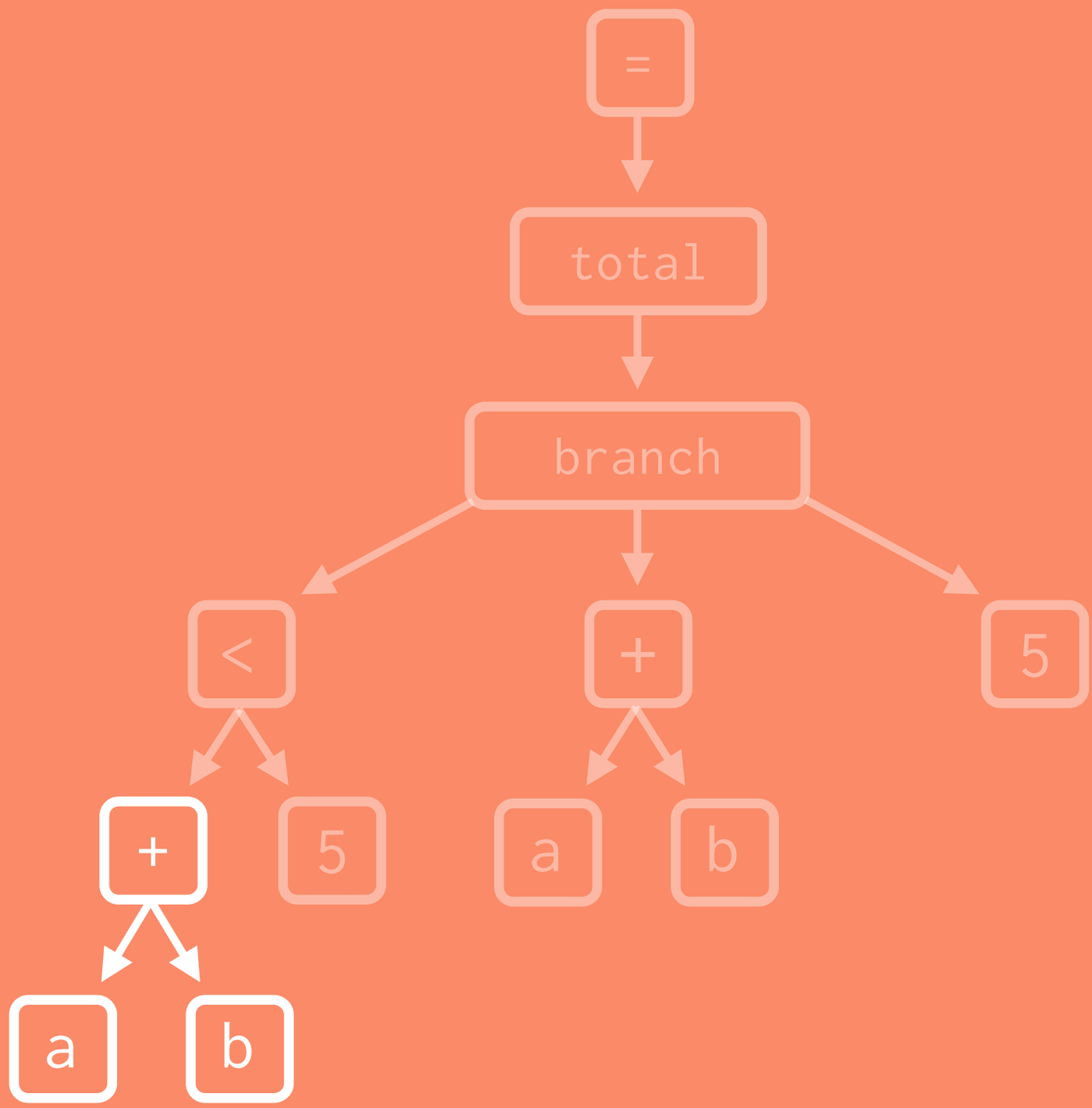
↓

MACHINE CODE

AST CONSTRUCTION



AST CONSTRUCTION



CODE STRING

↓

TOKENIZATION

PARSING

AST CONSTRUCTION

COMPILATION

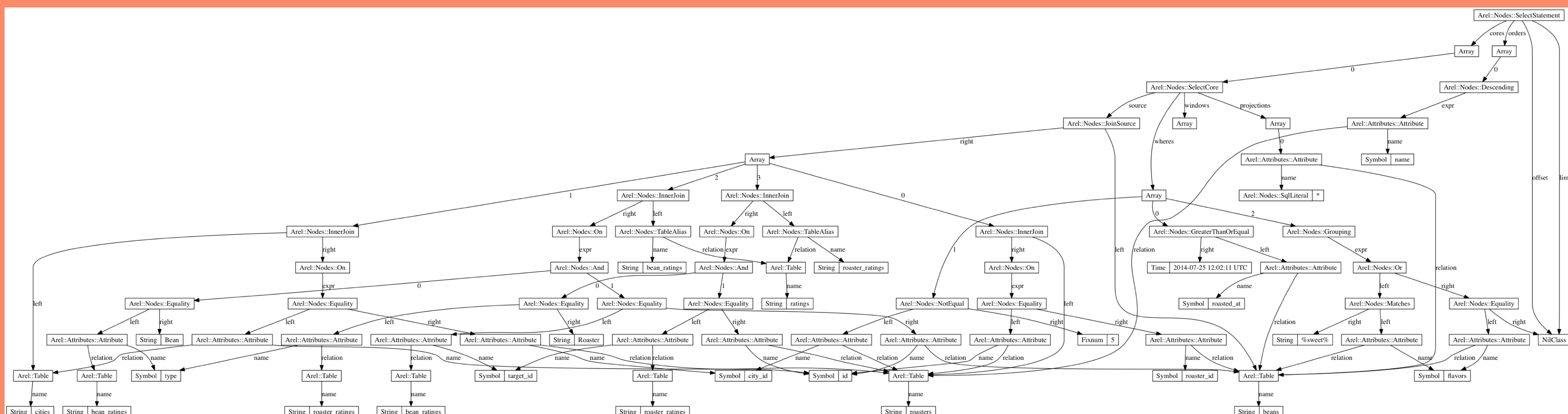
↓

MACHINE CODE

TOKENIZATION + PARSING

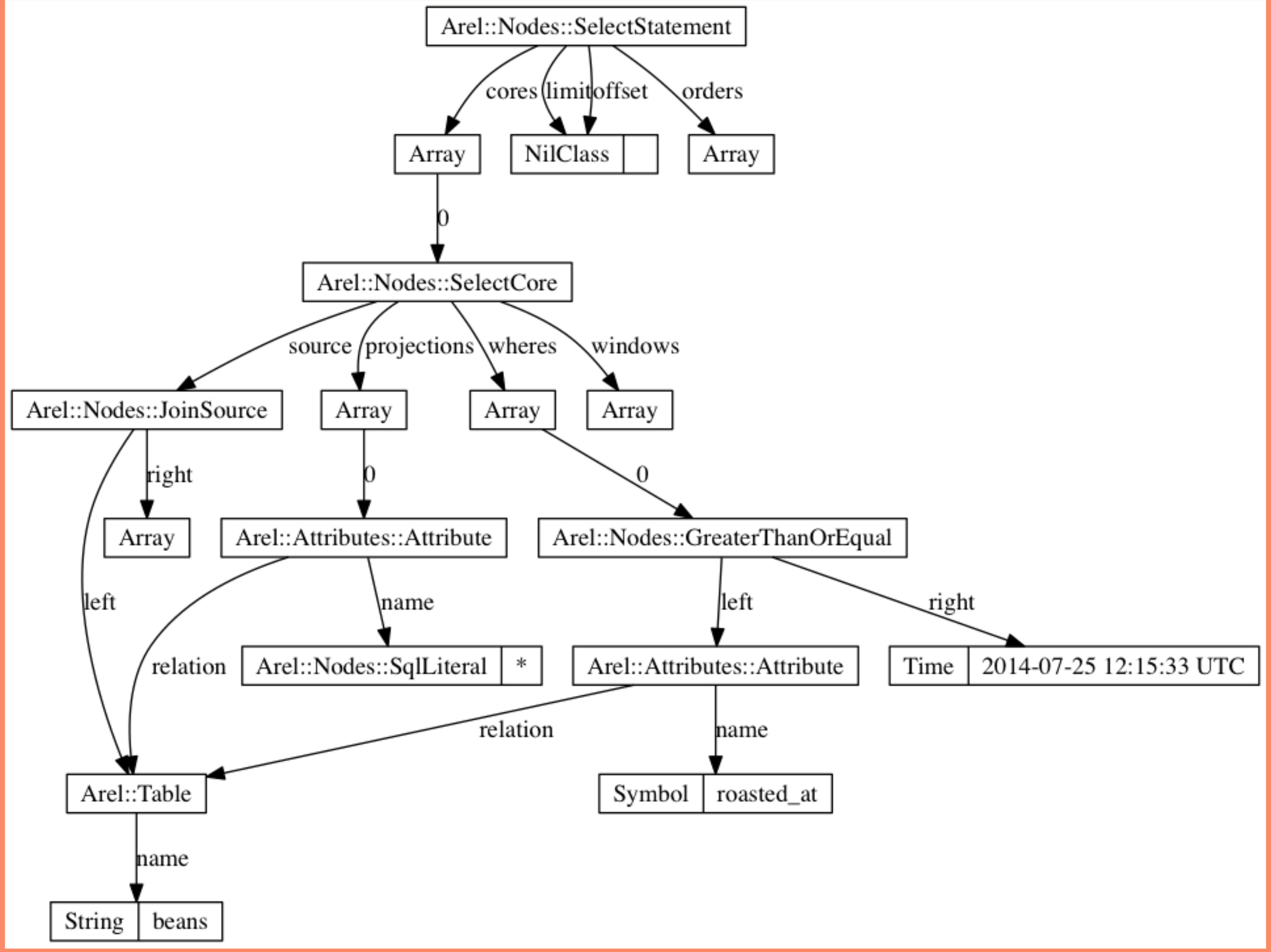
OBJECT-ORIENTED NODE GENERATION

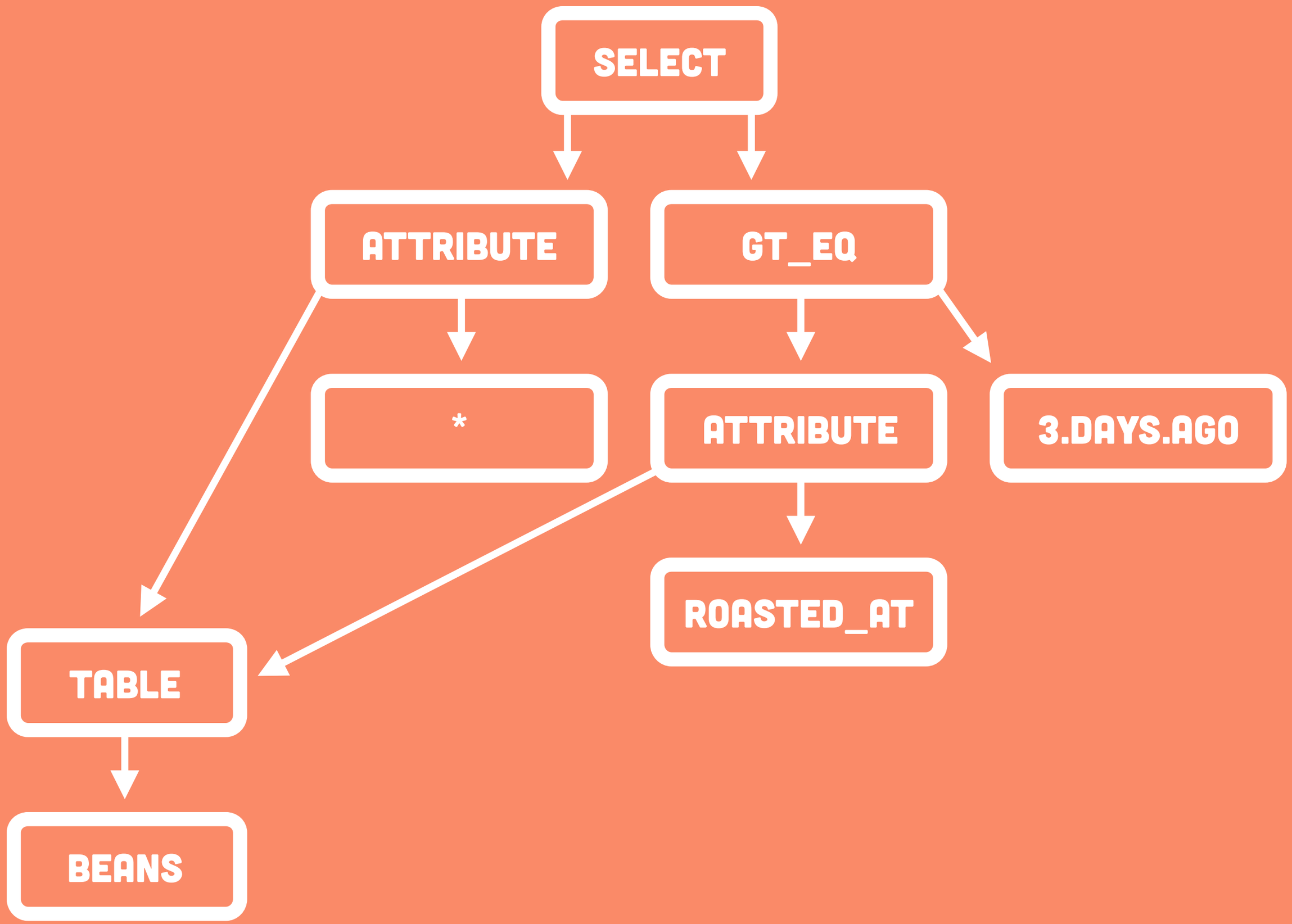
```
Bean.
  arel_table.project(Bean.arel_table[Arel.star]).
  join(Roaster.arel_table).on(Roaster.arel_table[:id].eq(Bean.arel_table[:roaster_id])).
  join(City.arel_table).on(City.arel_table[:id].eq(Roaster.arel_table[:city_id])).
  join(Rating.arel_table.alias("bean_ratings")).on(
    Arel::Table.new(:bean_ratings)[:type].eq("Bean"),
    Arel::Table.new(:bean_ratings)[:target_id].eq(Bean.arel_table[:id])
  ).
  join(Rating.arel_table.alias("roaster_ratings")).on(
    Arel::Table.new(:roaster_ratings)[:type].eq("Roaster"),
    Arel::Table.new(:roaster_ratings)[:target_id].eq(Roaster.arel_table[:id])
  ).
  where(Bean.arel_table[:roasted_at].gteq(3.days.ago)).
  where(Roaster.arel_table[:city_id].not_eq(5)).
  where(
    Bean.arel_table[:flavors].matches("%sweet%").or(
    Bean.arel_table[:flavors].eq(nil))
  ).
  having(
    Arel::Table.new(:bean_ratings)[:value].average.gteq(90).or(
    Arel::Table.new(:roaster_ratings)[:value].average.gteq(90).or(
    Arel::Table.new(:bean_ratings)[:id].count.eq(0).or(
    Arel::Table.new(:roaster_ratings)[:id].count.eq(0))))
  ).
  group(
    Bean.arel_table[:id],
    Roaster.arel_table[:name]
  ).
  order(Roaster.arel_table[:name].desc)
```

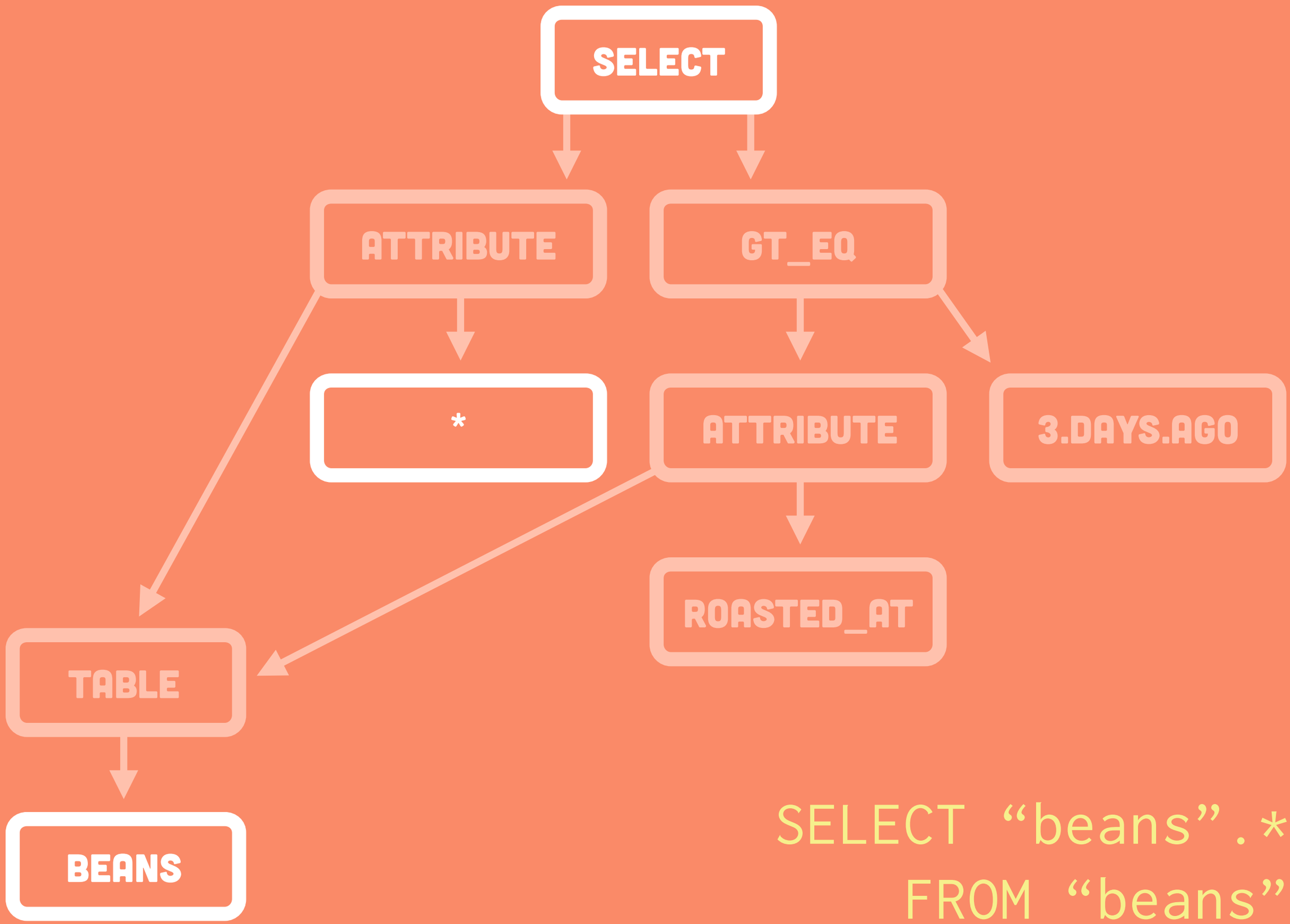


Arel::Nodes::SelectStatement
cores
orders
offset
limit
Array
Arel::Nodes::SelectCore
Arel::Nodes::Descending
source
windows
projections
wheres
expr
Arel::Nodes::JoinSource
Arel::Attributes::Attribute
right
left
Symbol
name
Arel::Nodes::SqlLiteral
*
Arel::Nodes::InnerJoin
Arel::Nodes::On
Arel::Nodes::TableAlias
Arel::Nodes::GreaterThanOrEqual
Arel::Nodes::Grouping
Arel::Nodes::And
Arel::Table
String
bean_ratings
roaster_ratings
ratings
Time
2014-07-25 12:02:11 UTC
roasted_at
Arel::Nodes::Or
Arel::Nodes::Equality
Arel::Nodes::NotEqual
Arel::Nodes::Matches
relation
Bean
Roaster
Fixnum
5
%sweet%
NilClass
type
target_id
city_id
id
roaster_id
flavors
cities
roasters
beans

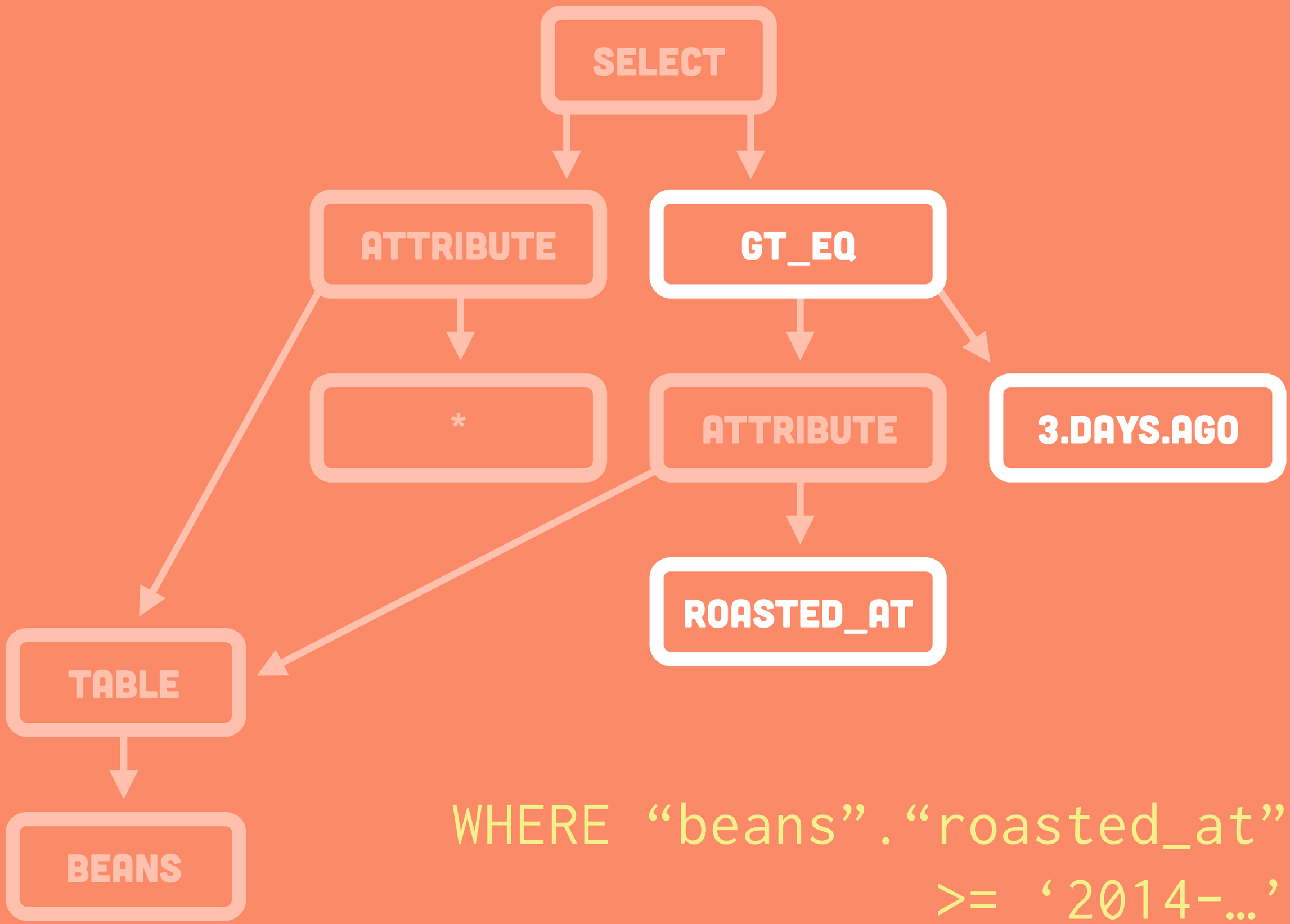
```
beans = Arel::Table.new(:beans)

beans.
  project(beans[Arel.star]).
  where(beans[:roasted_at].gteq(3.days.ago))
```


Arel::Nodes::SelectStatement
cores
limit
offset
orders
Array
NilClass
Array
0
Arel::Nodes::SelectCore
source
projections
wheres
windows
Arel::Nodes::JoinSource
Array
Array
Array
right
0
0
Array
Arel::Attributes::Attribute
Arel::Nodes::GreaterThanOrEqual
left
name
left
right
relation
Arel::Nodes::SqlLiteral
*
Arel::Attributes::Attribute
Time
2014-07-25 12:15:33 UTC
relation
name
Arel::Table
Symbol
roasted_at
name
String
beans

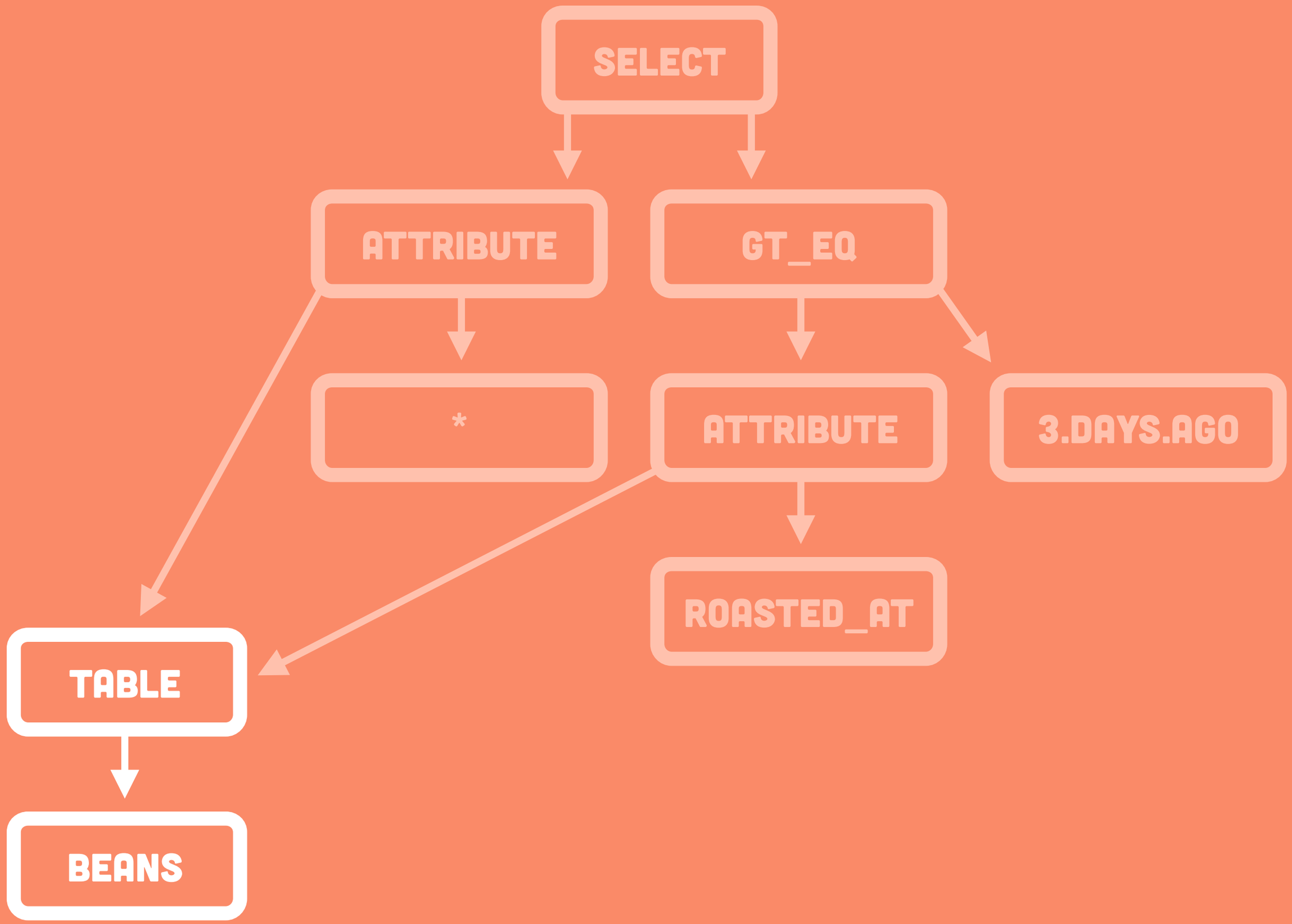

SELECT
ATTRIBUTE
GT_EQ
*
ATTRIBUTE
3.DAYS.AGO
ROASTED_AT
TABLE
BEANS

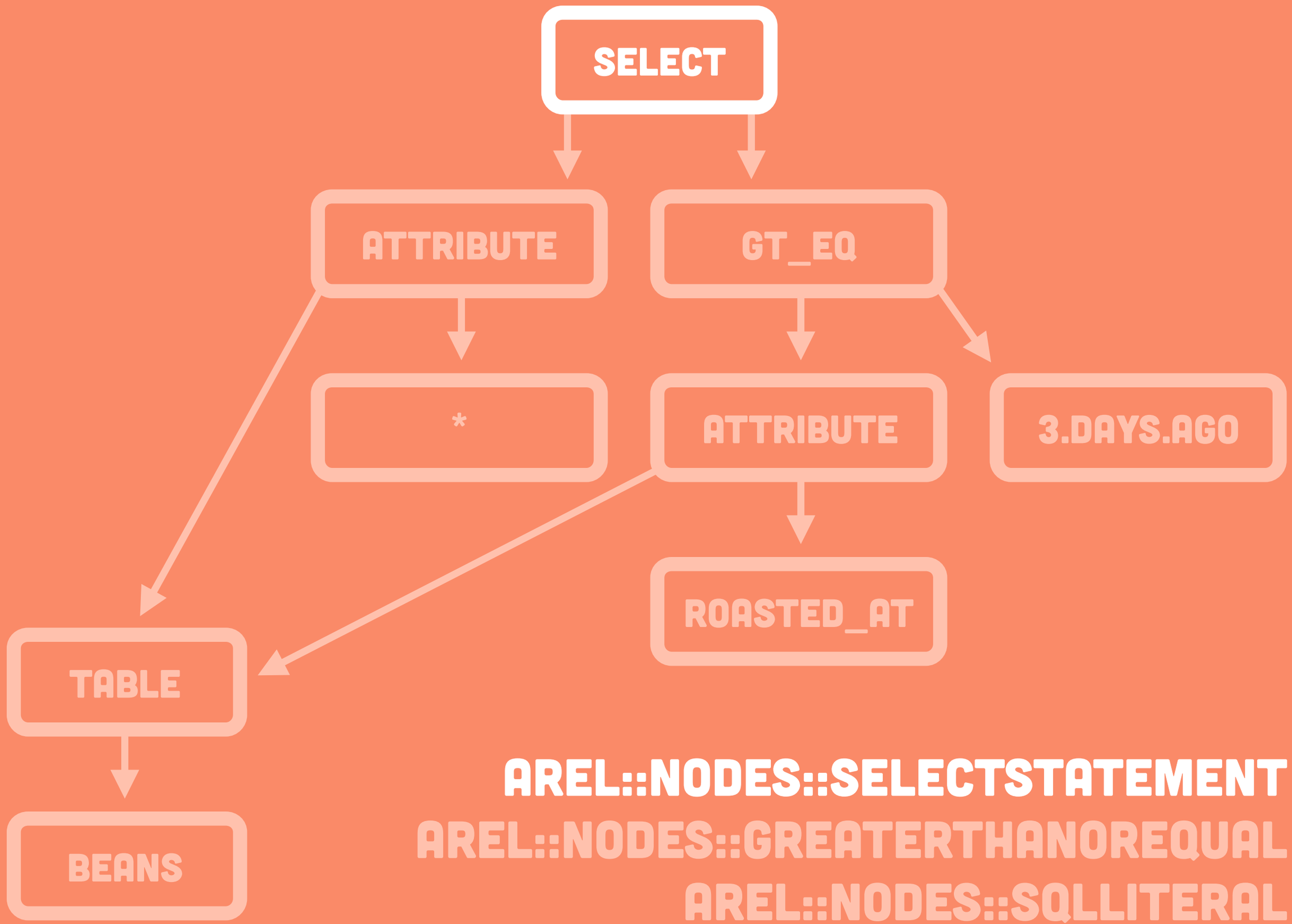



```
SELECT “beans”.*
  FROM “beans”
```




```
WHERE “beans”.“roasted_at”
>= ‘2014-…’
```

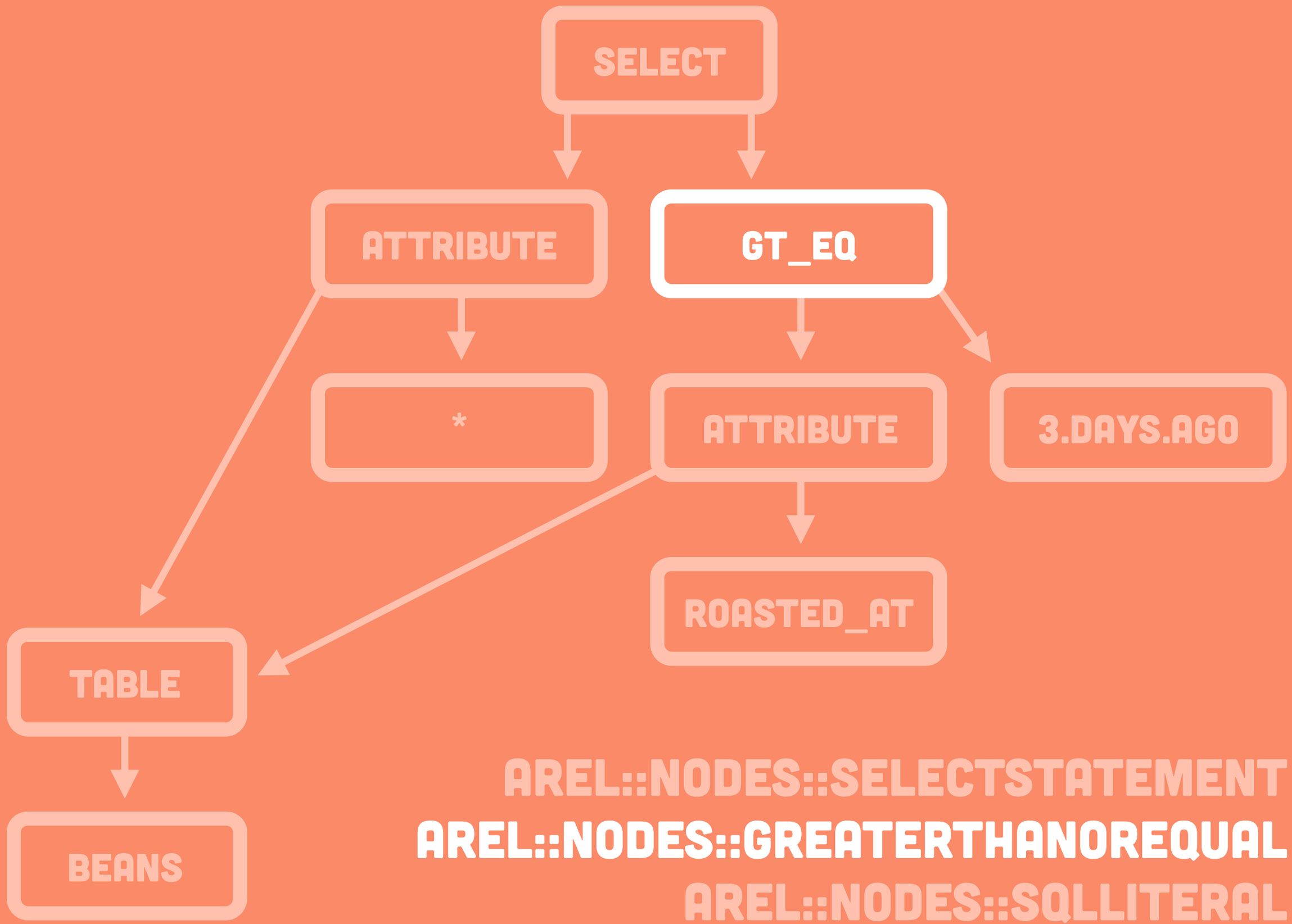

SELECT
ATTRIBUTE
GT_EQ
*
ATTRIBUTE
3.DAYS.AGO
ROASTED_AT
TABLE
BEANS

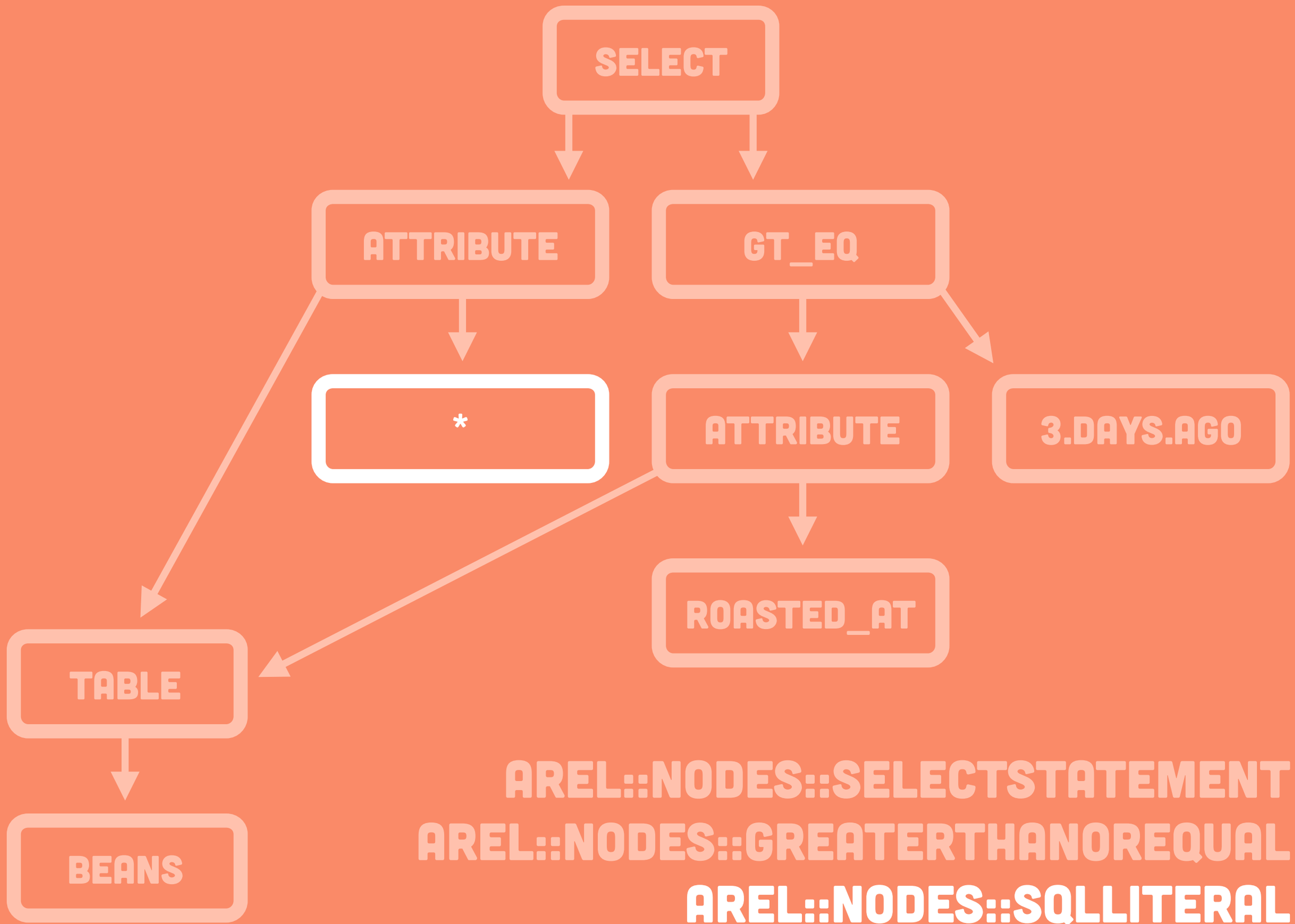


AREL::NODES::SELECTSTATEMENT

AREL::NODES::GREATERTHANOREQUAL

AREL::NODES::SQLLITERAL

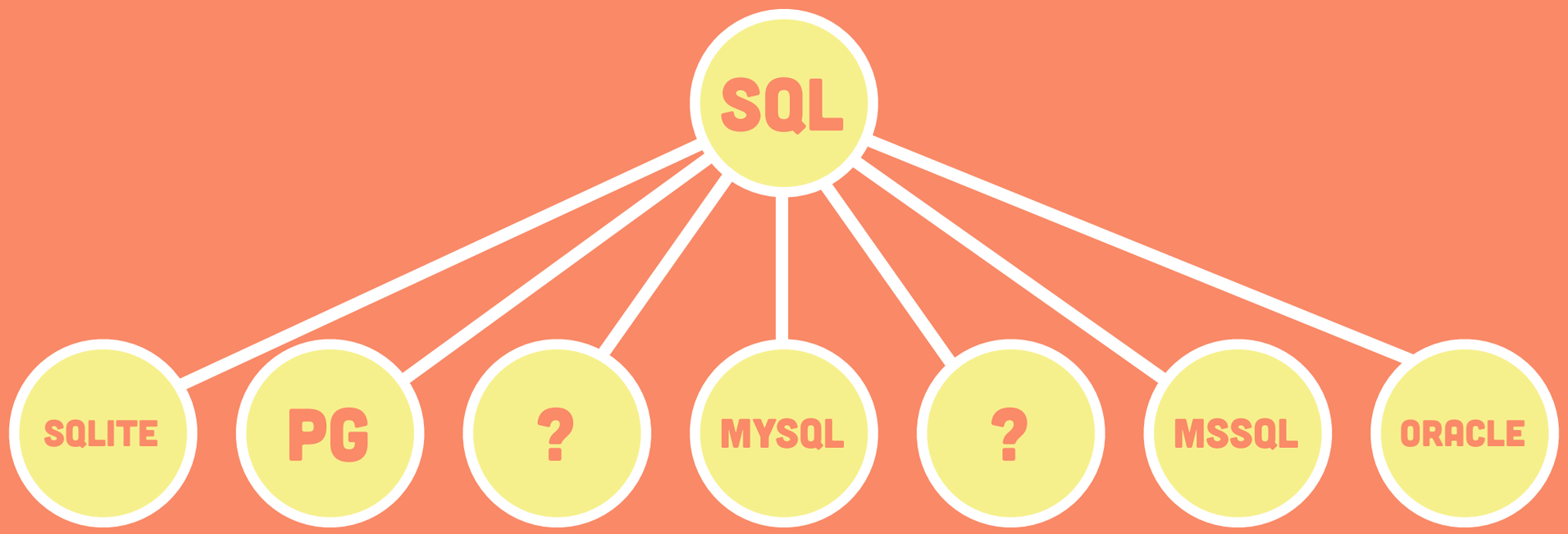

SELECT
ATTRIBUTE
GT_EQ
*
ATTRIBUTE
3.DAYS.AGO
ROASTED_AT
TABLE
BEANS
AREL::NODES::SELECTSTATEMENT
AREL::NODES::GREATERTHANOREQUAL
AREL::NODES::SQLLITERAL

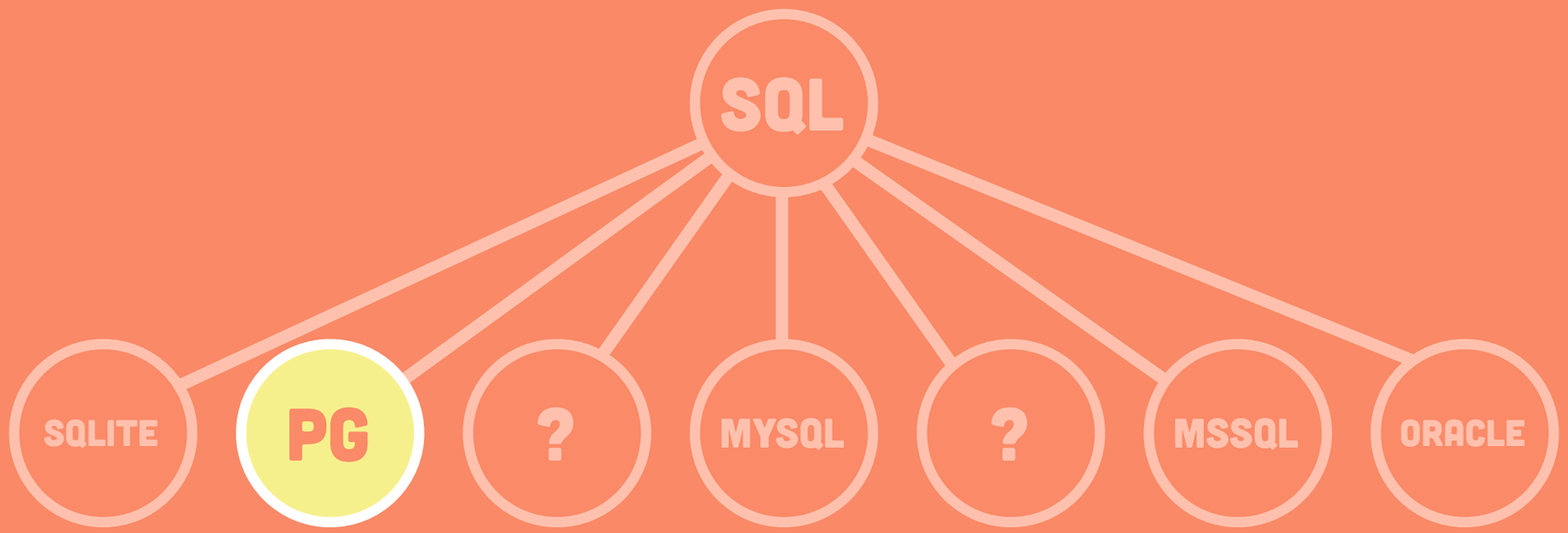


AREL::NODES::SELECTSTATEMENT
AREL::NODES::GREATERTHANOREQUAL
AREL::NODES::SQLLITERAL

THE VISITOR PATTERN

SQL


SQL
SQLITE
PG
?
MYSQL
?
MSSQL
ORACLE


SQL
SQLITE
PG
?
MYSQL
?
MSSQL
ORACLE

BASED ON THE
TYPE OF VISITOR
AND THE
TYPE OF NODE
IT IS VISITING, WE GET
DYNAMIC BEHAVIOR

AREL::VISITORS::TOSQL

```
def visit_Arel_Nodes_Matches(node, collector)
  collector = visit(node.left, collector)
  collector << " LIKE "
  visit(node.right, collector)
end
```

AREL::VISITORS::TOSQL

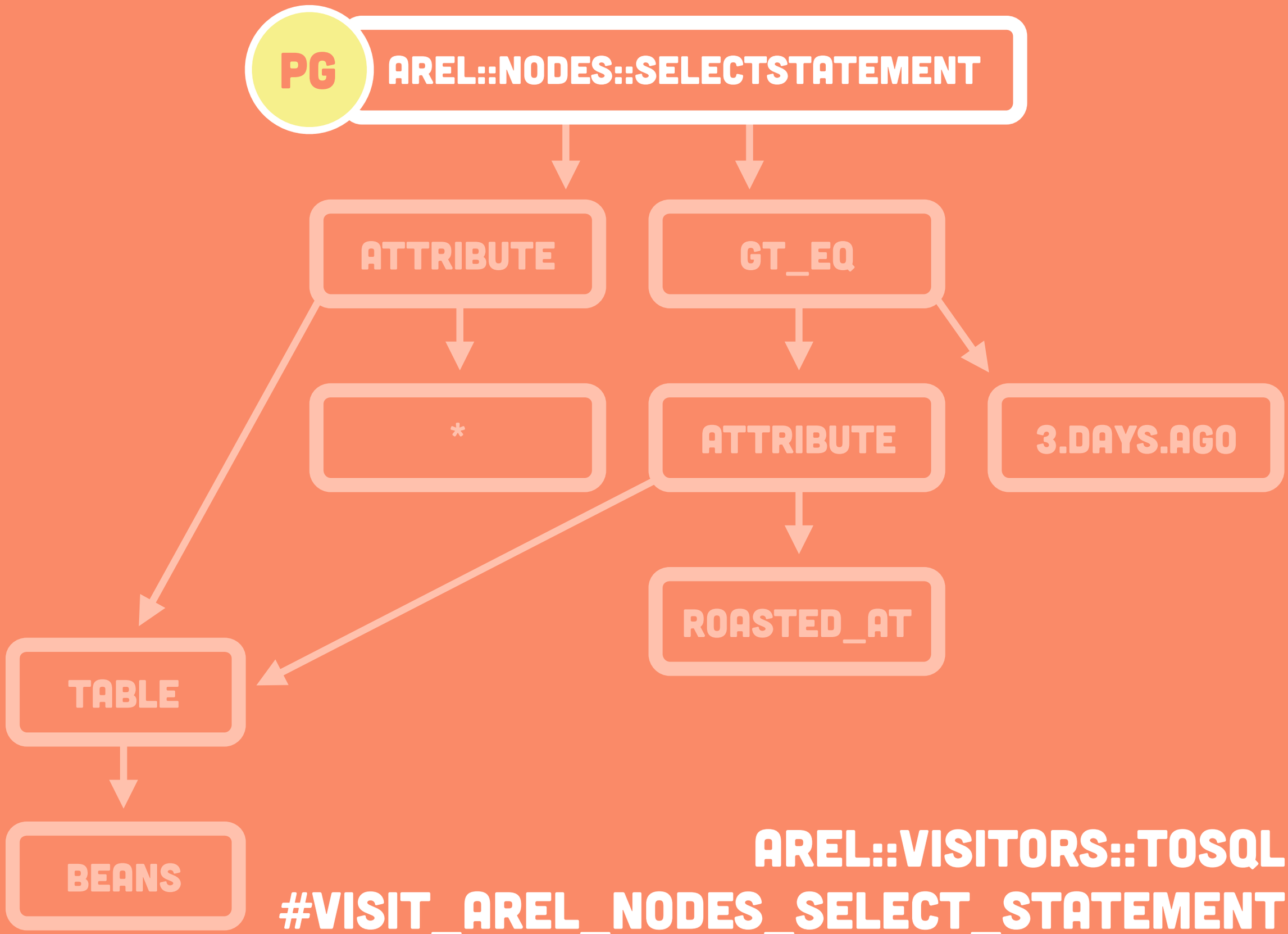
```
def visit_Arel_Nodes_Matches(node, collector)
  collector = visit(node.left, collector)
  collector << " LIKE "
  visit(node.right, collector)
end
```

AREL::VISITORS::TOSQL

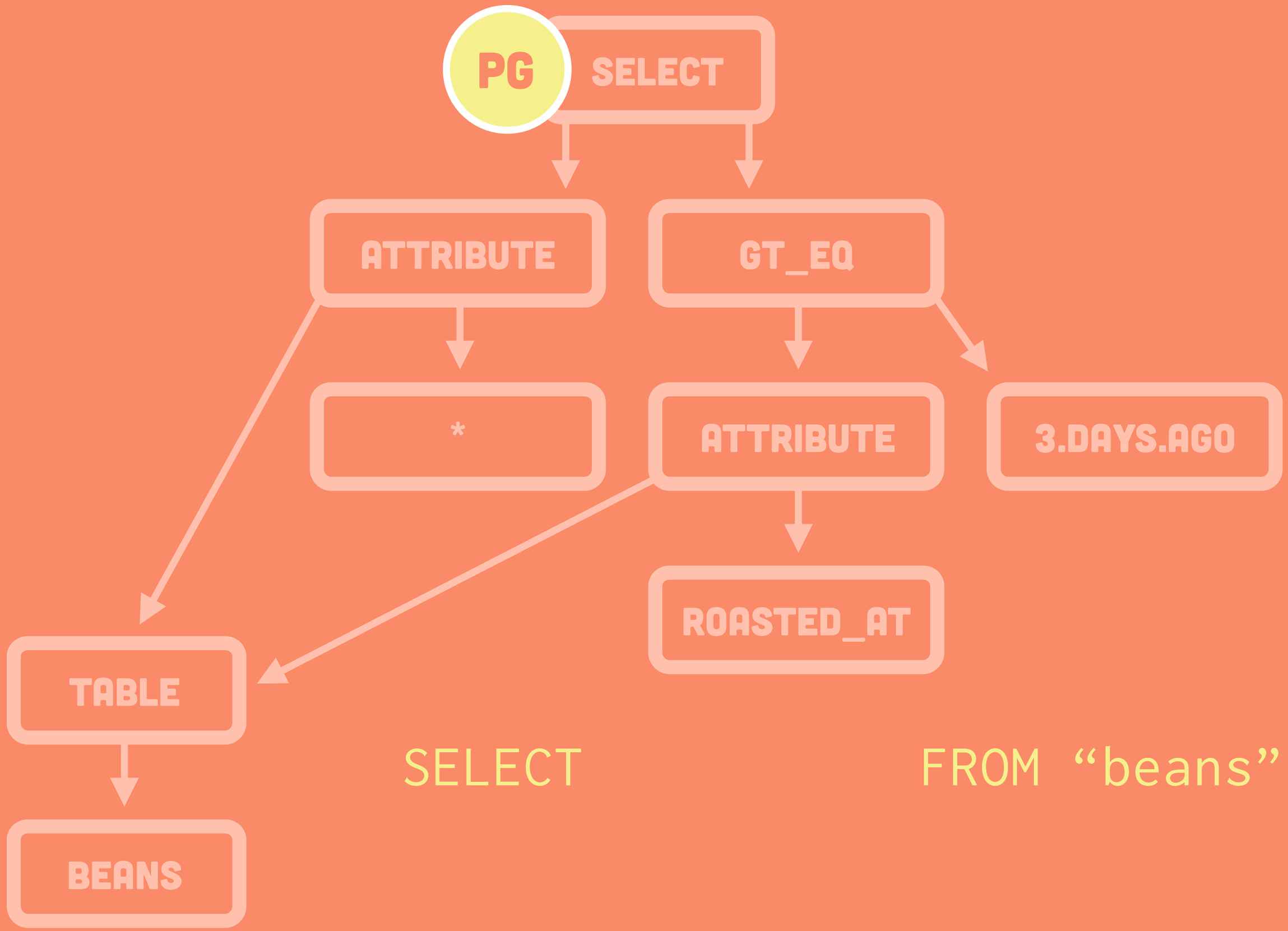
```
def visit_Arel_Nodes_Matches(node, collector)
  collector = visit(node.left, collector)
  collector << " LIKE "
  visit(node.right, collector)
end
```

AREL::VISITORS::TOSQL

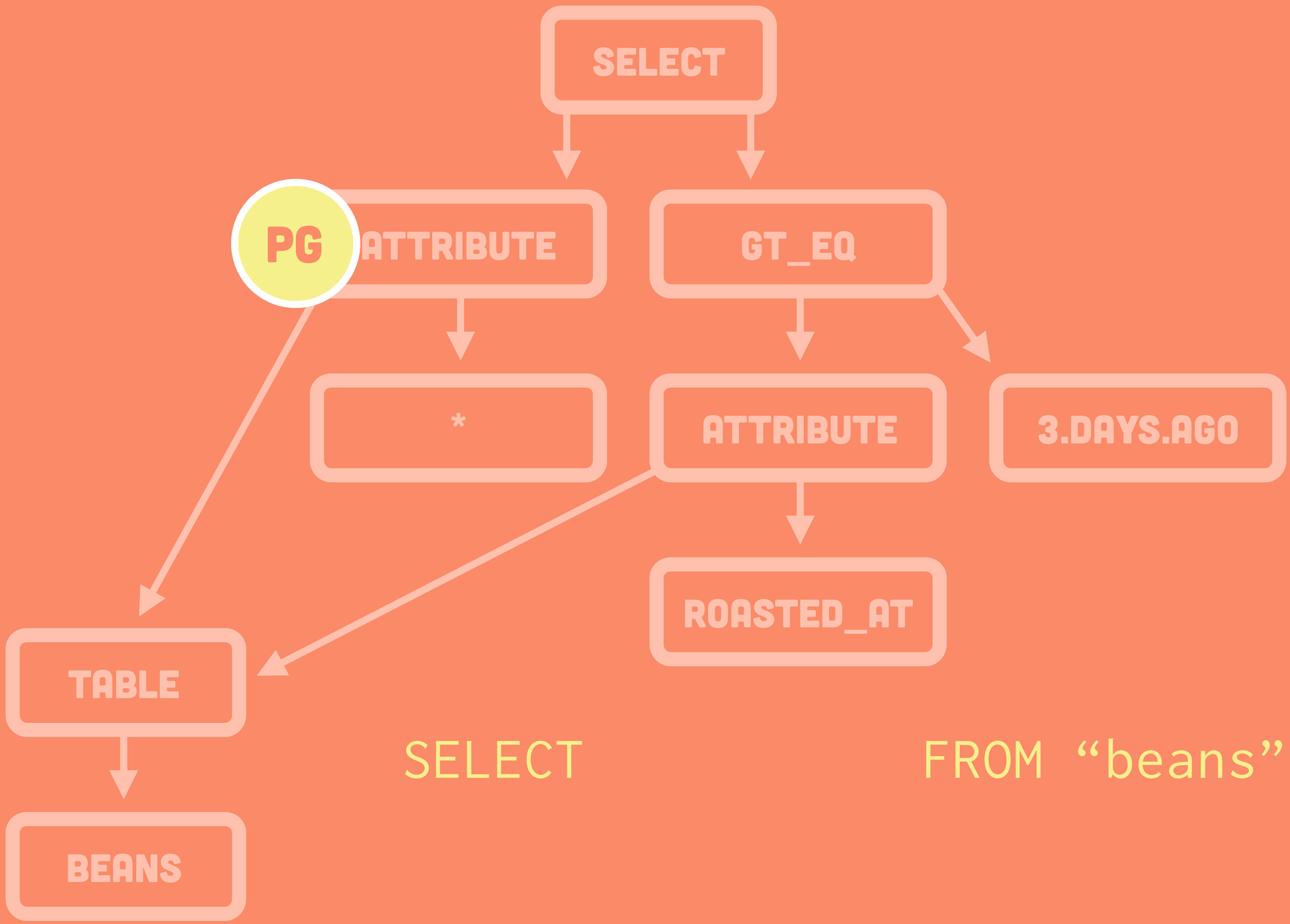
```
def visit_Arel_Nodes_Matches(node, collector)
  collector = visit(node.left, collector)
  collector << " LIKE "
  visit(node.right, collector)
end
```



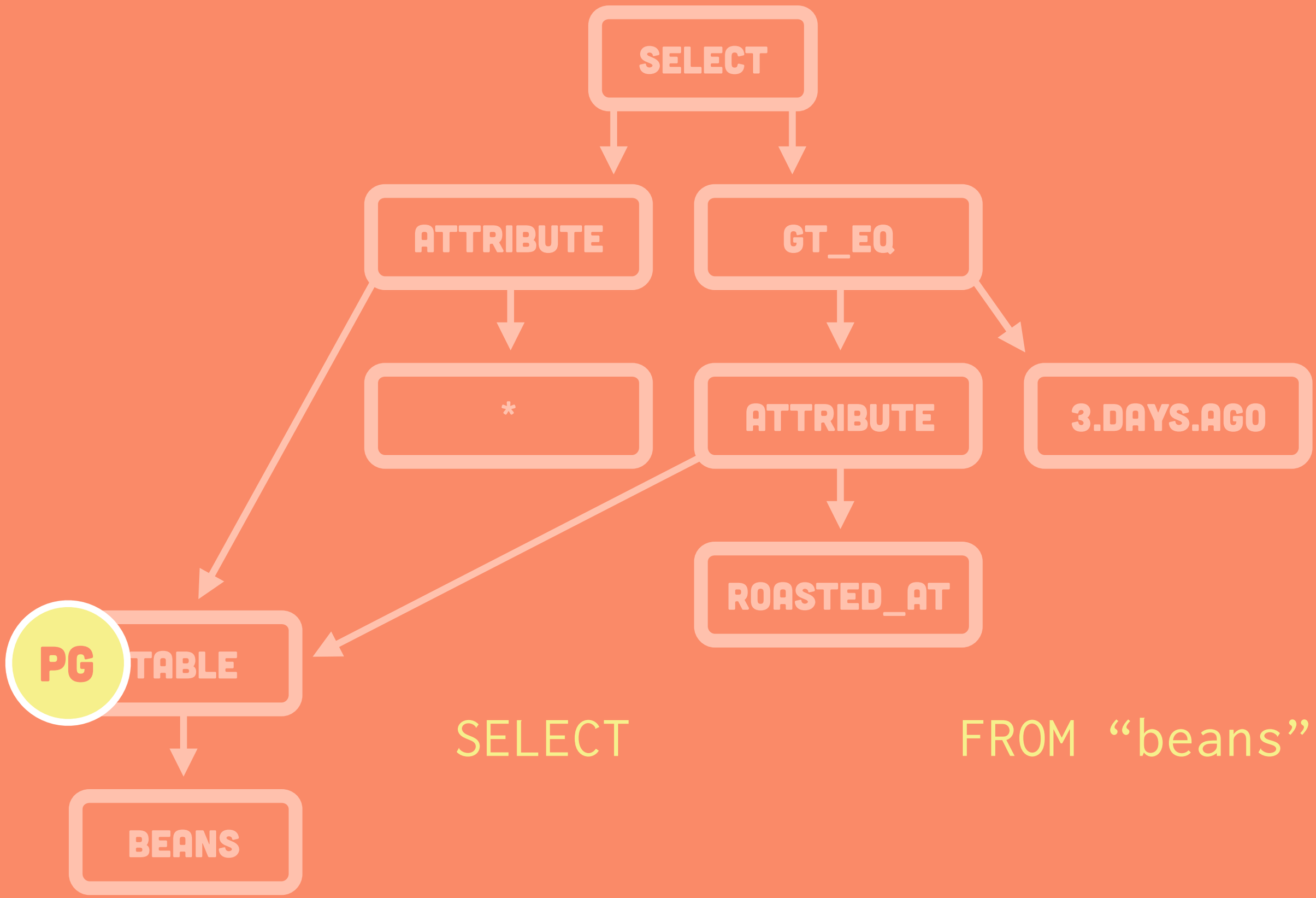
AREL::VISITORS::TOSQL
#VISIT_AREL_NODES_SELECT_STATEMENT



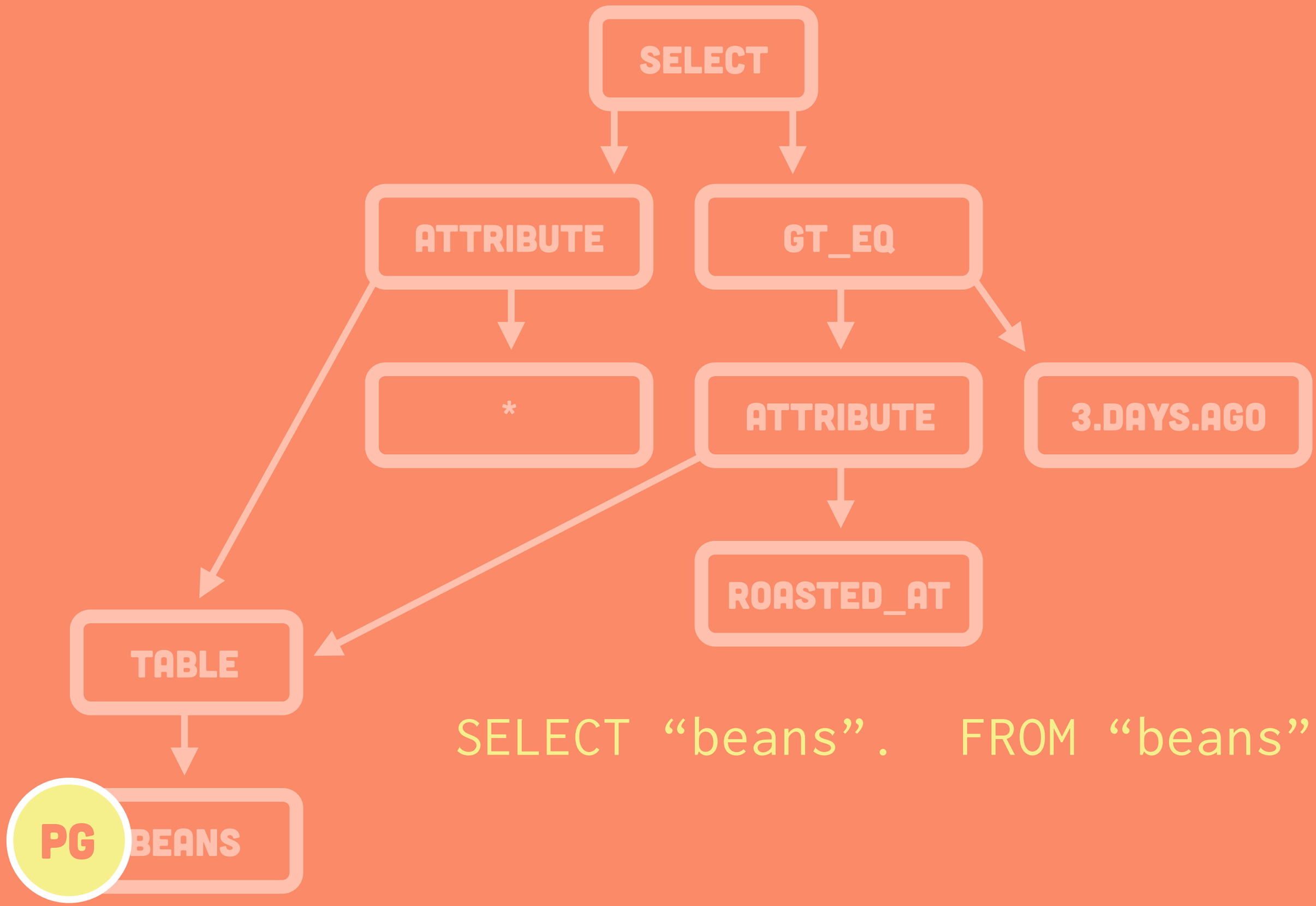

```
SELECT                    FROM “beans”
```



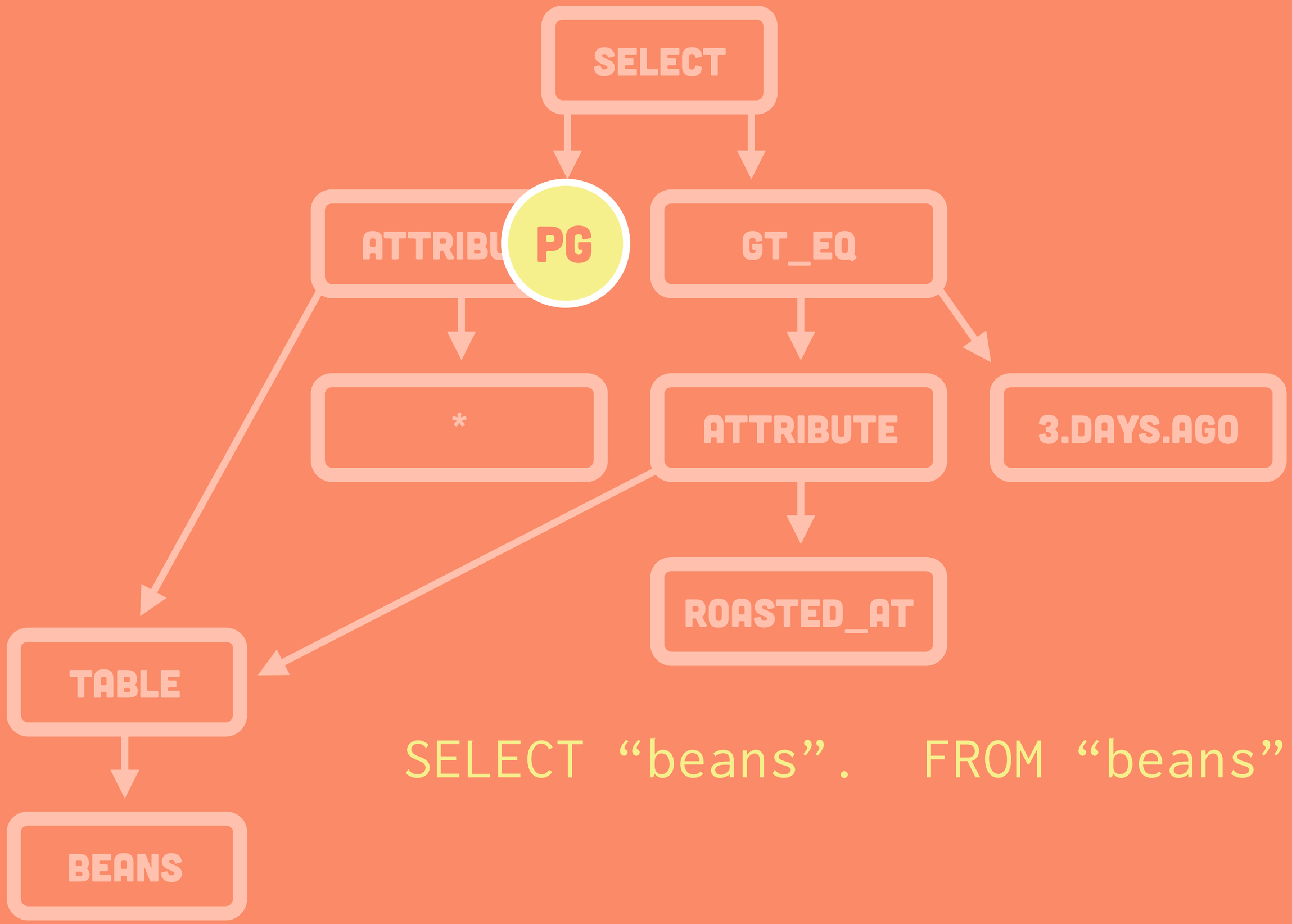

```
SELECT                    FROM “beans”
```



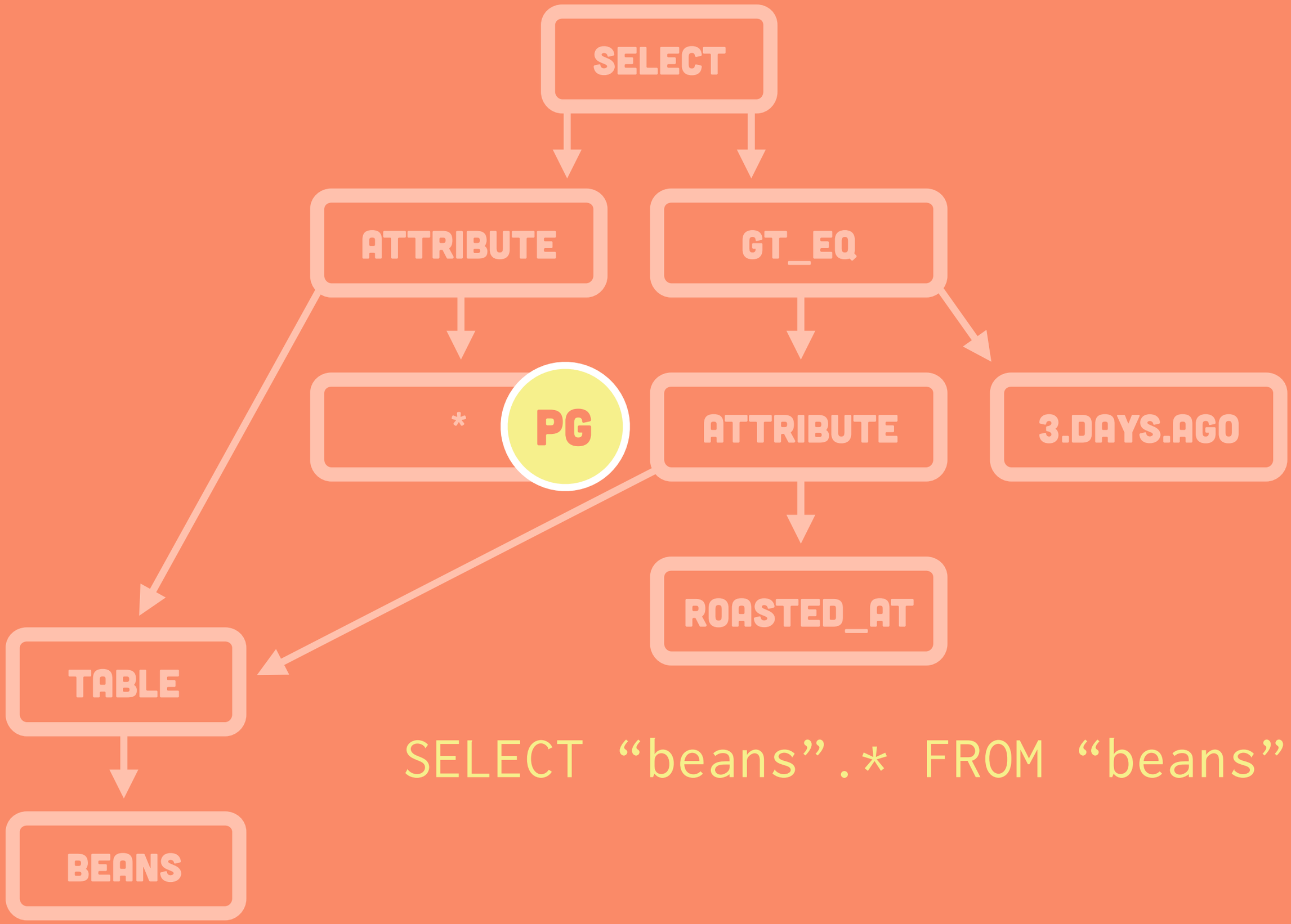

```
SELECT                    FROM “beans”
```



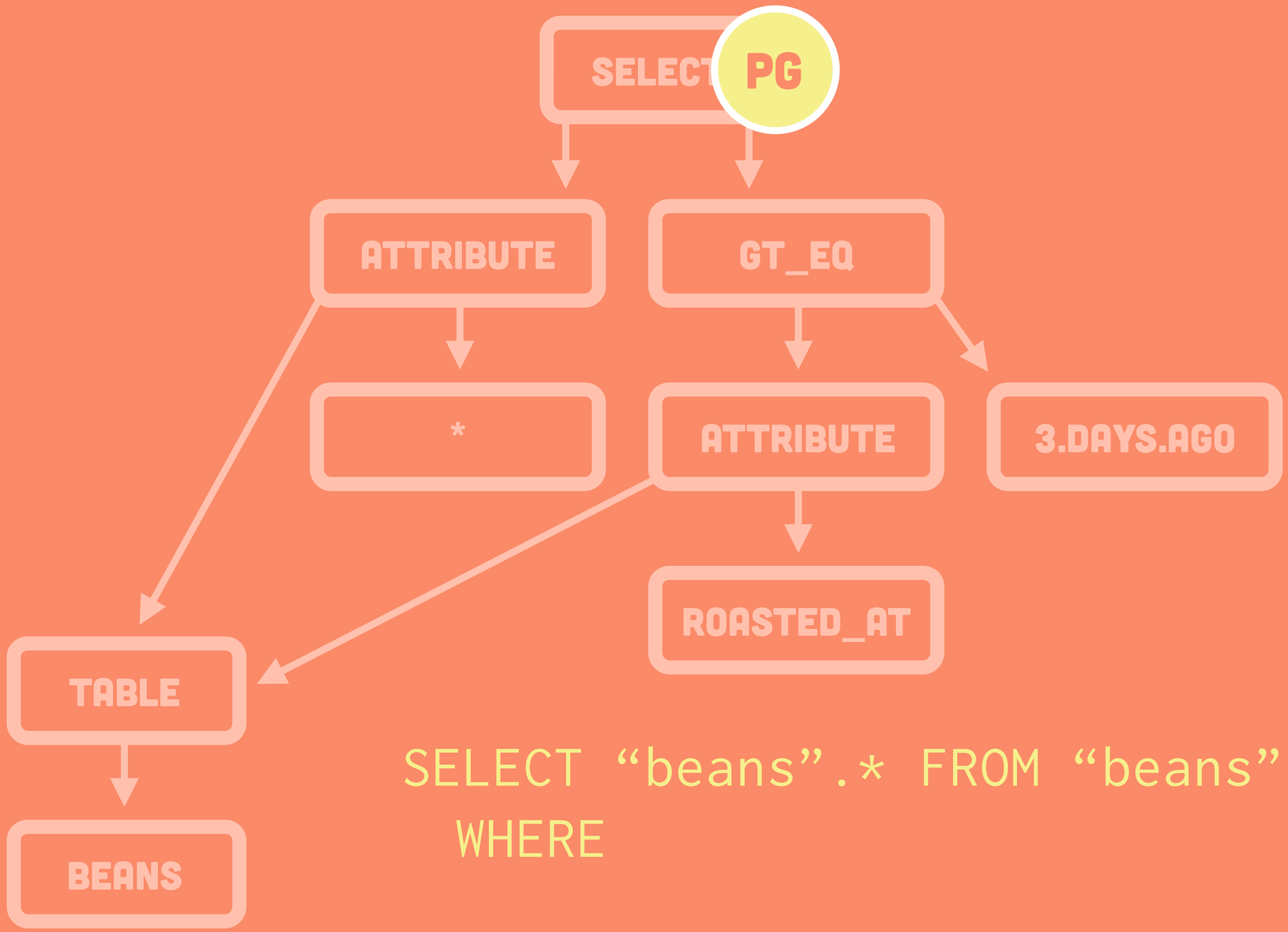

```
SELECT “beans”.  FROM “beans”
```



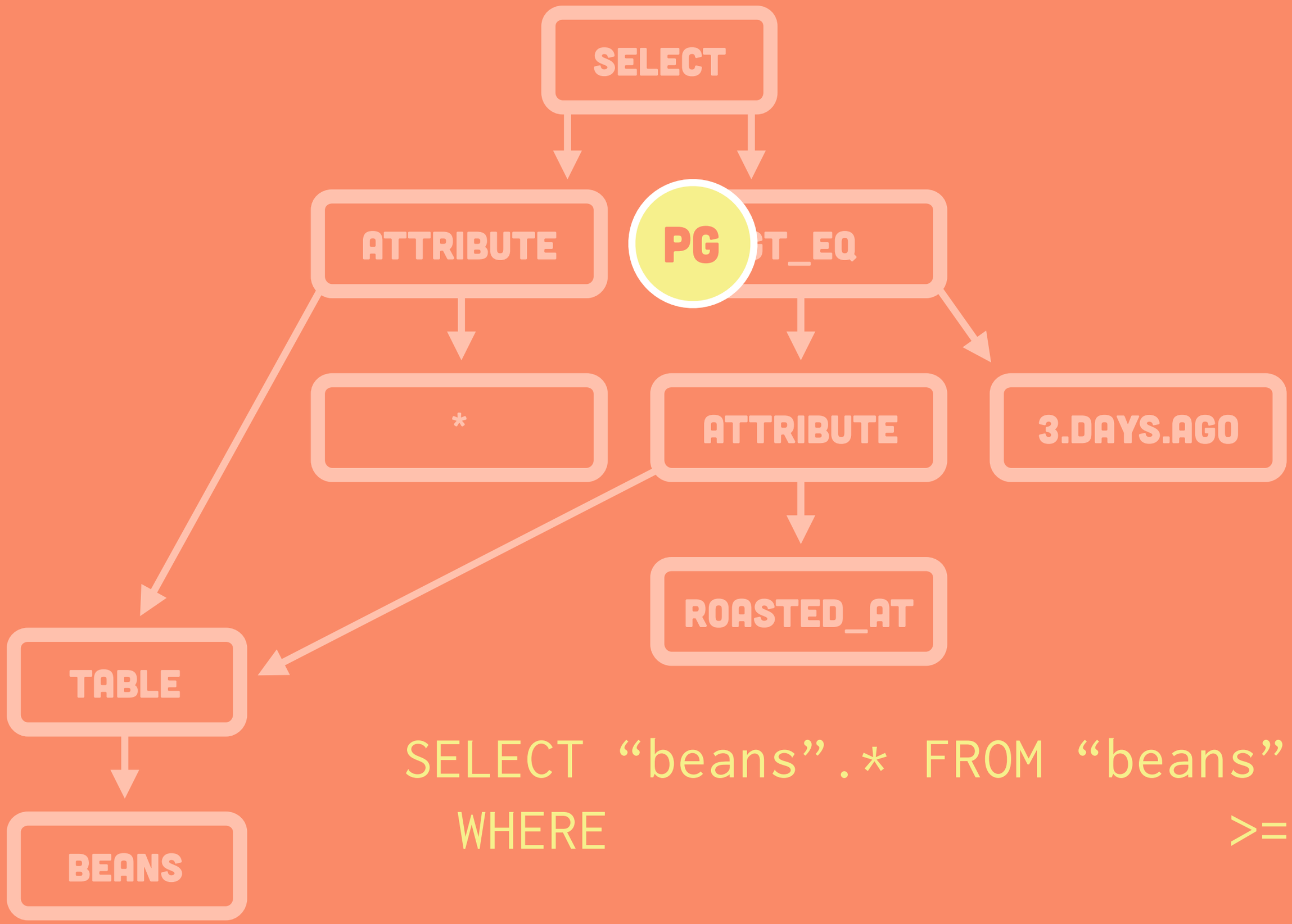

```
SELECT “beans”.  FROM “beans”
```



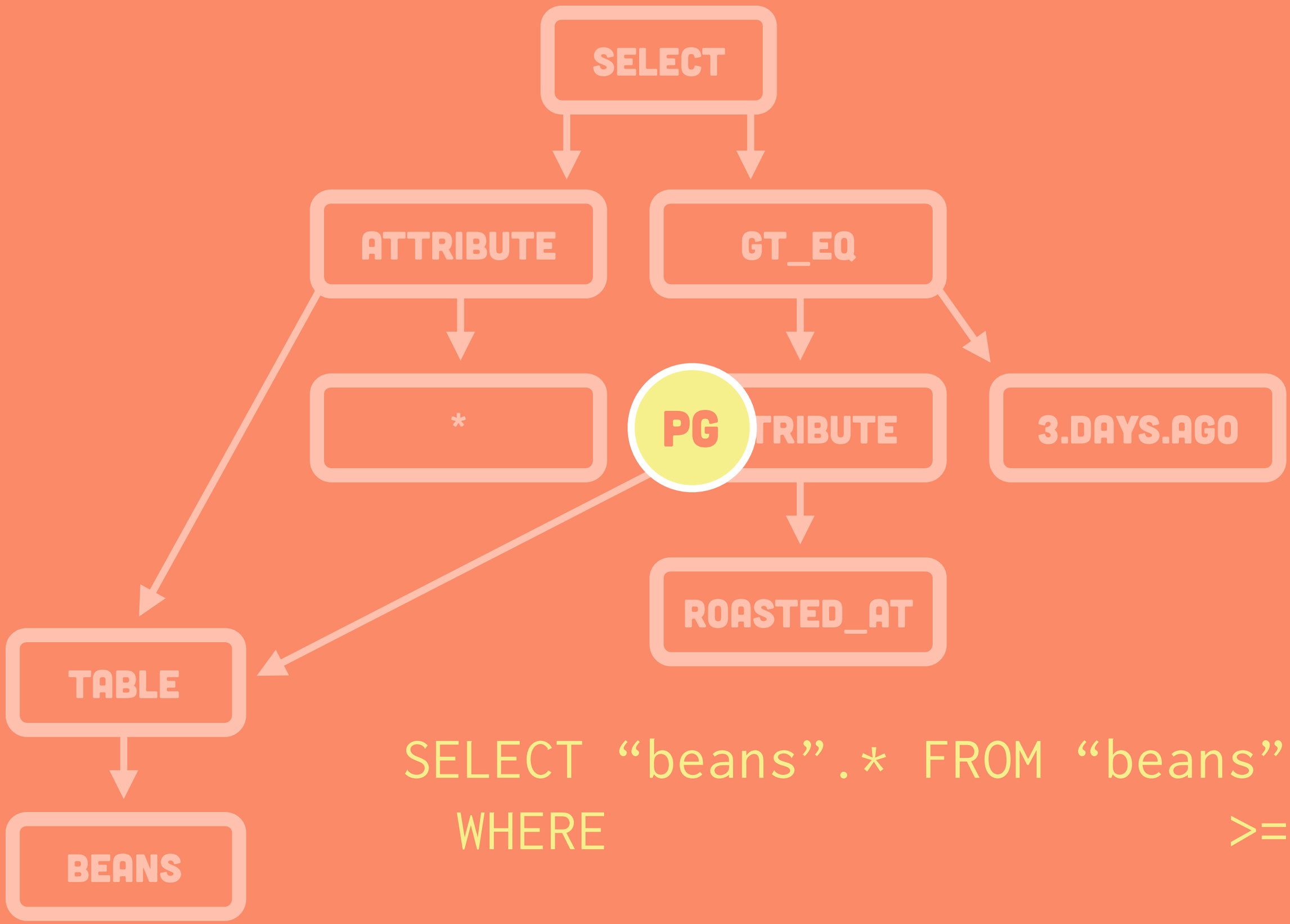

```
SELECT “beans”.* FROM “beans”
```



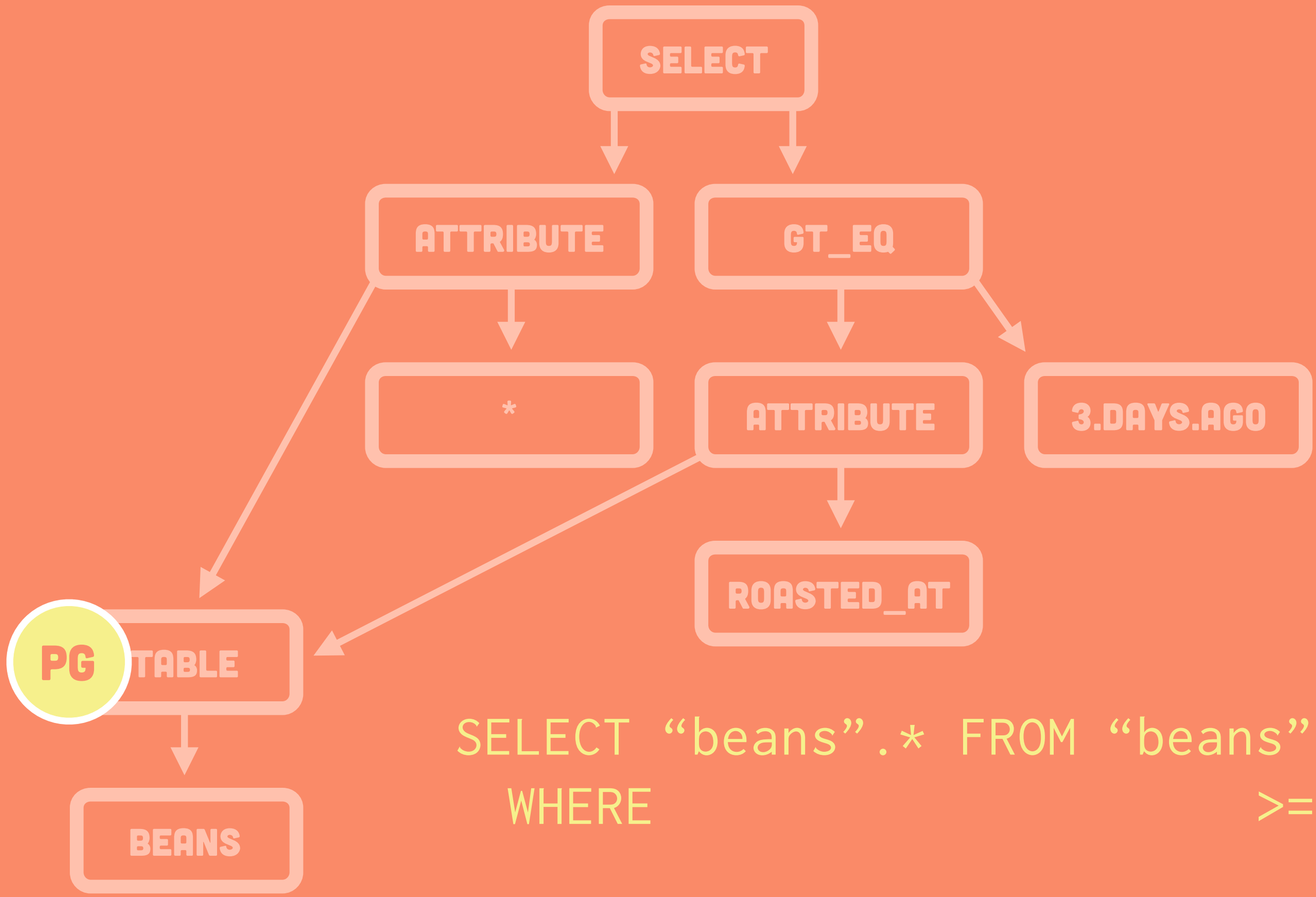

```
SELECT “beans”.* FROM “beans”
  WHERE
```



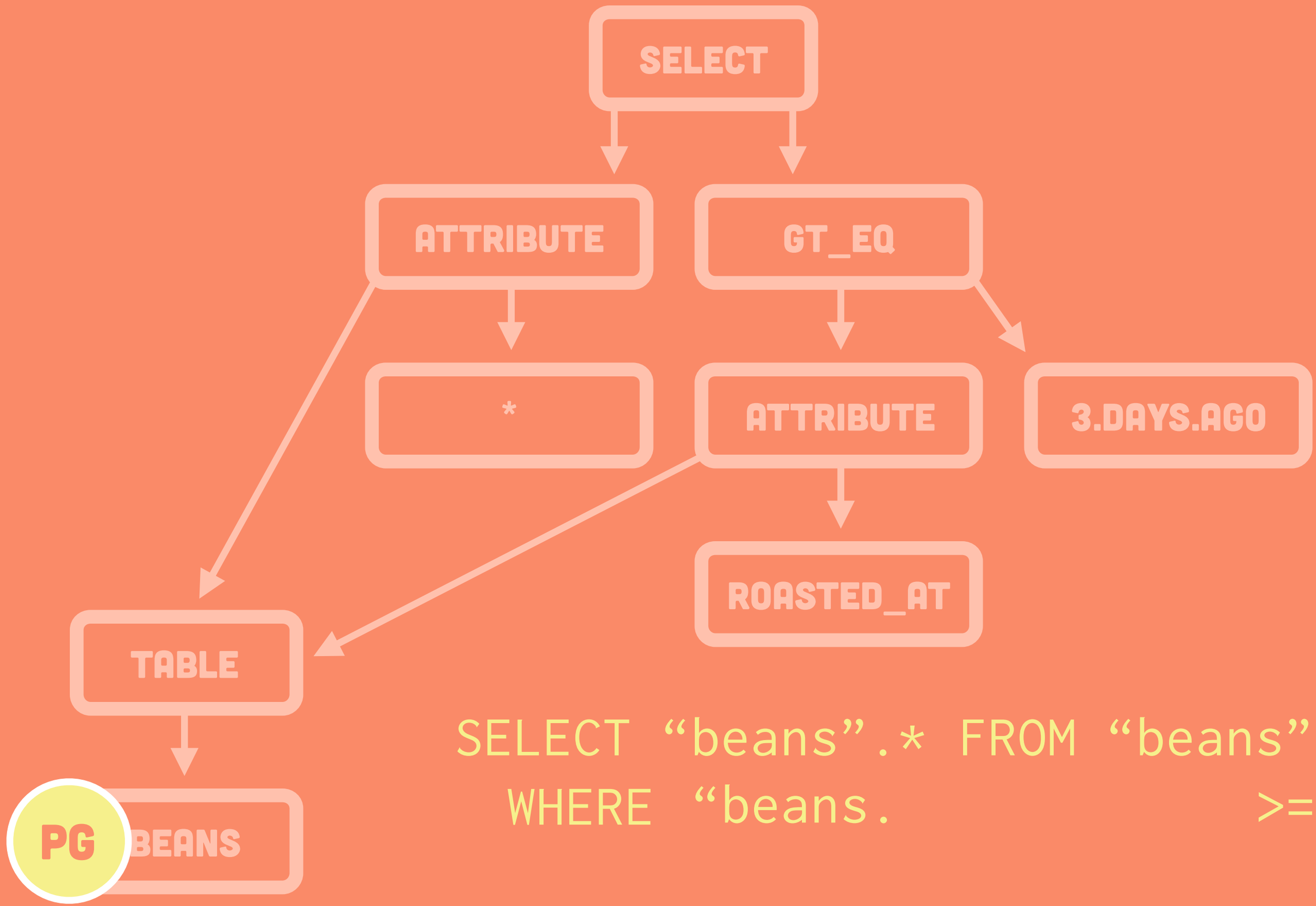

```
SELECT “beans”.* FROM “beans”
  WHERE                    >=
```



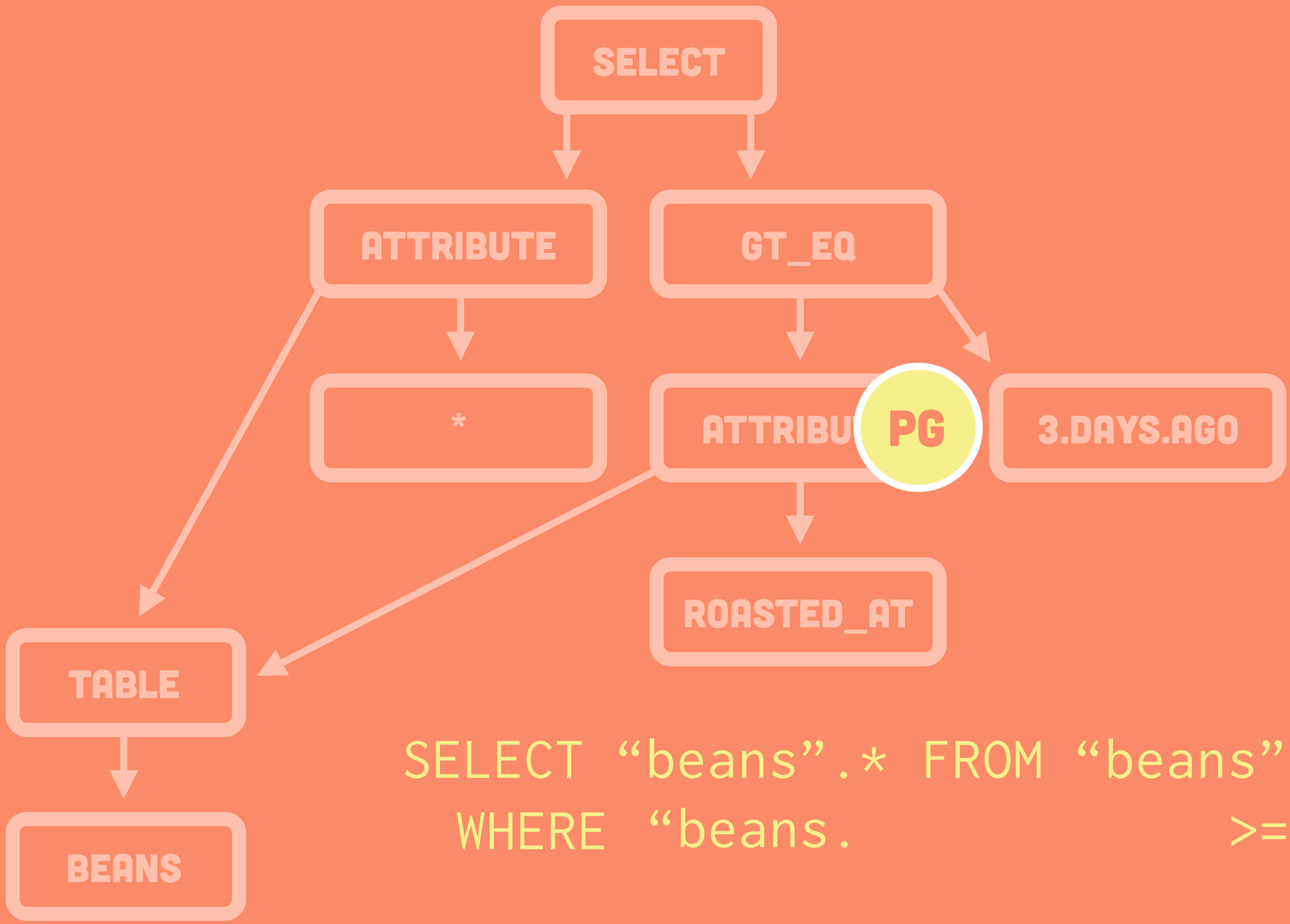

```
SELECT “beans”.* FROM “beans”
  WHERE                    >=
```



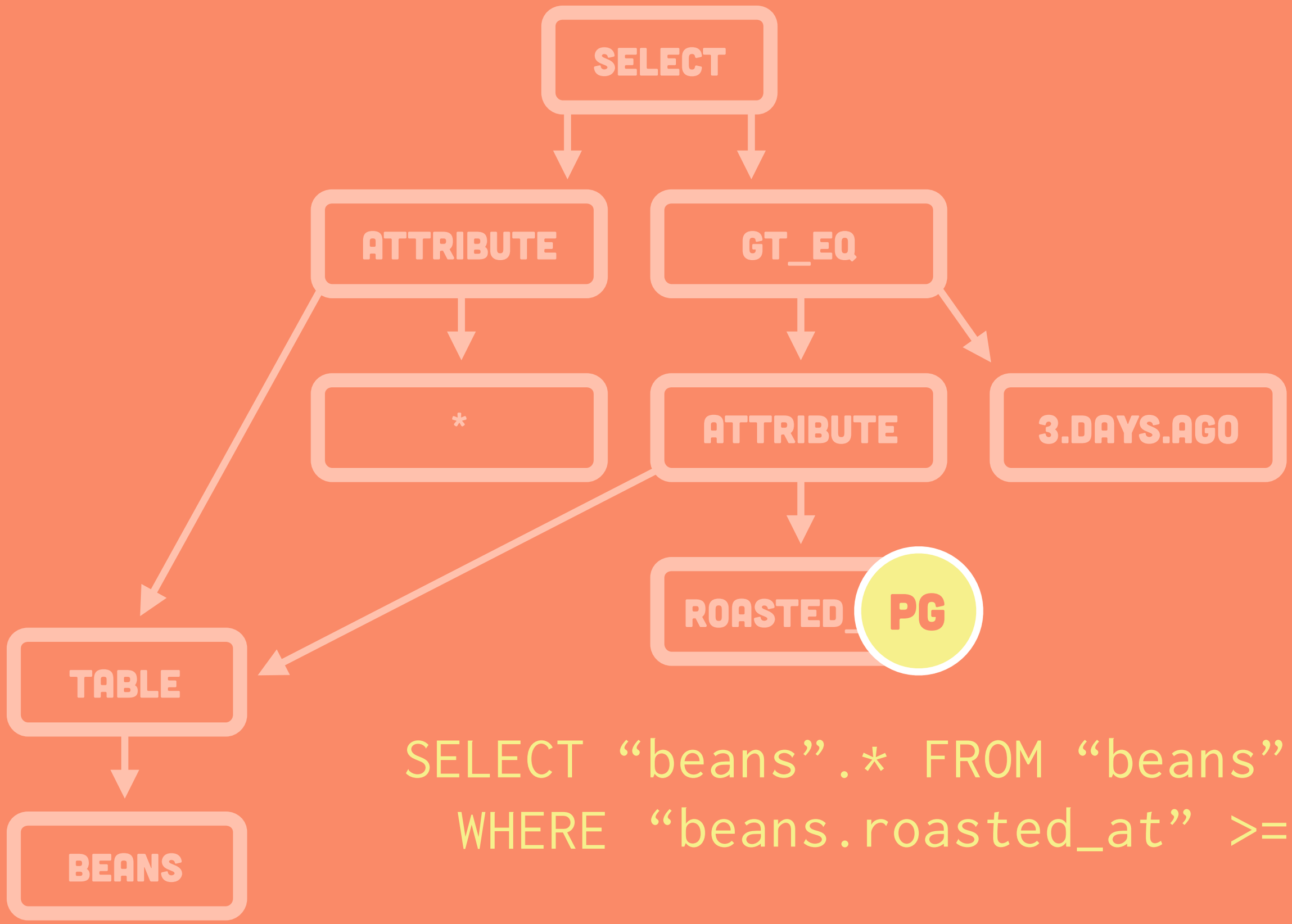

```
SELECT “beans”.* FROM “beans”
  WHERE                    >=
```



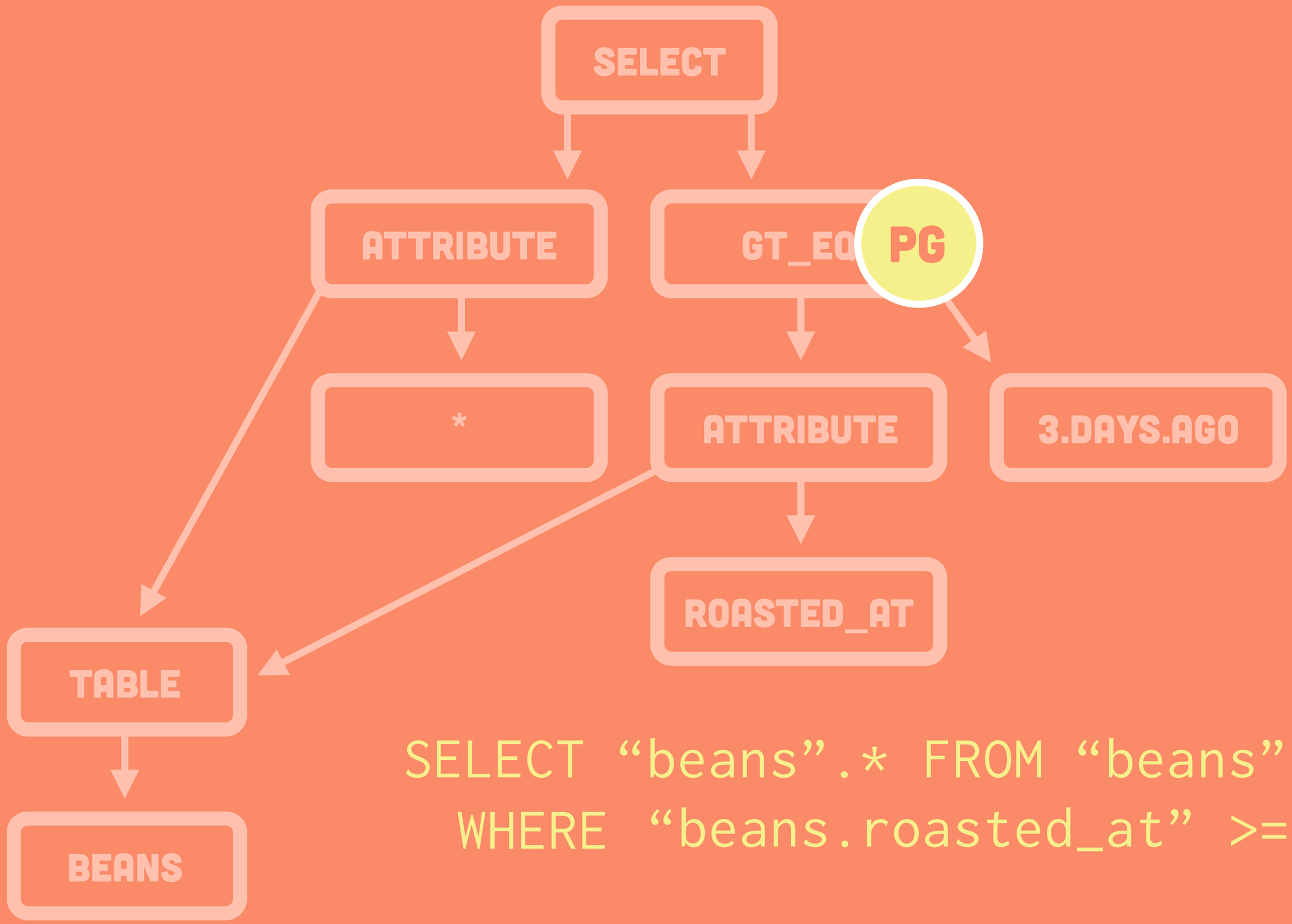

```
SELECT “beans”.* FROM “beans”
  WHERE “beans.                >=
```



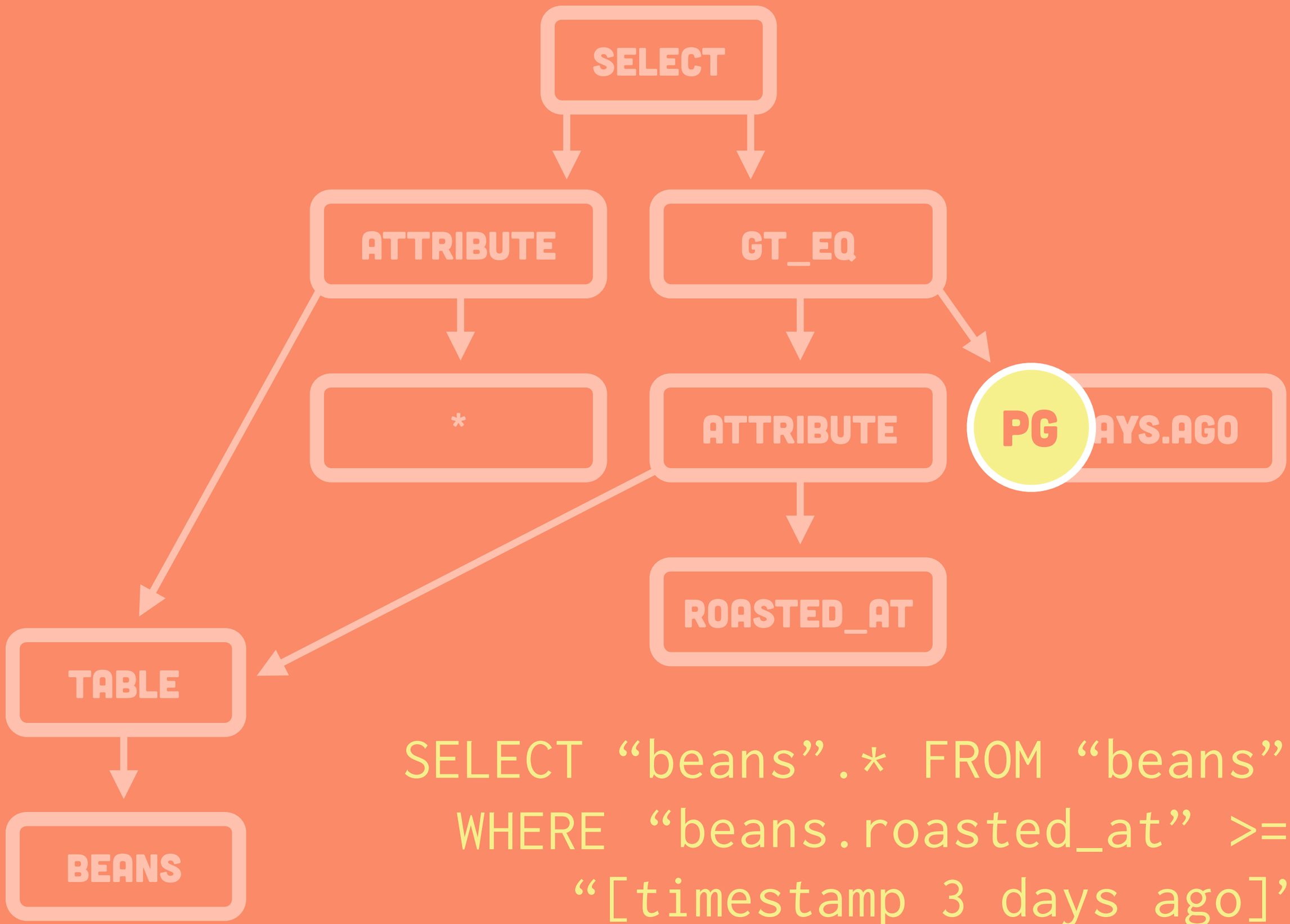

```
SELECT “beans”.* FROM “beans”
  WHERE “beans.                >=
```




```
SELECT “beans”.* FROM “beans”
  WHERE “beans.roasted_at” >=
```



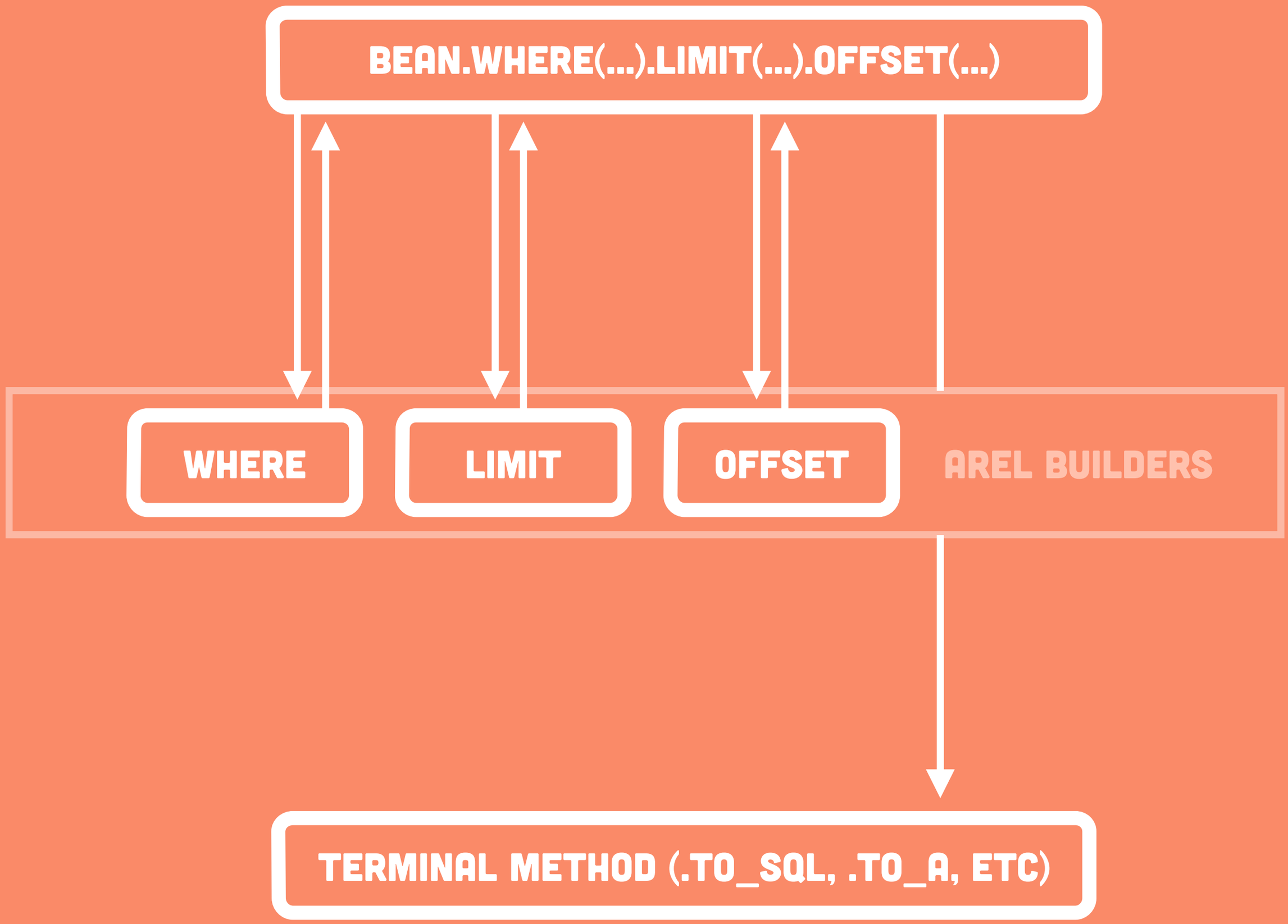

```
SELECT “beans”.* FROM “beans”
  WHERE “beans.roasted_at” >=
```

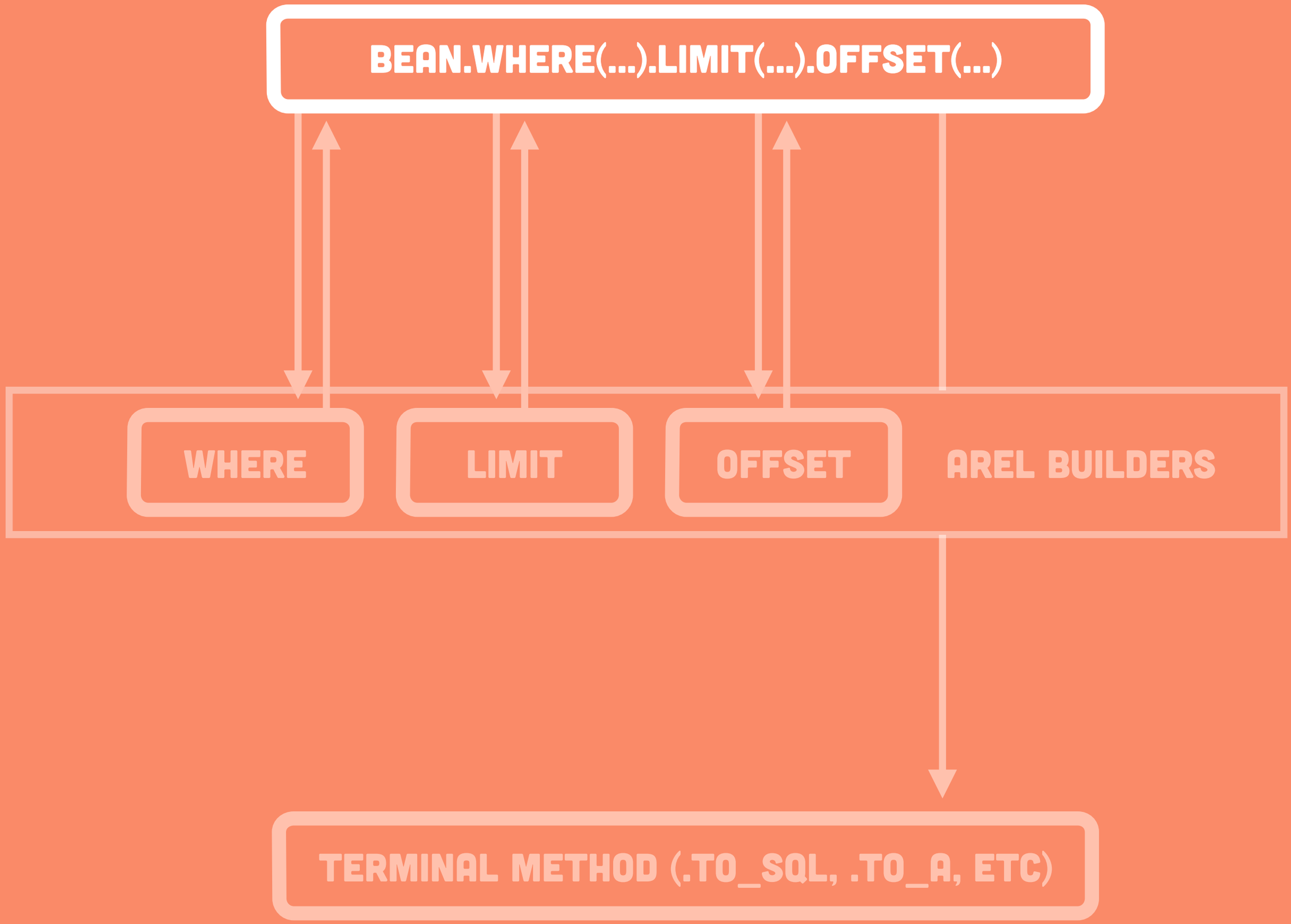


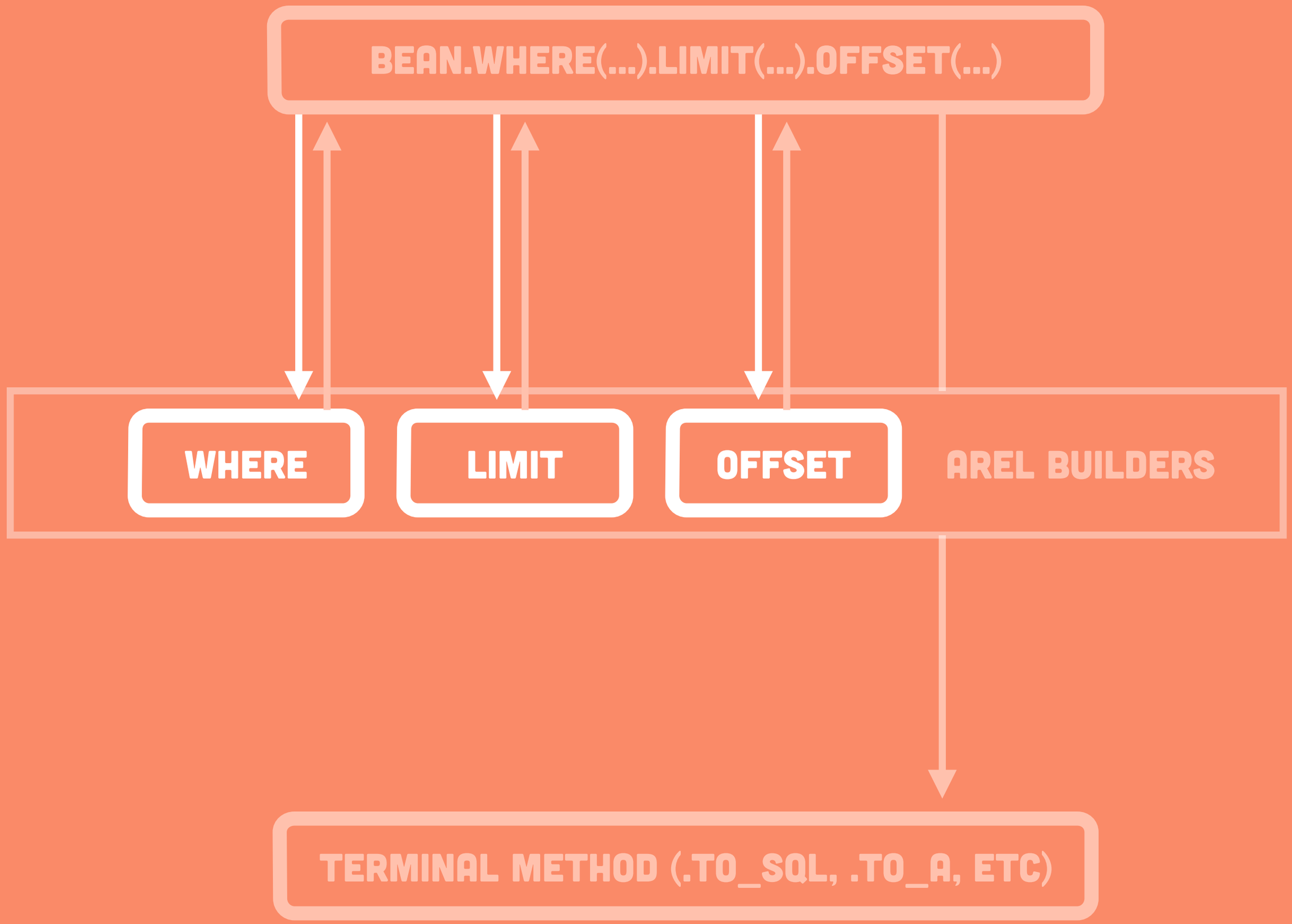

```
SELECT “beans”.* FROM “beans”
  WHERE “beans.roasted_at” >=
      “[timestamp 3 days ago]”
```

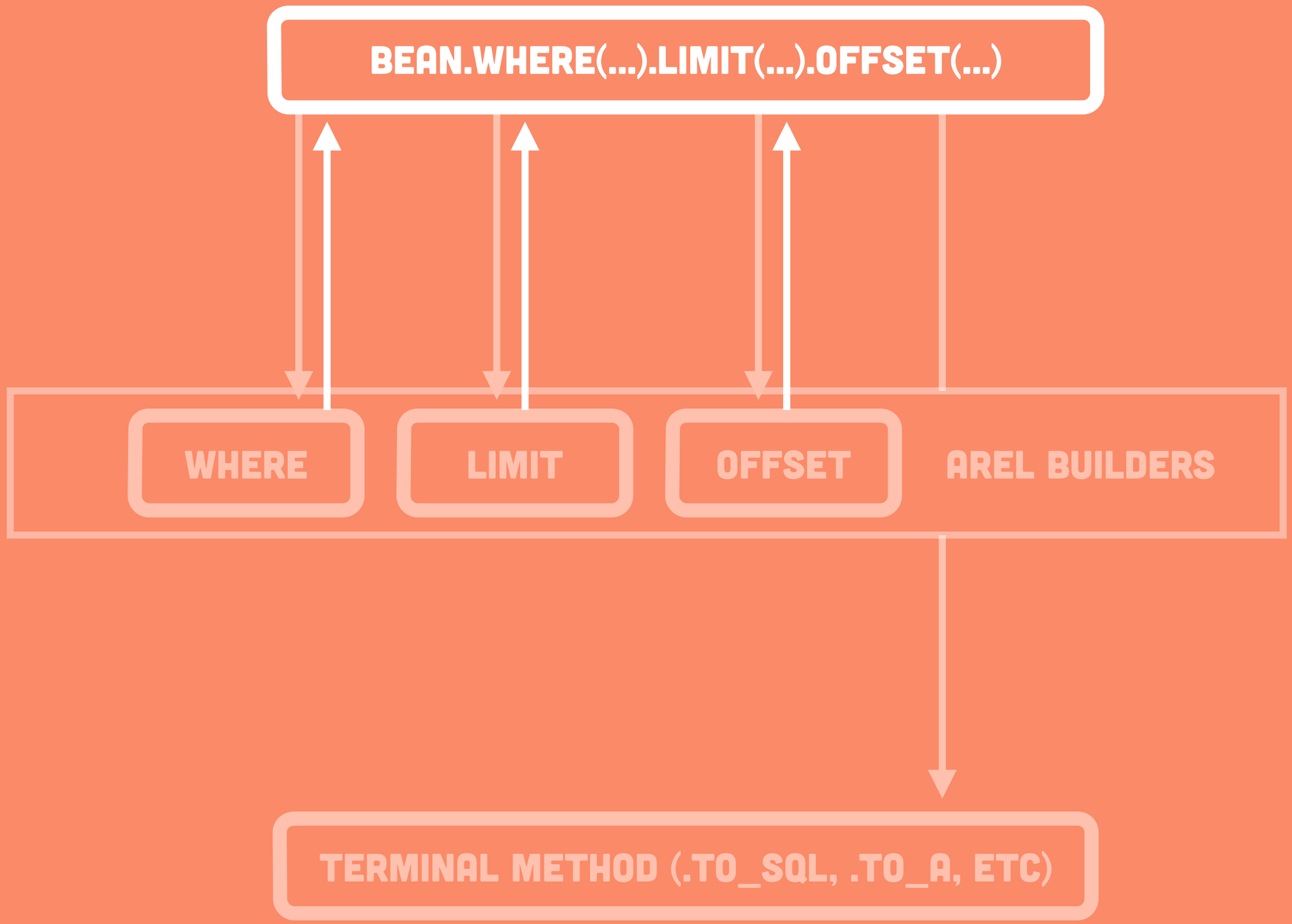
ACTIVERECORD::RELATION

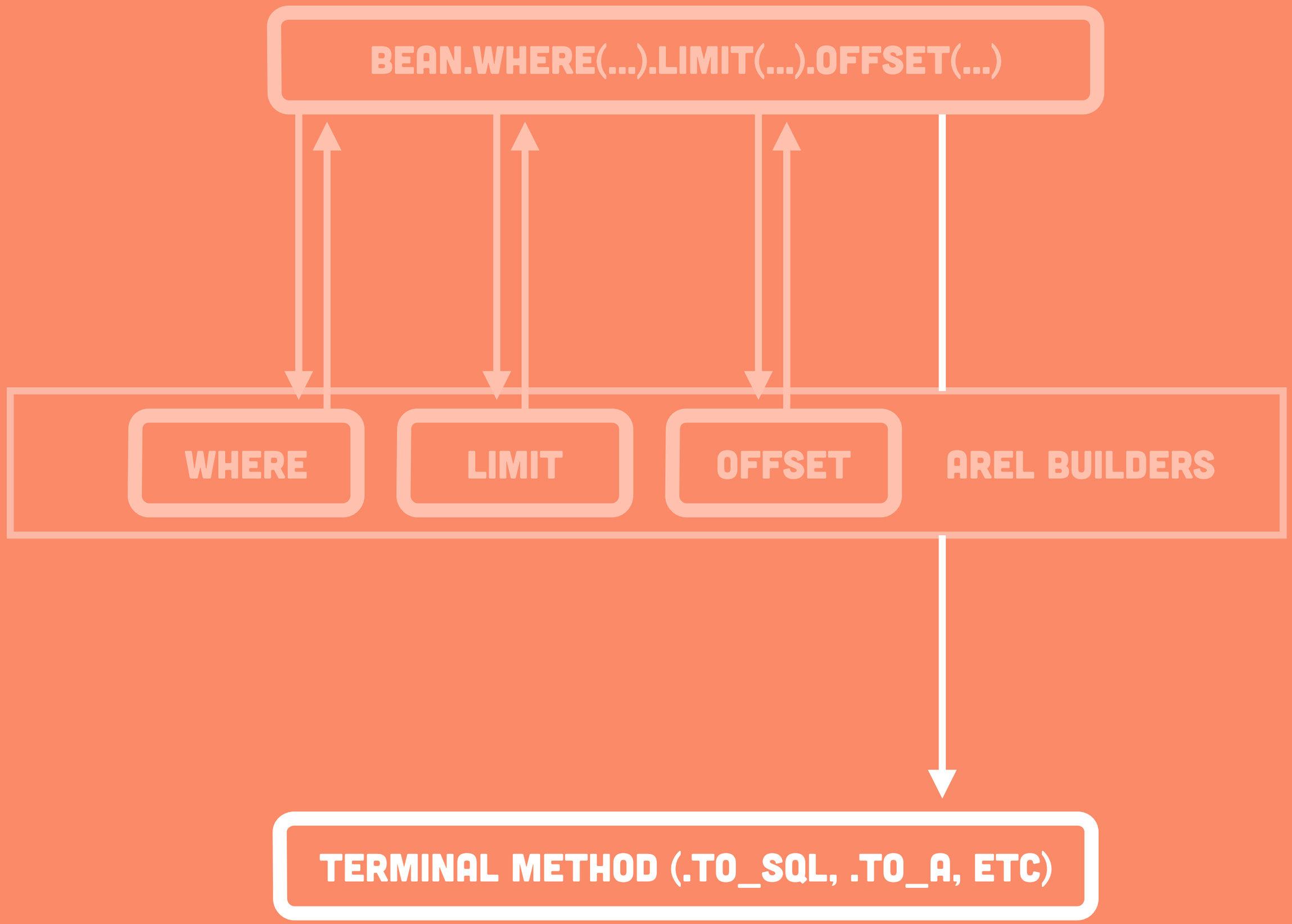
IN NAMING, ALWAYS FAVOR CLEVERNESS OVER CLARITY.


BEAN.WHERE(...).LIMIT(...).OFFSET(...)
WHERE
LIMIT
OFFSET
AREL BUILDERS
TERMINAL METHOD (.TO_SQL, .TO_A, ETC)


BEAN.WHERE(...).LIMIT(...).OFFSET(...)
WHERE
LIMIT
OFFSET
AREL BUILDERS
TERMINAL METHOD (.TO_SQL, .TO_A, ETC)


BEAN.WHERE(...).LIMIT(...).OFFSET(...)
WHERE
LIMIT
OFFSET
AREL BUILDERS
TERMINAL METHOD (.TO_SQL, .TO_A, ETC)


BEAN.WHERE(...).LIMIT(...).OFFSET(...)
WHERE
LIMIT
OFFSET
AREL BUILDERS
TERMINAL METHOD (.TO_SQL, .TO_A, ETC)


BEAN.WHERE(...).LIMIT(...).OFFSET(...)
WHERE
LIMIT
OFFSET
AREL BUILDERS
TERMINAL METHOD (.TO_SQL, .TO_A, ETC)

HOW DOES IT WORK?

BY EXPOSING METHODS TO BUILD AN ABSTRACT SYNTAX TREE, THEN USING A VISITOR TO GENERATE SQL STRINGS.

I. WHAT'S AN AREL?

II. WHEN SHOULD I USE IT?

III. HOW DOES IT WORK?

IV. BUT ISN'T THE CODE MESSY?

There's a thin line very quickly crossed where a query written in pure Arel is way more opaque and confusing than the SQL it generates.

@TWMILLS

```
Bean.
  arel_table.project(Bean.arel_table[Arel.star]).
  join(Roaster.arel_table).on(Roaster.arel_table[:id].eq(Bean.arel_table[:roaster_id])).
  join(City.arel_table).on(City.arel_table[:id].eq(Roaster.arel_table[:city_id])).
  join(Rating.arel_table.alias("bean_ratings")).on(
    Arel::Table.new(:bean_ratings)[:type].eq("Bean"),
    Arel::Table.new(:bean_ratings)[:target_id].eq(Bean.arel_table[:id])
  ).
  join(Rating.arel_table.alias("roaster_ratings")).on(
    Arel::Table.new(:roaster_ratings)[:type].eq("Roaster"),
    Arel::Table.new(:roaster_ratings)[:target_id].eq(Roaster.arel_table[:id])
  ).
  where(Bean.arel_table[:roasted_at].gteq(3.days.ago)).
  where(Roaster.arel_table[:city_id].not_eq(5)).
  where(
    Bean.arel_table[:flavors].matches("%sweet%").or(
    Bean.arel_table[:flavors].eq(nil))
  ).
  having(
    Arel::Table.new(:bean_ratings)[:value].average.gteq(90).or(
    Arel::Table.new(:roaster_ratings)[:value].average.gteq(90).or(
    Arel::Table.new(:bean_ratings)[:id].count.eq(0).or(
    Arel::Table.new(:roaster_ratings)[:id].count.eq(0))))
  ).
  group(
    Bean.arel_table[:id],
    Roaster.arel_table[:name]
  ).
  order(Roaster.arel_table[:name].desc)
```

YES.

(BUT WE CAN DO BETTER)

CONVENIENCE METHODS
DESCRIPTIVE METHODS
MIX & MATCH ACTIVERECORD

CONVENIENCE METHODS

```
ModelName.arel_table[:attribute]
Arel::Table.new(table_name)
```

```
Bean.
  arel_table.project(Bean.arel_table[Arel.star]).
  join(Roaster.arel_table).on(Roaster.arel_table[:id].eq(Bean.arel_table[:roaster_id])).
  join(City.arel_table).on(City.arel_table[:id].eq(Roaster.arel_table[:city_id])).
  join(Rating.arel_table.alias("bean_ratings")).on(
    Arel::Table.new(:bean_ratings)[:type].eq("Bean"),
    Arel::Table.new(:bean_ratings)[:target_id].eq(Bean.arel_table[:id])
  ).
  join(Rating.arel_table.alias("roaster_ratings")).on(
    Arel::Table.new(:roaster_ratings)[:type].eq("Roaster"),
    Arel::Table.new(:roaster_ratings)[:target_id].eq(Roaster.arel_table[:id])
  ).
  where(Bean.arel_table[:roasted_at].gteq(3.days.ago)).
  where(Roaster.arel_table[:city_id].not_eq(5)).
  where(
    Bean.arel_table[:flavors].matches("%sweet%").or(
    Bean.arel_table[:flavors].eq(nil))
  ).
  having(
    Arel::Table.new(:bean_ratings)[:value].average.gteq(90).or(
    Arel::Table.new(:roaster_ratings)[:value].average.gteq(90).or(
    Arel::Table.new(:bean_ratings)[:id].count.eq(0).or(
    Arel::Table.new(:roaster_ratings)[:id].count.eq(0))))
  ).
  group(
    Bean.arel_table[:id],
    Roaster.arel_table[:name]
  ).
  order(Roaster.arel_table[:name].desc)
```

CONVENIENCE METHODS

```
def beans
  Bean.arel_table
end

# more names?
# _, table, beans_table,
# arel_table, arel
```

CONVENIENCE METHODS

```
def alias_table(name)
  Arel::Table.new(name).alias(name)
end
```

CONVENIENCE METHODS

```
def beans;          Bean.arel_table;                  end
def ratings;        Rating.arel_table;                end
def roasters;       Roaster.arel_table;               end
def cities;         City.arel_table;                  end
def bean_ratings;   alias_table(:bean_ratings);   end
def roaster_ratings; alias_table(:roaster_ratings); end

def self.alias_table(name)
  Arel::Table.new(name.to_s).alias(name.to_s)
end
```

```
Bean.
  arel_table.project(Bean.arel_table[Arel.star]).
  join(Roaster.arel_table).on(Roaster.arel_table[:id].eq(Bean.arel_table[:roaster_id])).
  join(City.arel_table).on(City.arel_table[:id].eq(Roaster.arel_table[:city_id])).
  join(Rating.arel_table.alias("bean_ratings")).on(
    Arel::Table.new(:bean_ratings)[:type].eq("Bean"),
    Arel::Table.new(:bean_ratings)[:target_id].eq(Bean.arel_table[:id])
  ).
  join(Rating.arel_table.alias("roaster_ratings")).on(
    Arel::Table.new(:roaster_ratings)[:type].eq("Roaster"),
    Arel::Table.new(:roaster_ratings)[:target_id].eq(Roaster.arel_table[:id])
  ).
  where(Bean.arel_table[:roasted_at].gteq(3.days.ago)).
  where(Roaster.arel_table[:city_id].not_eq(5)).
  where(
    Bean.arel_table[:flavors].matches("%sweet%").or(
    Bean.arel_table[:flavors].eq(nil))
  ).
  having(
    Arel::Table.new(:bean_ratings)[:value].average.gteq(90).or(
    Arel::Table.new(:roaster_ratings)[:value].average.gteq(90).or(
    Arel::Table.new(:bean_ratings)[:id].count.eq(0).or(
    Arel::Table.new(:roaster_ratings)[:id].count.eq(0))))
  ).
  group(
    Bean.arel_table[:id],
    Roaster.arel_table[:name]
  ).
  order(Roaster.arel_table[:name].desc)
```

```
beans.
  project(beans[Arel.star]).
  join(roasters).on(roasters[:id].eq(beans[:roaster_id])).
  join(cities).on(cities[:id].eq(roasters[:city_id])).
  join(bean_ratings).on(
    bean_ratings[:type].eq("Bean"),
    bean_ratings[:target_id].eq(beans[:id])
  ).
  join(roaster_ratings).on(
    roaster_ratings[:type].eq("Roaster"),
    roaster_ratings[:target_id].eq(roasters[:id])
  ).
where(beans[:roasted_at].gteq(3.days.ago)).
where(roasters[:city_id].not_eq(5)).
where(beans[:flavors].matches("%sweet%").or(beans[:flavors].eq(nil))).
having(
  bean_ratings[:value].average.gteq(90).or(
  roaster_ratings[:value].average.gteq(90).or(
  bean_ratings[:id].count.eq(0).or(
  roaster_ratings[:id].count.eq(0))))
).
group(beans[:id], roasters[:name]).
order(roasters[:name].desc)
```

DESCRIPTIVE METHODS

WE'RE JUST WORKING WITH RUBY. WE KNOW HOW TO EXTRACT-METHOD.

DESCRIPTIVE METHODS

```
# .having(high_ratings)
def high_ratings(threshold = 90)
  bean_ratings[:value].average.gteq(threshold).or(
  roaster_ratings[:value].average.gteq(threshold))
end
```

DESCRIPTIVE METHODS

```
# .join(table).on(*beans_to_ratings)
def beans_to_ratings
  [
    bean_ratings[:model_type].eq(“Bean"),
    bean_ratings[:model_id].eq(beans[:id])
  ]
end
```

```
beans.
  project(beans[Arel.star]).
  join(roasters).on(roasters[:id].eq(beans[:roaster_id])).
  join(cities).on(cities[:id].eq(roasters[:city_id])).
  join(bean_ratings).on(
    bean_ratings[:type].eq("Bean"),
    bean_ratings[:target_id].eq(beans[:id])
  ).
  join(roaster_ratings).on(
    roaster_ratings[:type].eq("Roaster"),
    roaster_ratings[:target_id].eq(roasters[:id])
  ).
where(beans[:roasted_at].gteq(3.days.ago)).
where(roasters[:city_id].not_eq(5)).
where(beans[:flavors].matches("%sweet%").or(beans[:flavors].eq(nil))).
having(
  bean_ratings[:value].average.gteq(90).or(
  roaster_ratings[:value].average.gteq(90).or(
  bean_ratings[:id].count.eq(0).or(
  roaster_ratings[:id].count.eq(0))))
).
group(beans[:id], roasters[:name]).
order(roasters[:name].desc)
```

```
beans.
  project(beans[Arel.star]).
  join(roasters).on(beans_to_roasters).
  join(cities).on(cities_to_roasters).
  join(beans.alias(bean_ratings.name)).on(*beans_to_ratings).
  join(roasters.alias(roaster_ratings.name)).on(*roasters_to_ratings).
  where(roasted_within(3)).
  where(roasters[:city_id].not_eq(5)).
  where(flavor_matches("sweet").or(blank_flavor)).
  having(high_ratings.or(no_ratings)).
  group(beans[:id], roasters[:name]).
  order(roasters[:name].desc)
```

MIX AND MATCH

ACTIVERECORD SPEAKS AREL FLUENTLY.

MIX AND MATCH

```
# #<Arel::Nodes::InnerJoin>
a_join = beans.join(ratings).on(…).join_sources
Bean.joins(a_join)
```

MIX AND MATCH

```
# #<Arel::Nodes::GreaterThanOrEqual>
a_where = beans[:roasted_at].gteq(3.days.ago)
Bean.where(a_where)
```

MIX AND MATCH

```
def bean_ratings_join
  beans.join(ratings.alias(bean_ratings.name)).on(
    bean_ratings[:type].eq("Bean"),
    bean_ratings[:target_id].eq(beans[:id])
  ).join_sources
end

Bean.joins(:roaster, bean_ratings_join)
```

MIX AND MATCH

```
def bean_ratings_join
  beans.join(ratings.alias(bean_ratings.name)).on(
    bean_ratings[:type].eq("Bean"),
    bean_ratings[:target_id].eq(beans[:id])
  ).join_sources
end

Bean.joins(:roaster, bean_ratings_join)
```

MIX AND MATCH

```
def bean_ratings_join
  beans.join(ratings.alias(bean_ratings.name)).on(
    bean_ratings[:type].eq("Bean"),
    bean_ratings[:target_id].eq(beans[:id])
  ).join_sources
end

Bean.joins(:roaster, bean_ratings_join)
```

MIX AND MATCH

```
def roasted_since(roasted_since)
  beans[:roasted_at].gteq(roasted_since)
end
```

```
Bean.joins(:roaster, bean_ratings_join).
     where(roasted_since(3.days.ago))
```

MIX AND MATCH

```
def roasted_since(roasted_since)
  beans[:roasted_at].gteq(roasted_since)
end
```

```
Bean.joins(:roaster, bean_ratings_join).
     where(roasted_since(3.days.ago))
```

```
beans.
  project(beans[Arel.star]).
  join(roasters).on(beans_to_roasters).
  join(cities).on(cities_to_roasters).
  join(beans.alias(bean_ratings.name)).on(*beans_to_ratings).
  join(roasters.alias(roaster_ratings.name)).on(*roasters_to_ratings).
  where(roasted_within(3)).
  where(roasters[:city_id].not_eq(5)).
  where(flavor_matches("sweet").or(blank_flavor)).
  having(high_ratings.or(no_ratings)).
  group(beans[:id], roasters[:name]).
  order(roasters[:name].desc)
```

```
Bean.
  joins(
    {roaster: :city},
    bean_ratings_join,
    roaster_ratings_join
  ).
having(high_ratings.or(no_ratings)).
roasted_since(3.days.ago).
not_city(5).
not_flavor("sweet").
group(beans[:id], roasters[:name]).
order(roasters[:name].desc)
```

```
Bean.
  arel_table.project(Bean.arel_table[Arel.star]).
  join(Roaster.arel_table).on(Roaster.arel_table[:id].eq(Bean.arel_table[:roaster_id])).
  join(City.arel_table).on(City.arel_table[:id].eq(Roaster.arel_table[:city_id])).
  join(Rating.arel_table.alias("bean_ratings")).on(
    Arel::Table.new(:bean_ratings)[:type].eq("Bean"),
    Arel::Table.new(:bean_ratings)[:target_id].eq(Bean.arel_table[:id])
  ).
  join(Rating.arel_table.alias("roaster_ratings")).on(
    Arel::Table.new(:roaster_ratings)[:type].eq("Roaster"),
    Arel::Table.new(:roaster_ratings)[:target_id].eq(Roaster.arel_table[:id])
  ).
  where(Bean.arel_table[:roasted_at].gteq(3.days.ago)).
  where(Roaster.arel_table[:city_id].not_eq(5)).
  where(
    Bean.arel_table[:flavors].matches("%sweet%").or(
    Bean.arel_table[:flavors].eq(nil))
  ).
  having(
    Arel::Table.new(:bean_ratings)[:value].average.gteq(90).or(
    Arel::Table.new(:roaster_ratings)[:value].average.gteq(90).or(
    Arel::Table.new(:bean_ratings)[:id].count.eq(0).or(
    Arel::Table.new(:roaster_ratings)[:id].count.eq(0))))
  ).
  group(
    Bean.arel_table[:id],
    Roaster.arel_table[:name]
  ).
  order(Roaster.arel_table[:name].desc)
```

```
Bean.
  joins(
    {roaster: :city},
    bean_ratings_join,
    roaster_ratings_join
  ).
having(high_ratings.or(no_ratings)).
roasted_since(3.days.ago).
not_city(5).
not_flavor("sweet").
group(beans[:id], roasters[:name]).
order(roasters[:name].desc)
```

WHAT'S AN AREL?

AN IMPLEMENTATION OF RELATIONAL ALGEBRA, USED FOR PROGRAMATICALLY GENERATING SQL QUERIES.

WHEN SHOULD I USE IT?

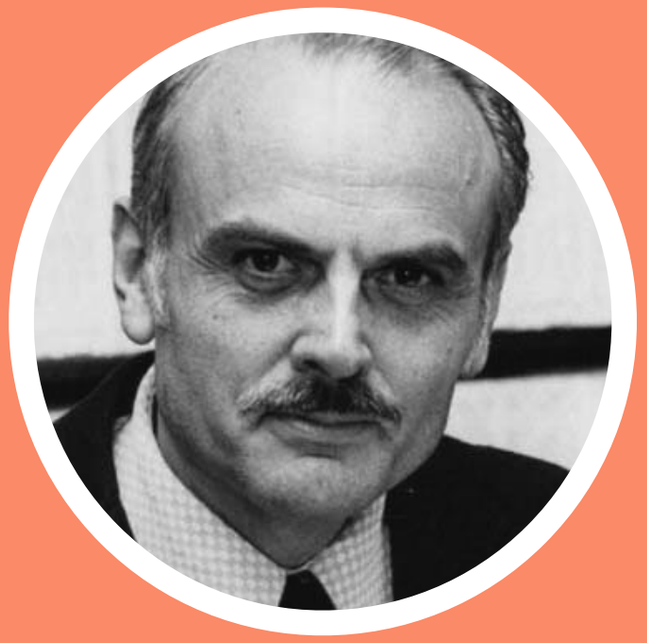
WHEN YOUR BESPOKE QUERIES ARE GETTING TACKY, YOU'RE GETTING REPETITIVE, OR YOU NEED MORE COMPOSABILITY.

HOW DOES IT WORK?

BY EXPOSING METHODS TO BUILD AN ABSTRACT SYNTAX TREE, THEN USING A VISITOR TO GENERATE SQL STRINGS.

BUT ISN'T THE CODE MESSY?

ONLY IF YOU DON'T GIVE IT THE SAME LOVE AND ATTENTION YOU ALREADY GIVE TO THE REST OF YOUR CODE.



THANKS!

FROM ME AND EDGAR