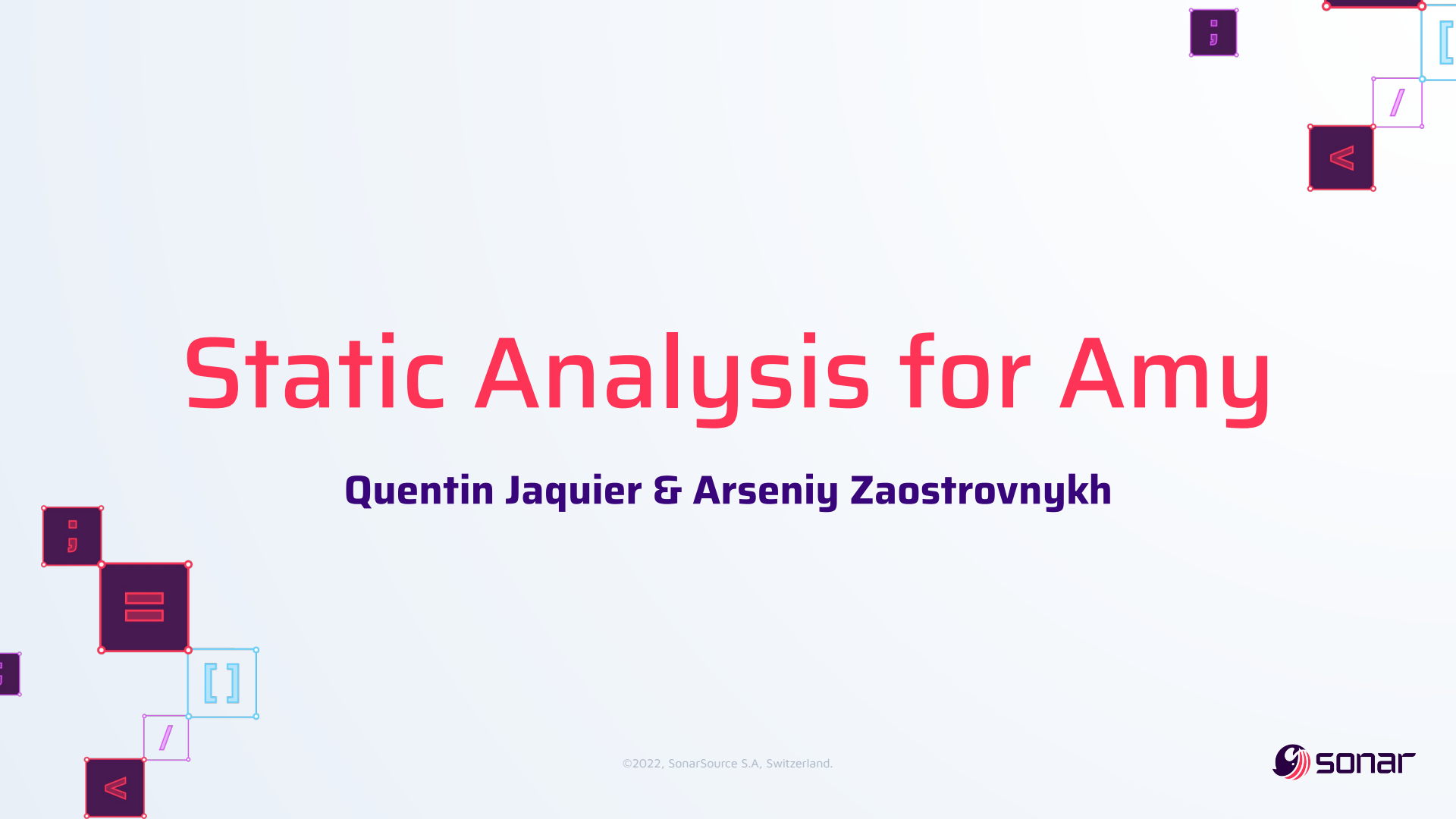


Static Analysis for Amy

Quentin Jaquier & Arseniy Zaostrovnykh

sonar

Quentin Jaquier

EPFL Alumni Master (2019)

sonar

Arseniy Zaostrovnykh

EPFL Alumni Ph.D. 2020 (DSLAB)
Software Verification
Compilers

sonar

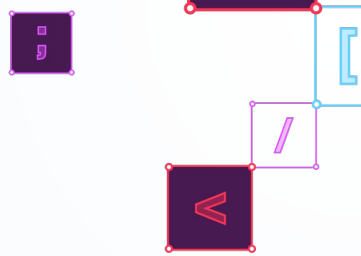
What is Sonar?

For developers and development teams

Geneva Austin Bochum Annecy Singapore; ~400 people

Code **quality & Security**

Mostly Open Source



What is Static Analysis?

Detecting code smells, bugs and vulnerabilities in the code without executing it.

sonar

Why do Static Analysis?

Reliability

Security

Maintainability

Lower security and operation risks: downtime, breaches

Lower maintenance costs: bug fixes and features are easier

What's Wrong With Tests?

Easy to forget some bug kinds

Impossible to test all execution scenarios

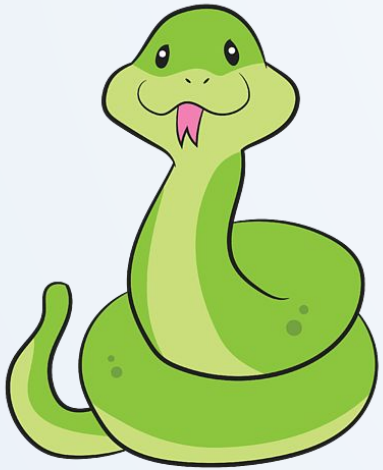
Miss code smells entirely

> Only check functionality and performance, *not* maintainability

Complementary

Why do Static Analysis?

NameError: name ‘true’ is not defined



after 20 hours of running script.py

script.py

```
#!/usr/bin/python
prods = long_20h_computation()
with open("res.json", "w") as outf:
    outf.write(json.dumps(prods, sort_keys=true))
```

static analyzer
goes even deeper

Outline

First hour

Intro to static analysis

Place for static analysis

AST-based analysis

Visitors & Matchers

Second hour

Taint Analysis

Symbolic Execution

Static Analysis Trade-off

Demo

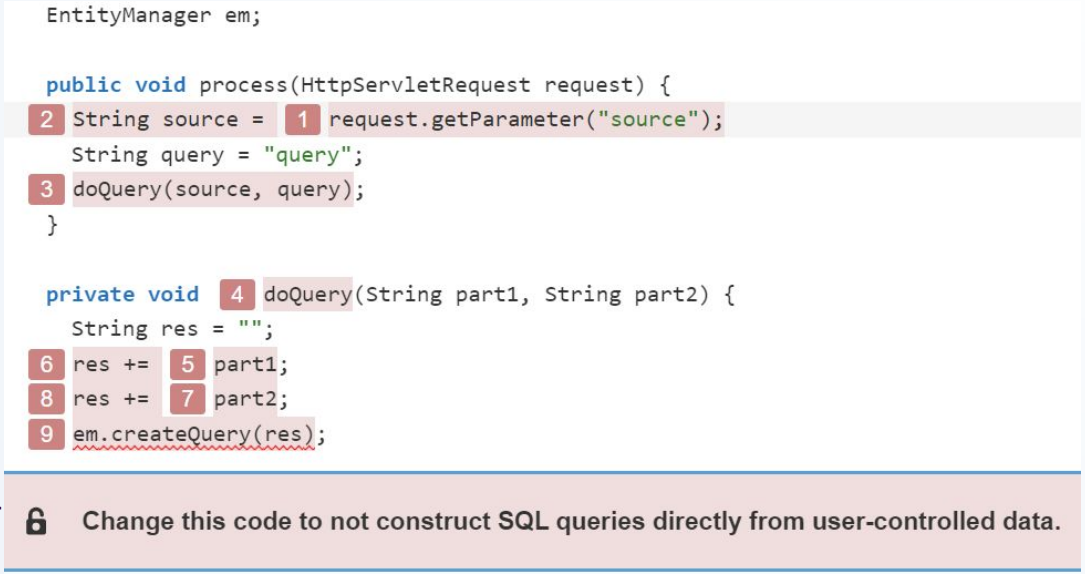
Static Analysis vs Compilers

Aren't compilers already doing this? Yes…

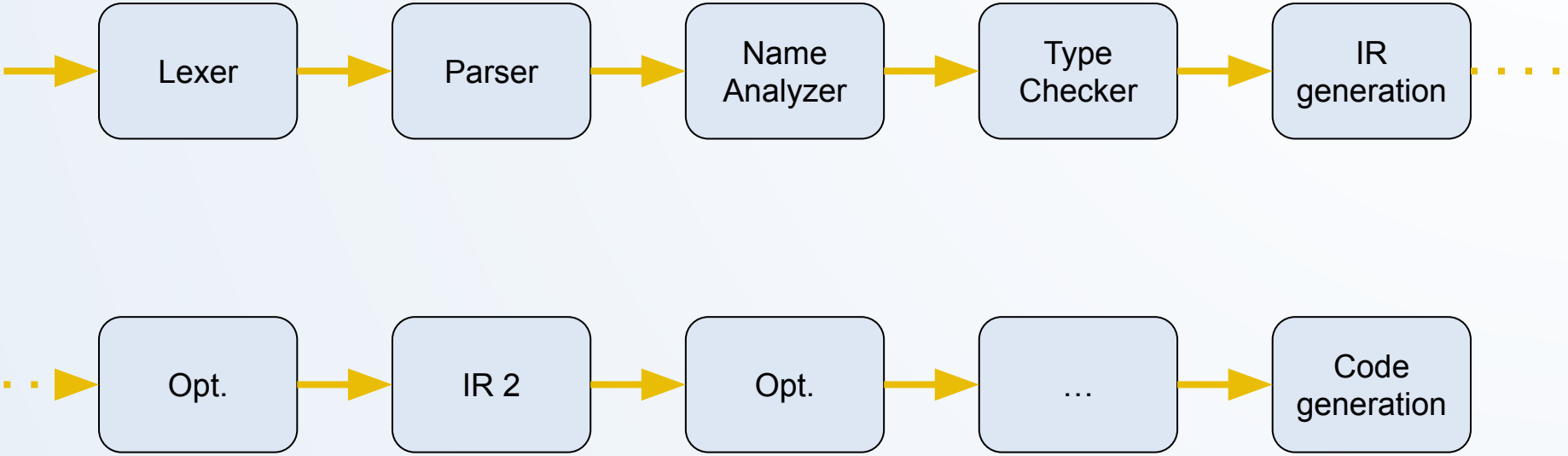
- Compilations errors
  - Illegale name shadowing
- Unused Variable

Similar, just with different constraints:

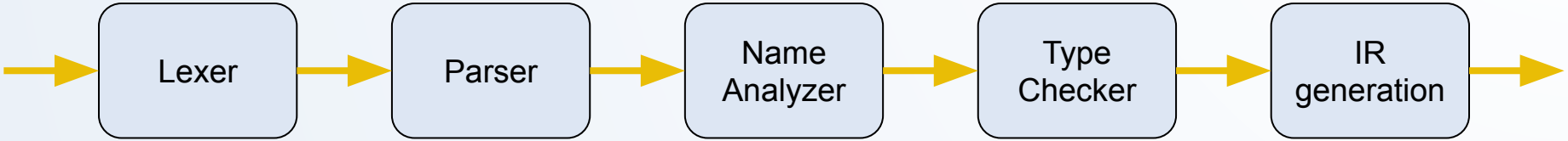
- Resources (People)
- Execution time
- Reporting: explain complex issue, tr
  output



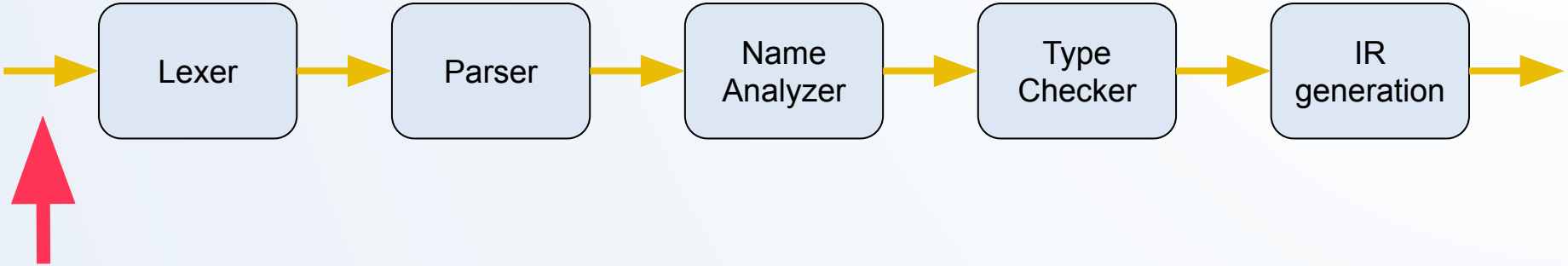
Where can we do static analysis?



Where can we do static analysis?

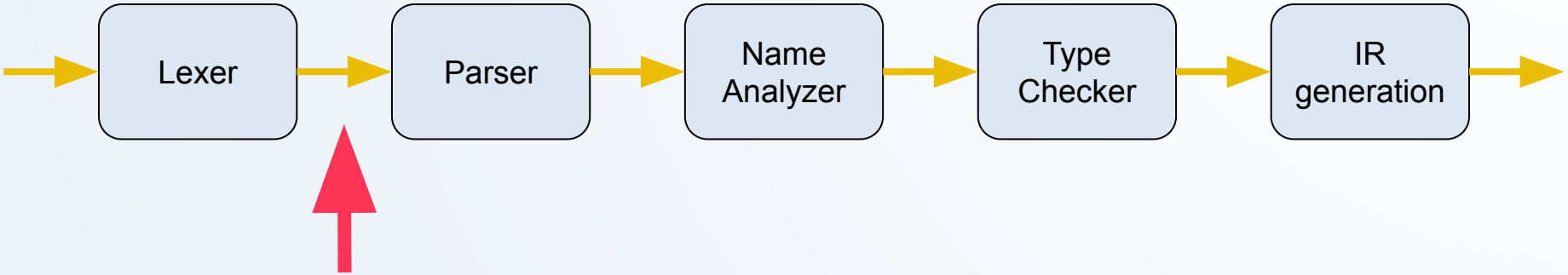


Before the Lexer?



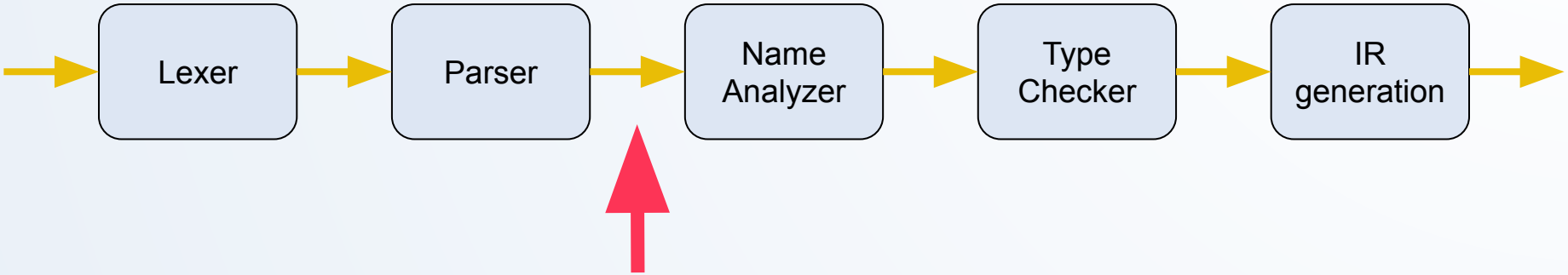
- Don’t use tabs - S105
- Don’t use “invisible” characters (*U+200B*, known as “zero width space”) - S2479

After the Lexer



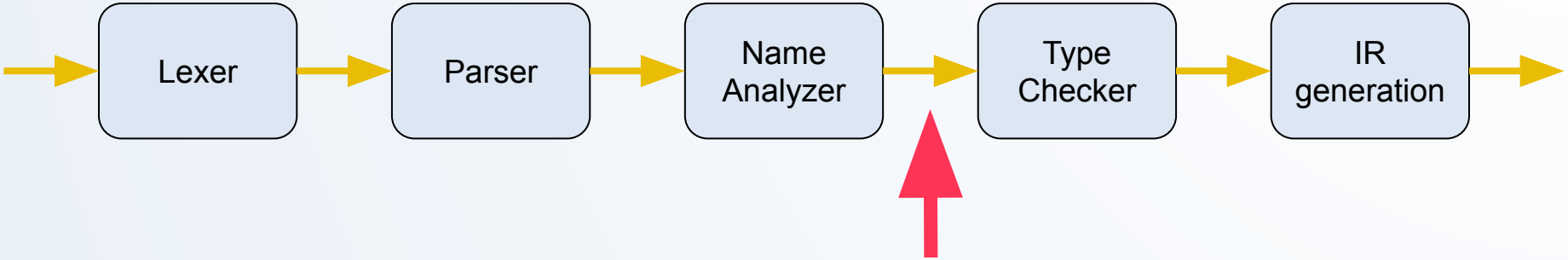
- Mixed comment style (// mixed with /* */) - S1917
- **TODO**/**FIXME** words - S1135

After the Parser



- Conventions rules
- Nested switch statement - S1821
- Cognitive/Cyclomatic Complexity - S3776

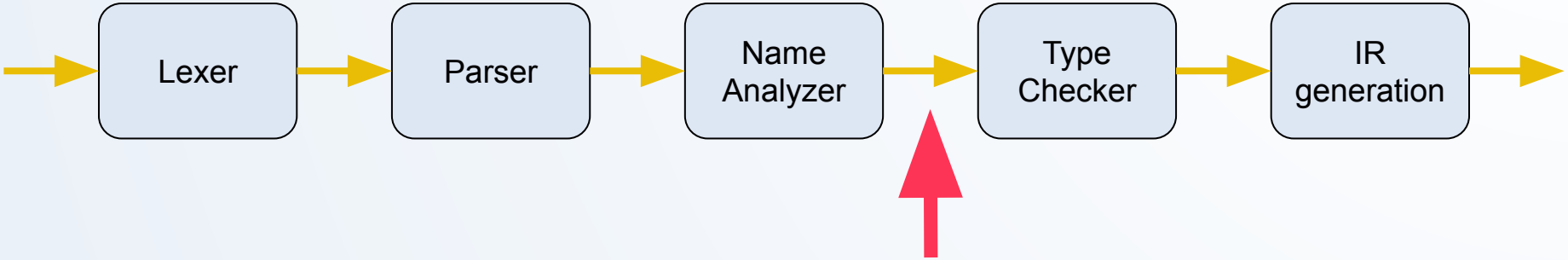
After the Name Analyzer



- Null pointer dereference - S2259

```
String s = null;

s.substring();
```

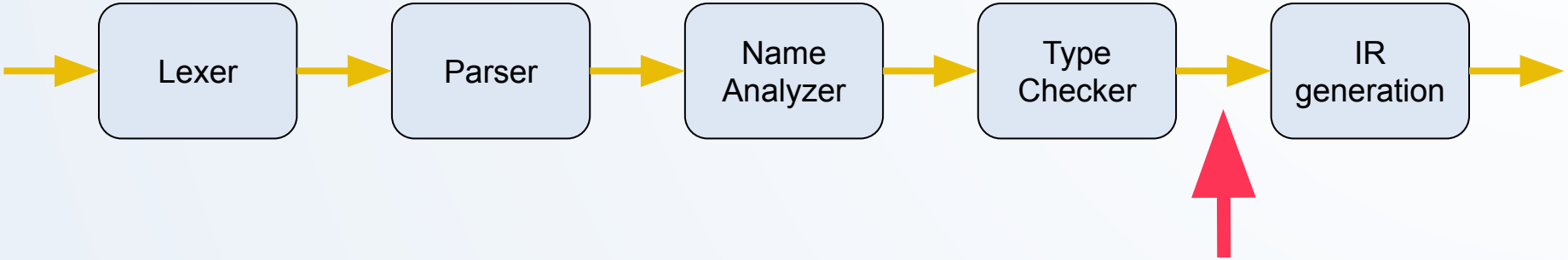
After the Name Analyzer



- Variable shadowing - S1524

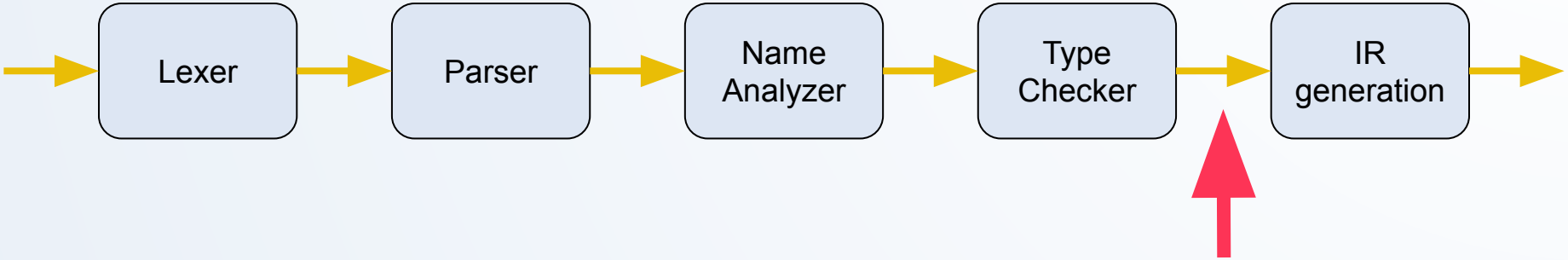
```
val x: Int = 1;
if (...) {
  val x: Int = 2;
  // ...
}
```

After the Type Checker



delux("something", "5c07")

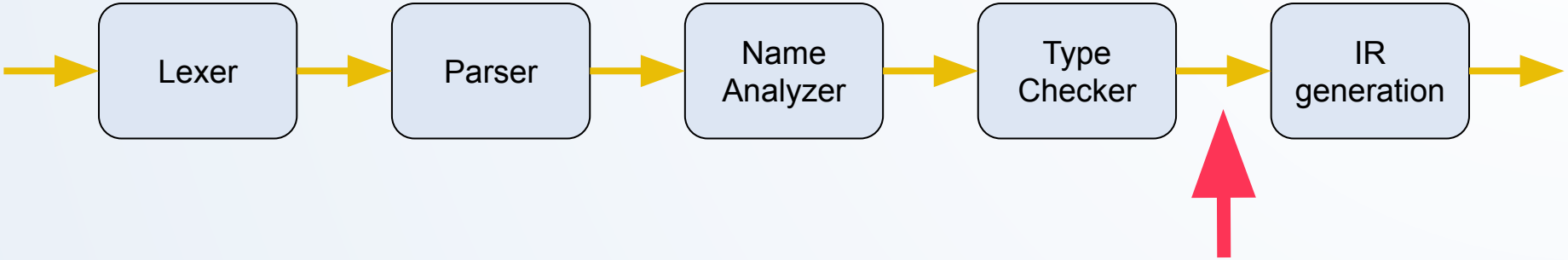
After the Type Checker



delux("something", "5c07")

springframework.security.**Encrypt**.delux(**String** password, **String** salt)

After the Type Checker

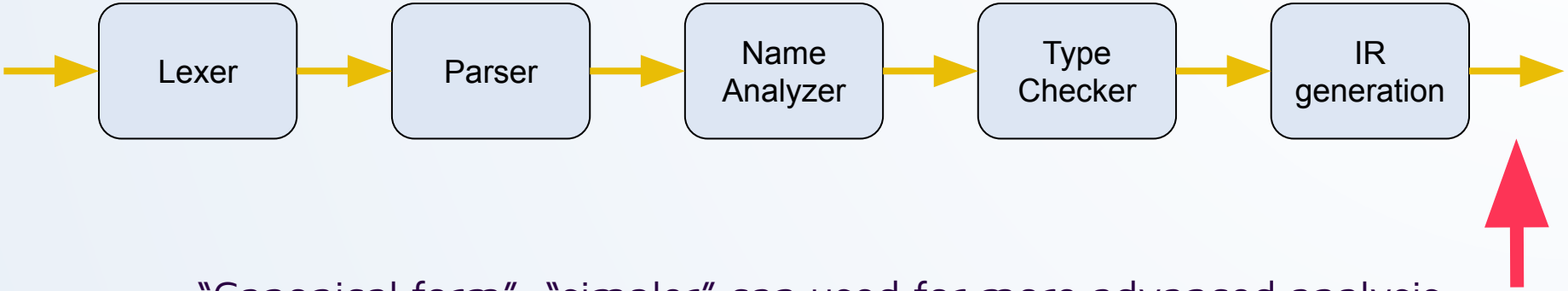


- Hard Coded credential - S2068

```
delux("something", "5c07")
```

springframework.security.**Encrypt**.delux(**String** password, **String** salt)

After IR generation



- “Canonical form”, “simpler” can used for more advanced analysis
  - Taint analysis and Symbolic execution
- Analysis of compiled dependencies (for Java)
- Write engine once, target multiple source languages (java, scala)

Outline

First hour

Intro to static analysis

Place for static analysis

AST-based analysis

- Visitors & Matchers

Second hour

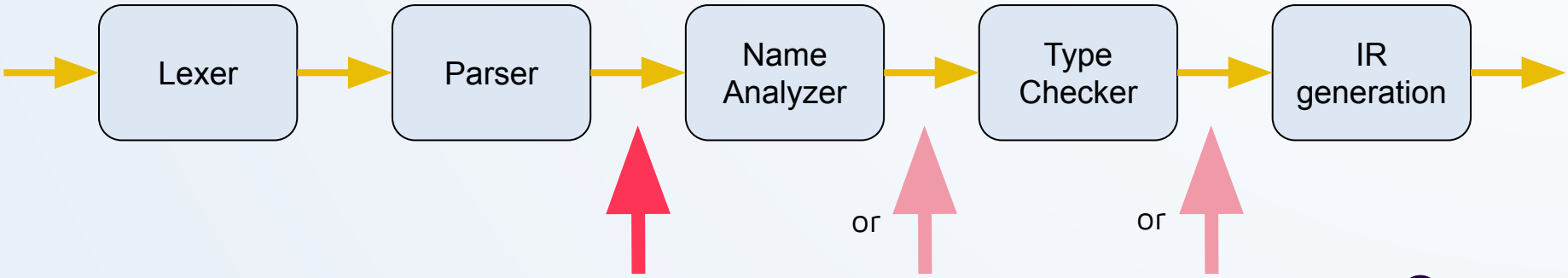
Taint Analysis

Symbolic Execution

Static Analysis Trade-off

Demo

AST Analysis



AST Analysis Concept

Traverse the AST

to detect patterns of a bug or a code smell

Keep token locations

to pinpoint found issues

Rule examples

If with a trivial condition

```
if (true) { ... } else { ... }
```

Redundant condition

```
if (x) { ... x ... } else { ... x ... }
```

Unused parameter

```
fn foo(x: Int(32), y: Int(32)) { x }
```

Running example

```
object Example

fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}

end Example
```

Running example

```
object Example

fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}

end Example
```

Running example

```
object Example

fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}

end Example
```

Running example

```
object Example

fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}

end Example
```

Running example

```
object Example

fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}

end Example
```

Running example

```
object Example

fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}

end Example
```

Running example

```
object Example

fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}

end Example
```

Running example

```
object Example

fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}

end Example
```

Running example

```
object Example

fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}

end Example
```

Running example

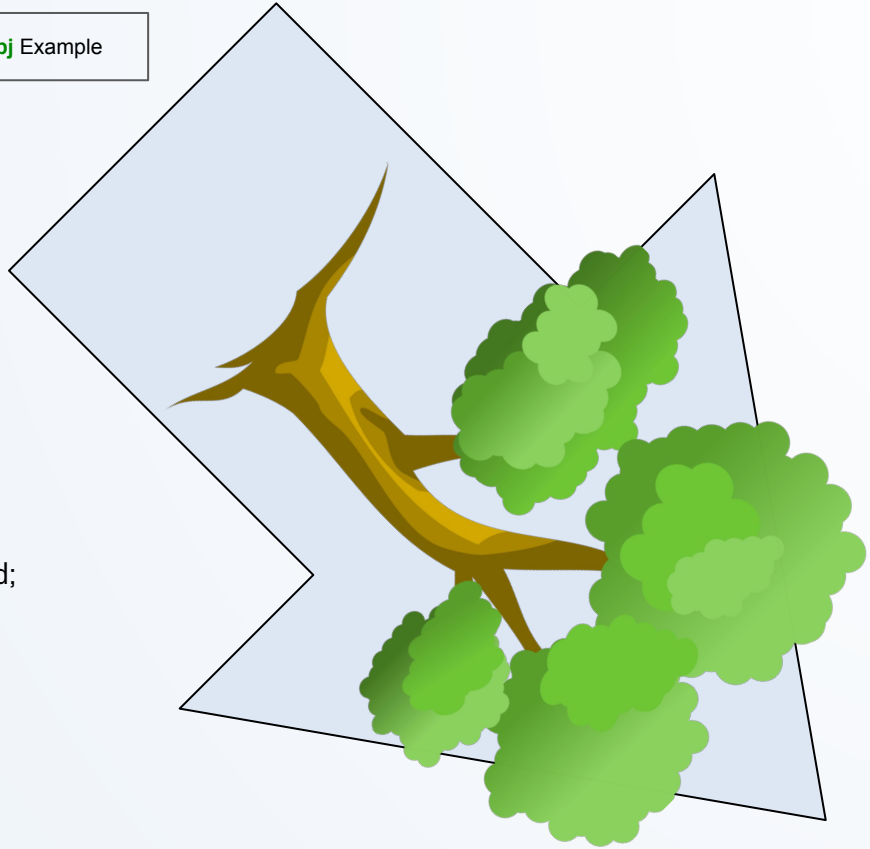
```
object Example

fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}

end Example
```

Running example

```
object Example

fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}

end Example
```

AST

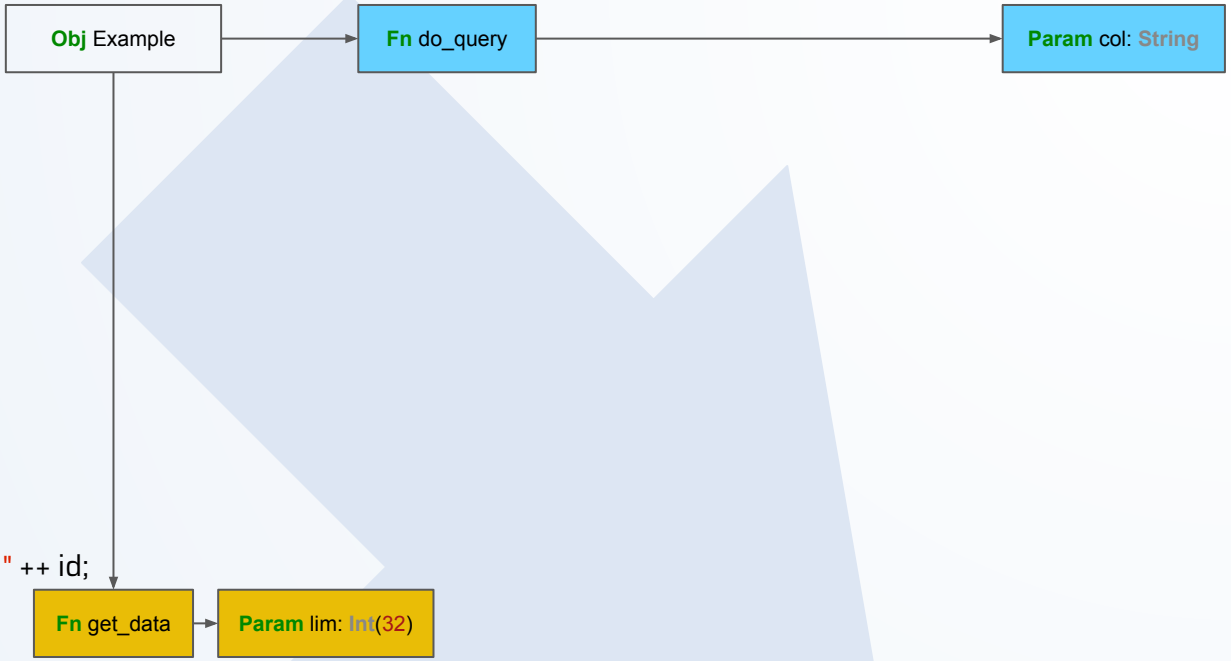
Obj Example



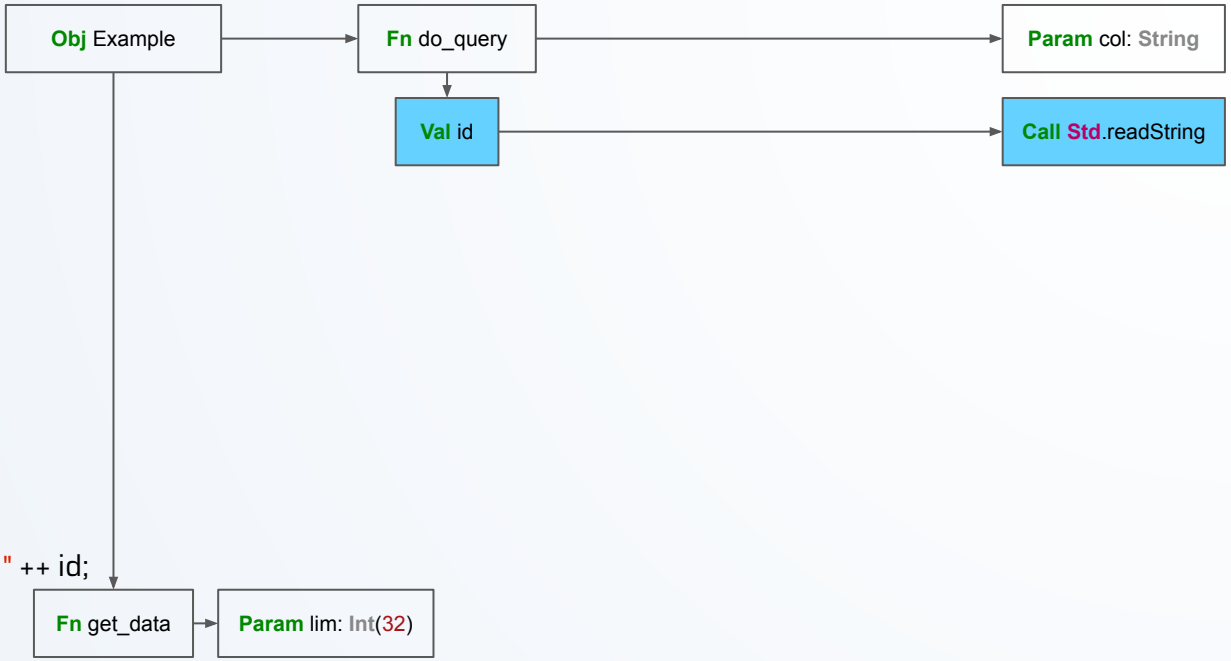
```
object Example
fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}

end Example
```

AST

```
object Example
fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String ={
  val column: String = Std.readString();
  do_query(column)
}

end Example
```



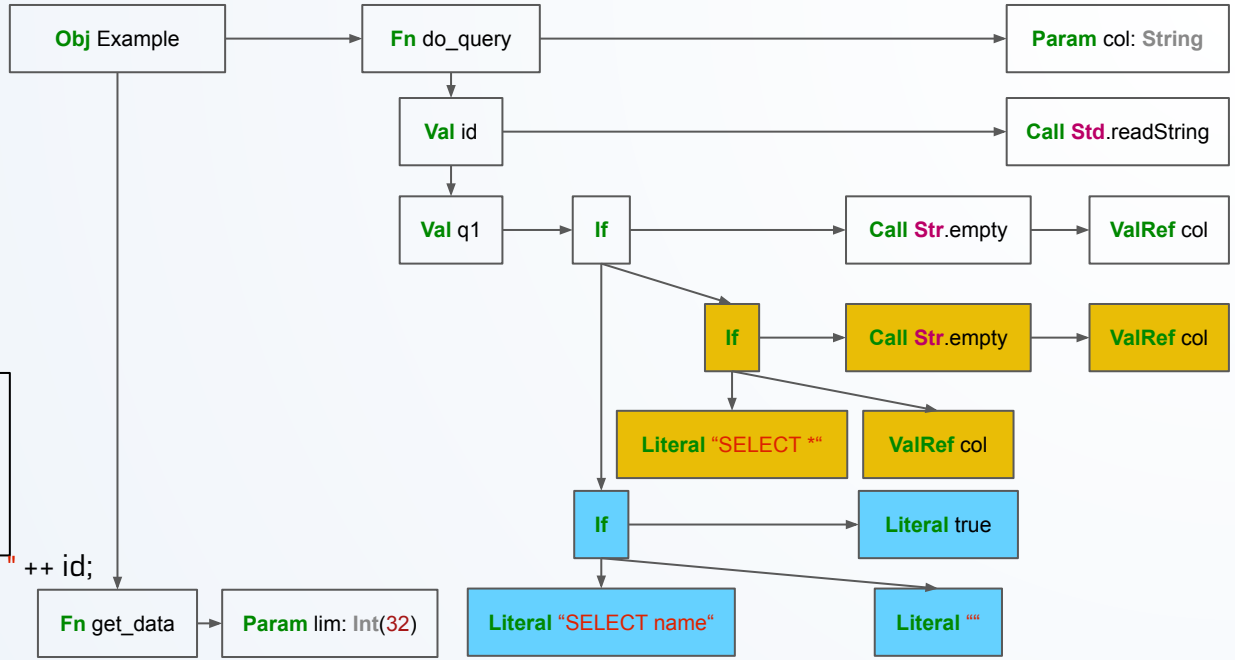
AST




```
object Example

fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}

end Example
```

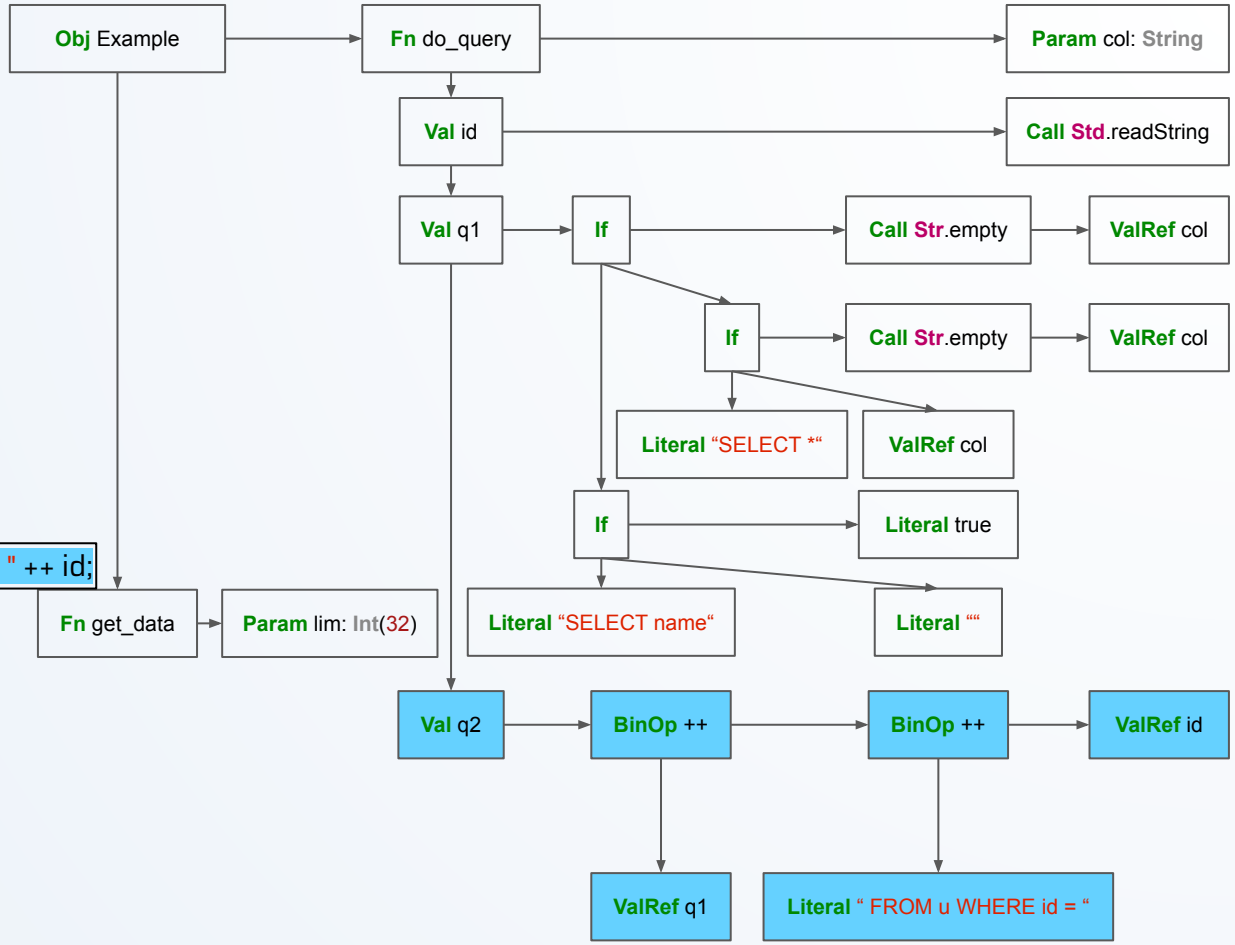
AST

```
object Example

fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}

end Example
```



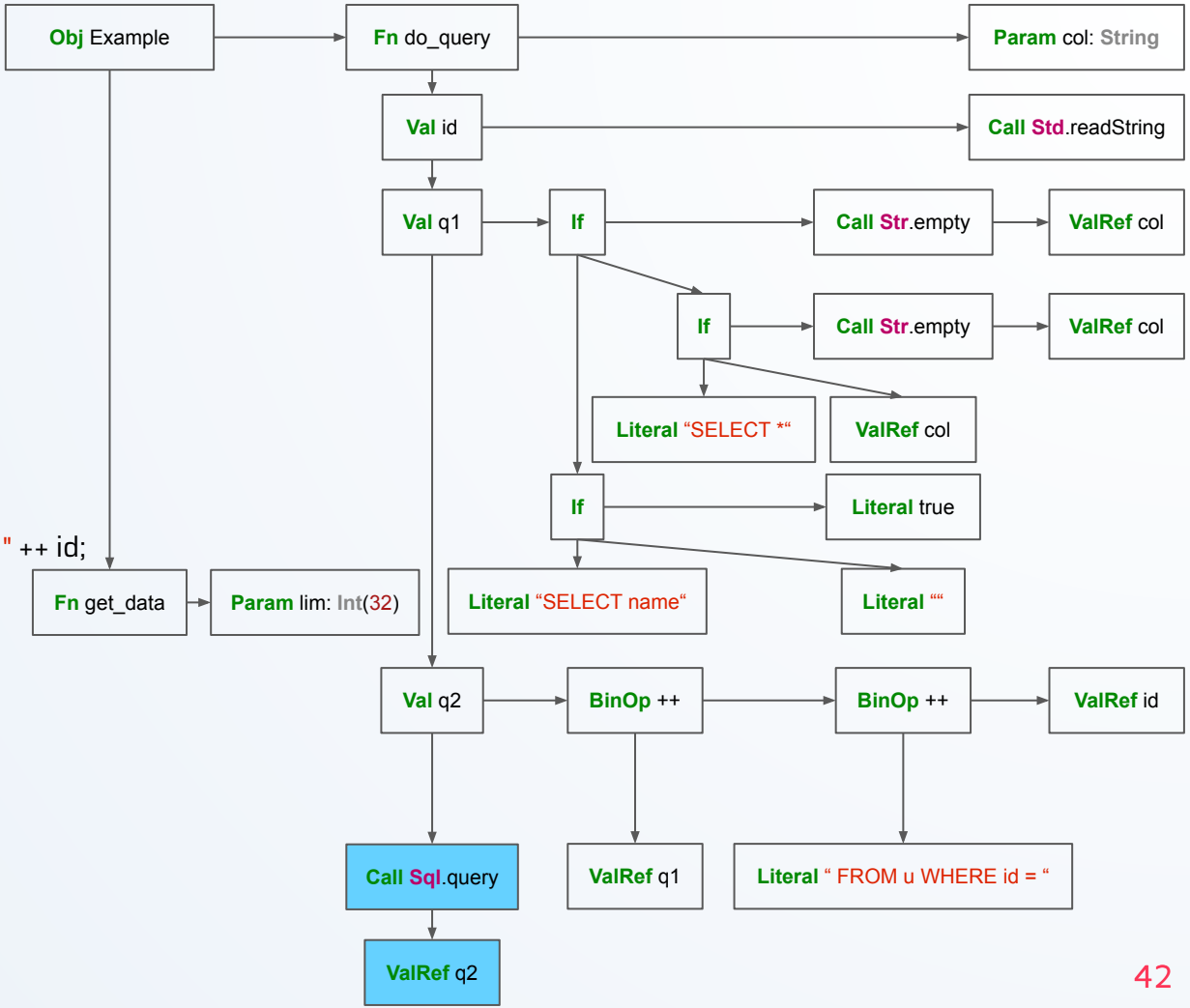
AST

```
object Example

fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}

end Example
```



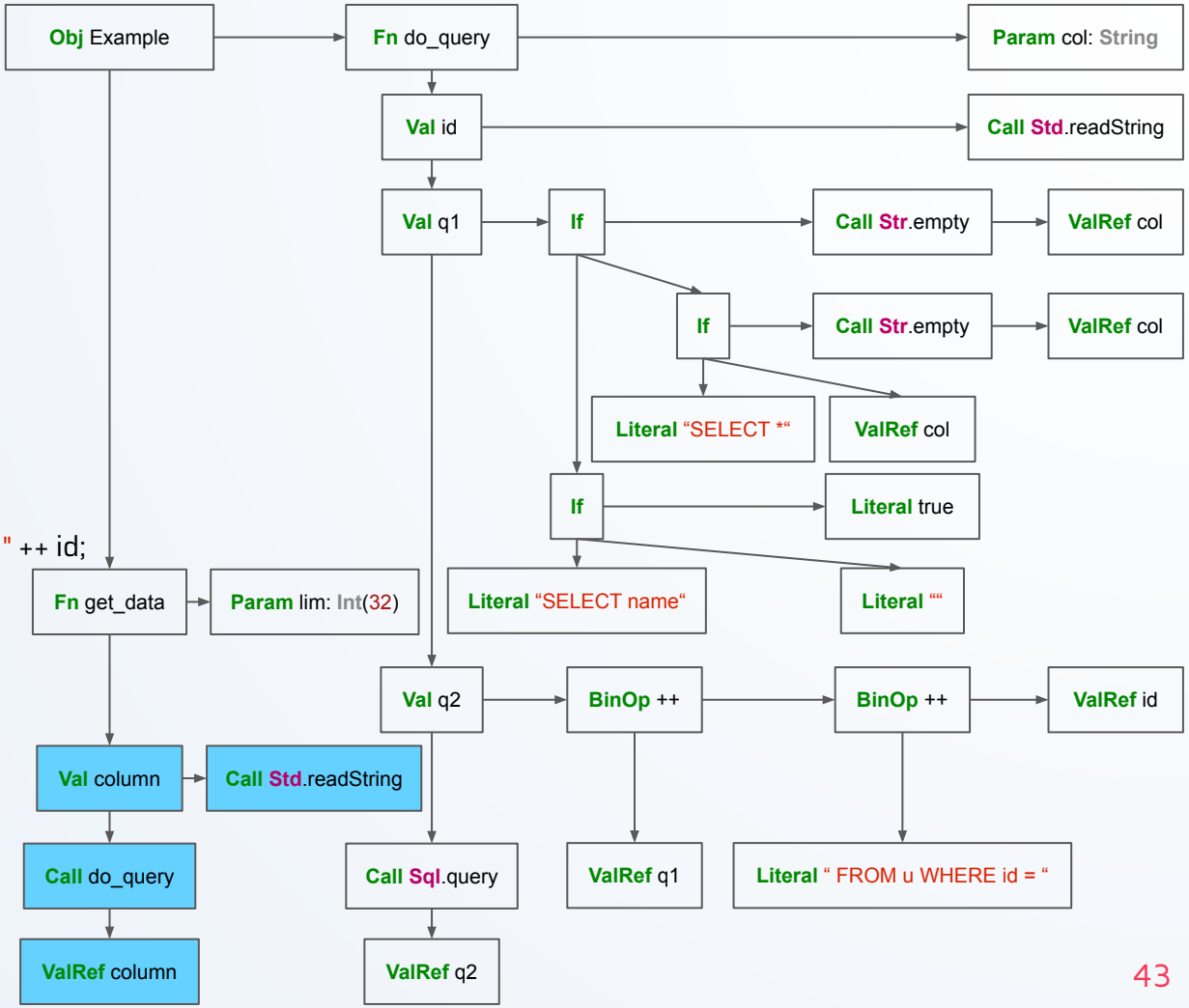
AST

```
object Example

fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}

end Example
```



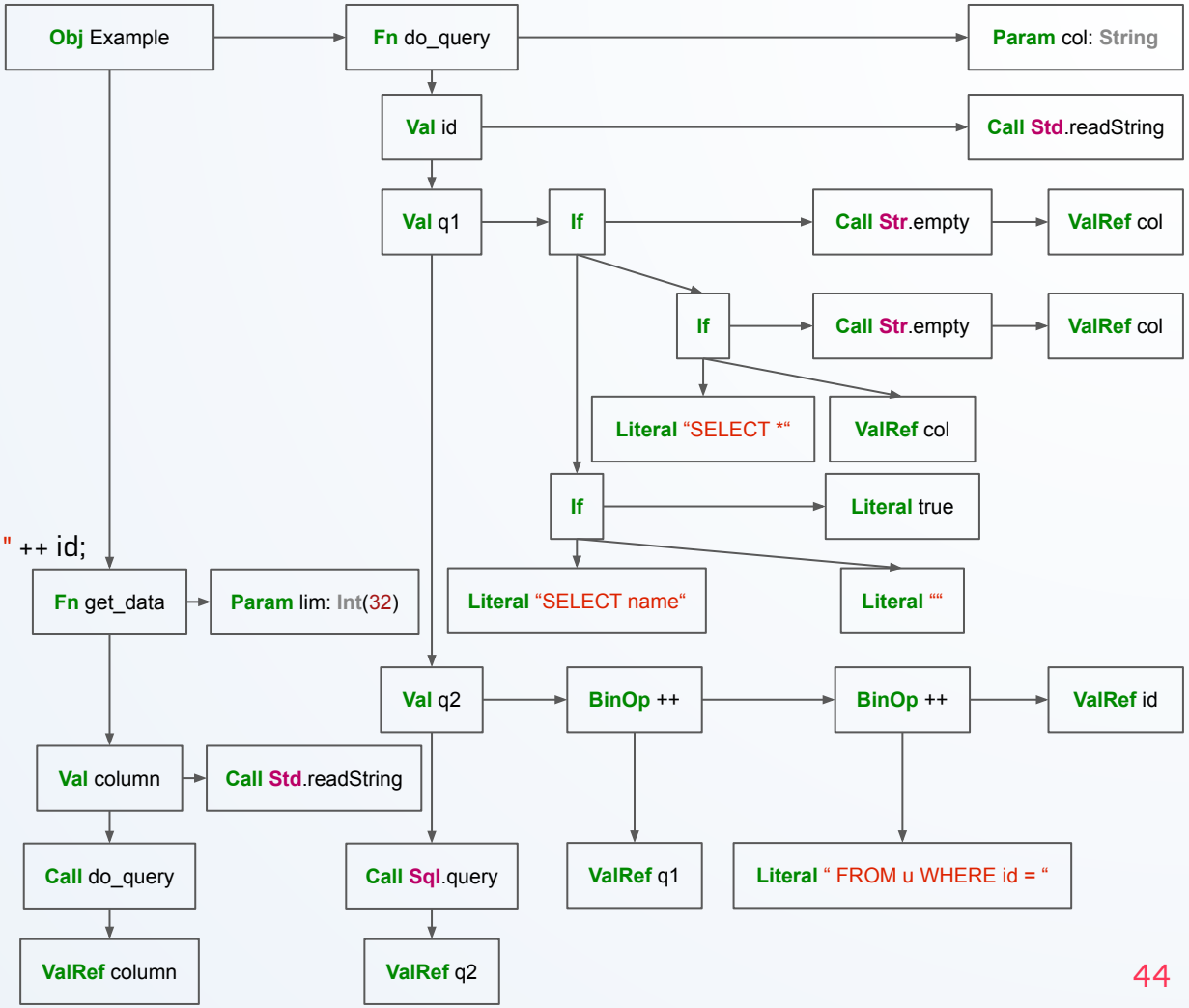
AST

```
object Example

fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}

end Example
```



AST

```
object Example

fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}

end Example
```



Outline

First hour

Intro to static analysis

Place for static analysis

AST-based analysis

→ Visitors & Matchers

Second hour

Taint Analysis

Symbolic Execution

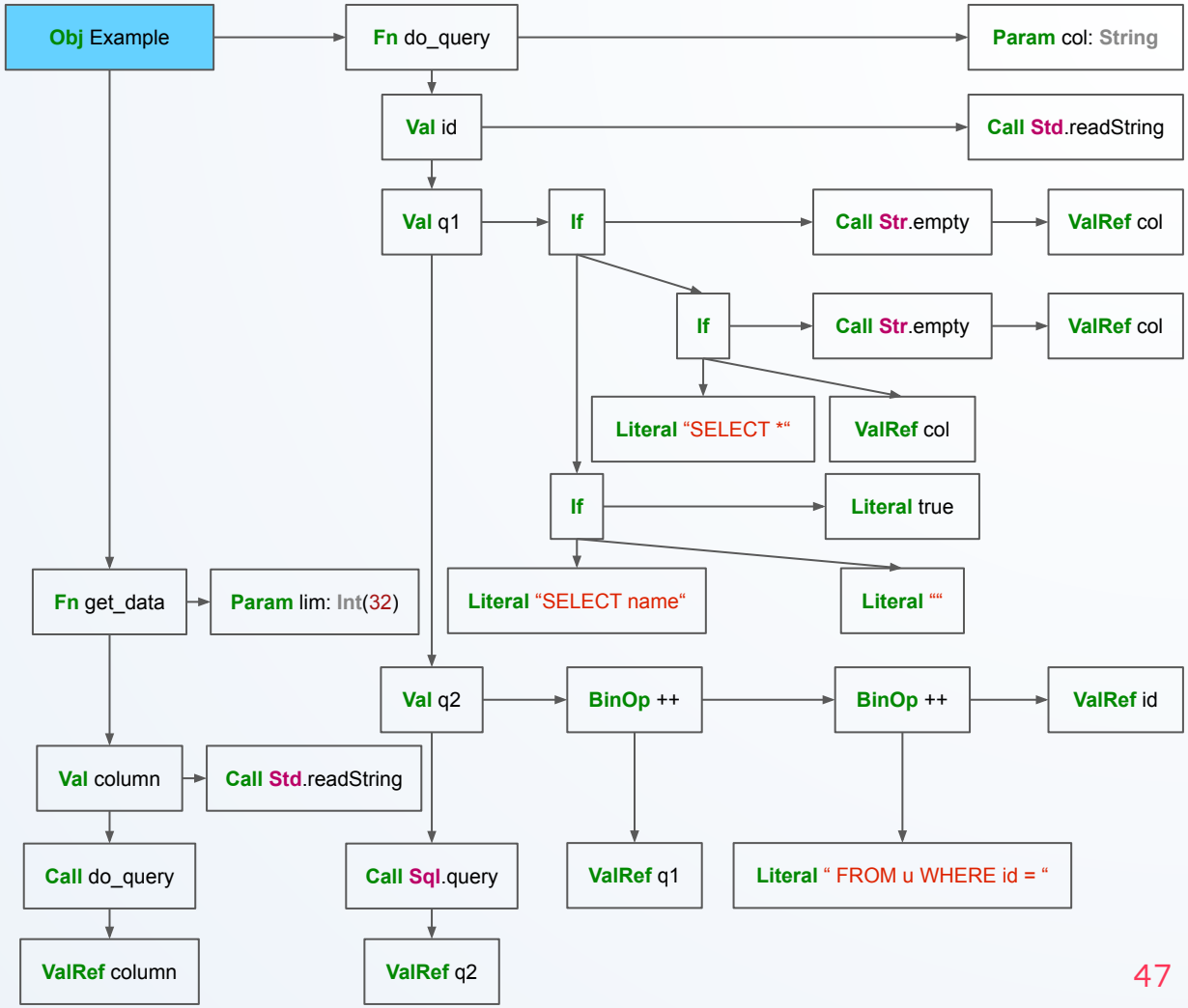
Static Analysis Trade-off

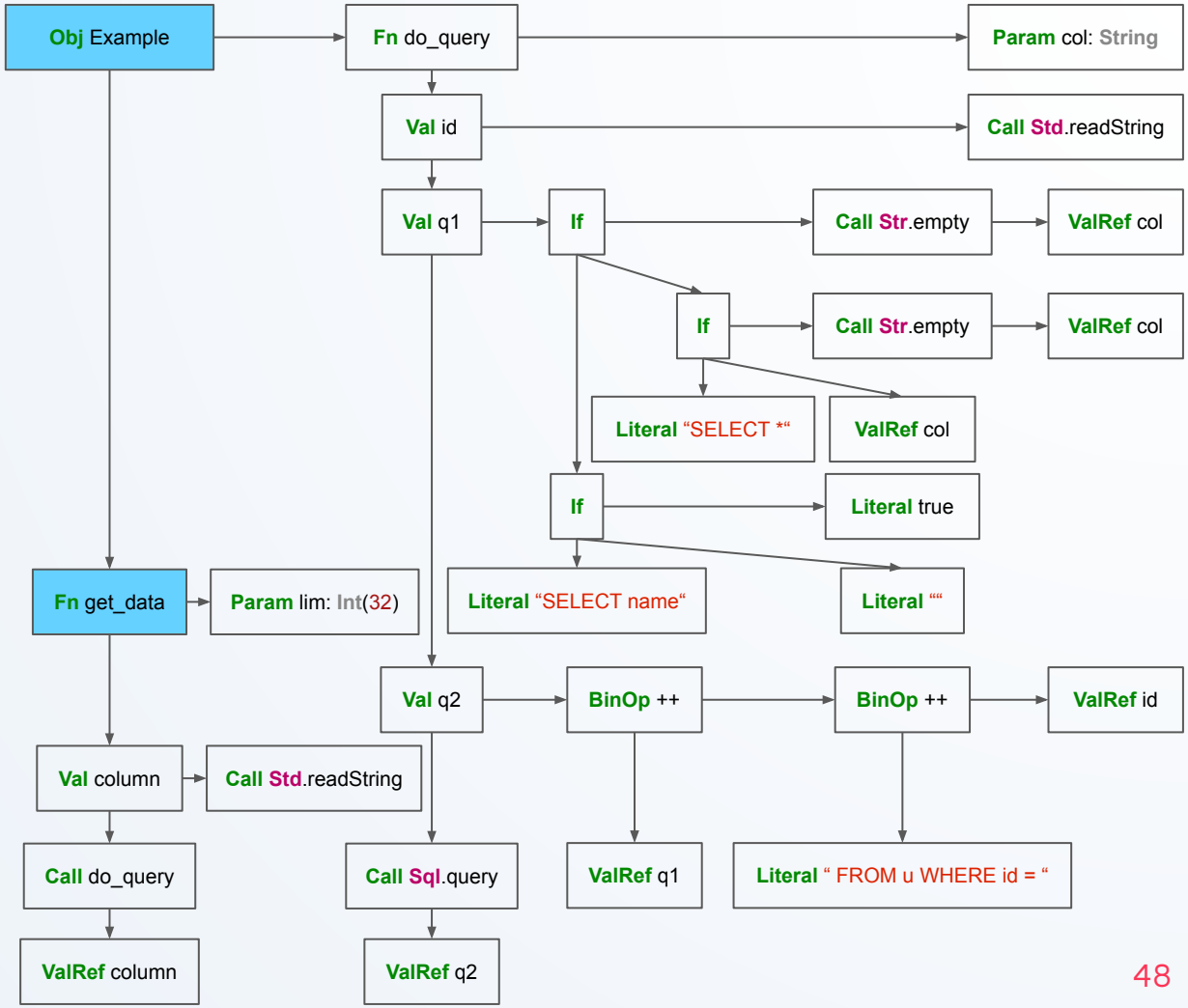
Demo

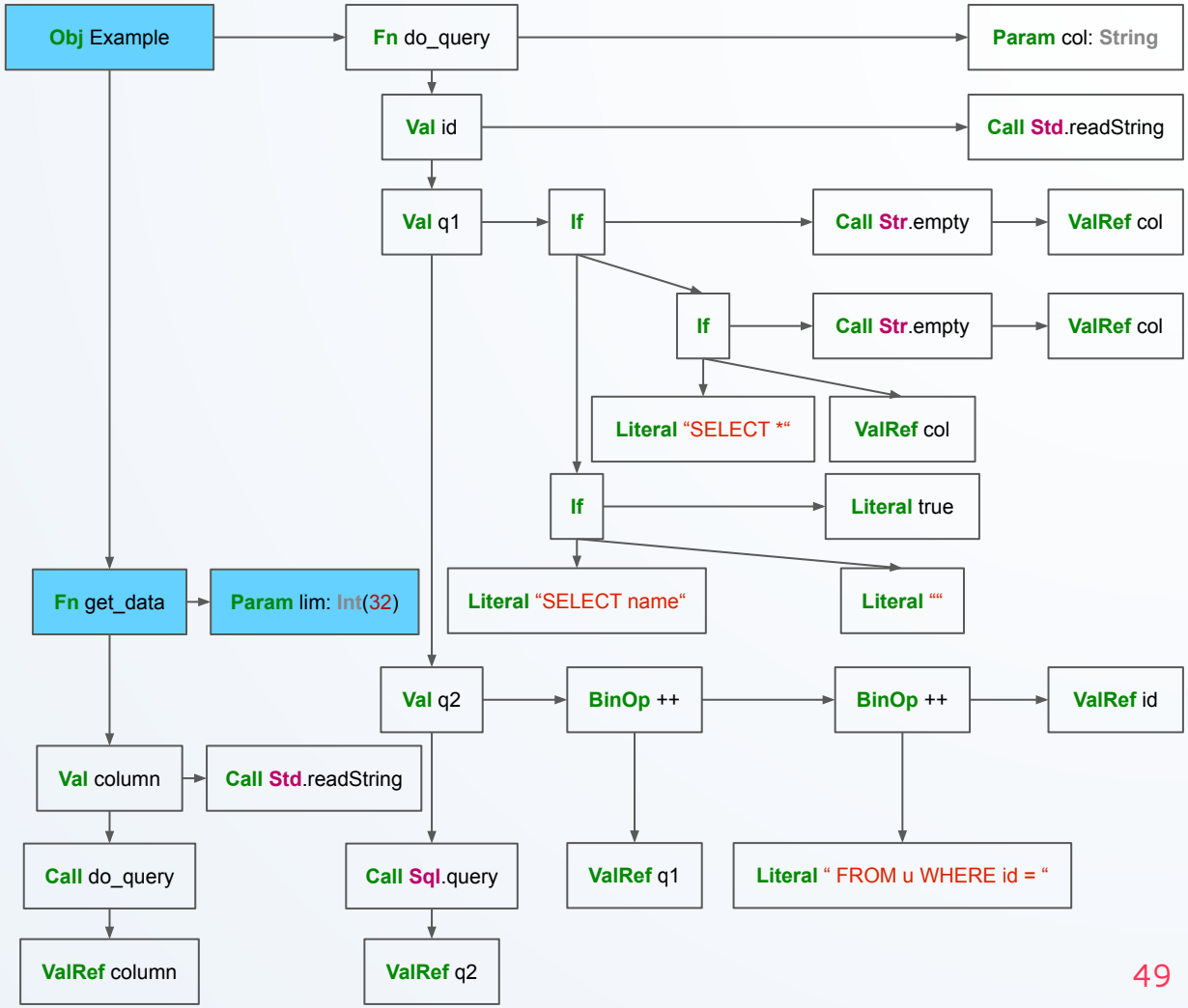
AST Visitors

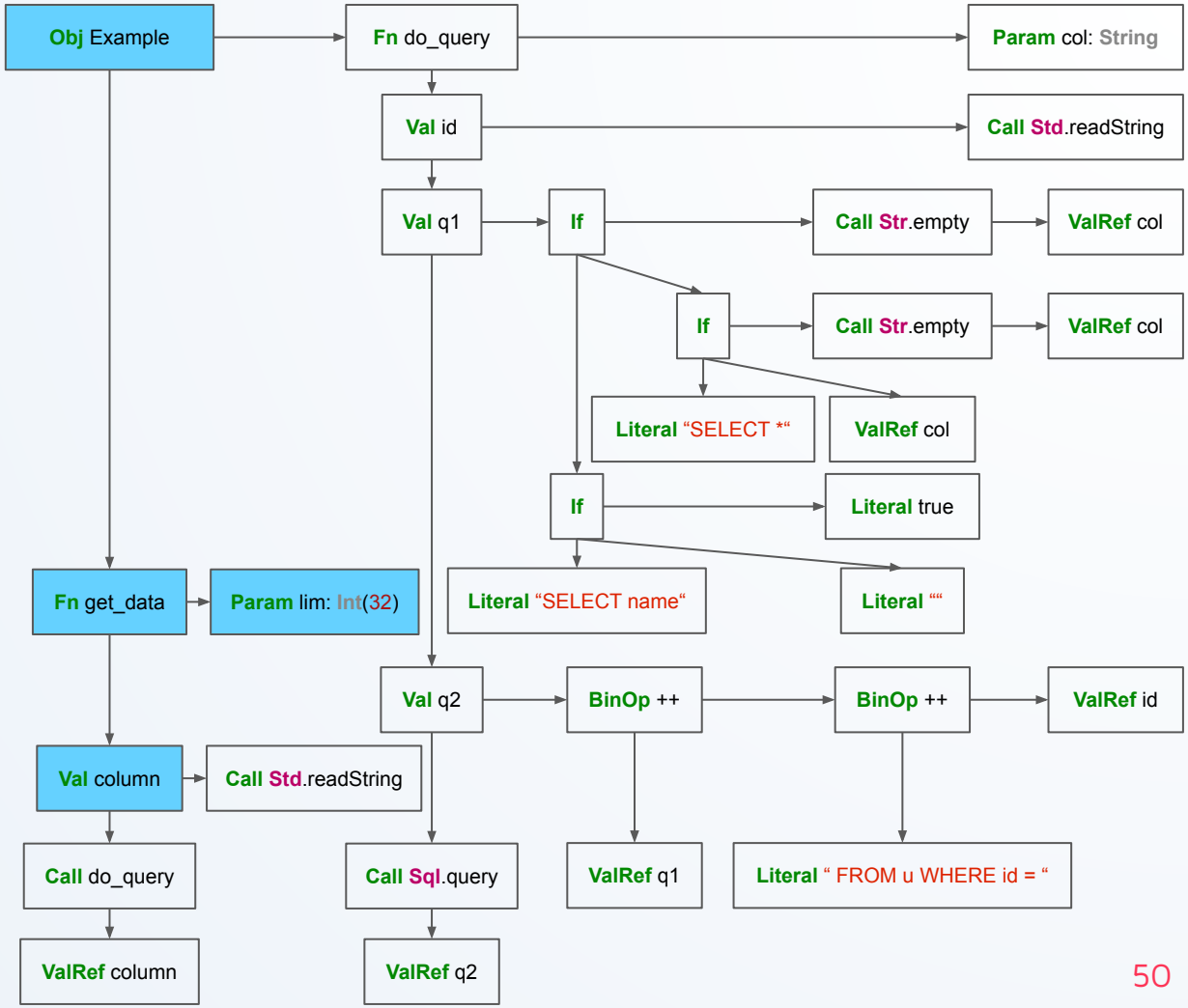
Intuitive way to work with trees

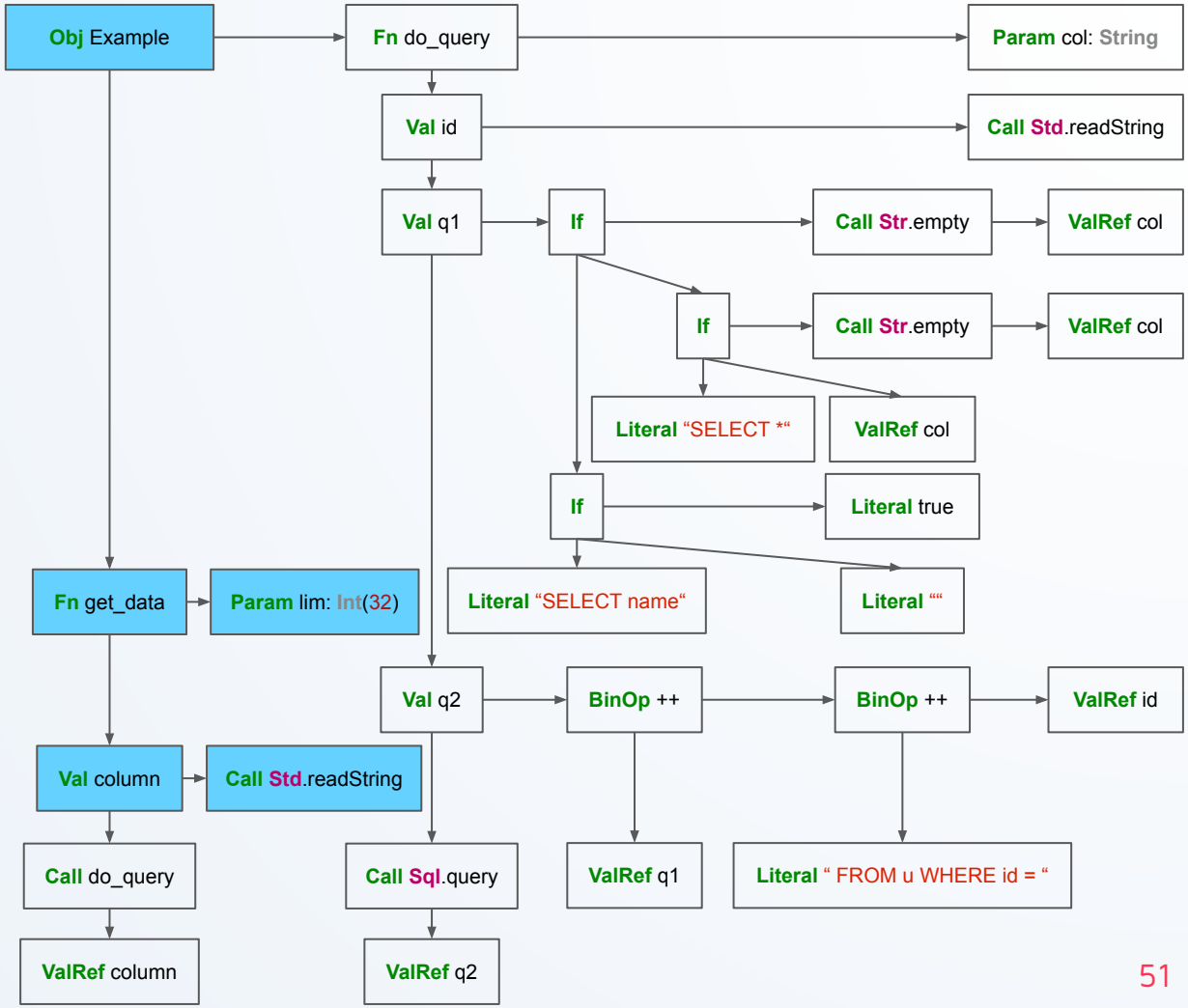
Used in your interpreter and compiler (e.g. for codegen)

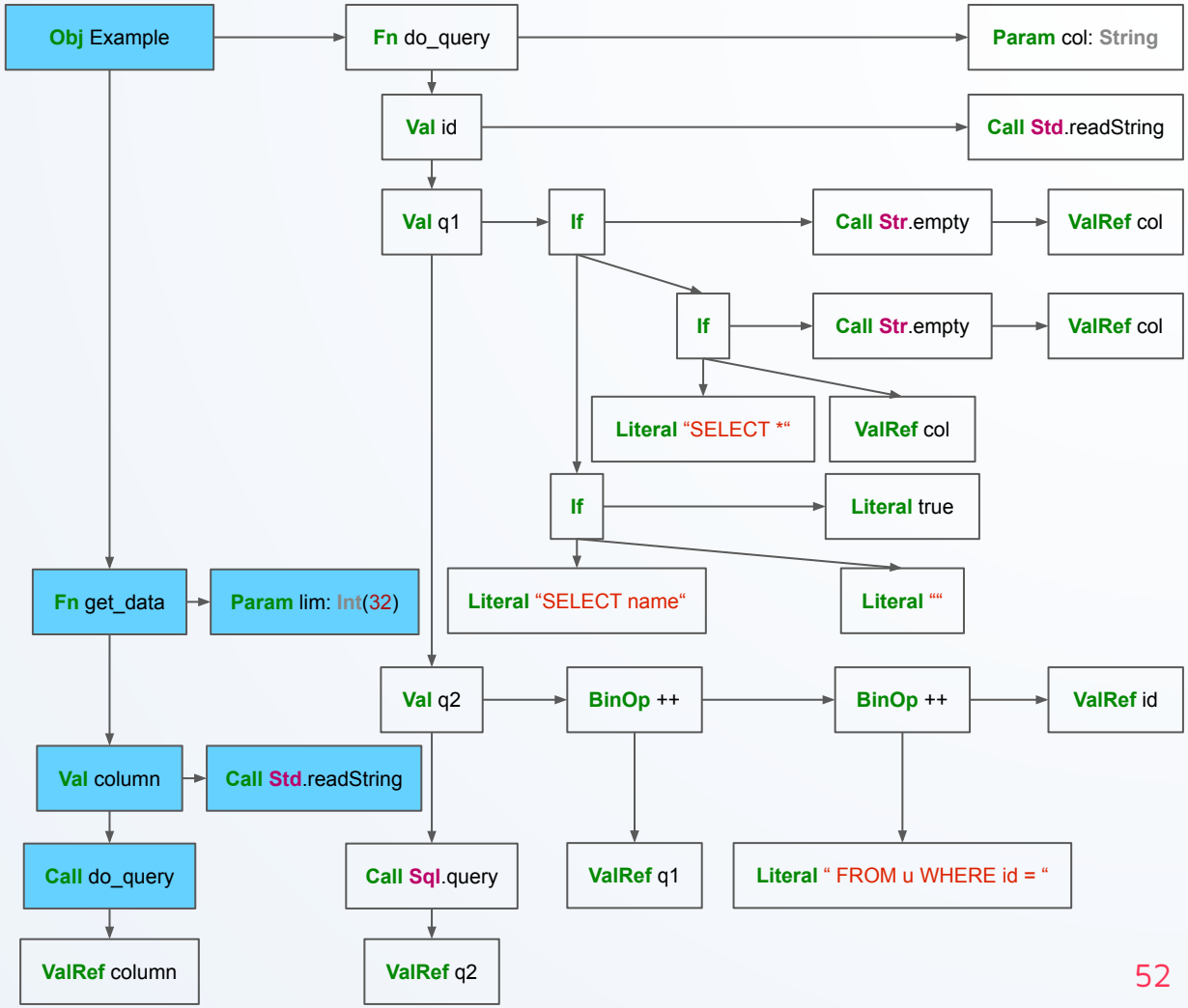

Obj Example
Fn do_query
Param col: String
Val id
Call Std.readString
Val q1
If
Call Str.empty
ValRef col
If
Call Str.empty
ValRef col
Literal “SELECT *“
ValRef col
If
Literal true
Literal “SELECT name“
Literal ““
Fn get_data
Param lim: Int(32)
Val q2
BinOp ++
BinOp ++
ValRef id
Val column
Call Std.readString
Call do_query
Call Sql.query
ValRef q1
Literal “ FROM u WHERE id = “
ValRef column
ValRef q2

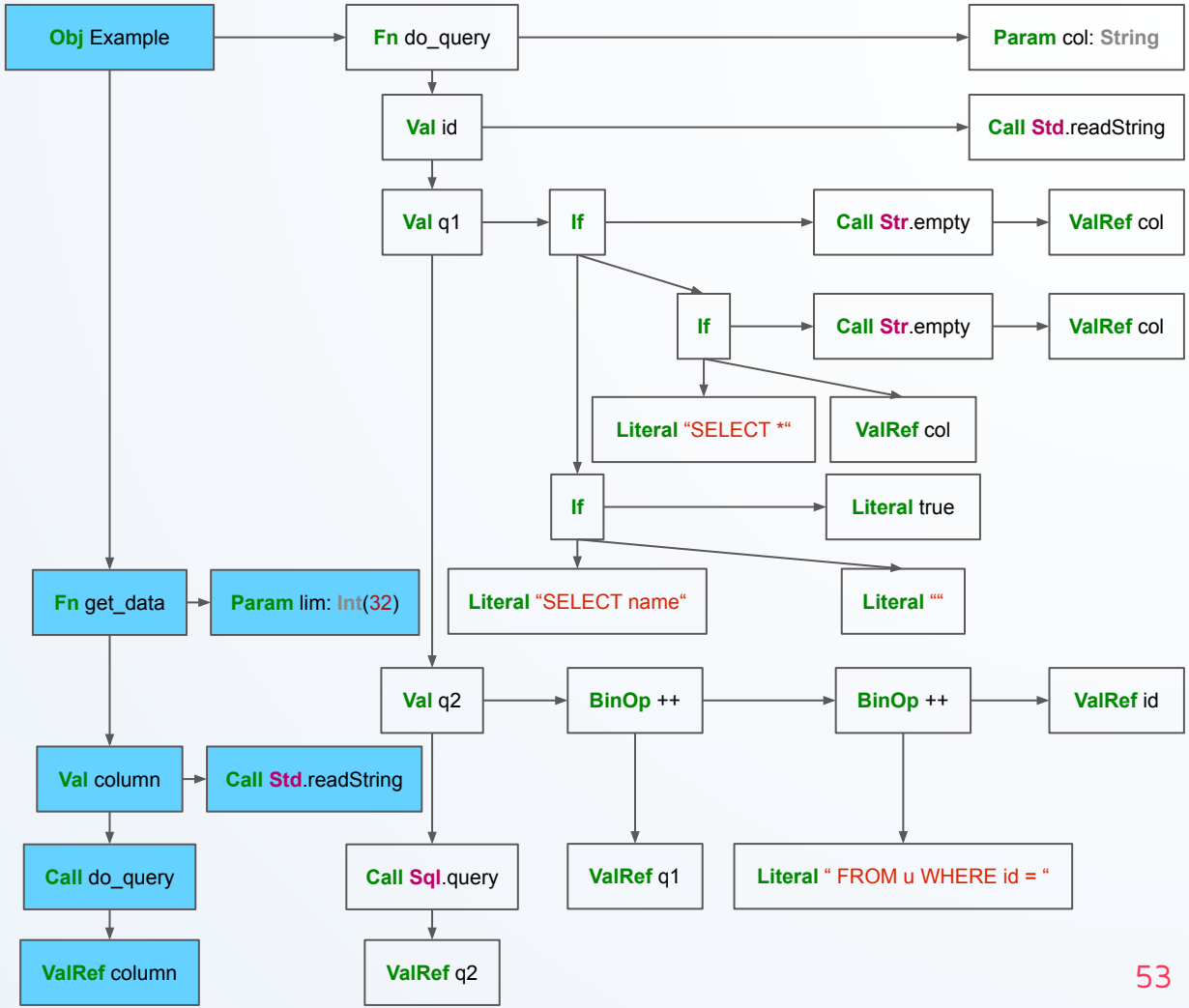

Obj Example
Fn do_query
Param col: String
Val id
Call Std.readString
Val q1
If
Call Str.empty
ValRef col
If
Call Str.empty
ValRef col
Literal “SELECT *“
ValRef col
If
Literal true
Literal “SELECT name“
Literal ““
Fn get_data
Param lim: Int(32)
Val q2
BinOp ++
BinOp ++
ValRef id
Val column
Call Std.readString
Call do_query
Call Sql.query
ValRef q1
Literal “ FROM u WHERE id = “
ValRef column
ValRef q2

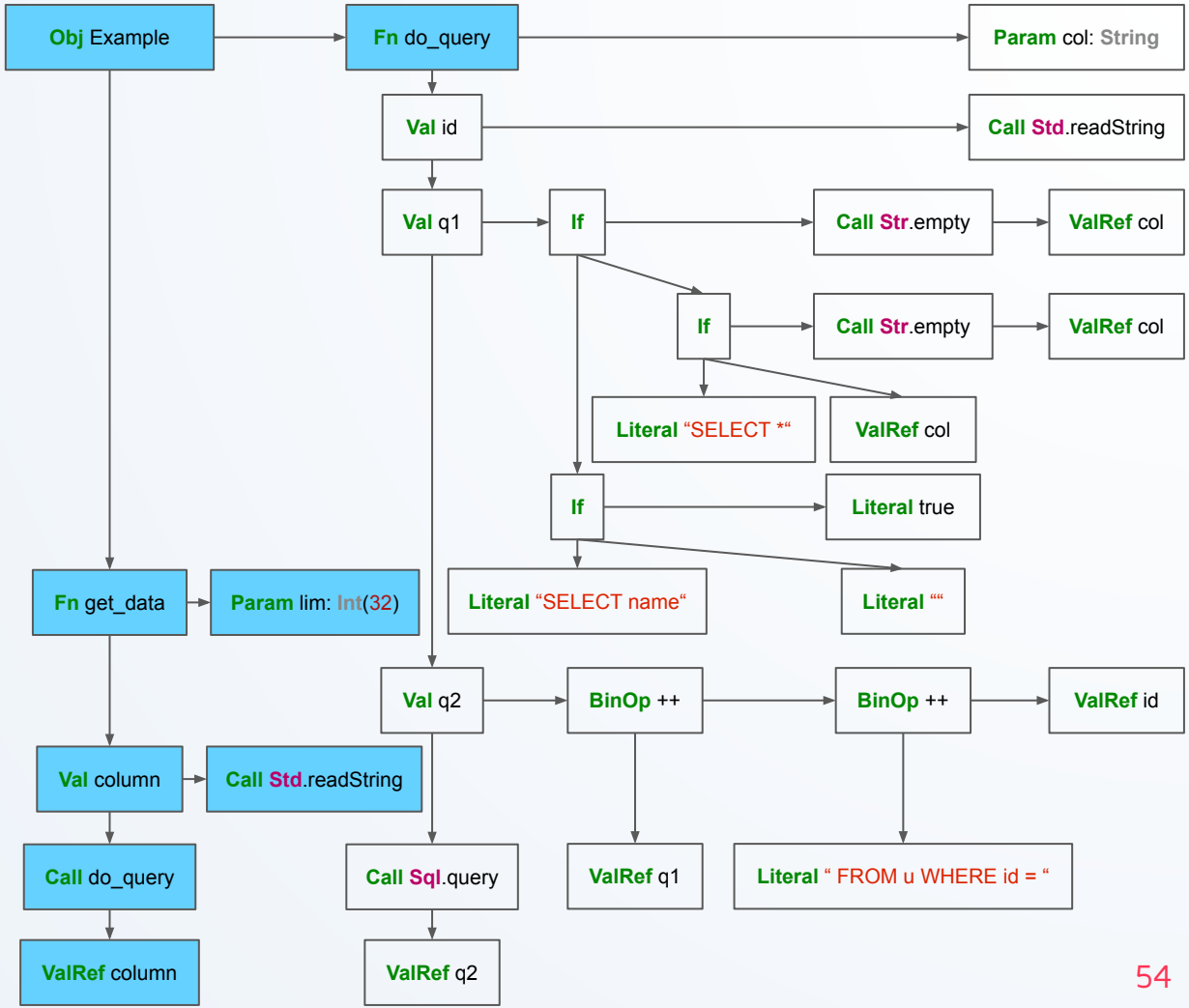

Obj Example
Fn do_query
Param col: String
Val id
Call Std.readString
Val q1
If
Call Str.empty
ValRef col
If
Call Str.empty
ValRef col
Literal "SELECT *"
ValRef col
If
Literal true
Literal "SELECT name"
Literal ""
Fn get_data
Param lim: Int(32)
Val q2
BinOp ++
BinOp ++
ValRef id
Val column
Call Std.readString
Call do_query
Call Sql.query
ValRef q1
Literal " FROM u WHERE id = "
ValRef column
ValRef q2


Obj Example
Fn do_query
Param col: String
Val id
Call Std.readString
Val q1
If
Call Str.empty
ValRef col
If
Call Str.empty
ValRef col
Literal “SELECT *“
ValRef col
If
Literal true
Literal “SELECT name“
Literal ““
Fn get_data
Param lim: Int(32)
Val q2
BinOp ++
BinOp ++
ValRef id
Val column
Call Std.readString
Call do_query
Call Sql.query
ValRef q1
Literal “ FROM u WHERE id = “
ValRef column
ValRef q2


Obj Example
Fn do_query
Param col: String
Val id
Call Std.readString
Val q1
If
Call Str.empty
ValRef col
If
Call Str.empty
ValRef col
Literal “SELECT *“
ValRef col
If
Literal true
Literal “SELECT name“
Literal ““
Fn get_data
Param lim: Int(32)
Val q2
BinOp ++
BinOp ++
ValRef id
Val column
Call Std.readString
Call do_query
Call Sql.query
ValRef q1
Literal “ FROM u WHERE id = “
ValRef column
ValRef q2


Obj Example
Fn do_query
Param col: String
Val id
Call Std.readString
Val q1
If
Call Str.empty
ValRef col
If
Call Str.empty
ValRef col
Literal “SELECT *“
ValRef col
If
Literal true
Literal “SELECT name“
Literal ““
Fn get_data
Param lim: Int(32)
Val q2
BinOp ++
BinOp ++
ValRef id
Val column
Call Std.readString
Call do_query
Call Sql.query
ValRef q1
Literal “ FROM u WHERE id = “
ValRef column
ValRef q2


Obj Example
Fn do_query
Param col: String
Val id
Call Std.readString
Val q1
If
Call Str.empty
ValRef col
If
Call Str.empty
ValRef col
Literal "SELECT *"
ValRef col
If
Literal true
Fn get_data
Param lim: Int(32)
Literal "SELECT name"
Literal ""
Val q2
BinOp ++
BinOp ++
ValRef id
Val column
Call Std.readString
Call do_query
Call Sql.query
ValRef q1
Literal " FROM u WHERE id = "
ValRef column
ValRef q2


Obj Example
Fn do_query
Param col: String
Val id
Call Std.readString
Val q1
If
Call Str.empty
ValRef col
If
Call Str.empty
ValRef col
Literal “SELECT *“
ValRef col
If
Literal true
Literal “SELECT name“
Literal ““
Fn get_data
Param lim: Int(32)
Val q2
BinOp ++
BinOp ++
ValRef id
Val column
Call Std.readString
Call do_query
Call Sql.query
ValRef q1
Literal “ FROM u WHERE id = “
ValRef column
ValRef q2

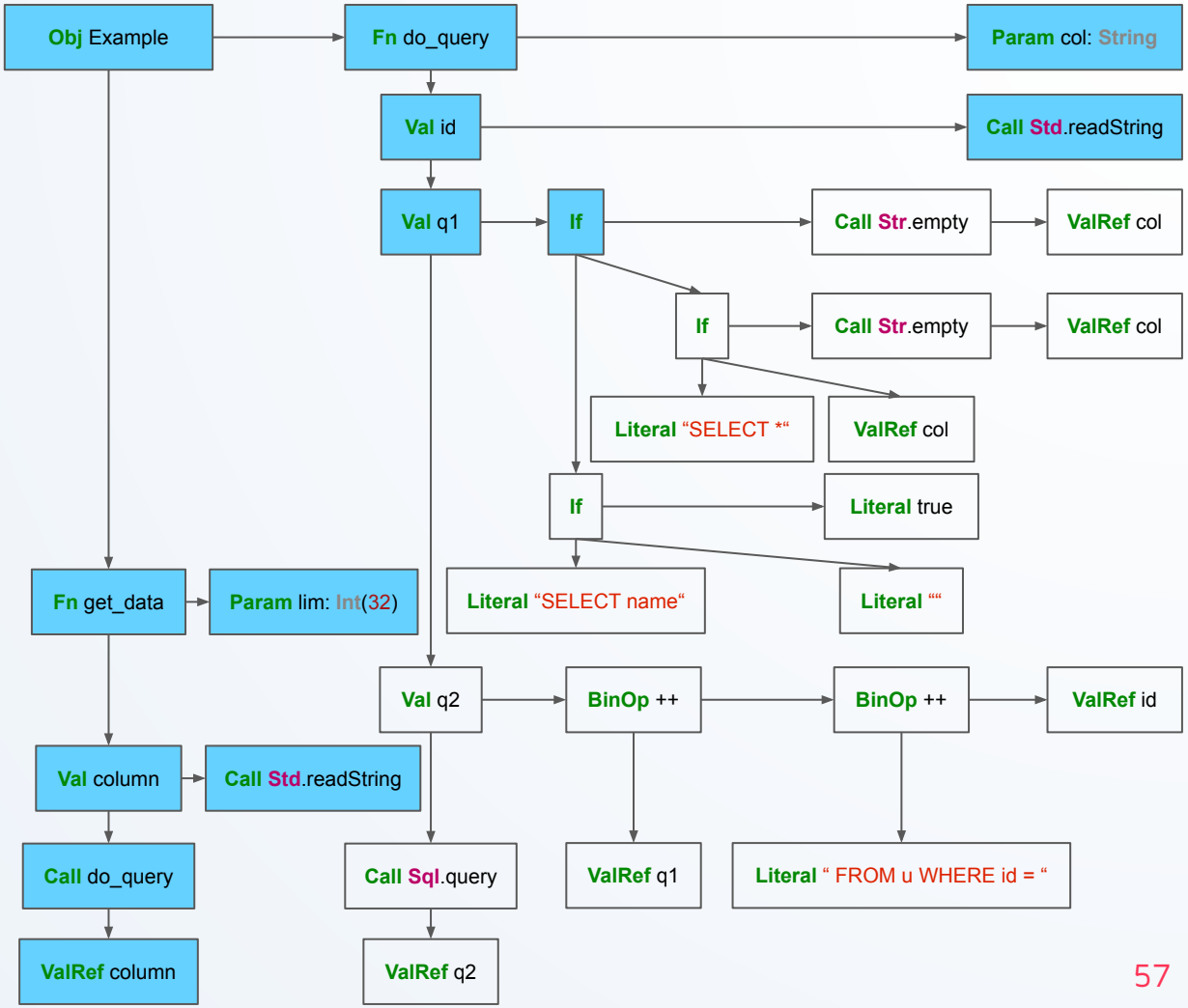
AST Visitors - Implementation

```
trait TreeVisitor {
  def visit(t: Tree) = t match {
    case Ite(cond, thenn, elze) =>
      visit(cond)
      visit(thenn)
      visit(elze)
    case FunDef(name, params, retType, body) =>
      visit(params)
      visit(body)
    // ...
}
```

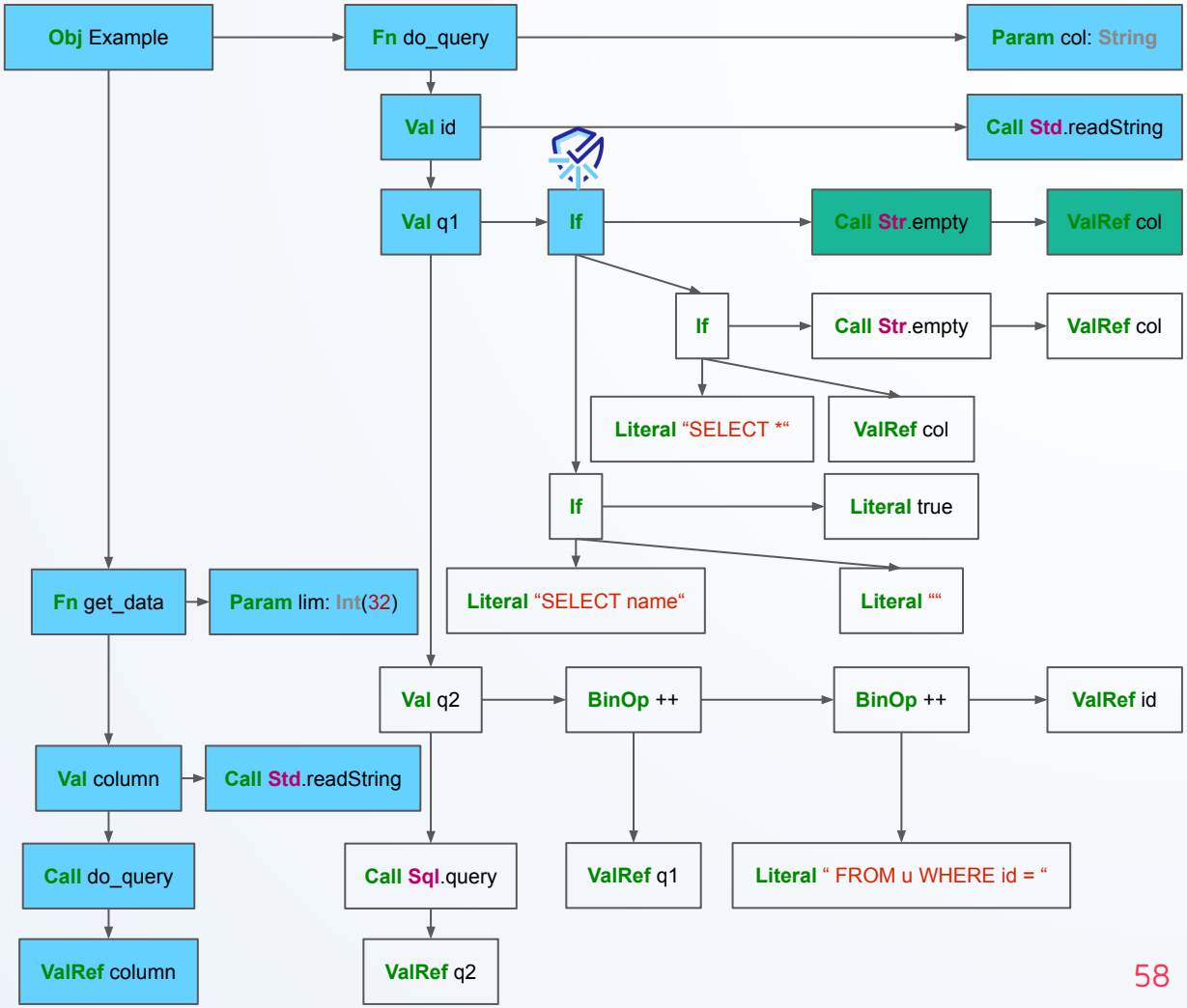
if with a trivial condition

```
object TrivialConditionCheck extends TreeVisitor {
  override def visit(tree :Tree) = tree match {
    case Ite(BooleanLiteral(_), _, _) =>
      reportIssue(tree)
      super.visit(tree)
    case _ => super.visit(tree)
  }
}
```

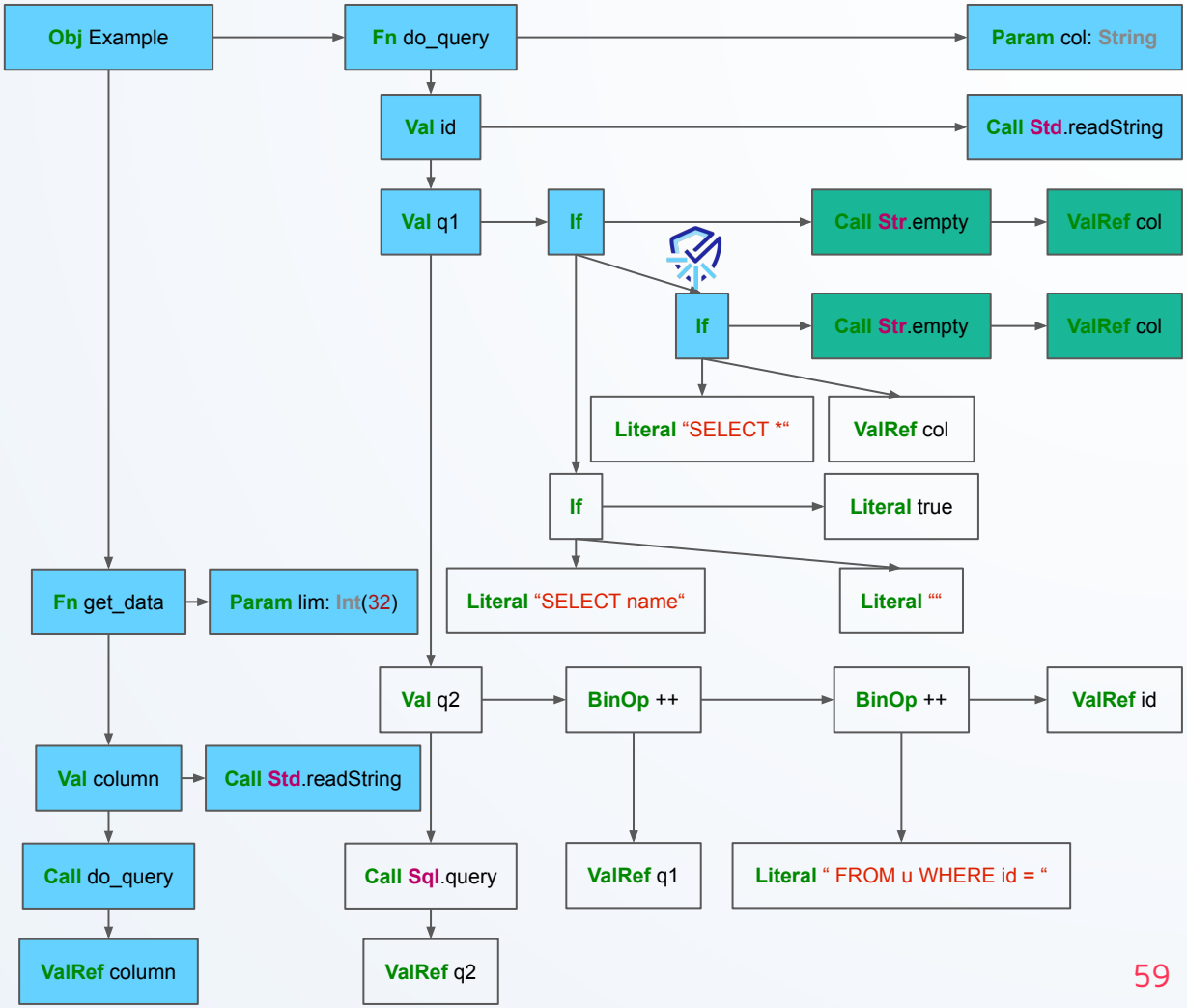
```
case Ite(BooleanLiteral(_), _, _) =>
  reportIssue(tree)
```



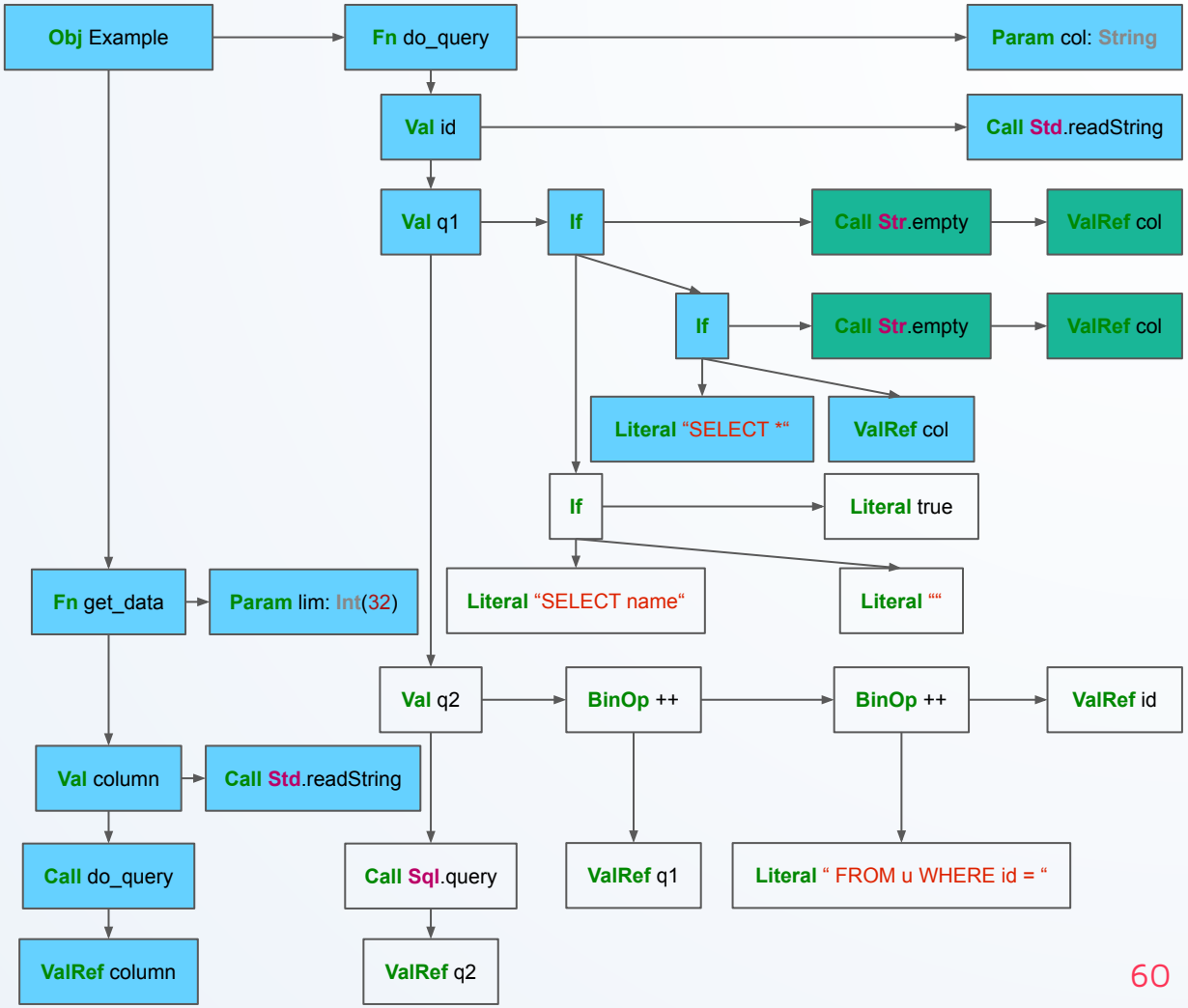
```
case Ite(BooleanLiteral(_), _, _) =>
  reportIssue(tree)
```



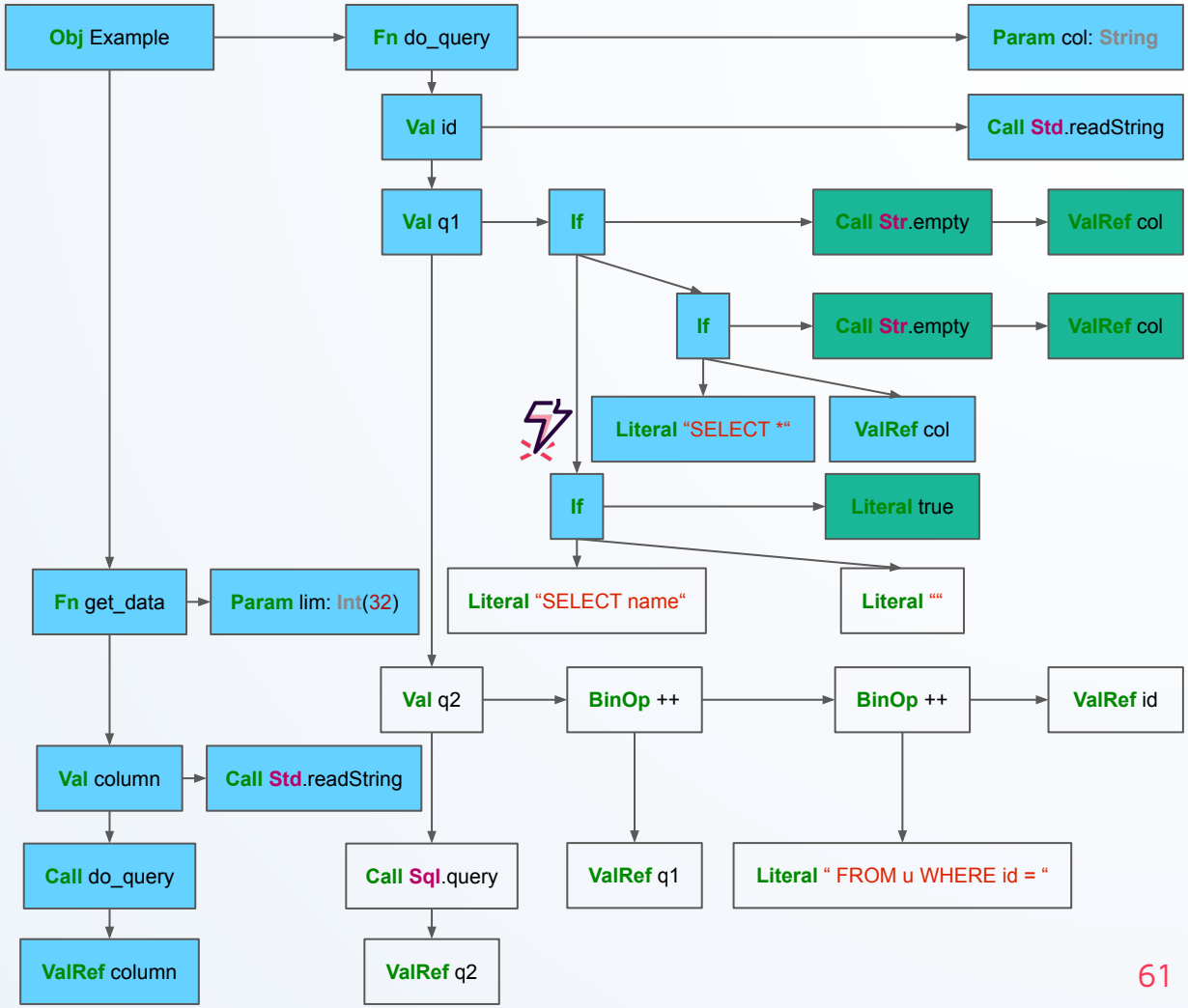
```
case Ite(BooleanLiteral(_), _, _) =>
  reportIssue(tree)
```



```
case Ite(BooleanLiteral(_), _, _) =>
  reportIssue(tree)
```



```
case Ite(BooleanLiteral(_), _, _) =>
  reportIssue(tree)
```



Unused Parameters

```
object UnusedParameterCheck extends TreeVisitor {
 override def visit(t: Tree) = t match {
  case FunDef(_, params, _, body) =>
   params.foreach(p => {
    val visitor = new UsageVisitor(p.name)
    visitor.visit(body)
    if (!visitor.seen) {
     reportIssue(p)
    }
   })
  case _ => super.visit(t)
 }
}
```

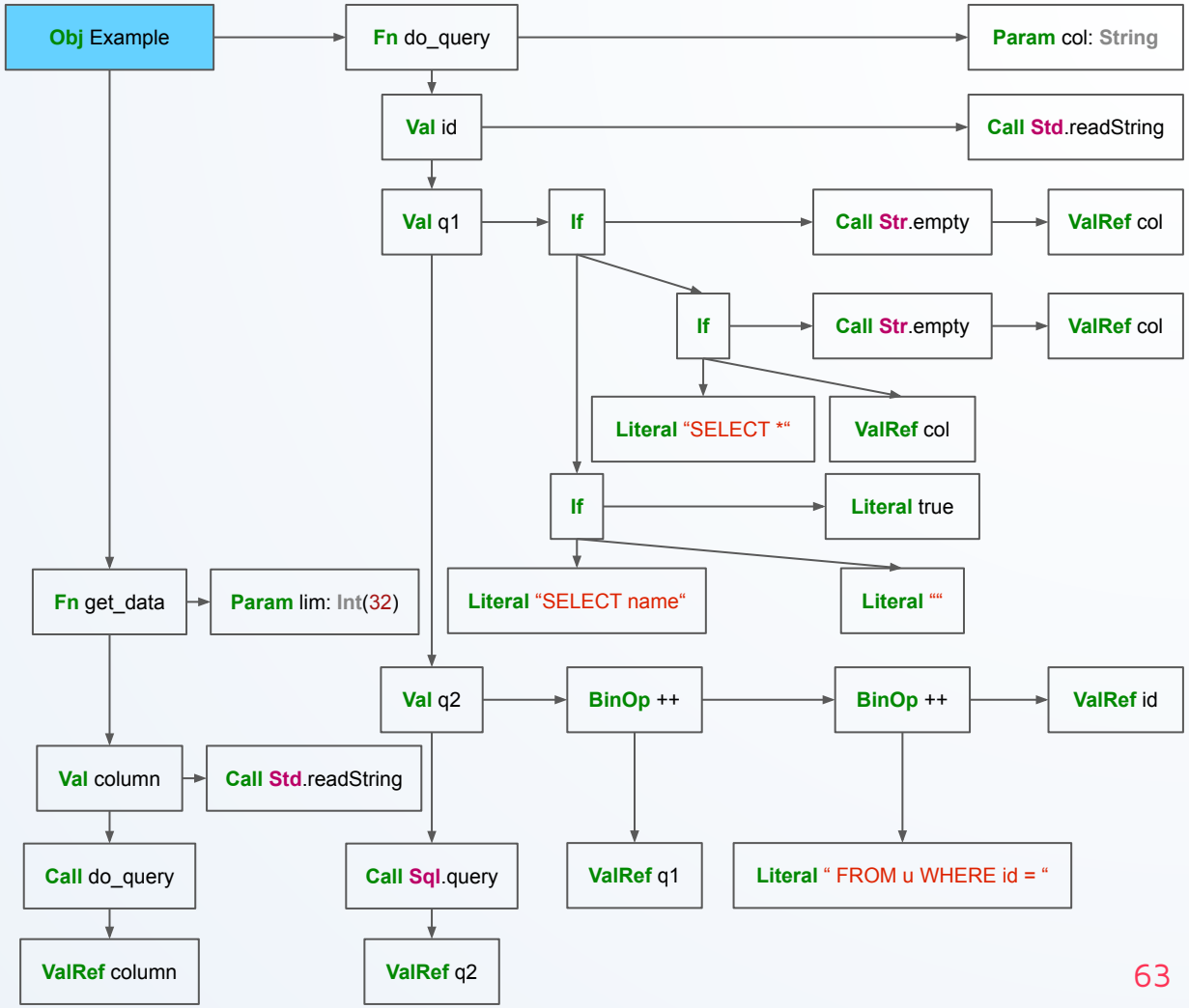
```
class UsageVisitor(target: Name, var seen: Boolean = false)
extends TreeVisitor {
 override def visit(t: Tree) = t match {
  case Variable(name) if name == target =>
   seen = true
  case _ => super.visit(t)
 }
}
```

UnusedParameterCheck

```
case FunDef(_, params, _, body) =>
  params.foreach(p => {
    val visitor = new UsageVisitor(p.name)
    visitor.visit(body)
    if (!visitor.seen) {
      reportIssue(p)
    }
```

UsageVisitor

```
case Variable(name) if name == target =>
  seen = true
```

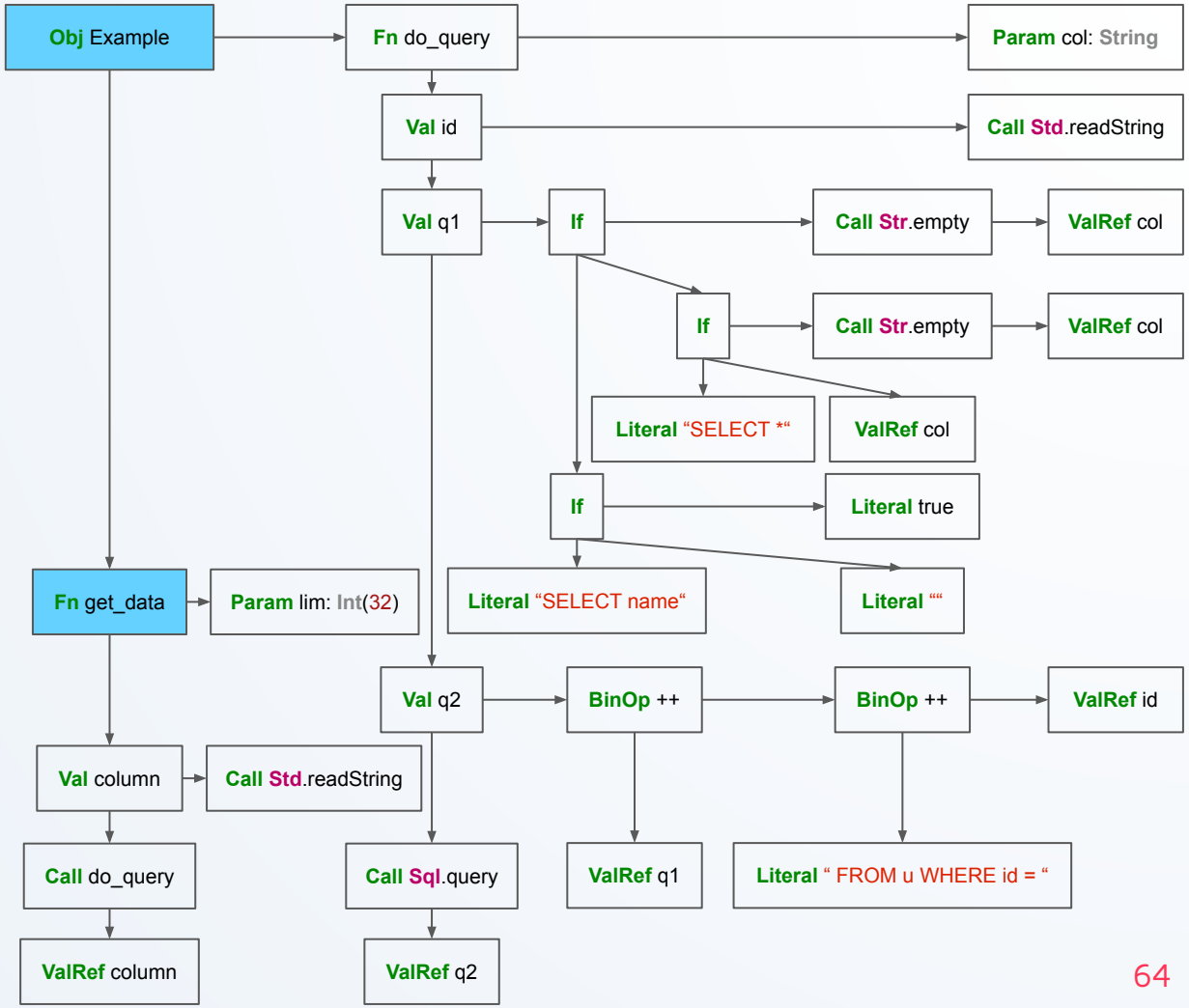


UnusedParameterCheck

```
case FunDef(_, params, _, body) =>
  params.foreach(p => {
    val visitor = new UsageVisitor(p.name)
    visitor.visit(body)
    if (!visitor.seen) {
      reportIssue(p)
    }
```

UsageVisitor

```
case Variable(name) if name == target =>
  seen = true
```

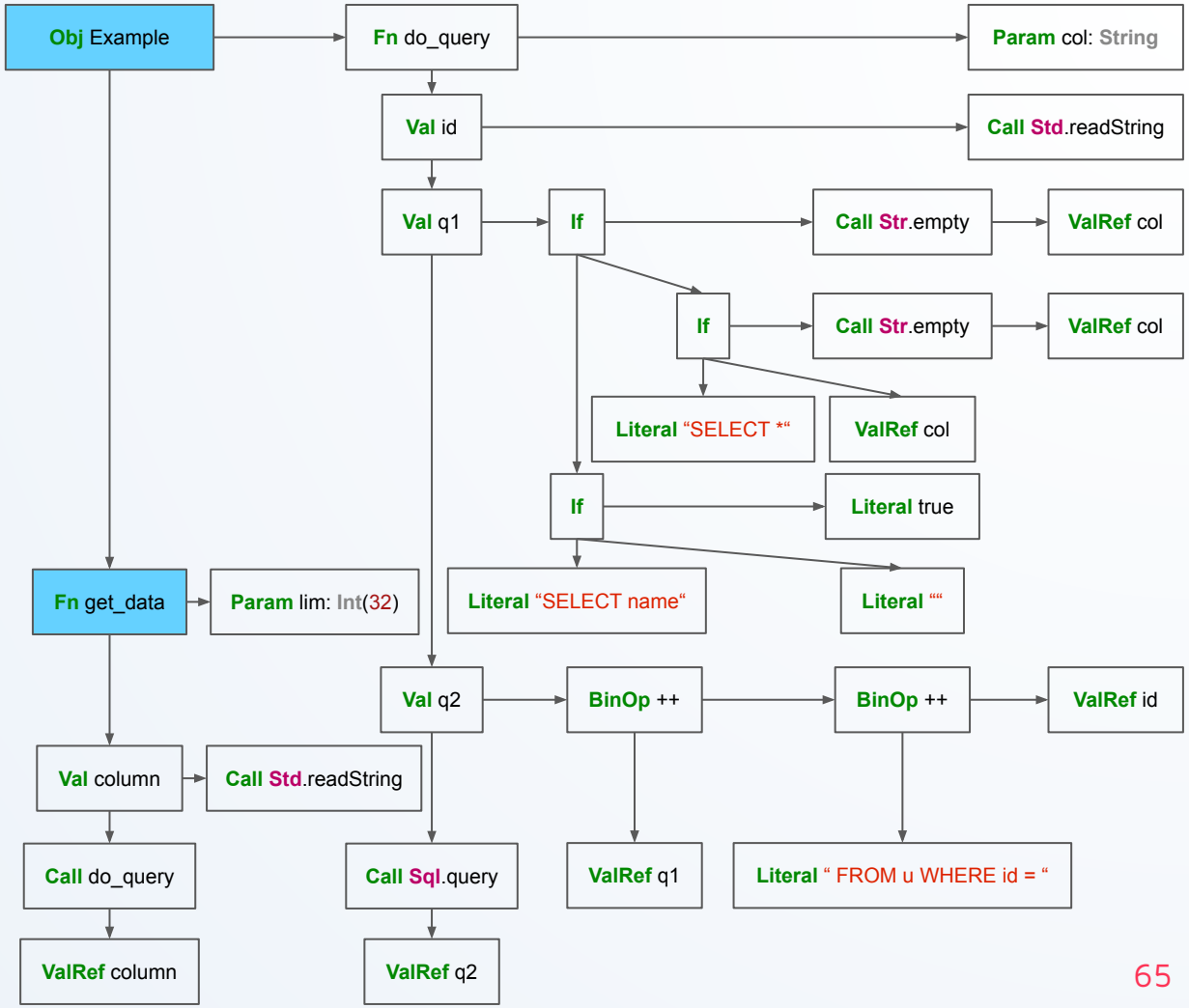


UnusedParameterCheck

```
case FunDef(_, params, _, body) =>
  params.foreach(p => {
    val visitor = new UsageVisitor(p.name)
    visitor.visit(body)
    if (!visitor.seen) {
      reportIssue(p)
    }
```

UsageVisitor

```
case Variable(name) if name == target =>
  seen = true
```



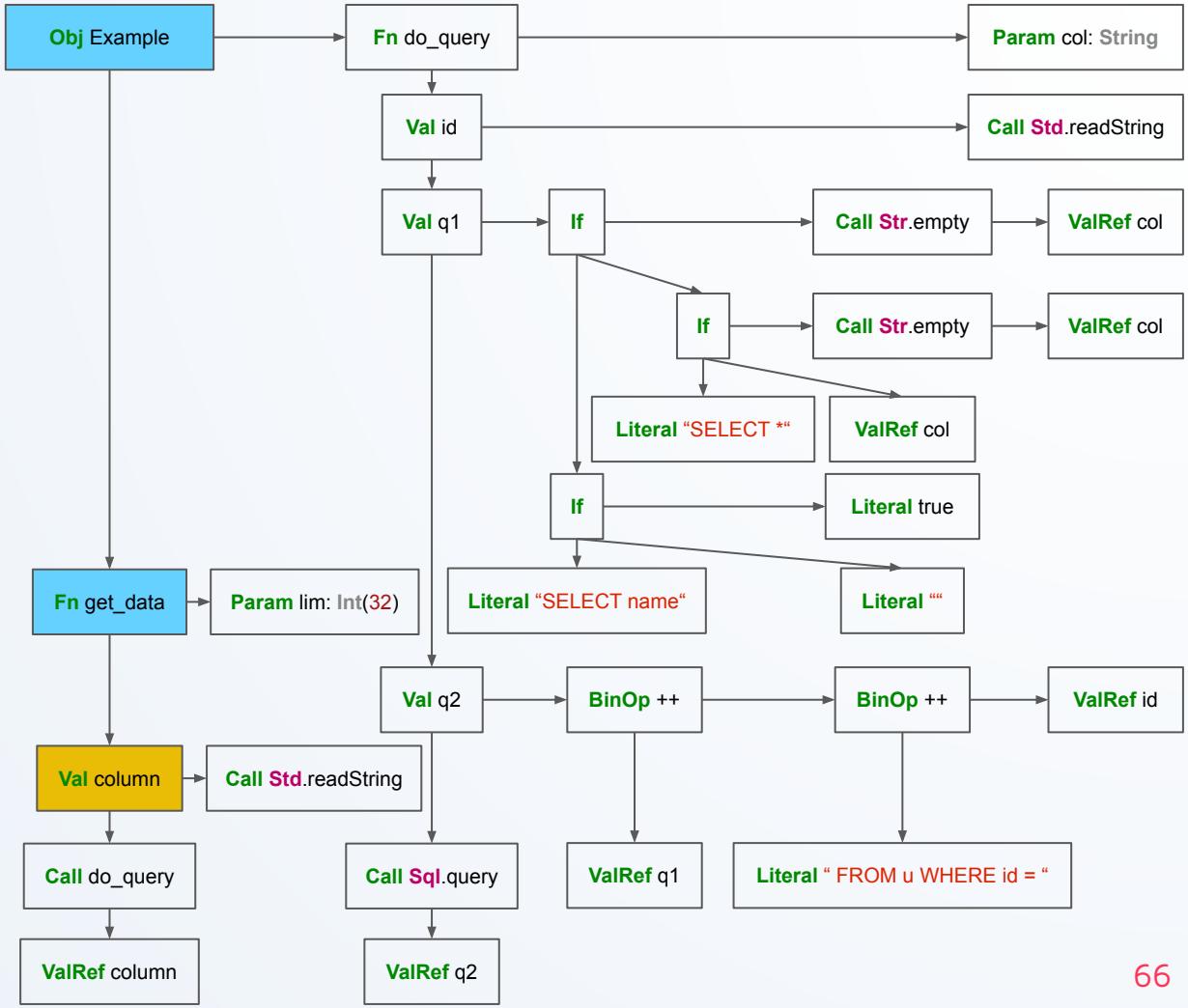
target = lim
seen = false

UnusedParameterCheck

```
case FunDef(_, params, _, body) =>
  params.foreach(p => {
    val visitor = new UsageVisitor(p.name)
    visitor.visit(body)
    if (!visitor.seen) {
      reportIssue(p)
    }
```

UsageVisitor

```
case Variable(name) if name == target =>
  seen = true
```



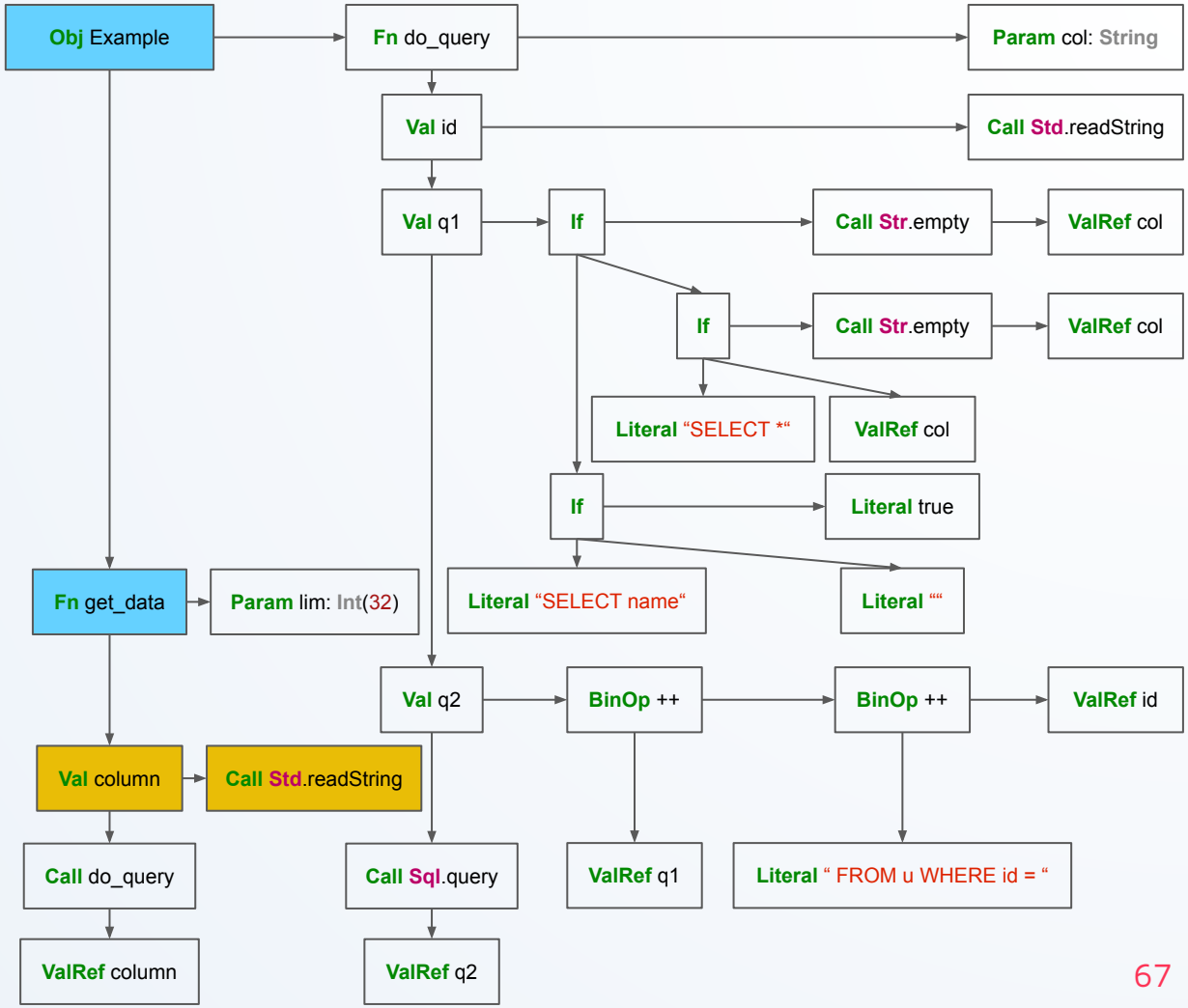
target = lim
seen = false

UnusedParameterCheck

```
case FunDef(_, params, _, body) =>
  params.foreach(p => {
    val visitor = new UsageVisitor(p.name)
    visitor.visit(body)
    if (!visitor.seen) {
      reportIssue(p)
    }
```

UsageVisitor

```
case Variable(name) if name == target =>
  seen = true
```



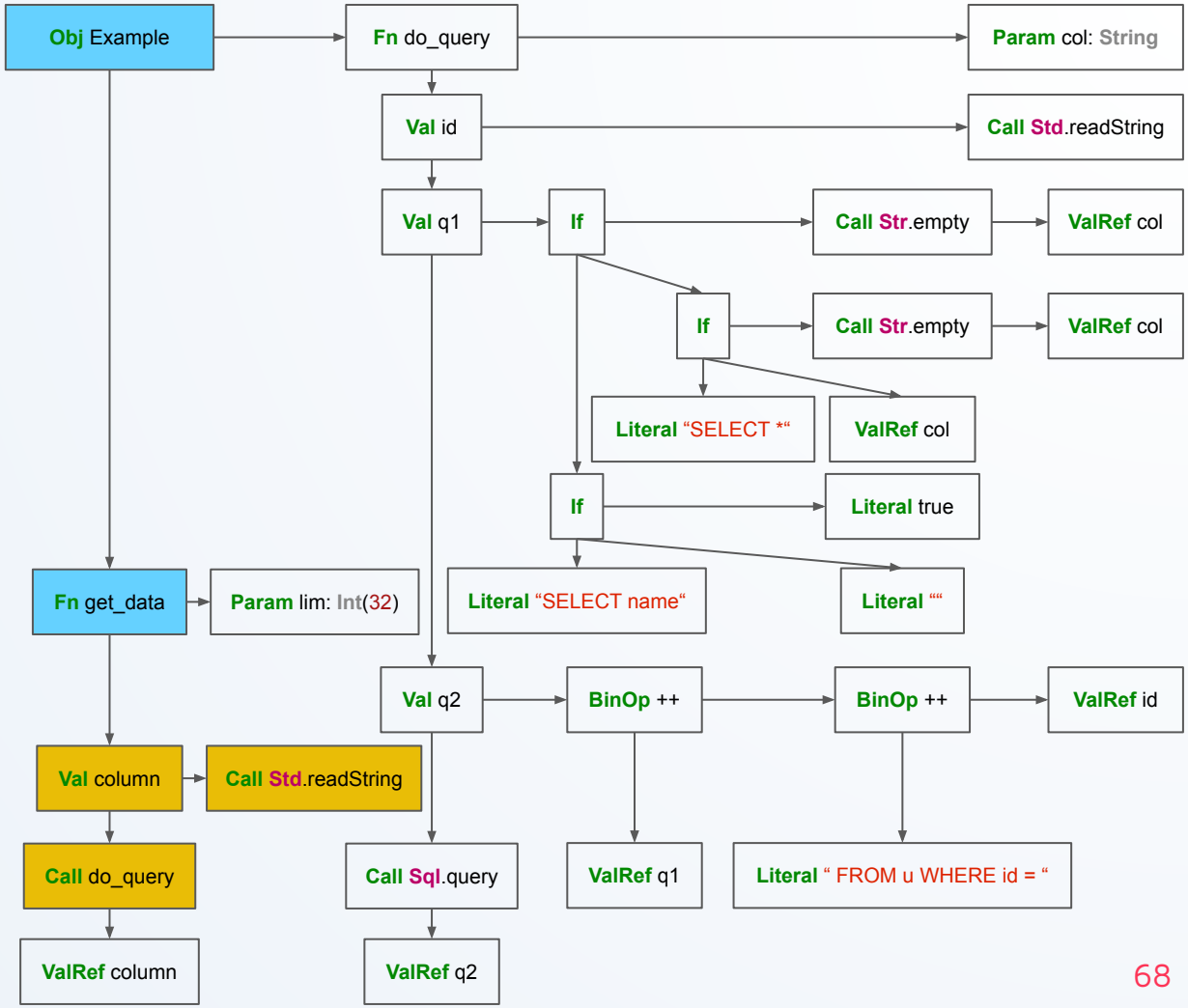
target = lim
seen = false

UnusedParameterCheck

```
case FunDef(_, params, _, body) =>
  params.foreach(p => {
    val visitor = new UsageVisitor(p.name)
    visitor.visit(body)
    if (!visitor.seen) {
      reportIssue(p)
    }
```

UsageVisitor

```
case Variable(name) if name == target =>
  seen = true
```



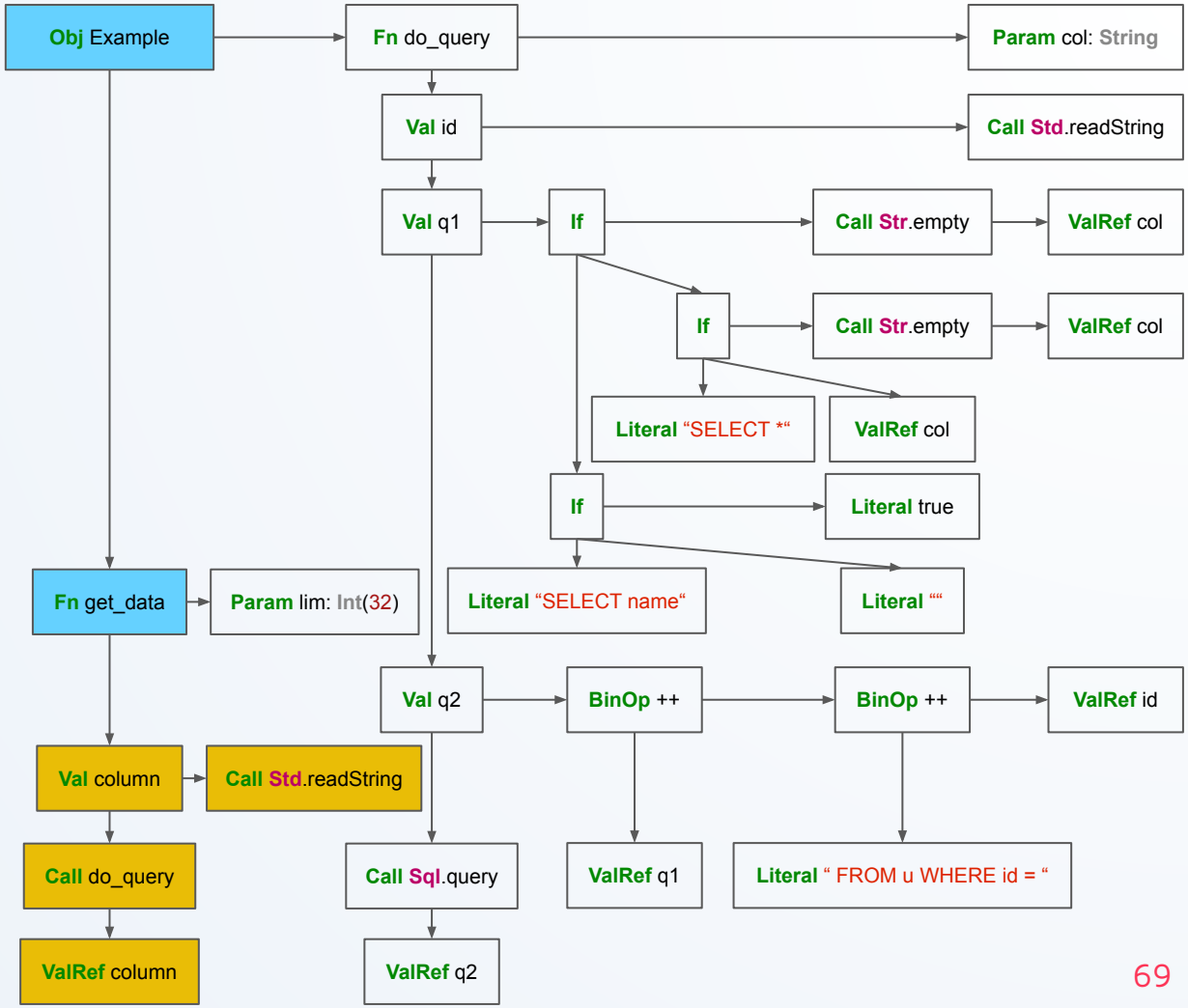
target = lim
seen = false

UnusedParameterCheck

```
case FunDef(_, params, _, body) =>
  params.foreach(p => {
    val visitor = new UsageVisitor(p.name)
    visitor.visit(body)
    if (!visitor.seen) {
      reportIssue(p)
    }
```

UsageVisitor

```
case Variable(name) if name == target =>
  seen = true
```



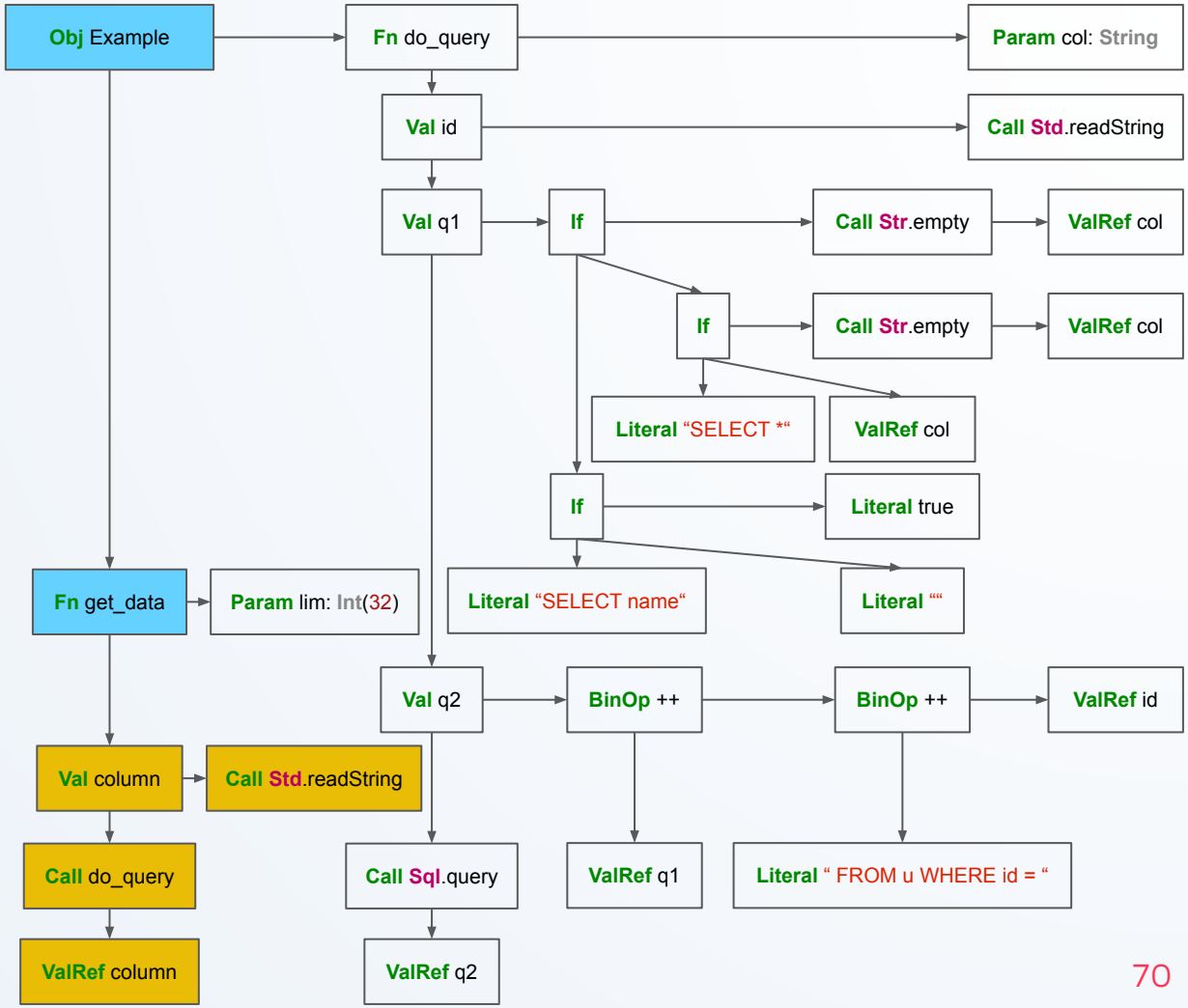
target = lim
seen = false

UnusedParameterCheck

```
case FunDef(_, params, _, body) =>
  params.foreach(p => {
    val visitor = new UsageVisitor(p.name)
    visitor.visit(body)
    if (!visitor.seen) {
      reportIssue(p)
    }
```

UsageVisitor

```
case Variable(name) if name == target =>
  seen = true
```



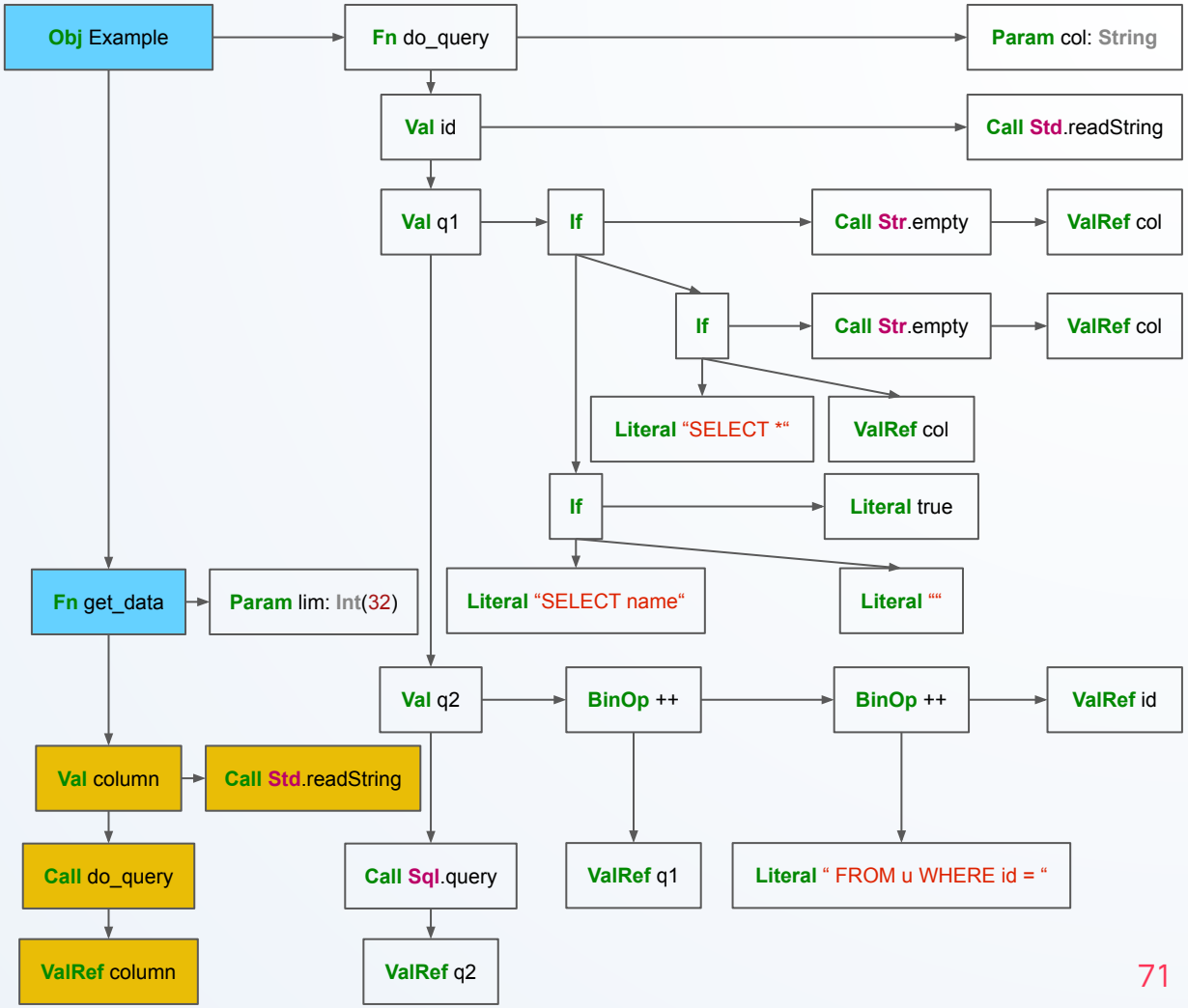
target = lim
seen = false

UnusedParameterCheck

```
case FunDef(_, params, _, body) =>
  params.foreach(p => {
    val visitor = new UsageVisitor(p.name)
    visitor.visit(body)
    if (!visitor.seen) {
      reportIssue(p)
    }
```

UsageVisitor

```
case Variable(name) if name == target =>
  seen = true
```

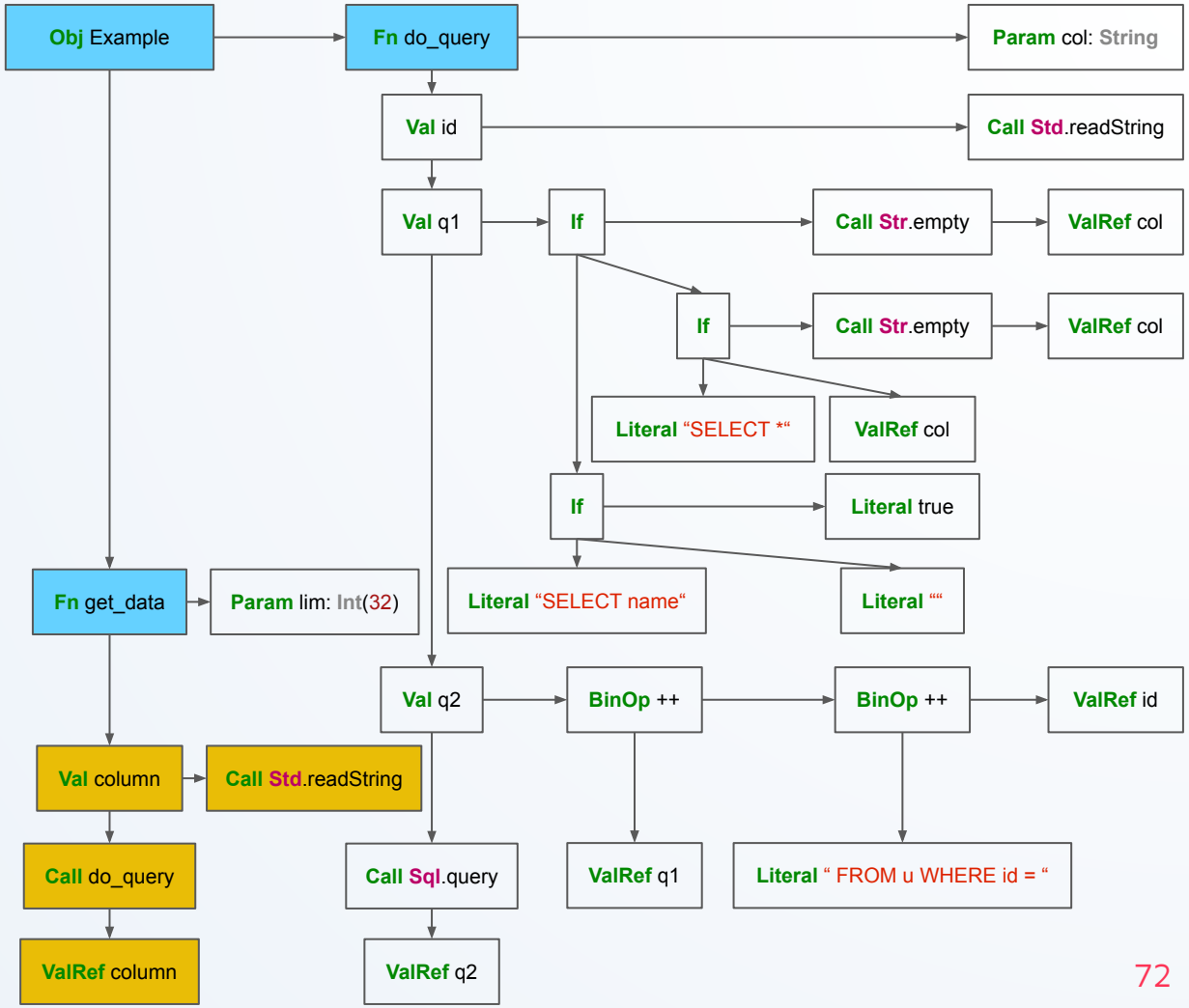


UnusedParameterCheck

```
case FunDef(_, params, _, body) =>
  params.foreach(p => {
    val visitor = new UsageVisitor(p.name)
    visitor.visit(body)
    if (!visitor.seen) {
      reportIssue(p)
    }
```

UsageVisitor

```
case Variable(name) if name == target =>
  seen = true
```



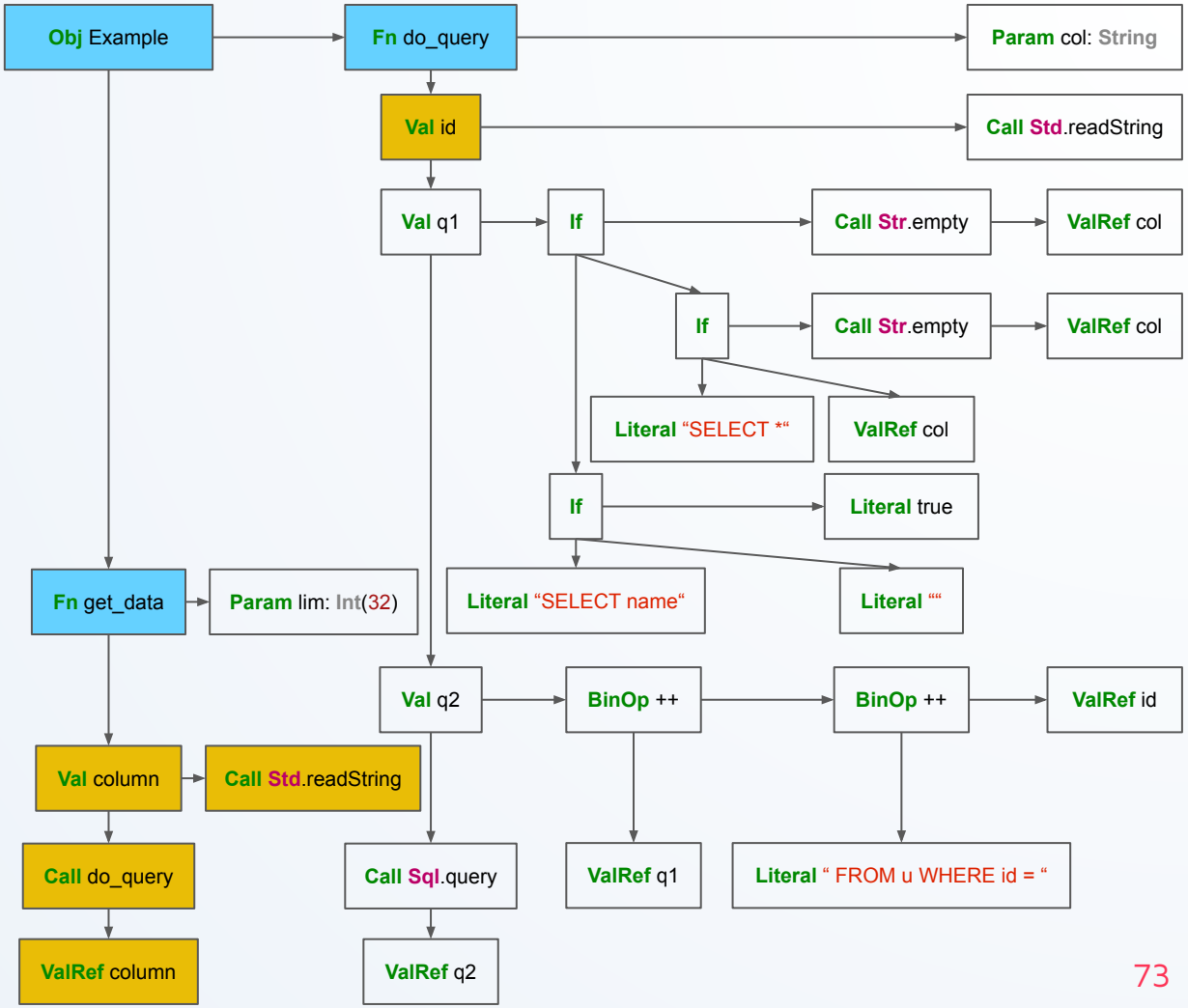
target = col
seen = false

UnusedParameterCheck

```
case FunDef(_, params, _, body) =>
  params.foreach(p => {
    val visitor = new UsageVisitor(p.name)
    visitor.visit(body)
    if (!visitor.seen) {
      reportIssue(p)
    }
```

UsageVisitor

```
case Variable(name) if name == target =>
  seen = true
```



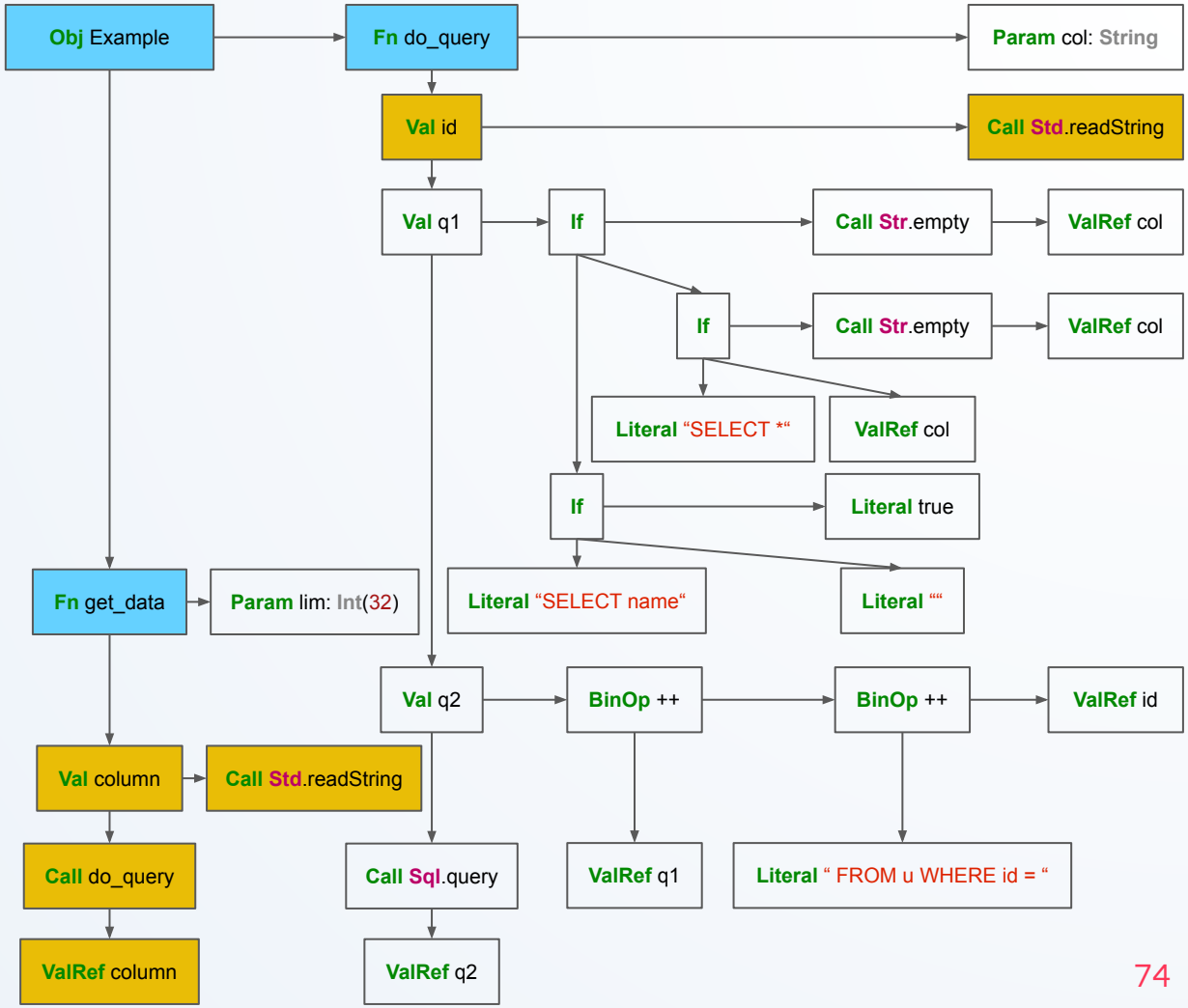
target = col
seen = false

UnusedParameterCheck

```
case FunDef(_, params, _, body) =>
  params.foreach(p => {
    val visitor = new UsageVisitor(p.name)
    visitor.visit(body)
    if (!visitor.seen) {
      reportIssue(p)
    }
```

UsageVisitor

```
case Variable(name) if name == target =>
  seen = true
```



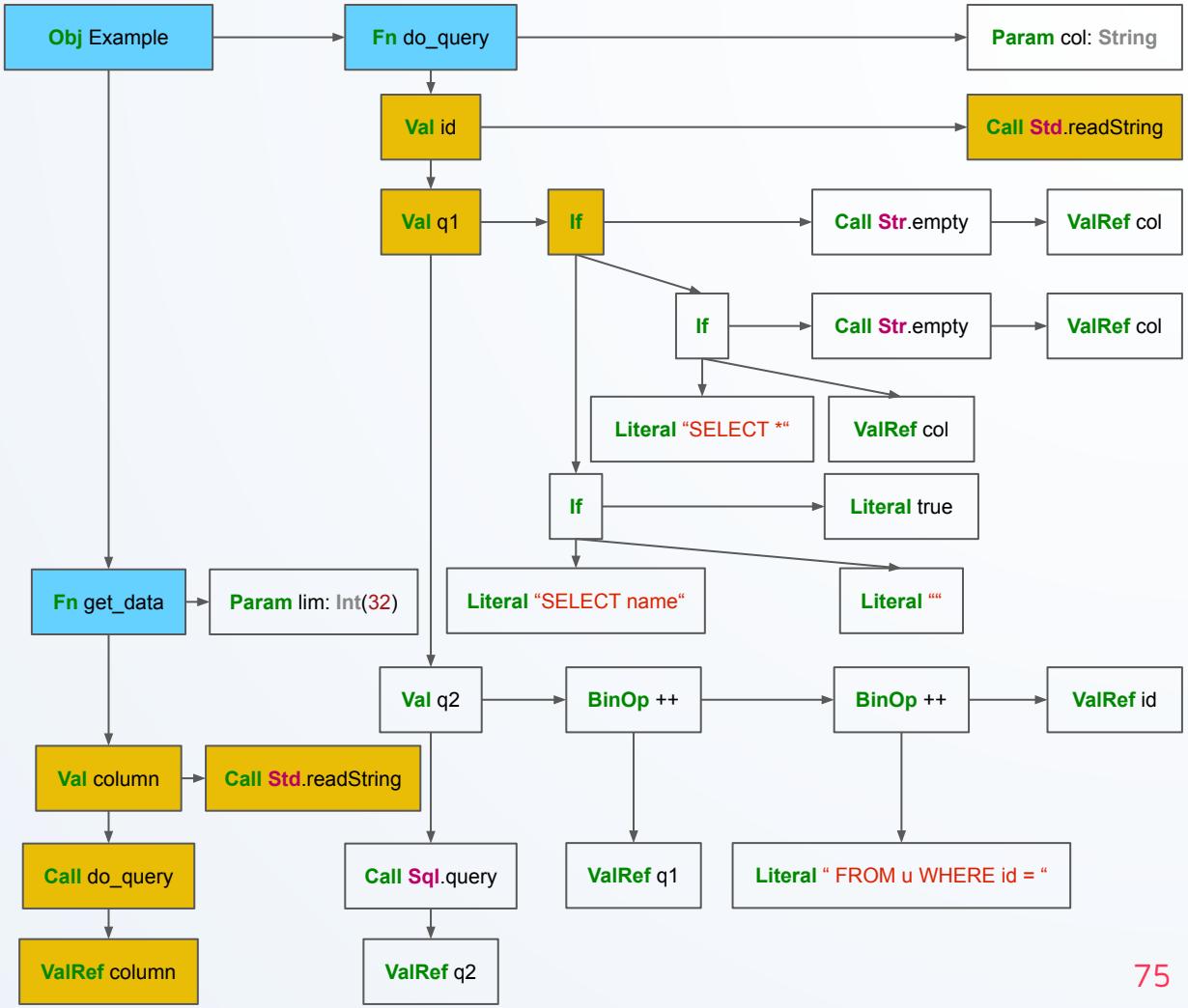
target = col
seen = false

UnusedParameterCheck

```
case FunDef(_, params, _, body) =>
  params.foreach(p => {
    val visitor = new UsageVisitor(p.name)
    visitor.visit(body)
    if (!visitor.seen) {
      reportIssue(p)
    }
```

UsageVisitor

```
case Variable(name) if name == target =>
  seen = true
```



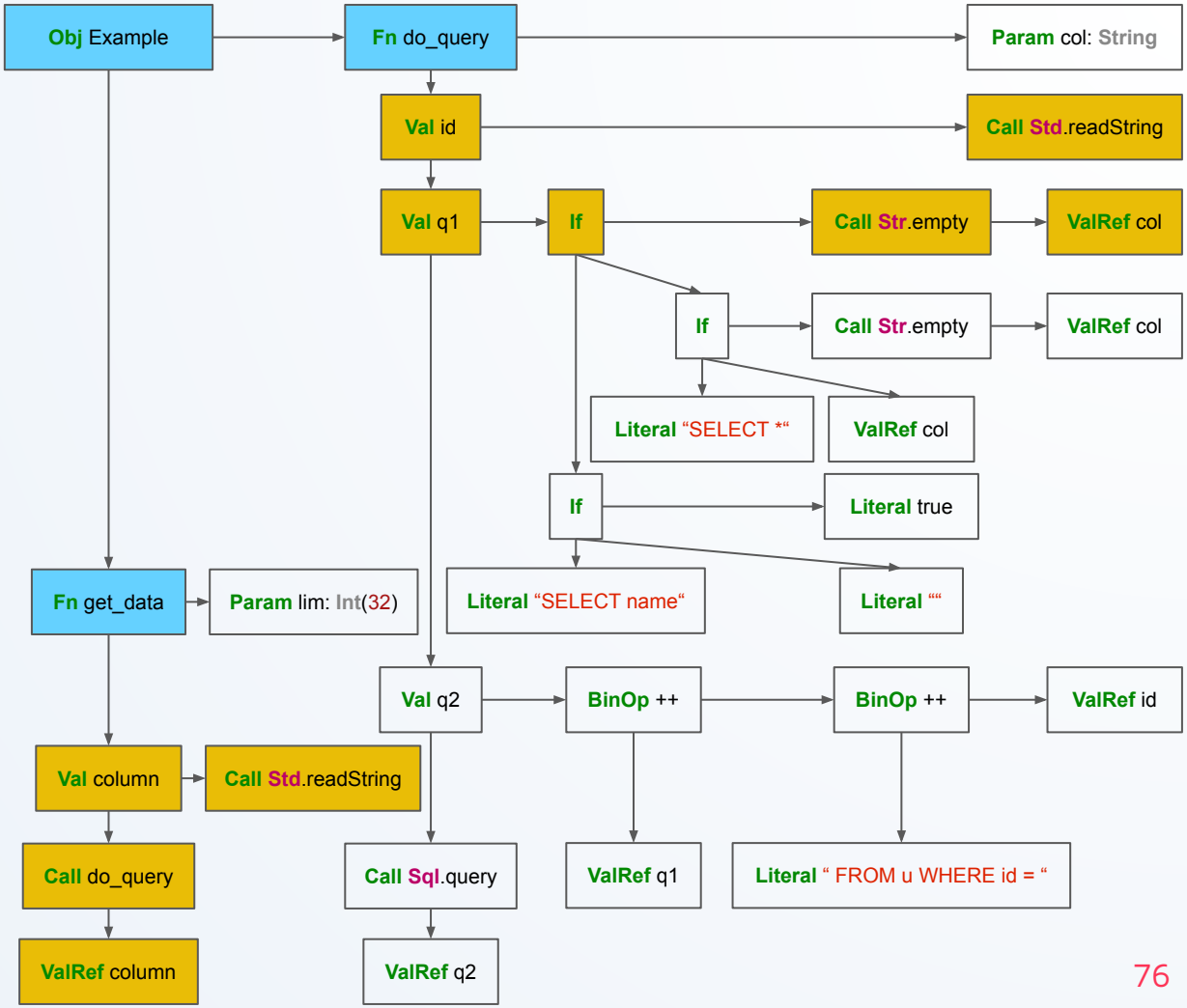
target = col
seen = false

UnusedParameterCheck

```
case FunDef(_, params, _, body) =>
  params.foreach(p => {
    val visitor = new UsageVisitor(p.name)
    visitor.visit(body)
    if (!visitor.seen) {
      reportIssue(p)
    }
```

UsageVisitor

```
case Variable(name) if name == target =>
  seen = true
```



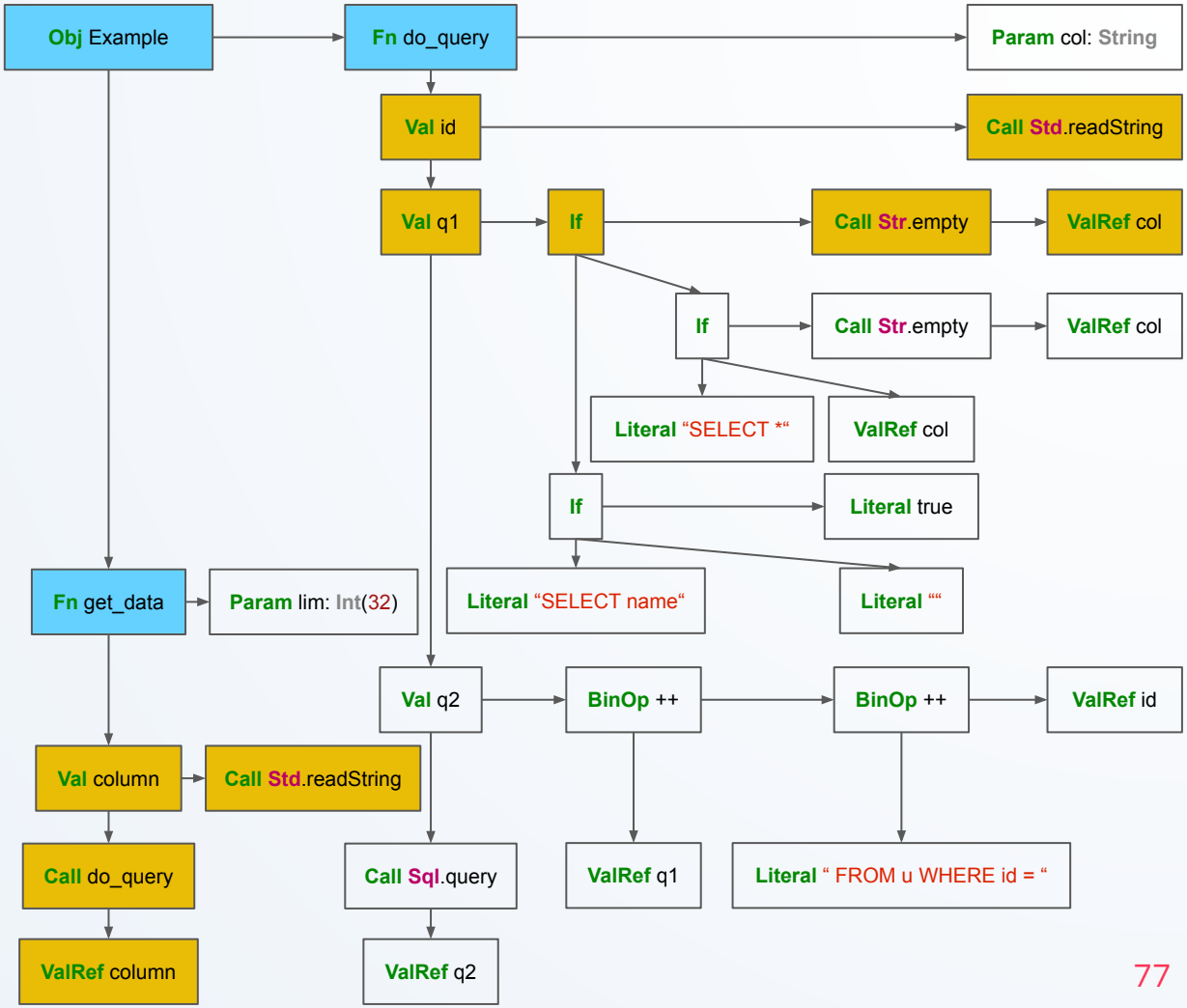
target = col
seen = true

UnusedParameterCheck

```
case FunDef(_, params, _, body) =>
  params.foreach(p => {
    val visitor = new UsageVisitor(p.name)
    visitor.visit(body)
    if (!visitor.seen) {
      reportIssue(p)
    }
```

UsageVisitor

```
case Variable(name) if name == target =>
  seen = true
```



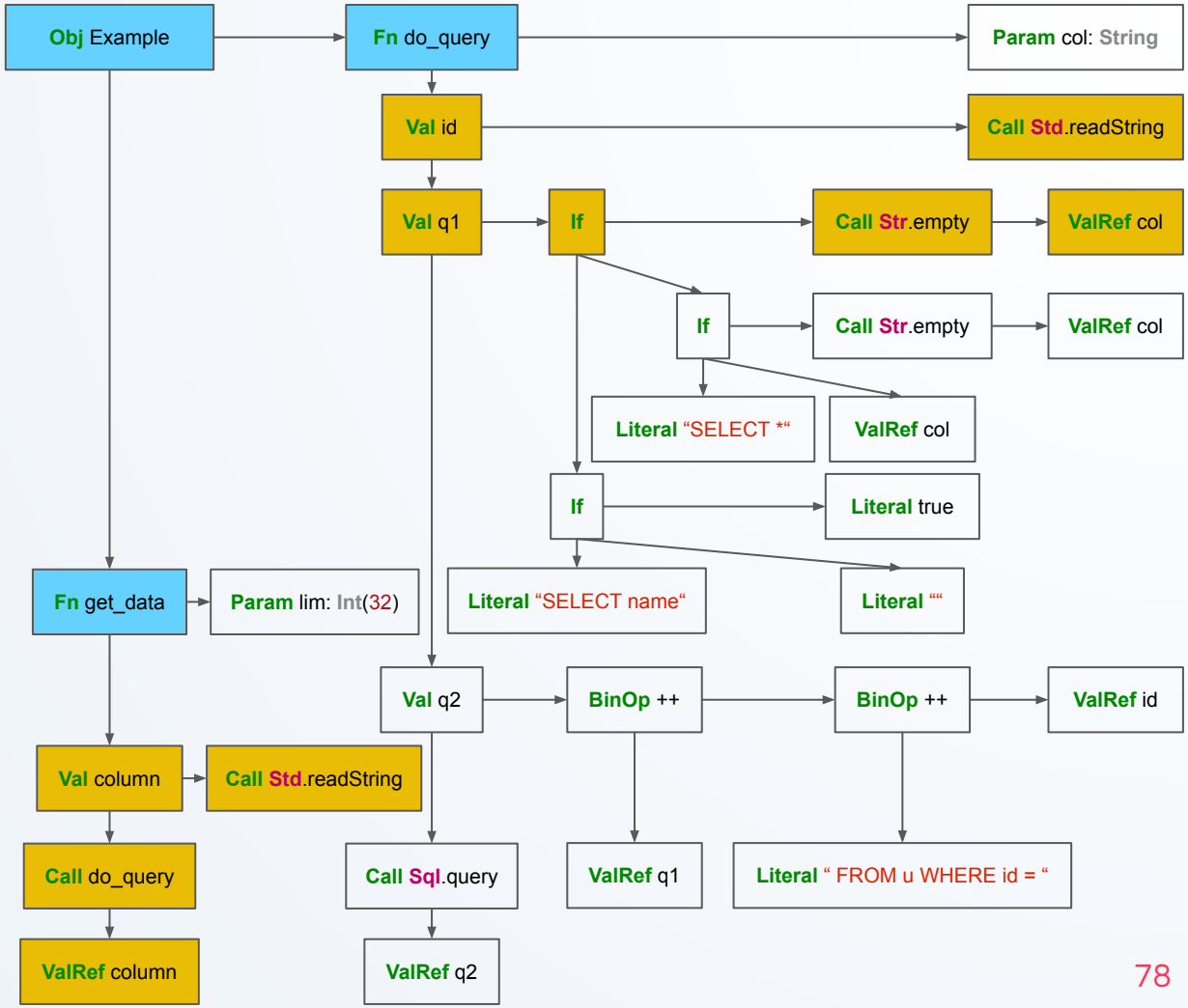
target = col
seen = true

UnusedParameterCheck

```
case FunDef(_, params, _, body) =>
  params.foreach(p => {
    val visitor = new UsageVisitor(p.name)
    visitor.visit(body)
    if (!visitor.seen) {
      reportIssue(p)
    }
```

UsageVisitor

```
case Variable(name) if name == target =>
  seen = true
```



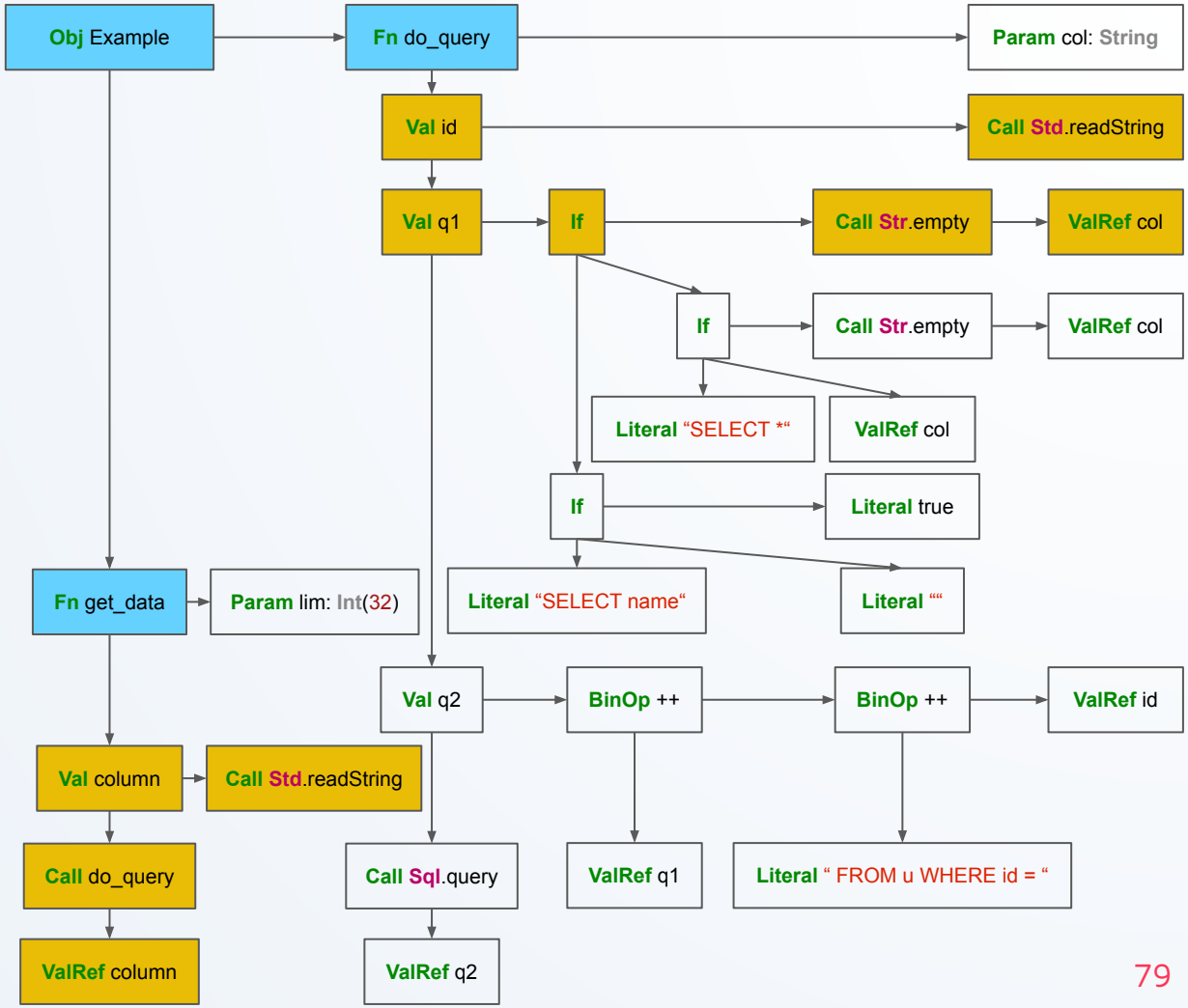
target = col
seen = true

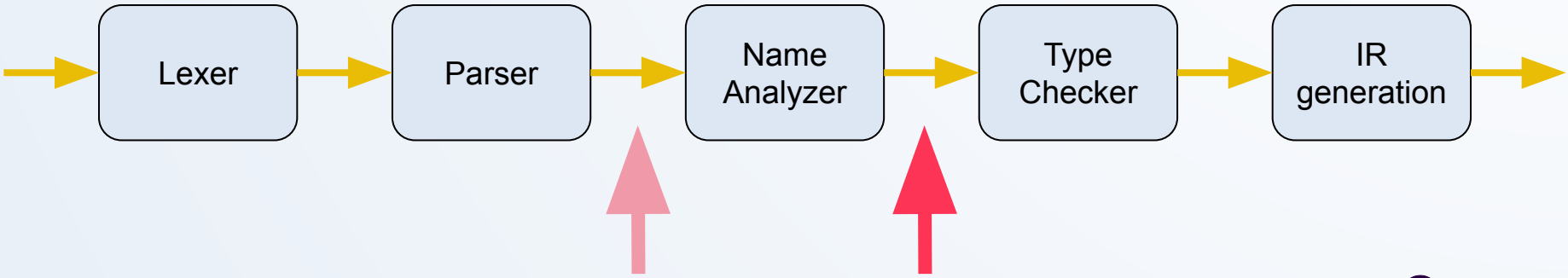
UnusedParameterCheck

```
case FunDef(_, params, _, body) =>
  params.foreach(p => {
    val visitor = new UsageVisitor(p.name)
    visitor.visit(body)
    if (!visitor.seen) {
      reportIssue(p)
    }
```

UsageVisitor

```
case Variable(name) if name == target =>
  seen = true
```




Lexer
Parser
Name Analyzer
Type Checker
IR generation

Redundant condition

```
object RedundantConditionCheck extends TreeVisitor {
 override def visit(t: Tree) = t match {
  case Ite(cond, thenn, elze) =>
   val visitor = new ReportVisitor(cond)
   visitor.visit(thenn)
   visitor.visit(elze)
   super.visit(t)
  case _ => super.visit(t)
 }
}
```

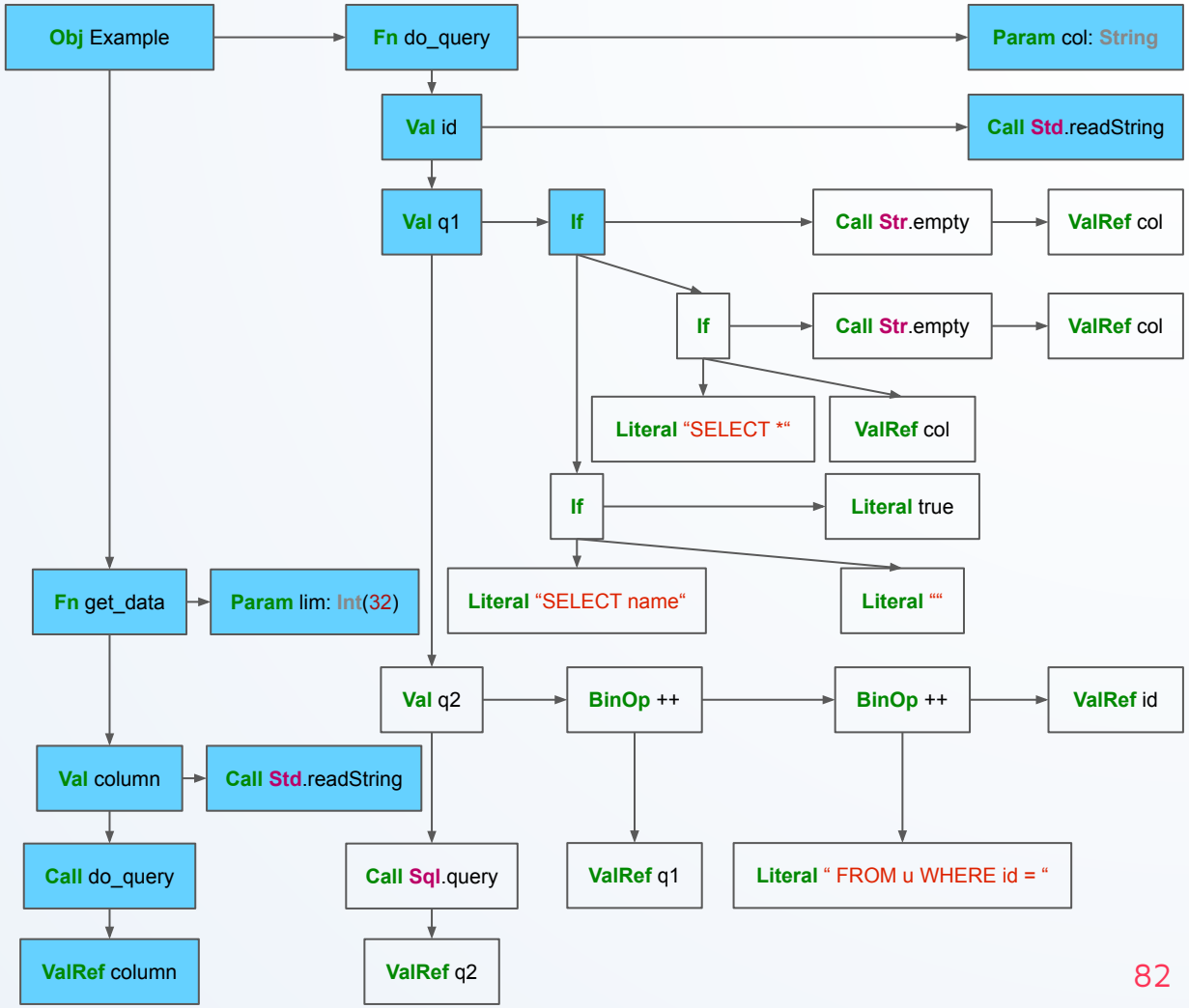
```
class ReportVisitor(upperCondition: Tree) extends TreeVisitor {
 override def visit(t: Tree) = t match {
  case Ite(cond, _, _) =>
   if (equals(cond, upperCondition)) {
    reportIssue(cond)
   }
   super.visit(t)
  case _ => super.visit(t)
 }
}
```

RedundantConditionCheck

```
case Ite(cond, thenn, elze) =>
    val visitor = new ReportVisitor(cond)
    visitor.visit(thenn)
    visitor.visit(elze)
```

ReportVisitor

```
  case Ite(cond, _, _) =>
   if (equals(cond, upperCondition)) {
    reportIssue(cond)
   }
```



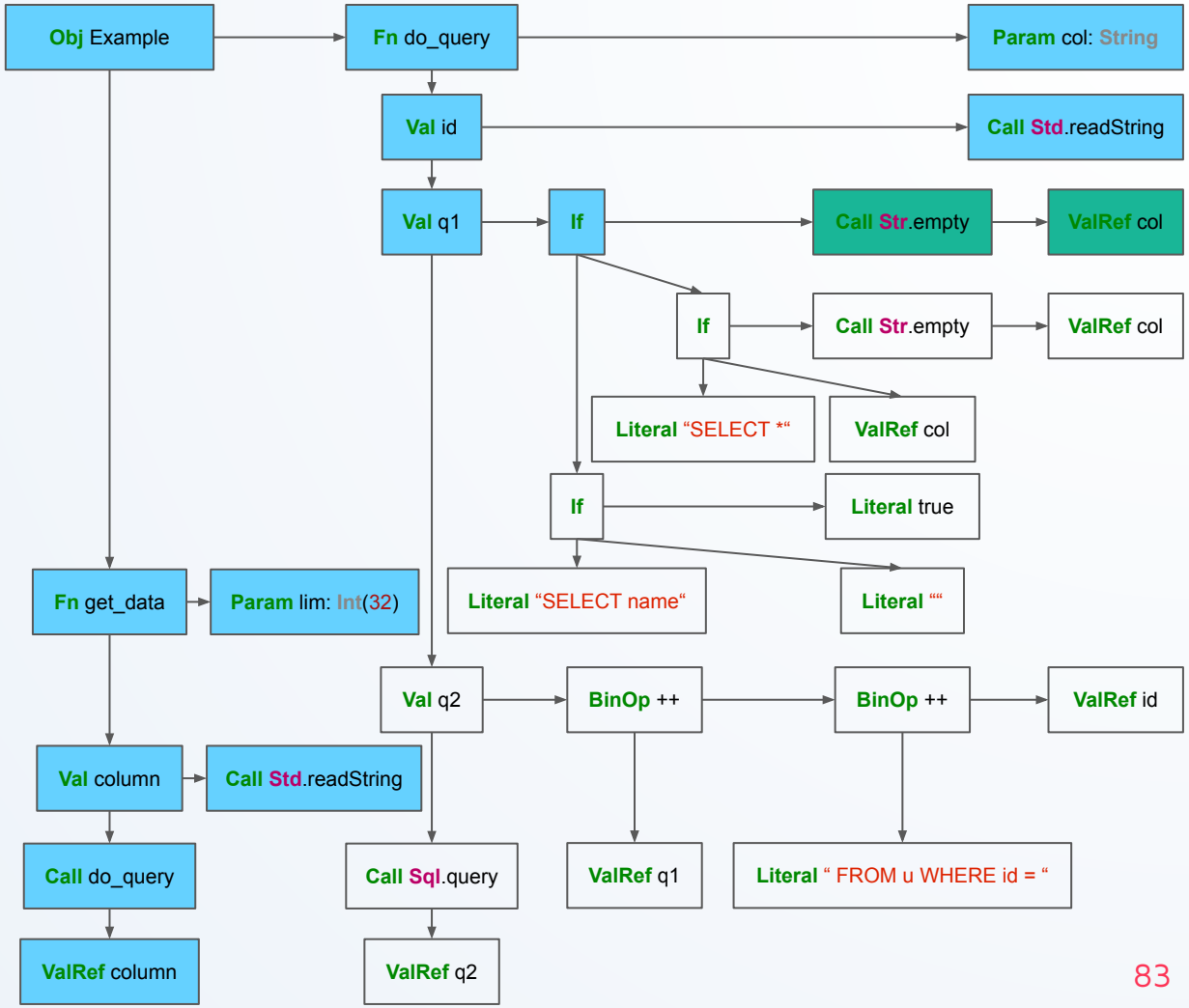
upperCondition = Str.empty(col)

RedundantConditionCheck

```
case Ite(cond, thenn, elze) =>
    val visitor = new ReportVisitor(cond)
    visitor.visit(thenn)
    visitor.visit(elze)
```

ReportVisitor

```
  case Ite(cond, _, _) =>
   if (equals(cond, upperCondition)) {
    reportIssue(cond)
   }
```



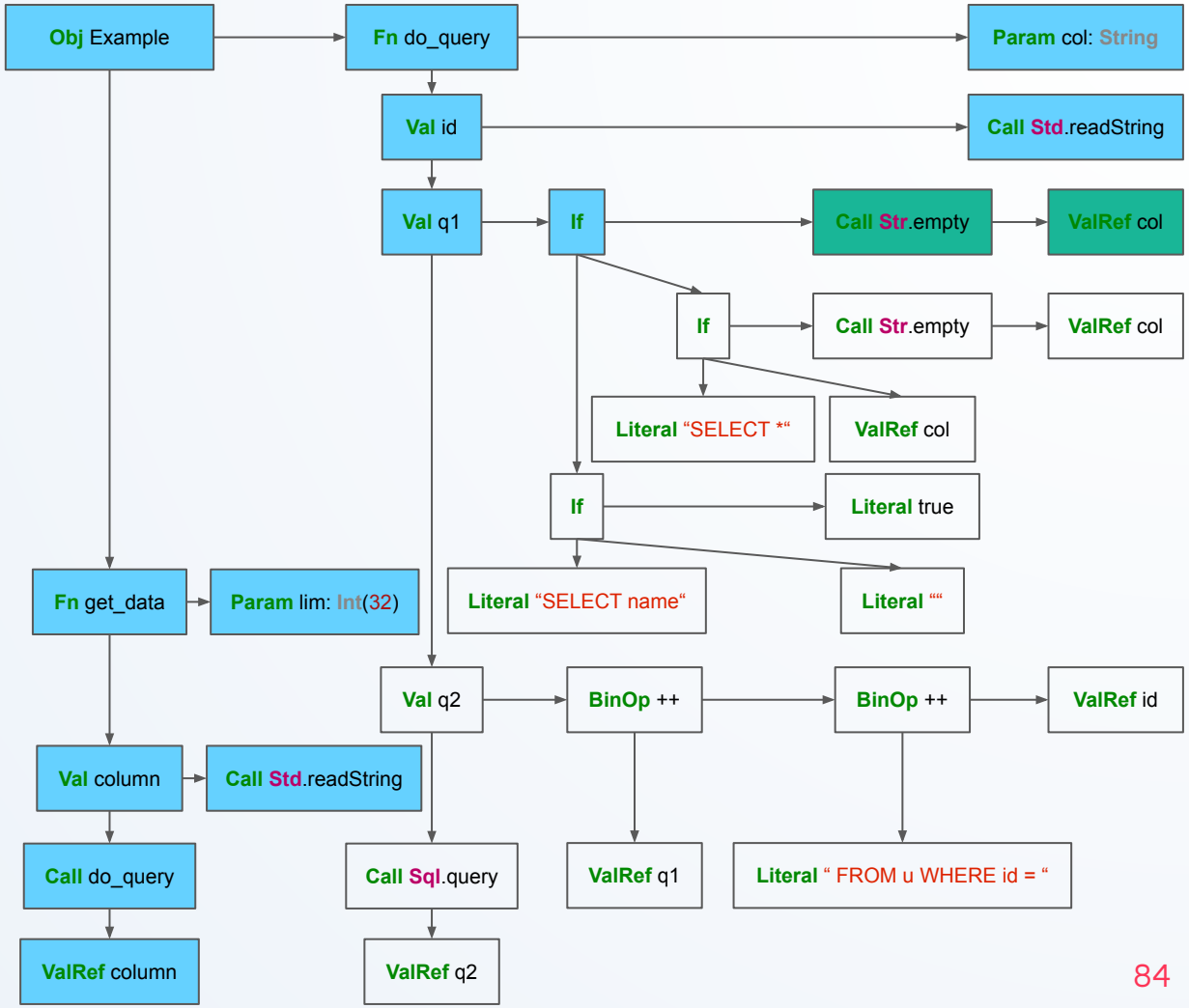
upperCondition = Str.empty(col)

RedundantConditionCheck

```
case Ite(cond, thenn, elze) =>
    val visitor = new ReportVisitor(cond)
    visitor.visit(thenn)
    visitor.visit(elze)
```

ReportVisitor

```
  case Ite(cond, _, _) =>
   if (equals(cond, upperCondition)) {
    reportIssue(cond)
   }
```



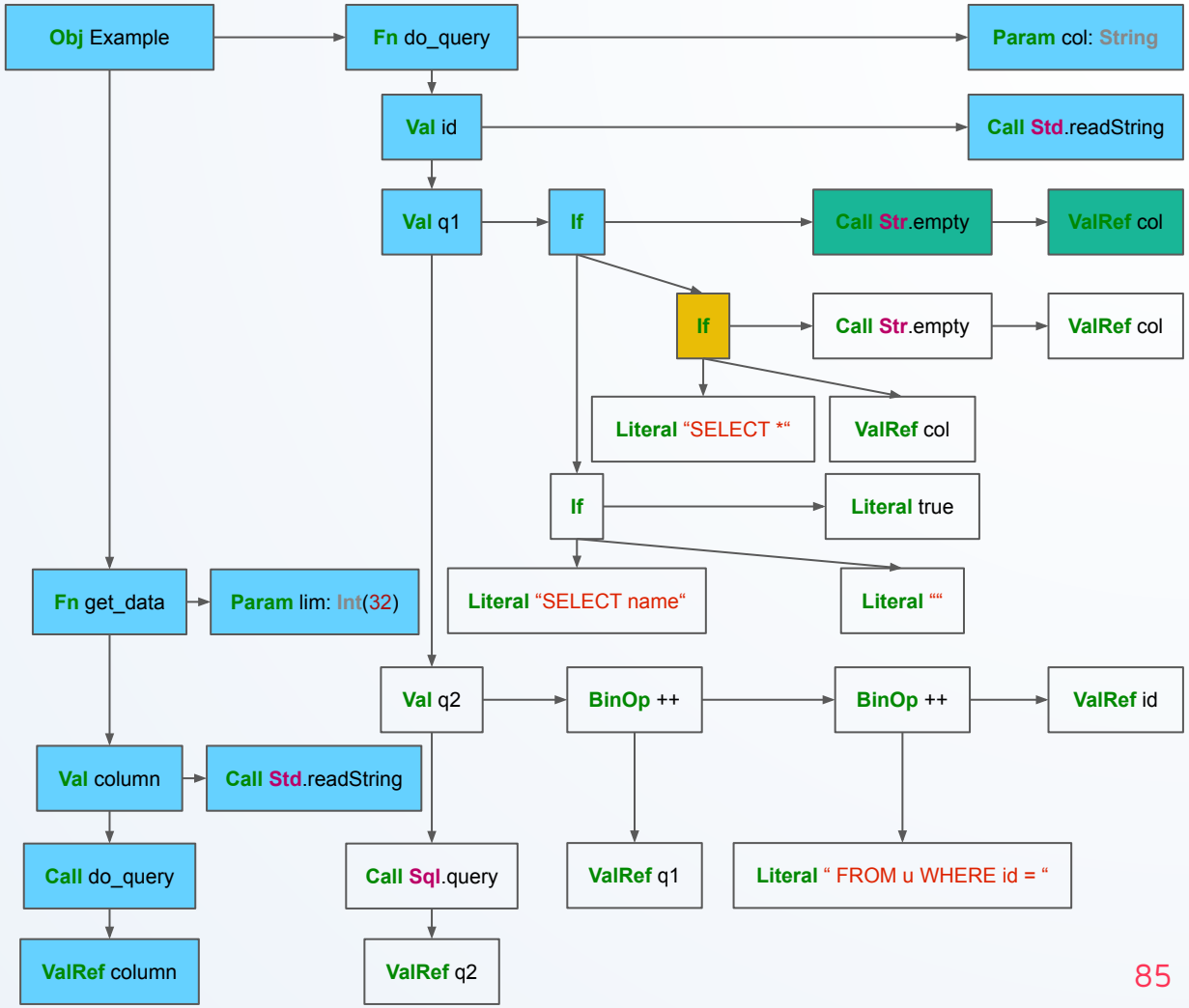
upperCondition = Str.empty(col)

RedundantConditionCheck

```
case Ite(cond, thenn, elze) =>
    val visitor = new ReportVisitor(cond)
    visitor.visit(thenn)
    visitor.visit(elze)
```

ReportVisitor

```
case Ite(cond, _, _) =>
  if (equals(cond, upperCondition)) {
    reportIssue(cond)
  }
```



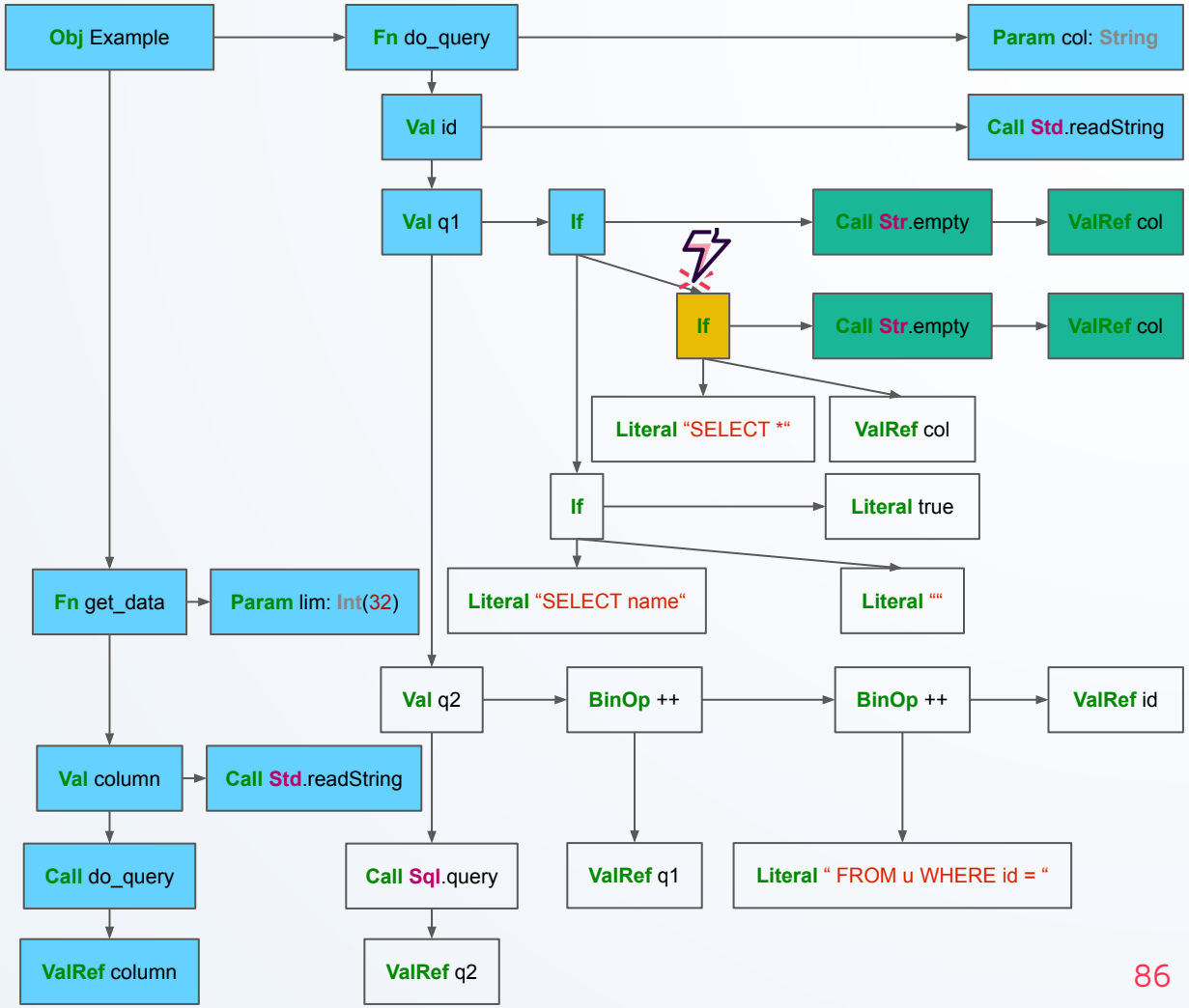
upperCondition = Str.empty(col)

RedundantConditionCheck

```
case Ite(cond, thenn, elze) =>
    val visitor = new ReportVisitor(cond)
    visitor.visit(thenn)
    visitor.visit(elze)
```

ReportVisitor

```
  case Ite(cond, _, _) =>
   if (equals(cond, upperCondition)) {
    reportIssue(cond)
   }
```



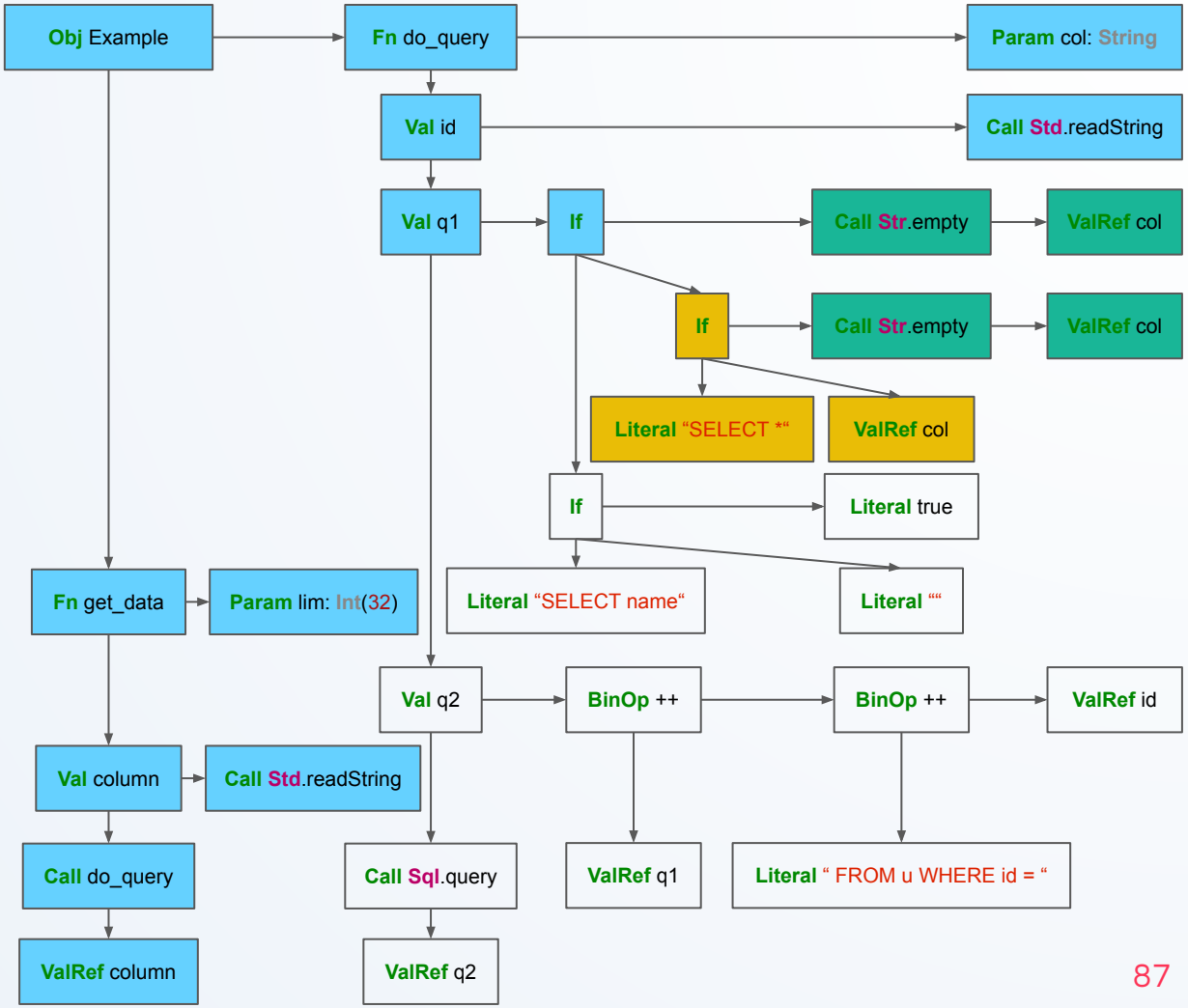
upperCondition = Str.empty(col)

RedundantConditionCheck

```
case Ite(cond, thenn, elze) =>
    val visitor = new ReportVisitor(cond)
    visitor.visit(thenn)
    visitor.visit(elze)
```

ReportVisitor

```
  case Ite(cond, _, _) =>
   if (equals(cond, upperCondition)) {
    reportIssue(cond)
   }
```



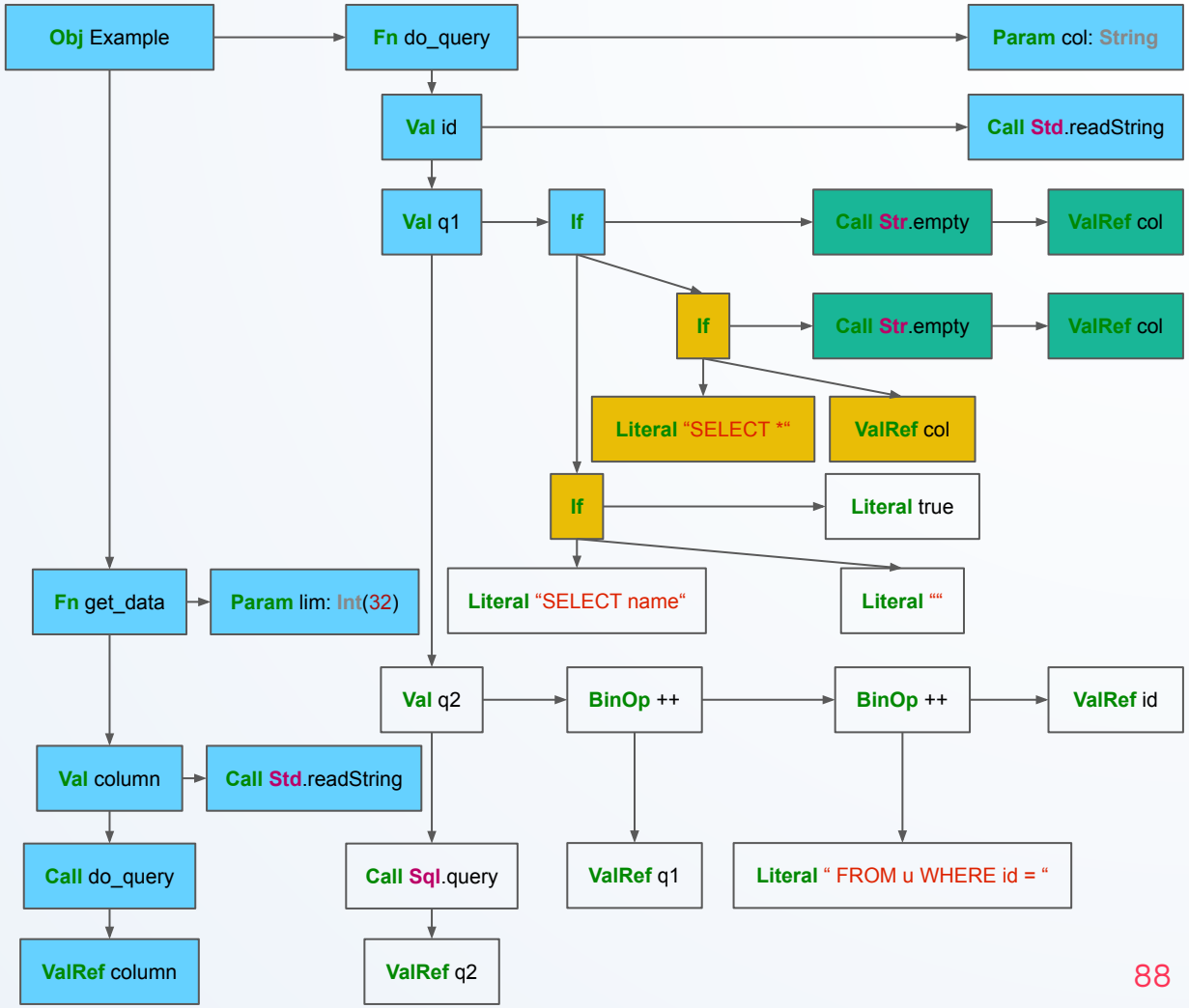
upperCondition = Str.empty(col)

RedundantConditionCheck

```
case Ite(cond, thenn, elze) =>
    val visitor = new ReportVisitor(cond)
    visitor.visit(thenn)
    visitor.visit(elze)
```

ReportVisitor

```
  case Ite(cond, _, _) =>
   if (equals(cond, upperCondition)) {
    reportIssue(cond)
   }
```



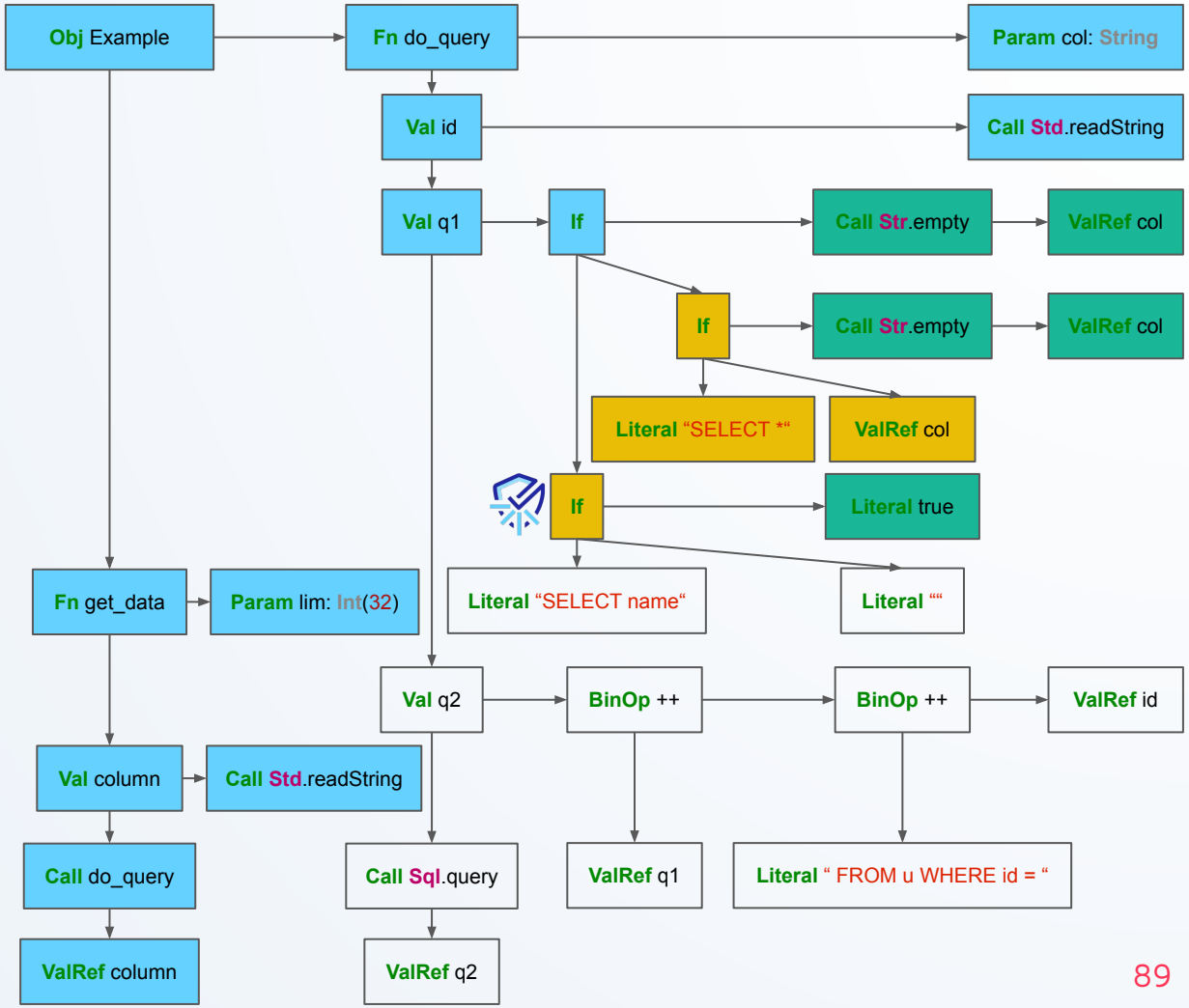
upperCondition = Str.empty(col)

RedundantConditionCheck

```
case Ite(cond, thenn, elze) =>
    val visitor = new ReportVisitor(cond)
    visitor.visit(thenn)
    visitor.visit(elze)
```

ReportVisitor

```
  case Ite(cond, _, _) =>
   if (equals(cond, upperCondition)) {
    reportIssue(cond)
   }
```



Outline

First hour

Intro to static analysis

Place for static analysis

AST-based analysis

Visitors → Matchers

Second hour

Taint Analysis

Symbolic Execution

Static Analysis Trade-off

Demo

AST Matchers

Domain-specific language

More expressive

Less flexible

Complicated under the hood

if with a trivial condition

```
object TrivialConditionCheck extends TreeVisitor {
 override def visit(tree :Tree) = tree match {
  case Ite(BooleanLiteral(_), _, _) =>
   reportIssue(tree)
   super.visit(tree)
  case _ => super.visit(tree)
 }
}
```

if with a trivial condition

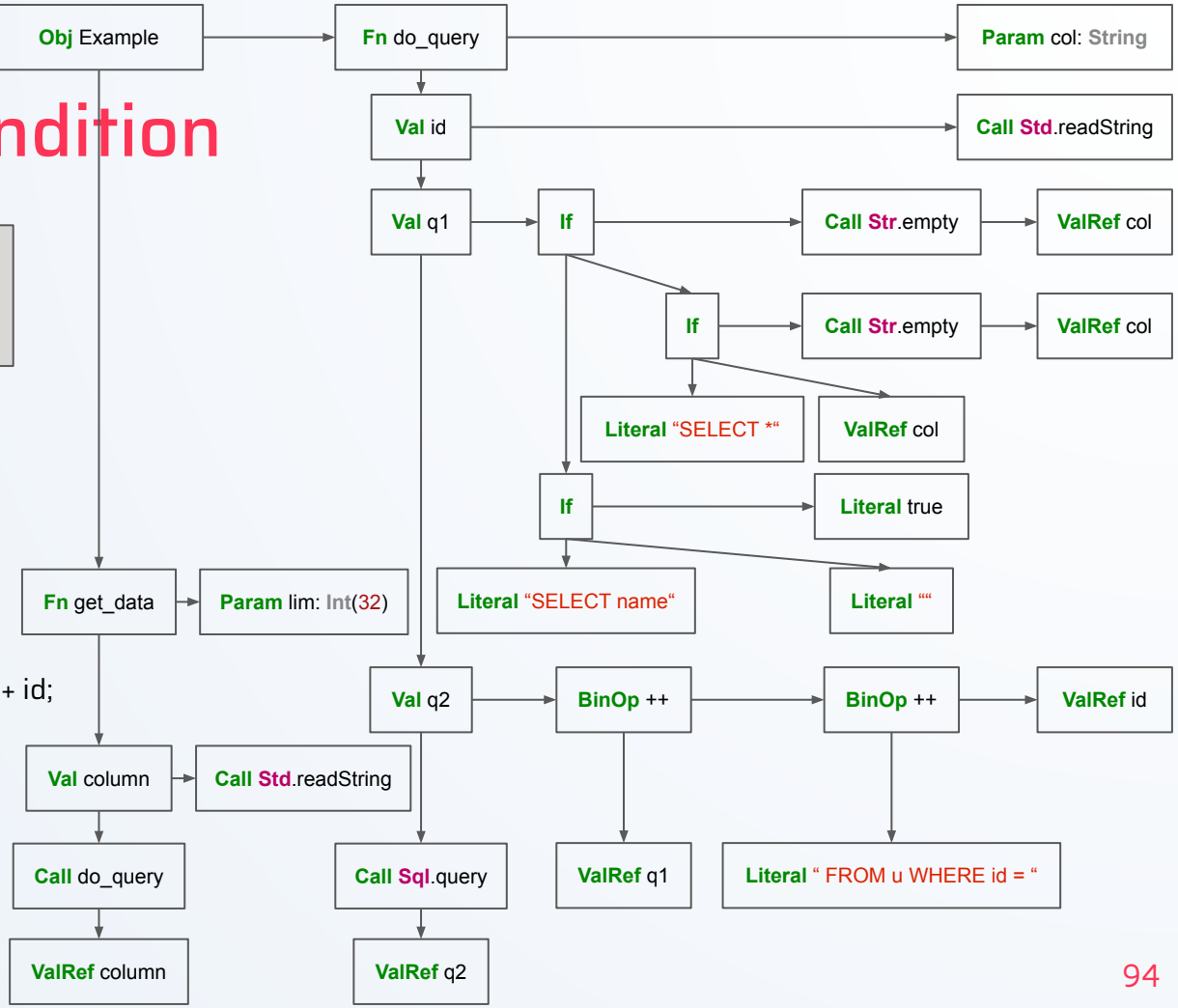
ite(hasCondition(literal))

```
case Ite(BooleanLiteral(_), _, _) =>
  reportIssue(tree)
  super.visit(tree)
```

if with a trivial condition

ite(hasCondition(literal))

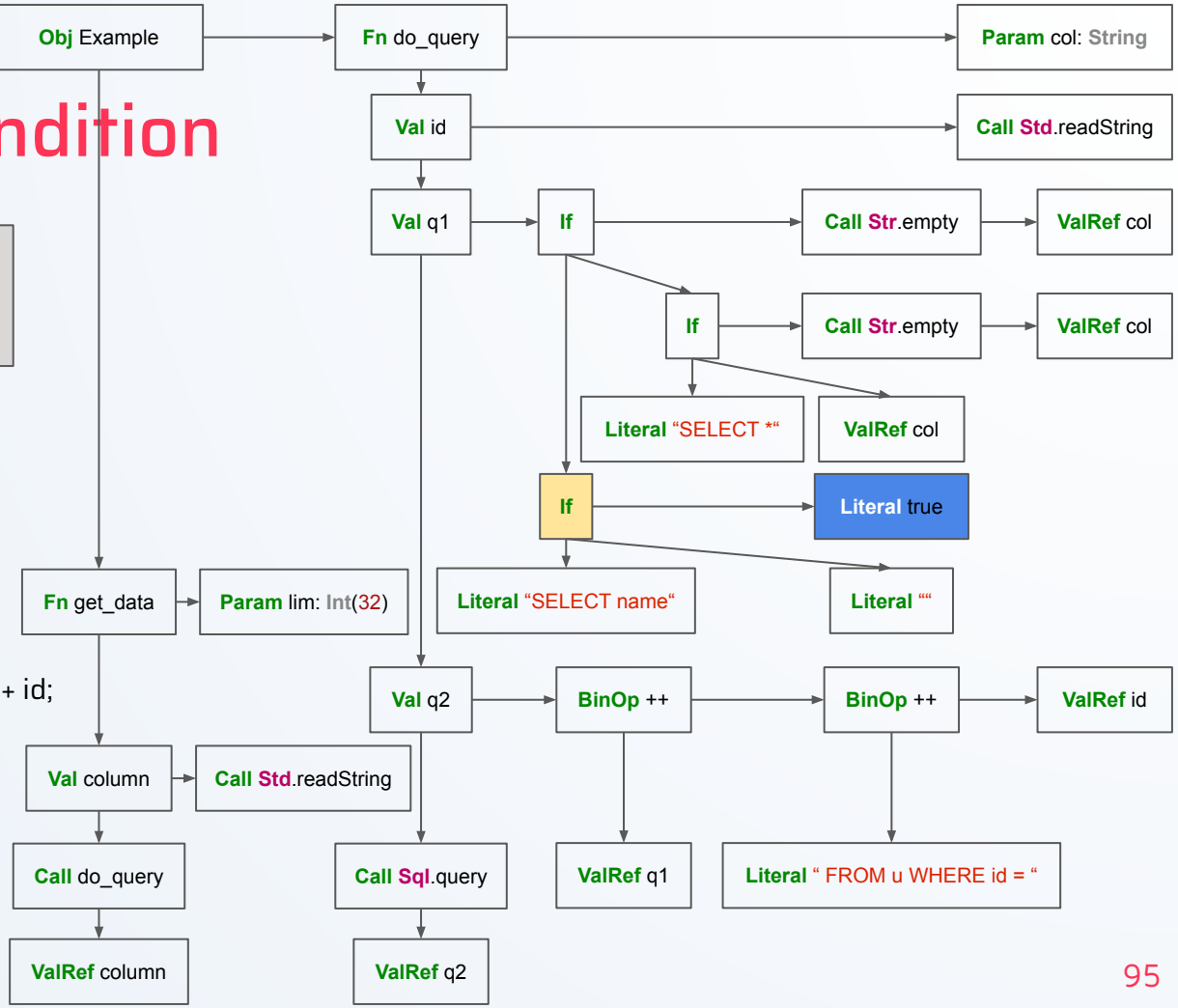
```
object Example

fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}

end Example
```



if with a trivial condition

```
ite(hasCondition(literal))
```

```
object Example

fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}

end Example
```



Matching patterns in AST

Nodes: “function”, “ite”, “literal”, “call”, “valRef”

Properties: “hasName”, “hasType”

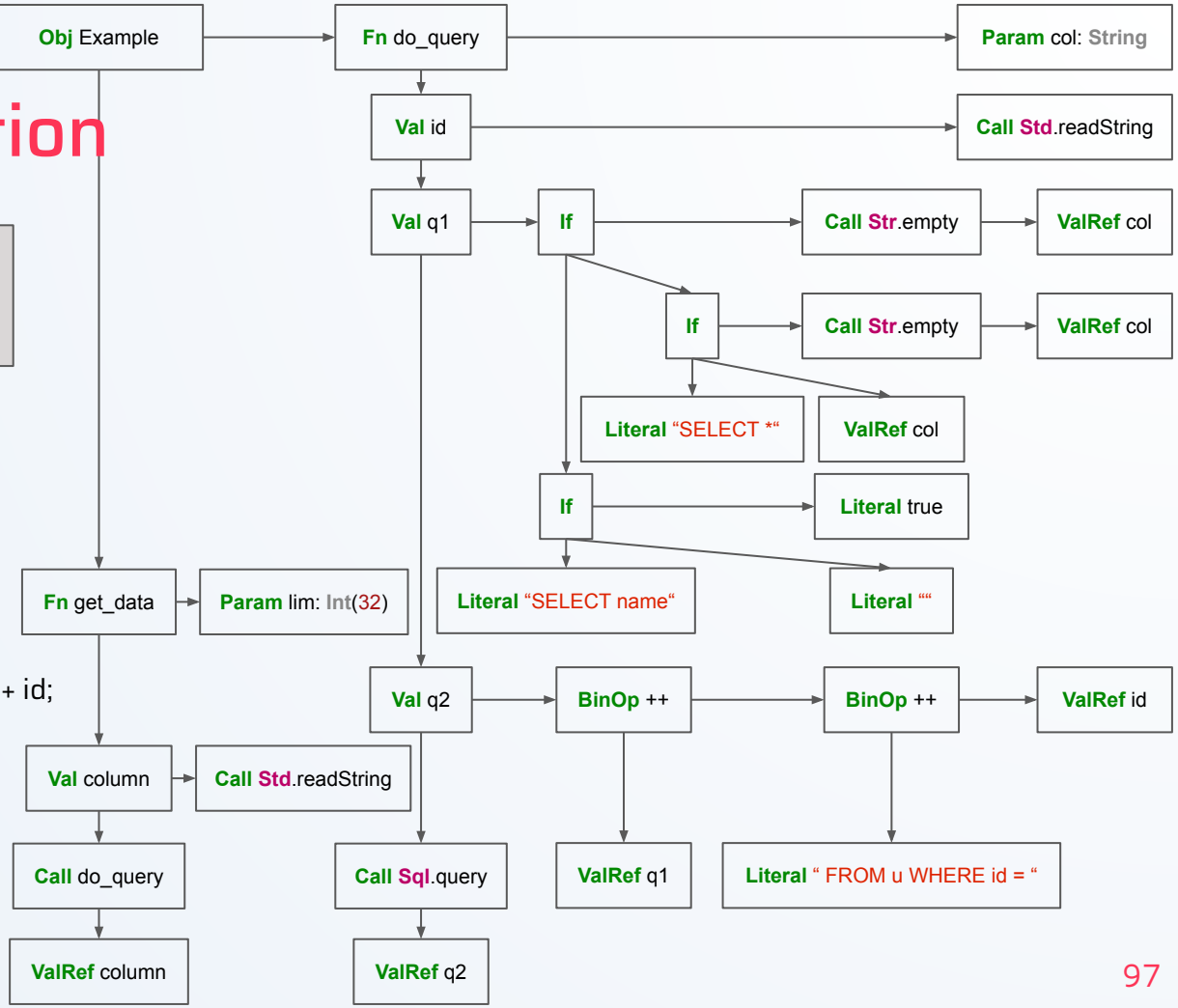
Relationships: “hasDescendant”, “hasCondition”, “hasParam”

Combinators: “not”, “bind” / “equalTo”, “or”; *implicit* “and”

Redundant condition

```
ite(
  hasCondition(bind("condition")),
  hasDescendant(expr(equalTo("condition"))))
```

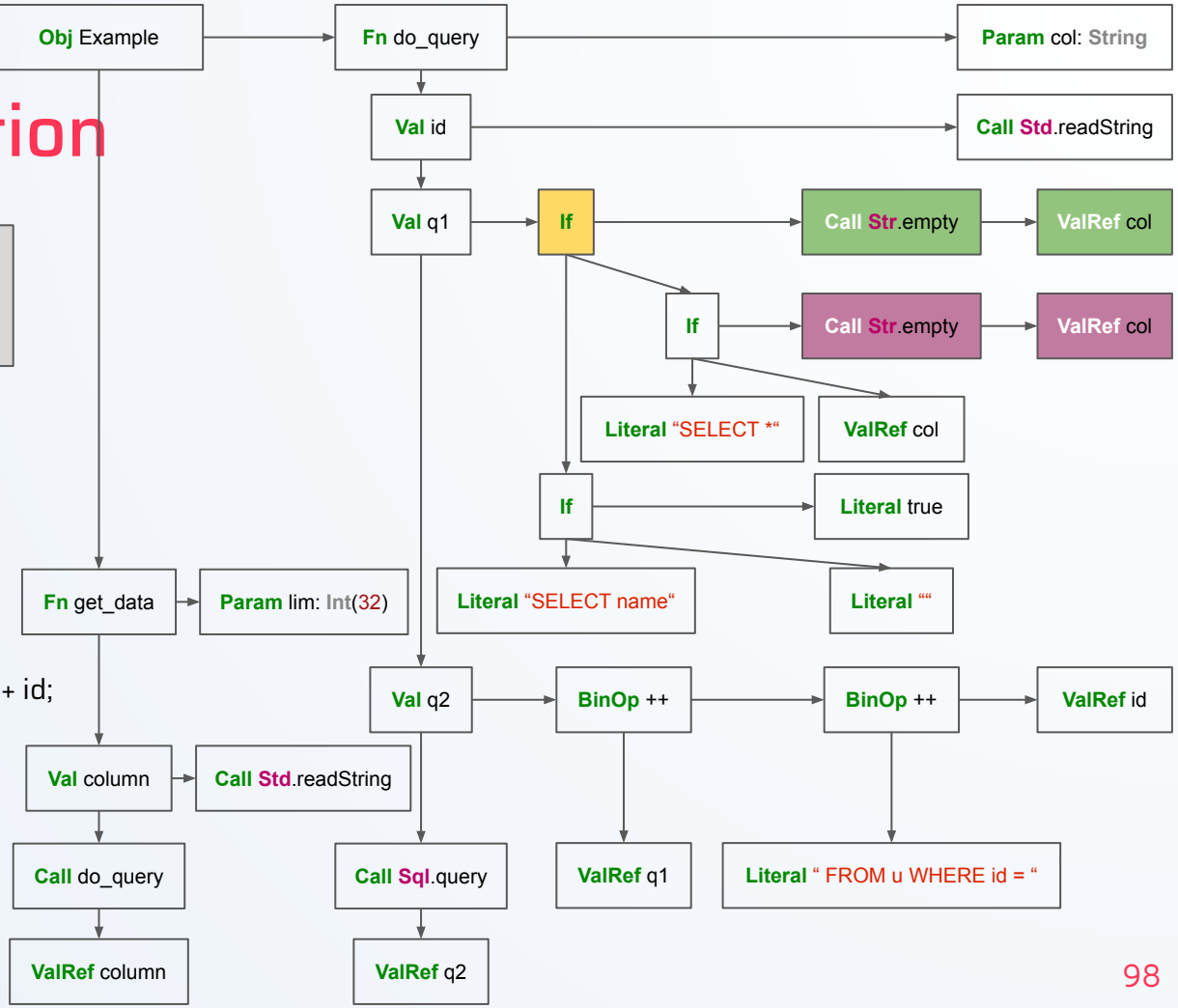
```
object Example

fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}

end Example
```



Redundant condition

```
ite(
  hasCondition(bind("condition")),
  hasDescendant(expr(equalTo("condition"))))
```

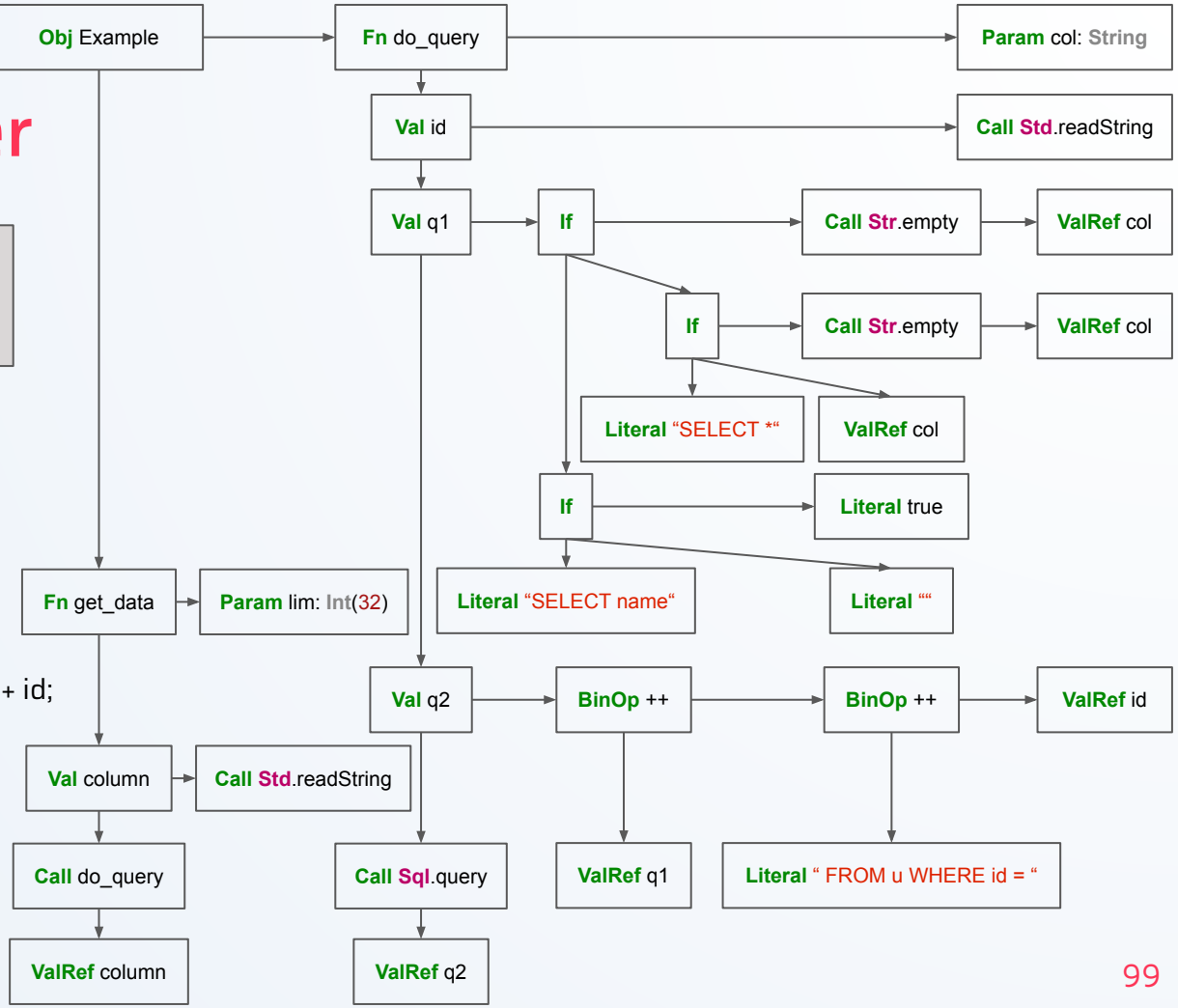
```
object Example

fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}

end Example
```



Unused parameter

```
function(
  hasParam(bind("param")),
  not(hasDescendant(valRef(to("param")))))
```

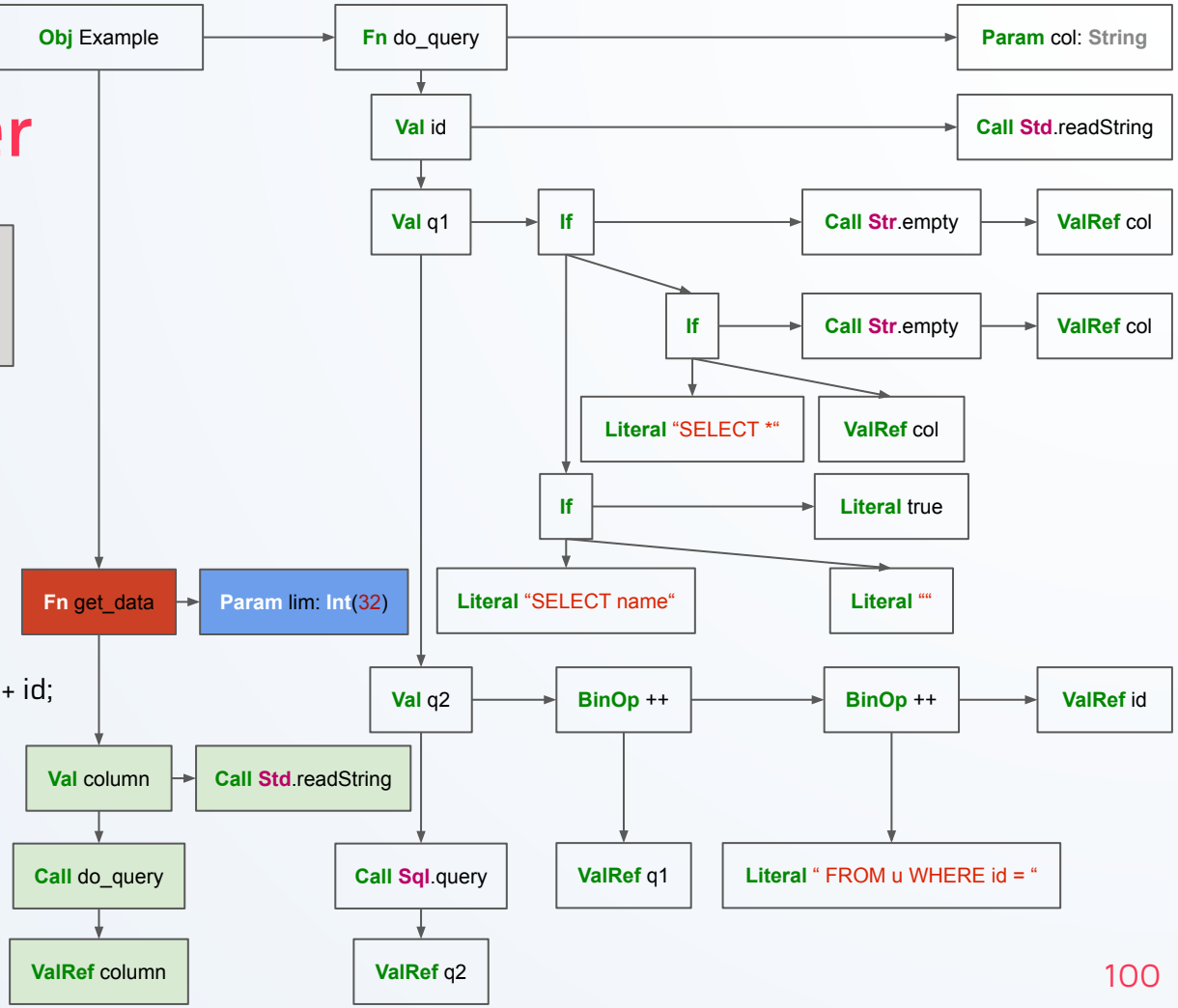
```
object Example

fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}

end Example
```



Unused parameter

```
function(
  hasParam(bind("param")),
  not(hasDescendant(valRef(to("param")))))
```

```
object Example

fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}

end Example
```



AST Analysis Limitations: Infeasible Rules

Dead code

ite(hasCondition(literal))

```
if (false) { ... } else { ... }
```

AST Analysis Limitations: Infeasible Rules

Dead code

```
or(ite(hasCondition(literal)),
    val(hasInit(literal), bind("x"), hasDescendant(ite(hasCondition(valRef("x")))))))
```

```
val x: Bool = false;
if (x) { ... } else { ... }
```

AST Analysis Limitations: Infeasible Rules

Dead code

```
or(or(ite(hasCondition(literal)),
        val(hasInit(literal), bind("x"), hasDescendant(ite(hasCondition(valRef("x"))))),
    val(hasInit(literal), bind("y"),
        hasDescendant(val(hasInit(valRef(equalTo("y"))), bind("x"), hasDescendant(ite(hasCondition(valRef("x")))))))
```

```
val x: Bool = false;
val y: Bool = x;
if (y) { ... } else { ... }
```

AST Analysis Limitations: Infeasible Rules

Dead code

Unused function

Null-pointer dereference

Division by zero

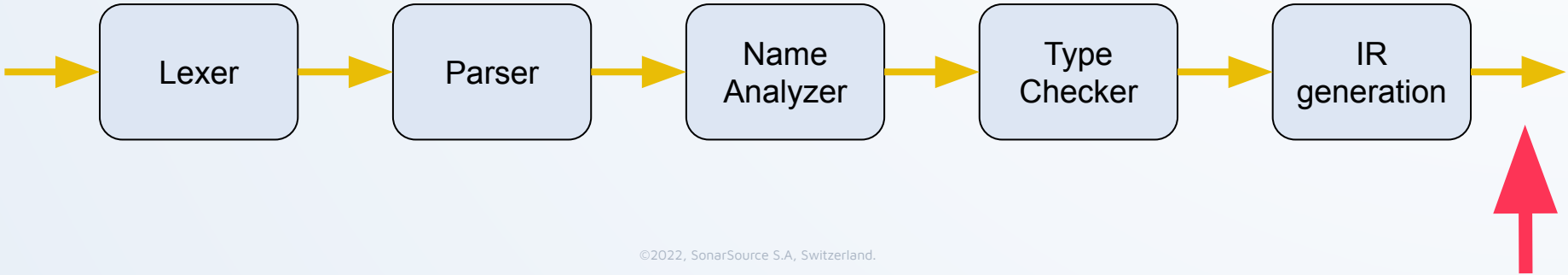
AST Analysis Limitations

Track values / data flow

Track execution order / control flow

Single function or translation unit

Control-Flow Graph Analysis



Control-Flow Graph (CFG)

Bring together instructions that execute together

Simplify-out unimportant AST details

Represent possible control-flow transfers

- ite branch, match, loop

CFG is useful in a Compiler

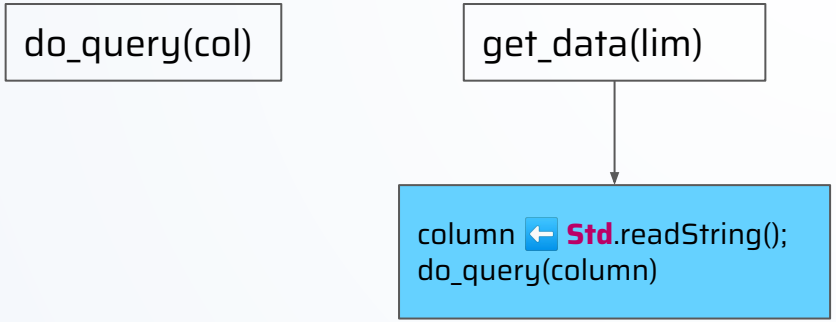
Register allocation

Insertion of garbage-collector checkpoints

Optimizations (e.g., constant propagation, loop hoisting)

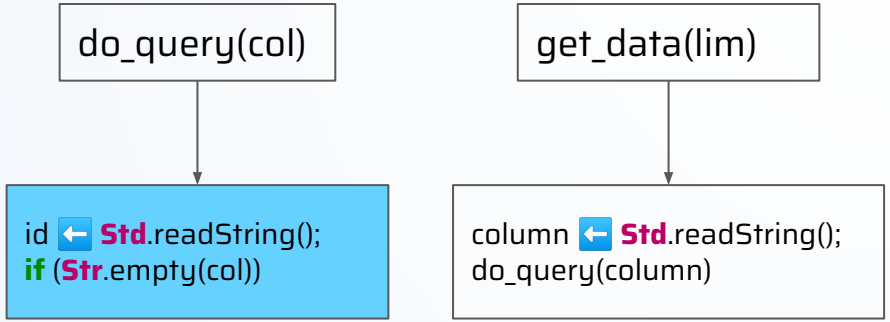
Code generation

CFG




```
object Example

fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}

end Example
```

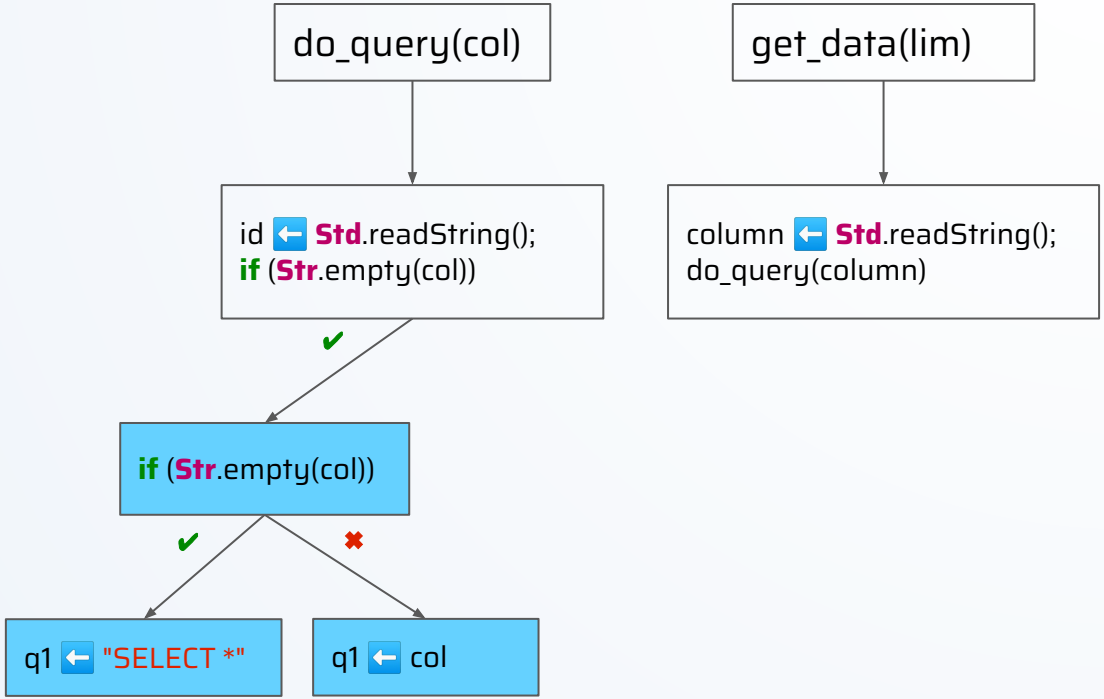
CFG




```
object Example

fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}

end Example
```

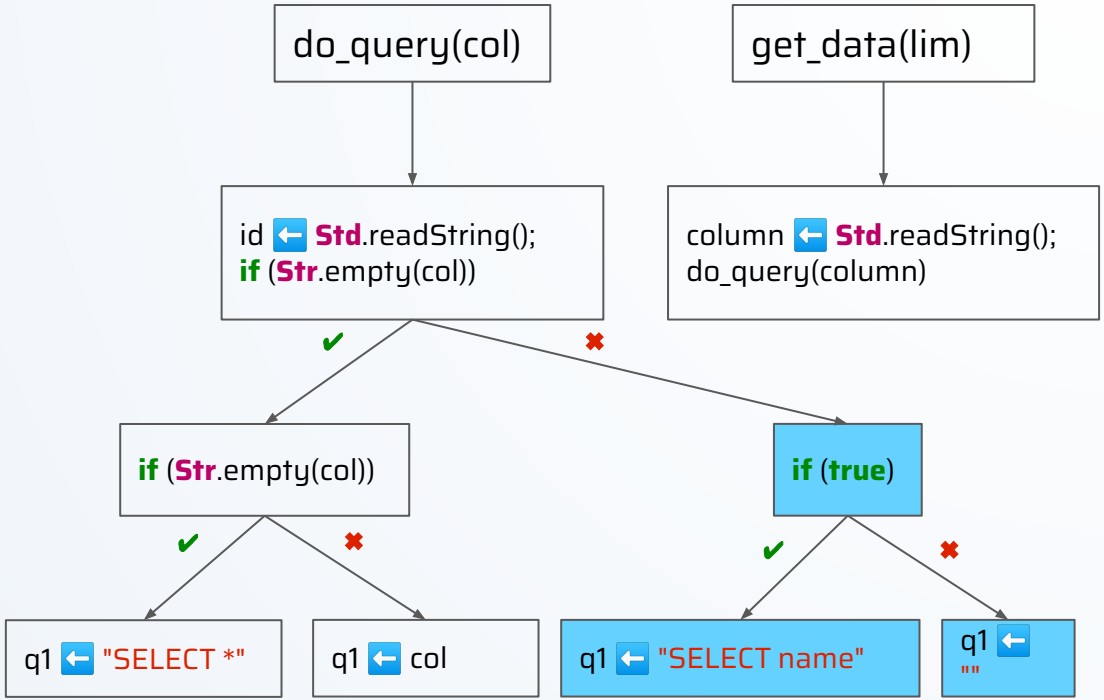
CFG

```
object Example

fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}

end Example
```



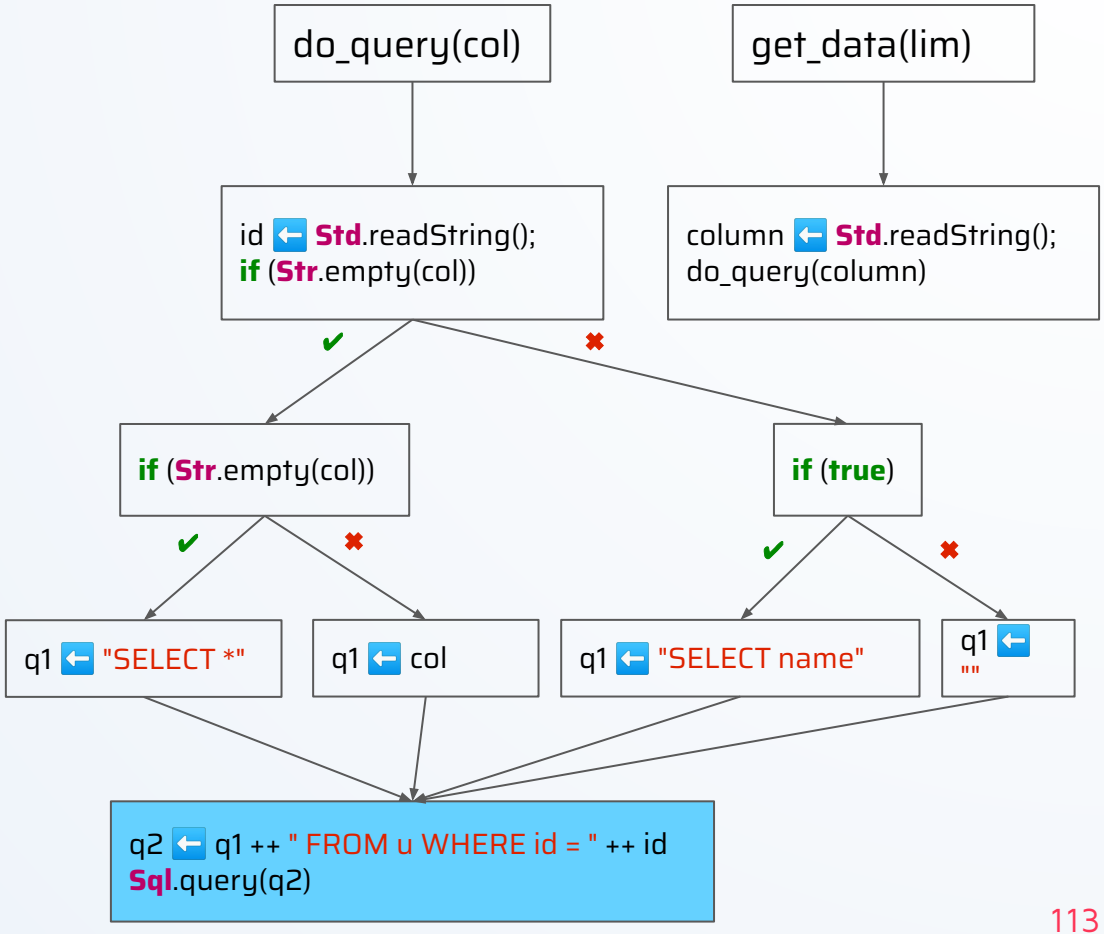
CFG

```
object Example

fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}

end Example
```



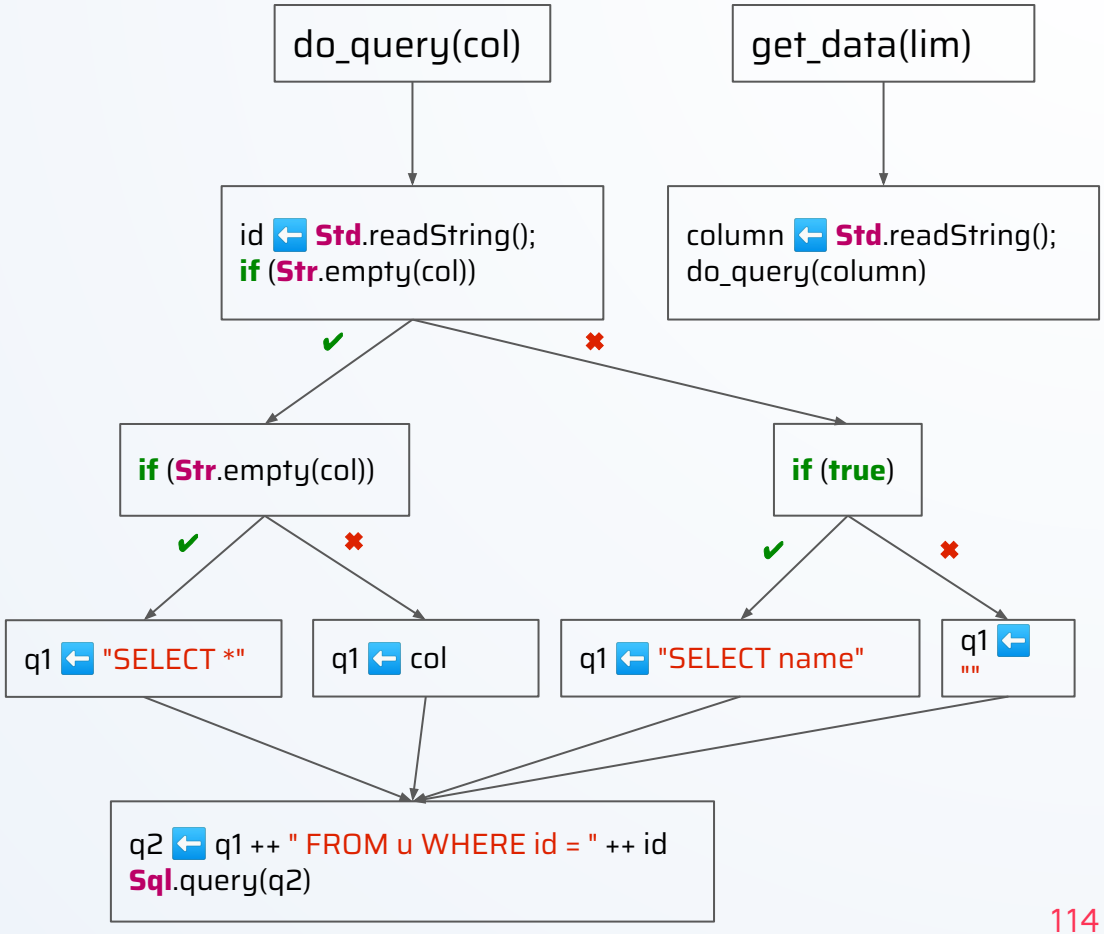
CFG

```
object Example

fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}

end Example
```



CFG

```
object Example

fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}

end Example
```



Outline

First hour

Intro to static analysis

Place for static analysis

AST-based analysis

- Visitors & Matchers

Second hour

→ Taint Analysis

Symbolic Execution

Static Analysis Trade-off

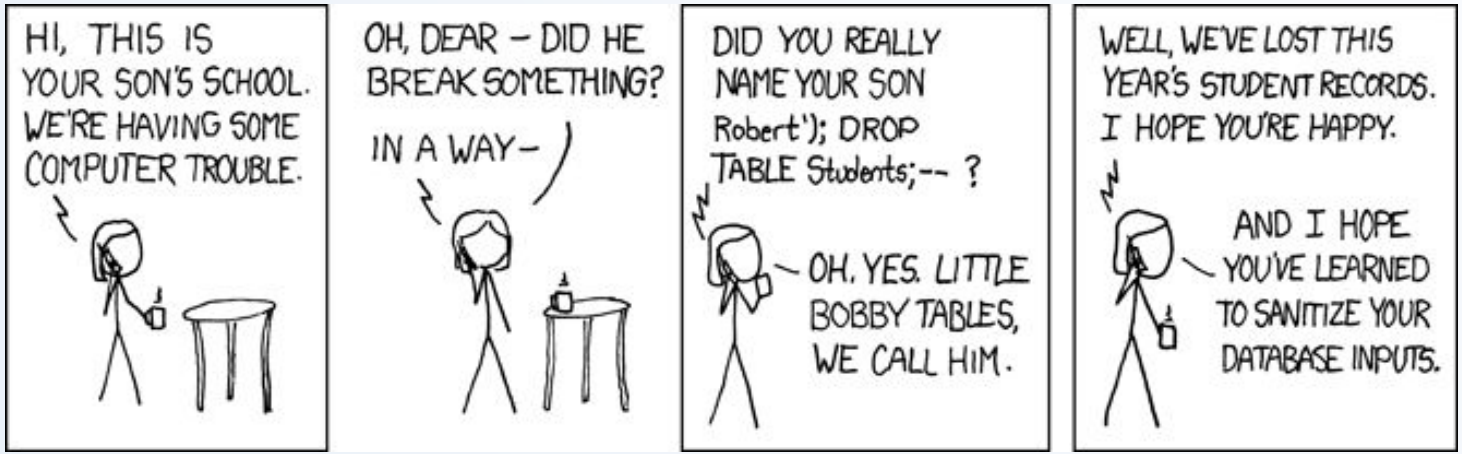
Demo

Taint Analysis

Find injection vulnerabilities

input →computation → sensitive query

Example: SQL injection



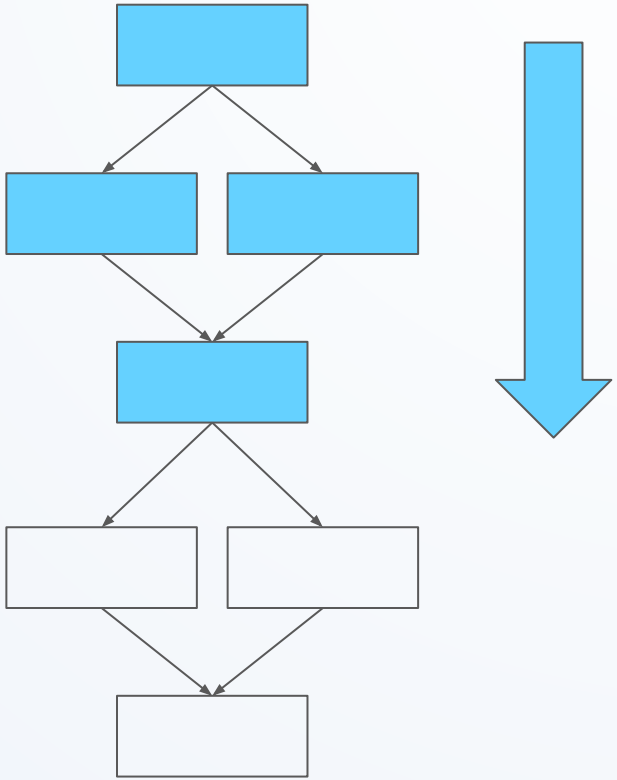
Taint Analysis

Simulate execution

Ignore branch conditions

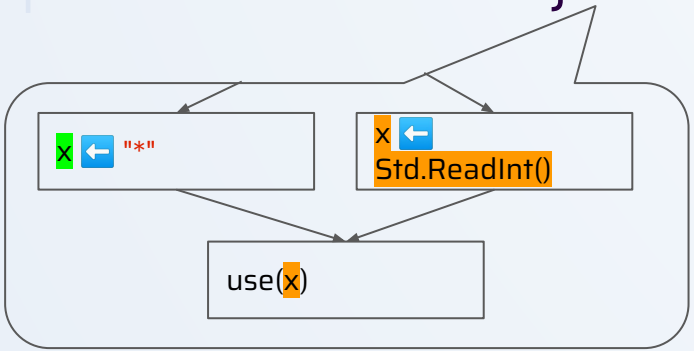
Use only “tainted” / “safe” for values

Explore all branches joining the taint



Taint Analysis

Explore all branches joining



Rules

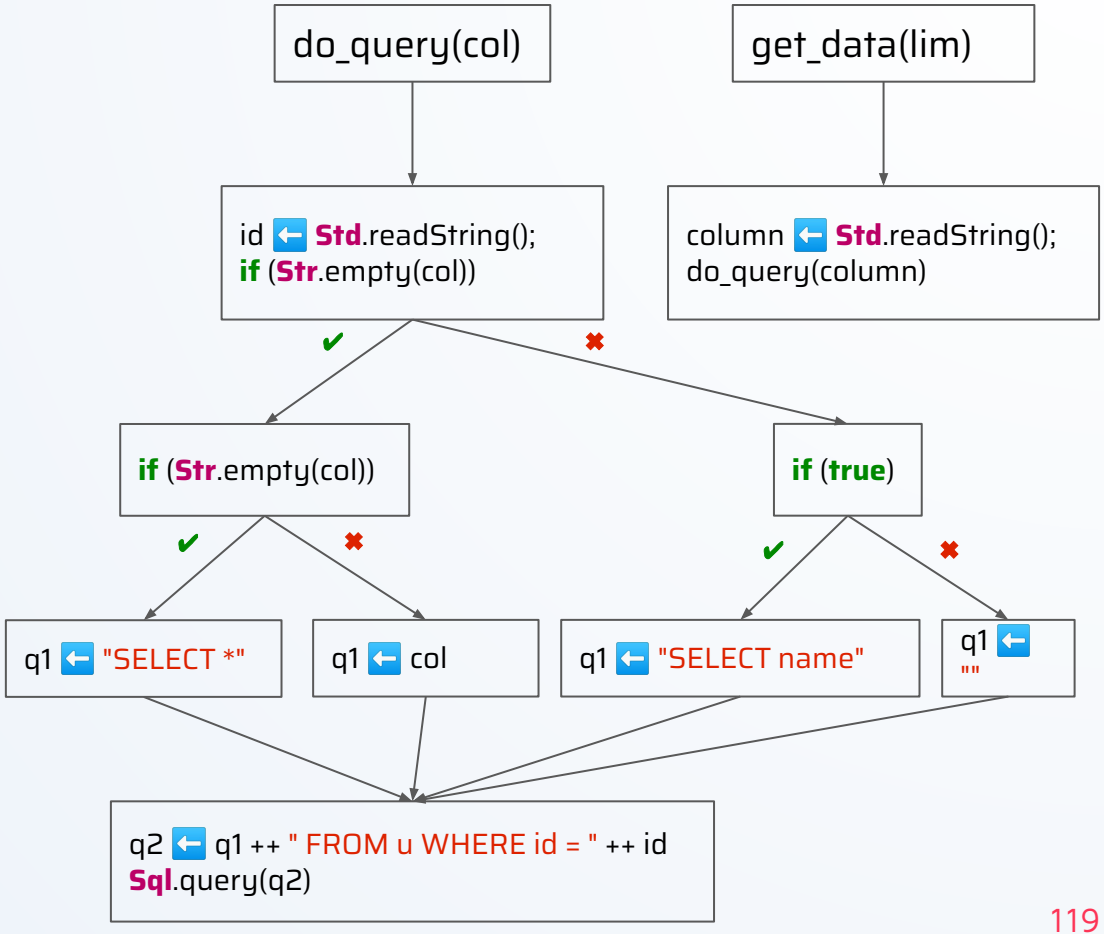
“safe” + “safe” ➡ “safe”
“tainted” + “safe” ➡ “tainted”
“safe” + “tainted” ➡ “tainted”
“tainted” + “tainted” ➡ “tainted”

For
- branch joins
- value combinations

```
val x: String <- y + z
```

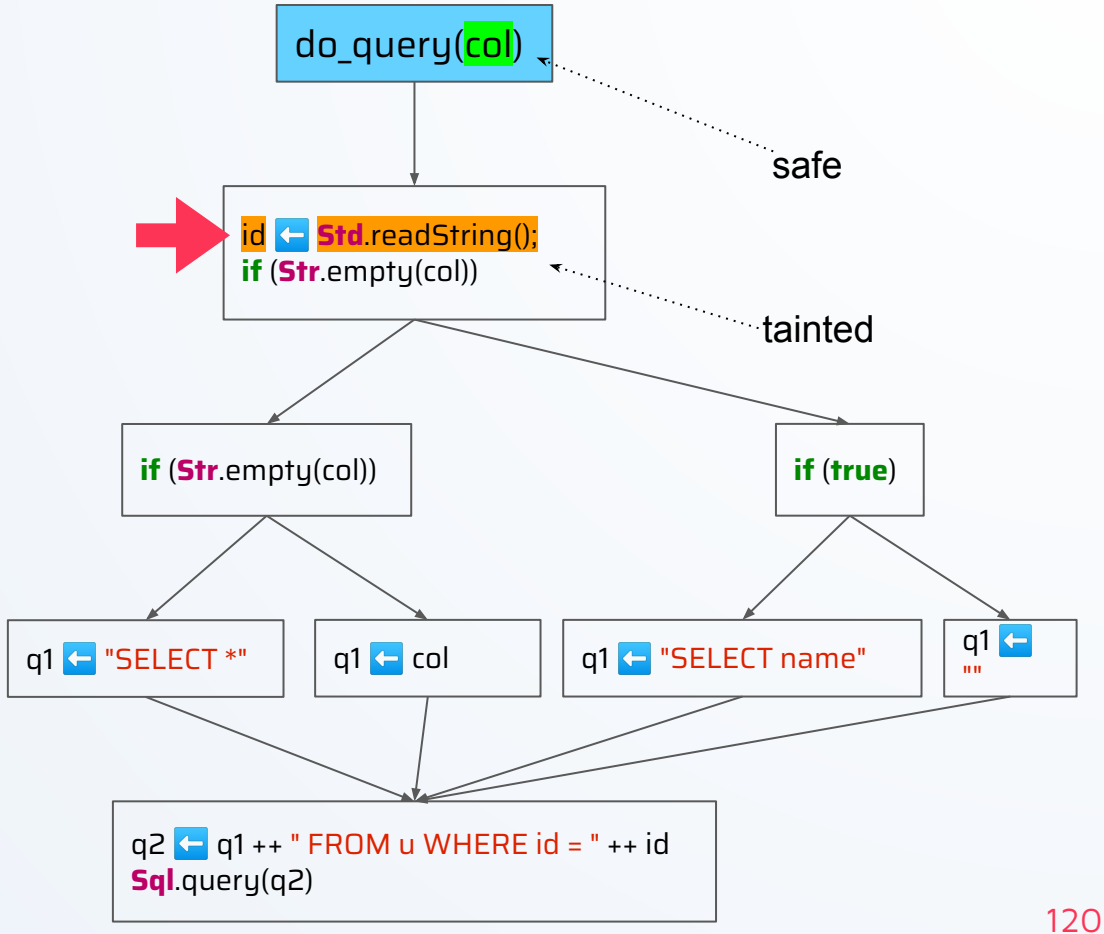
Taint Analysis

```
object Example

fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}

end Example
```



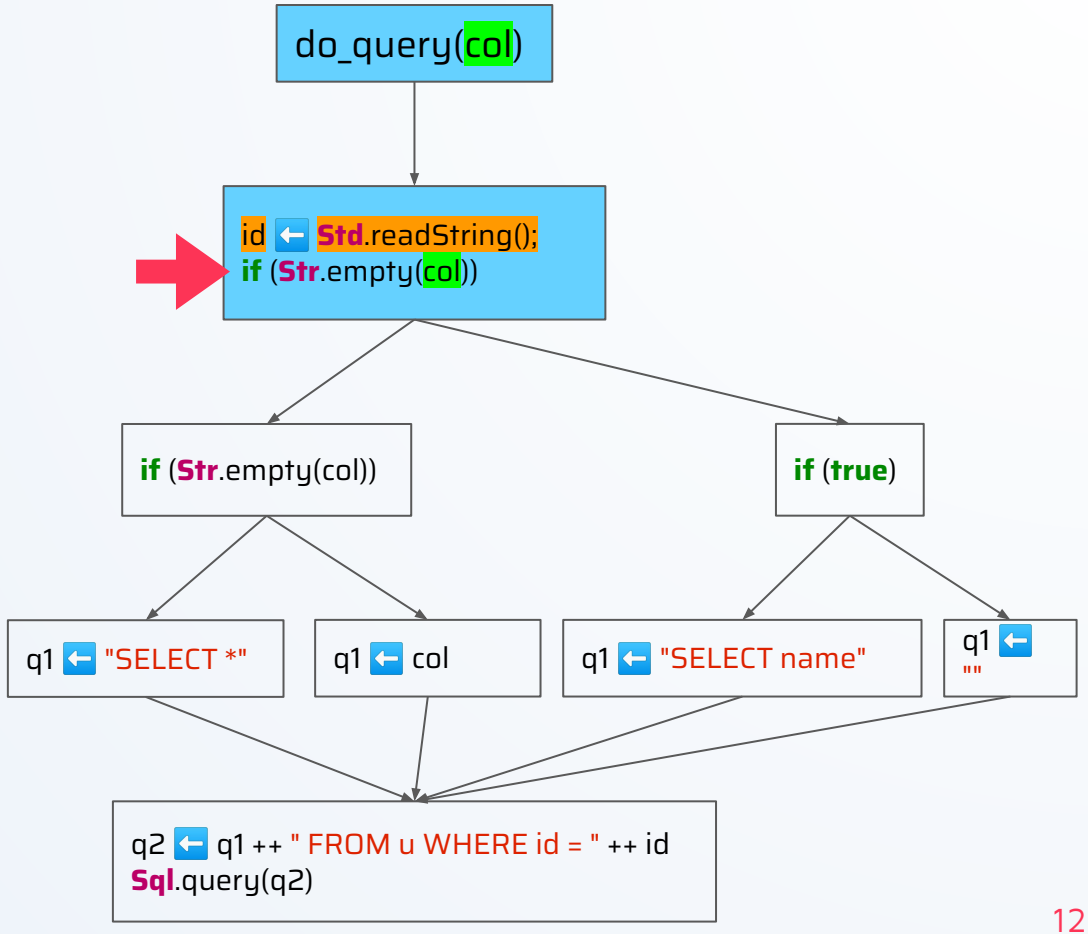
Taint Analysis

```
object Example

fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}

end Example
```



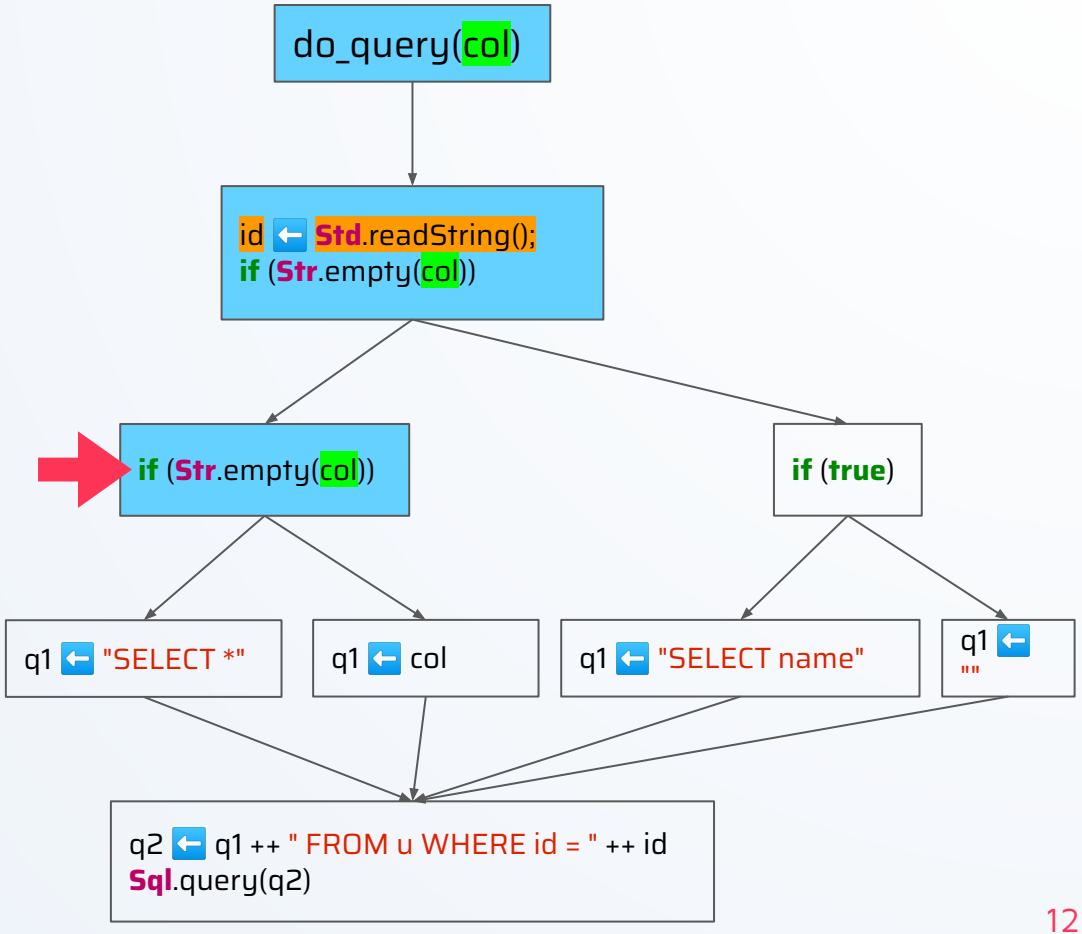
Taint Analysis

```
fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}
```



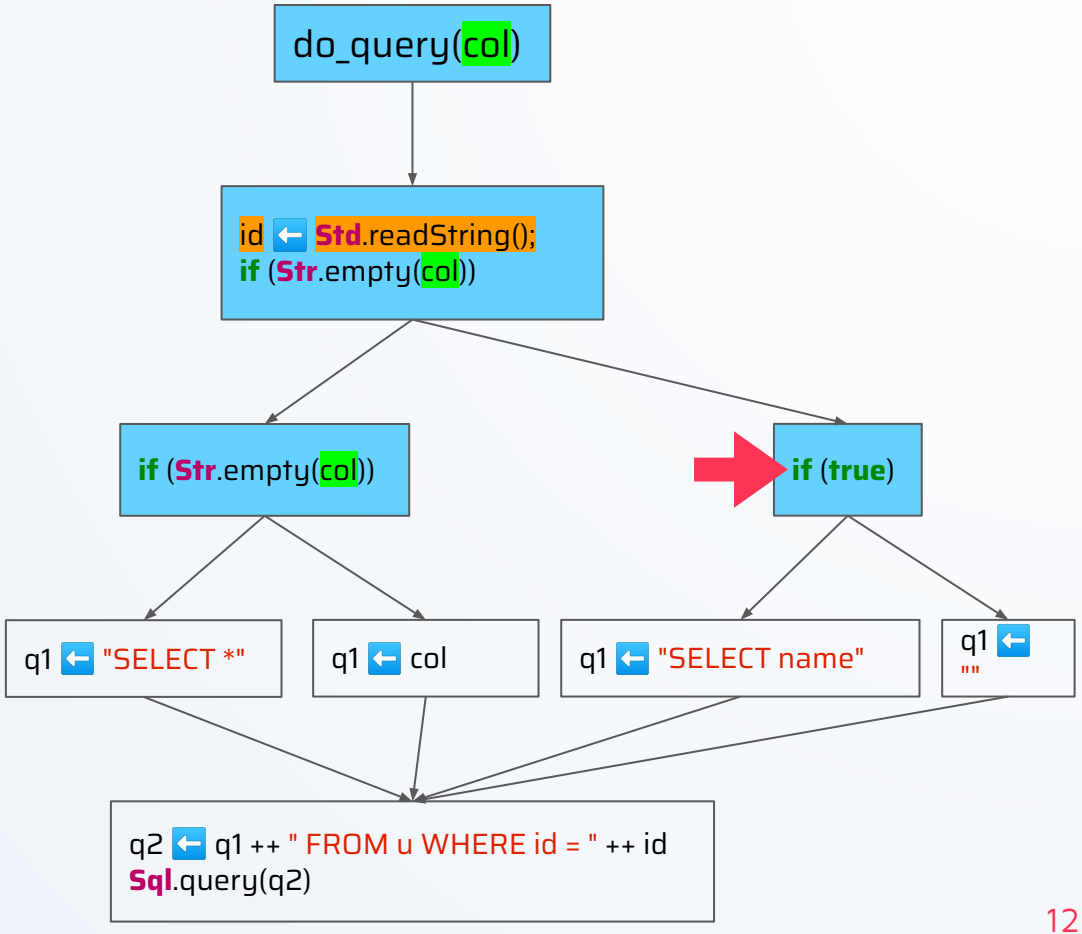
Taint Analysis

```
fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}
```



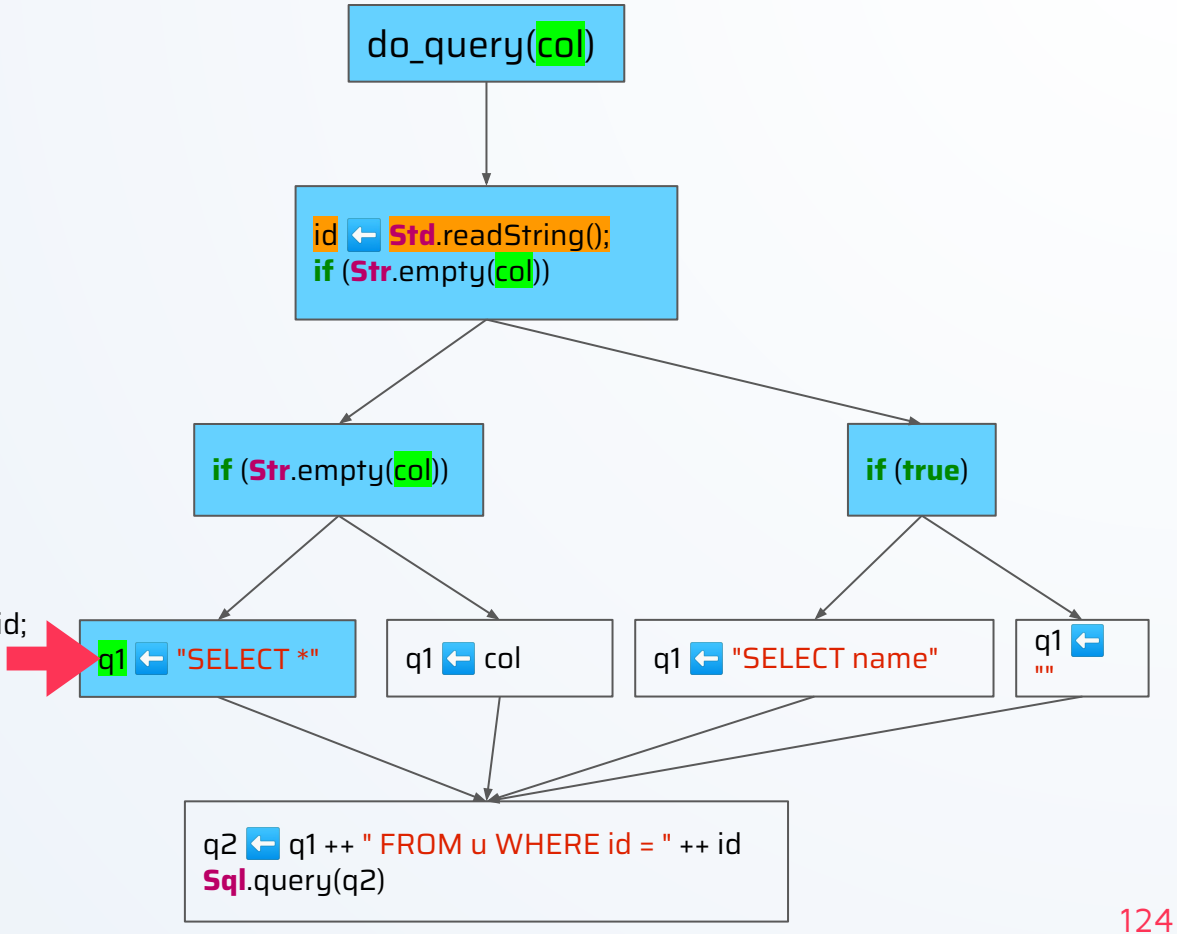
Taint Analysis

```
fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}
```



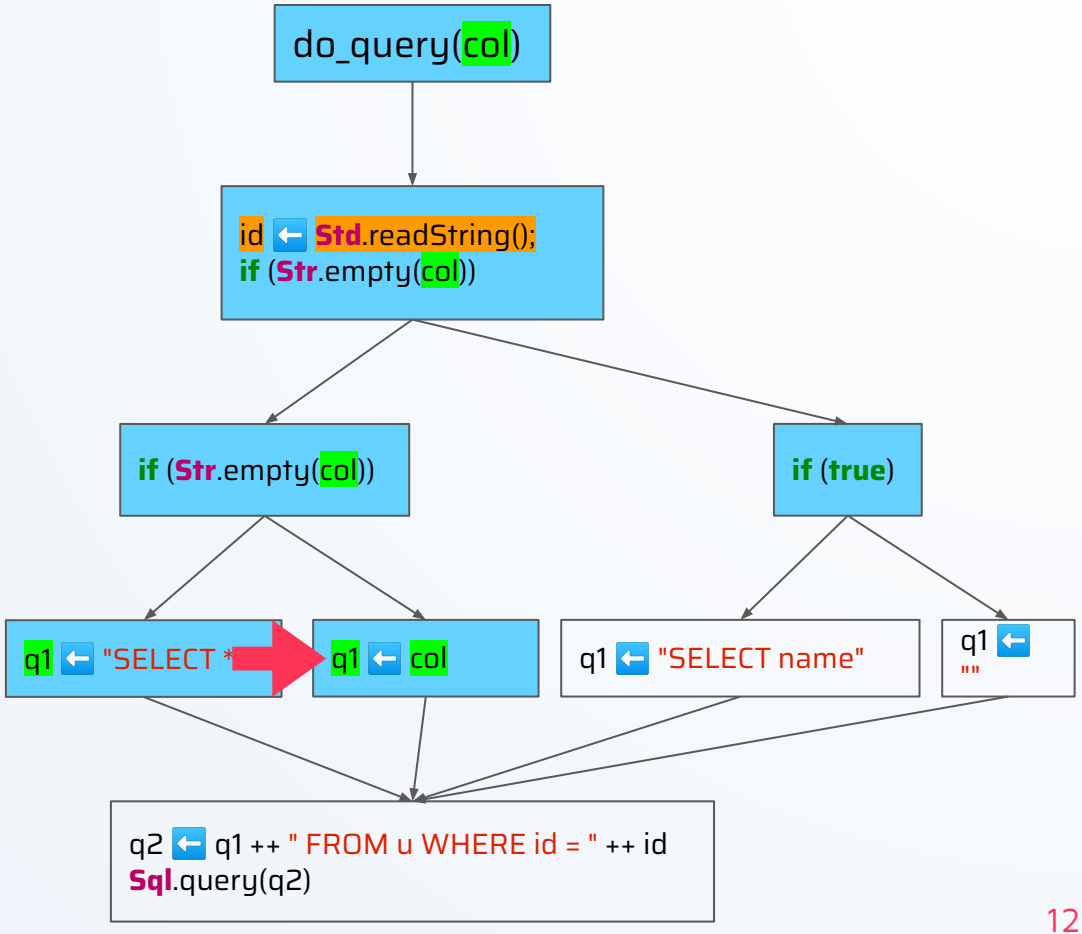
Taint Analysis

```
fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}
```



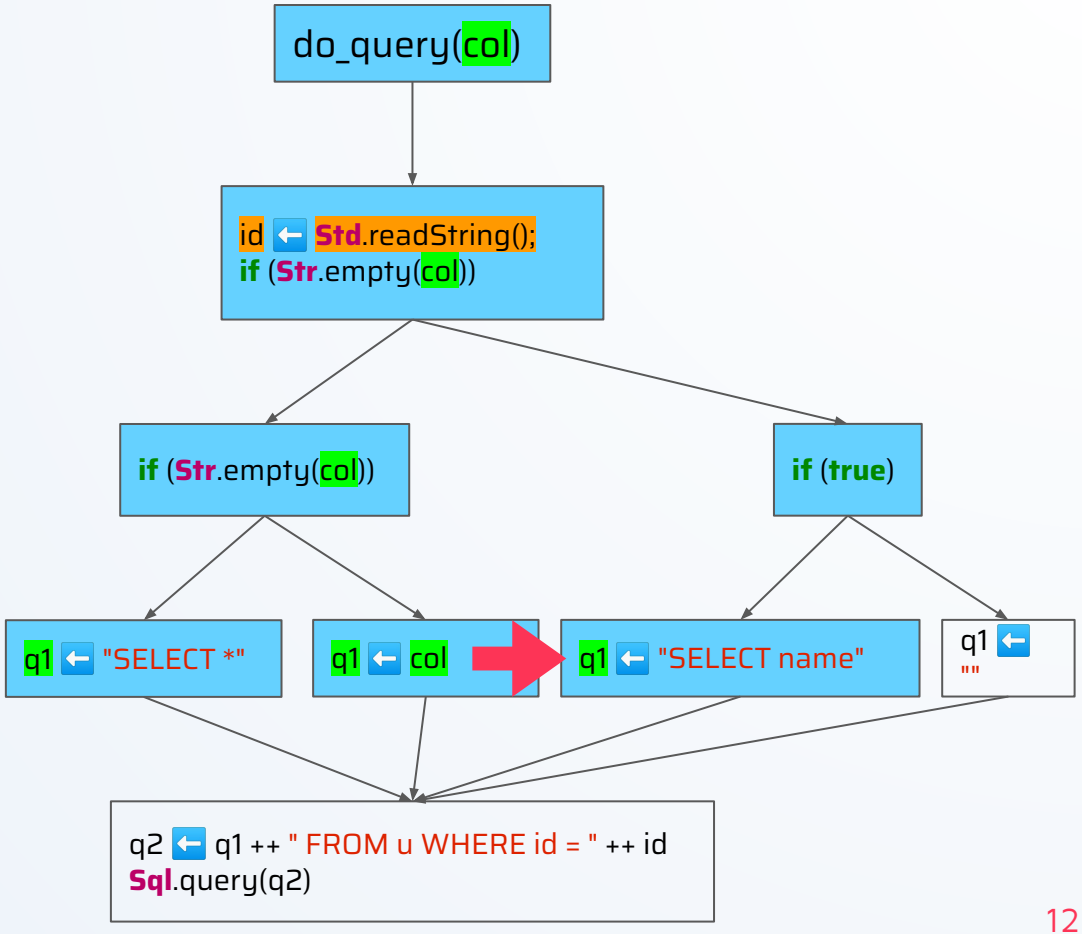
Taint Analysis

```
fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}
```



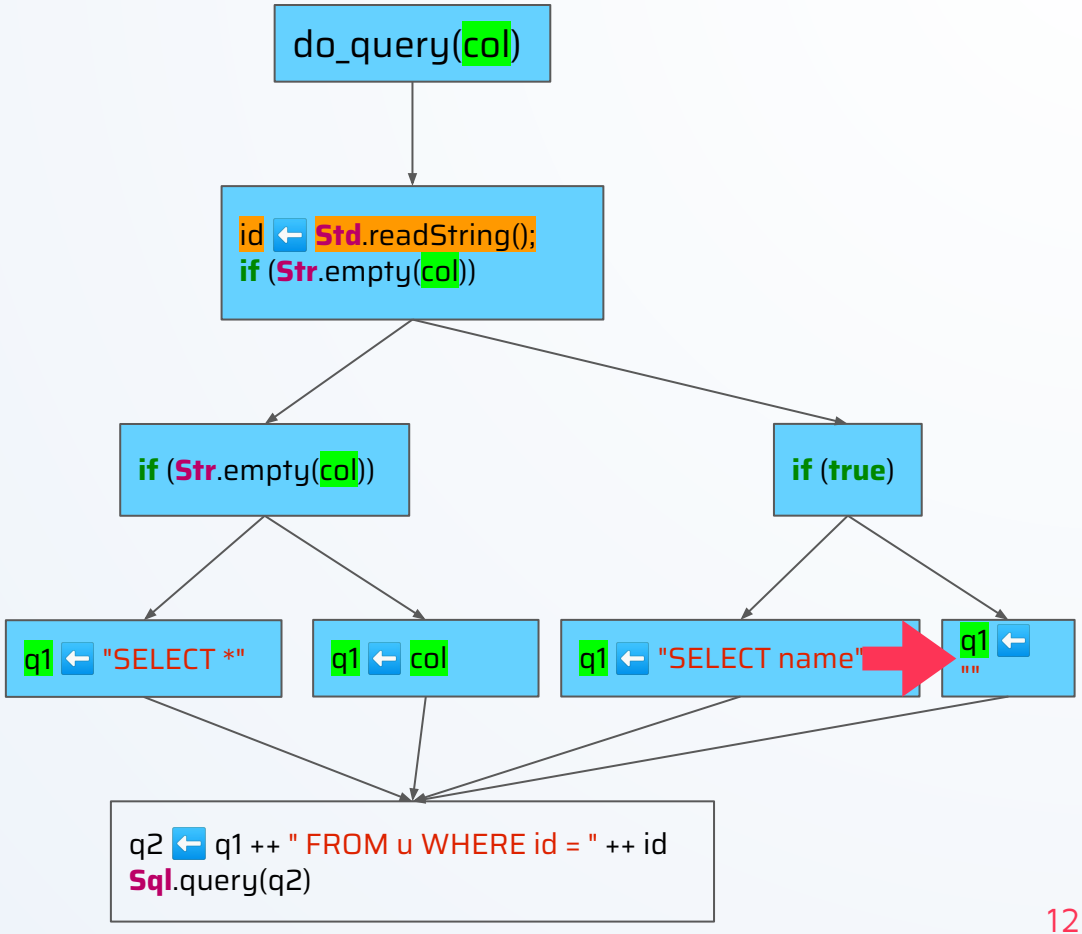
Taint Analysis

```
fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}
```



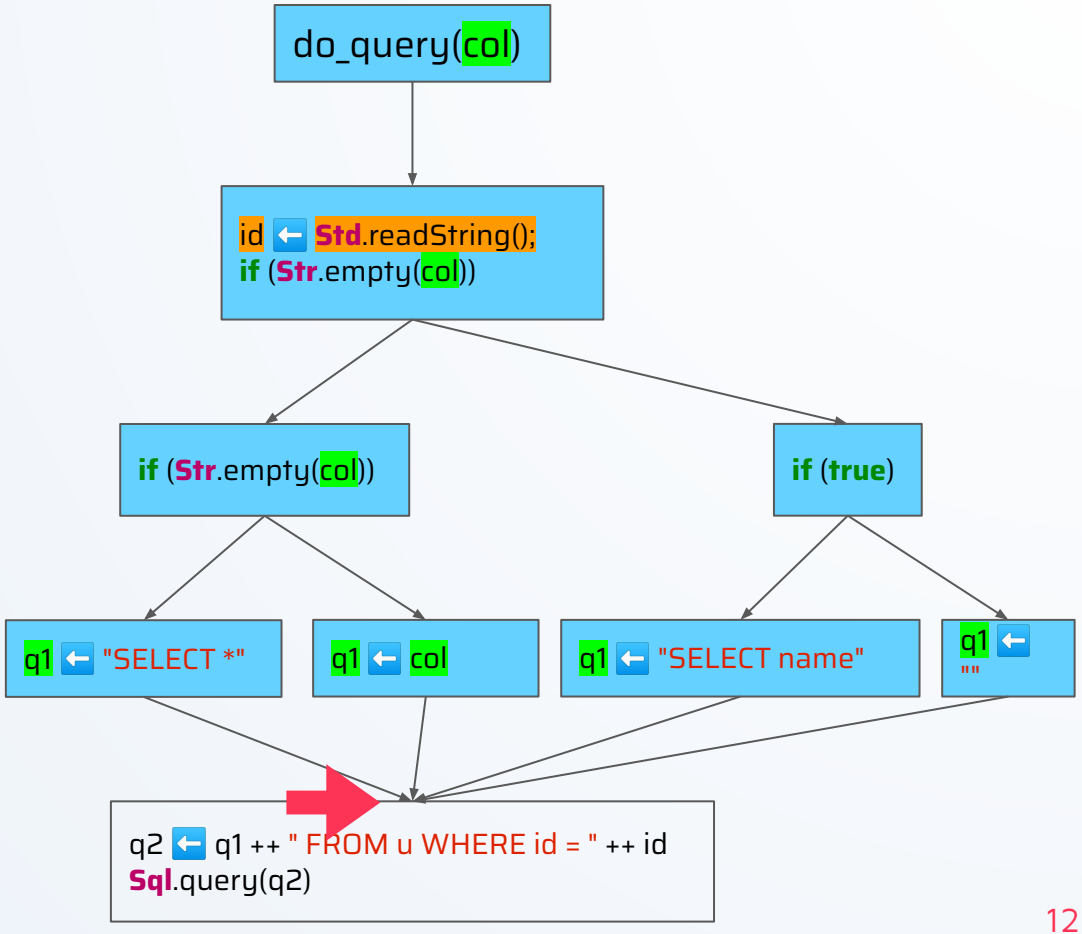
Taint Analysis

```
fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}
```



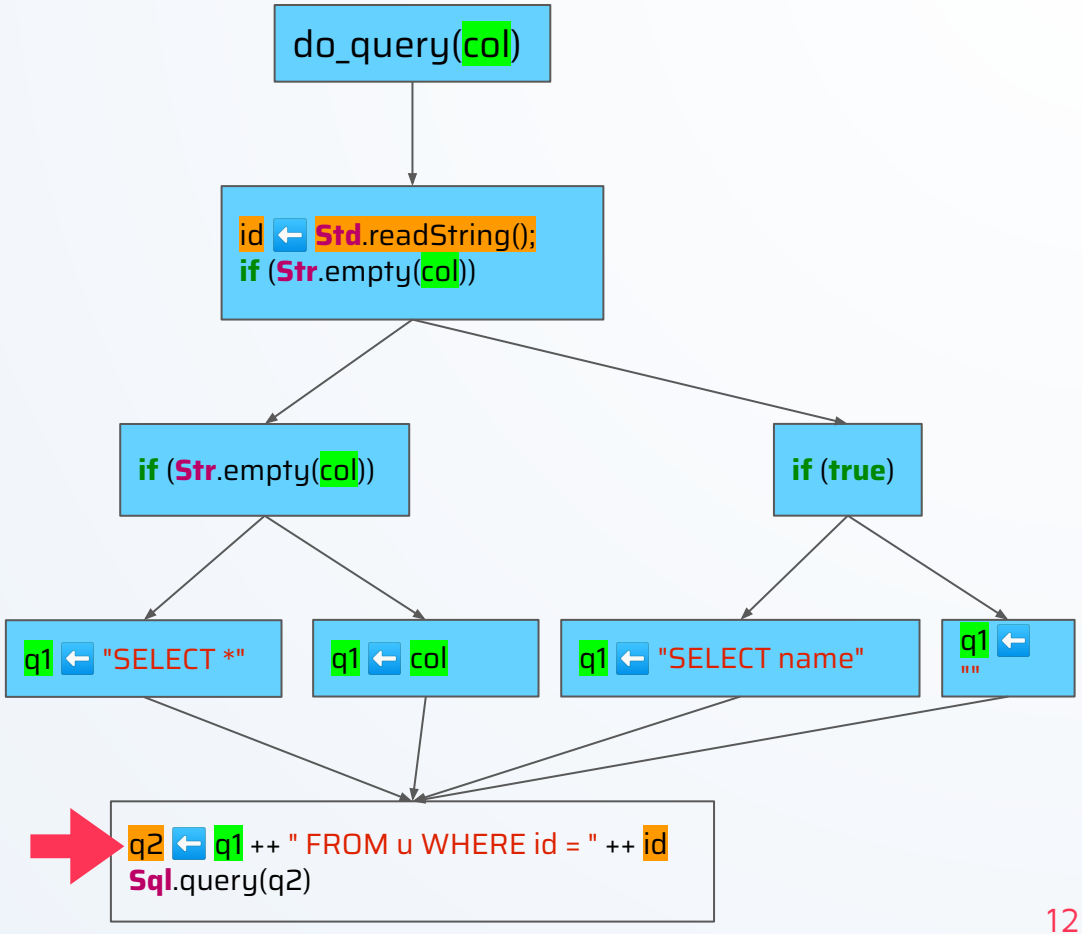
Taint Analysis

```
fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}
```



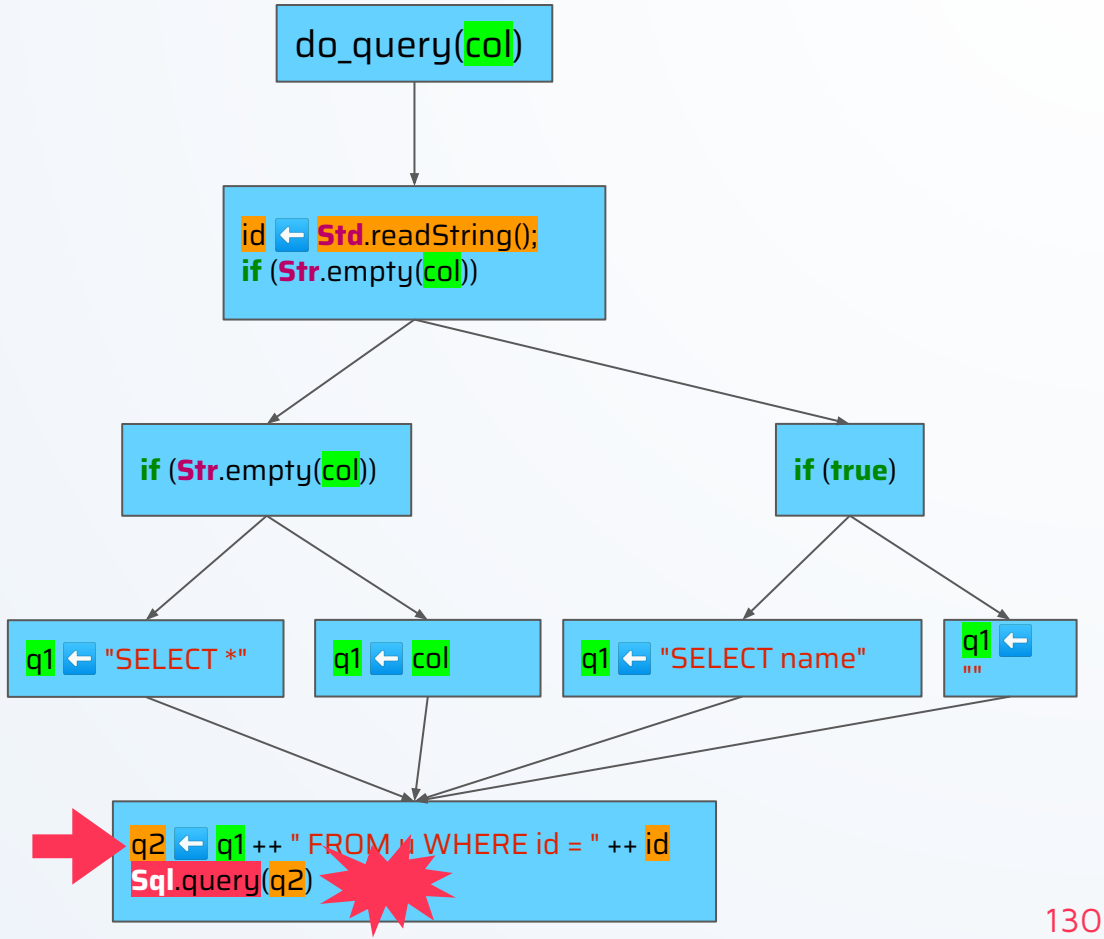
Taint Analysis

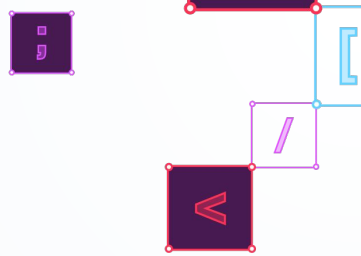
```
fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}
```



Taint Analysis

```
fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}
```



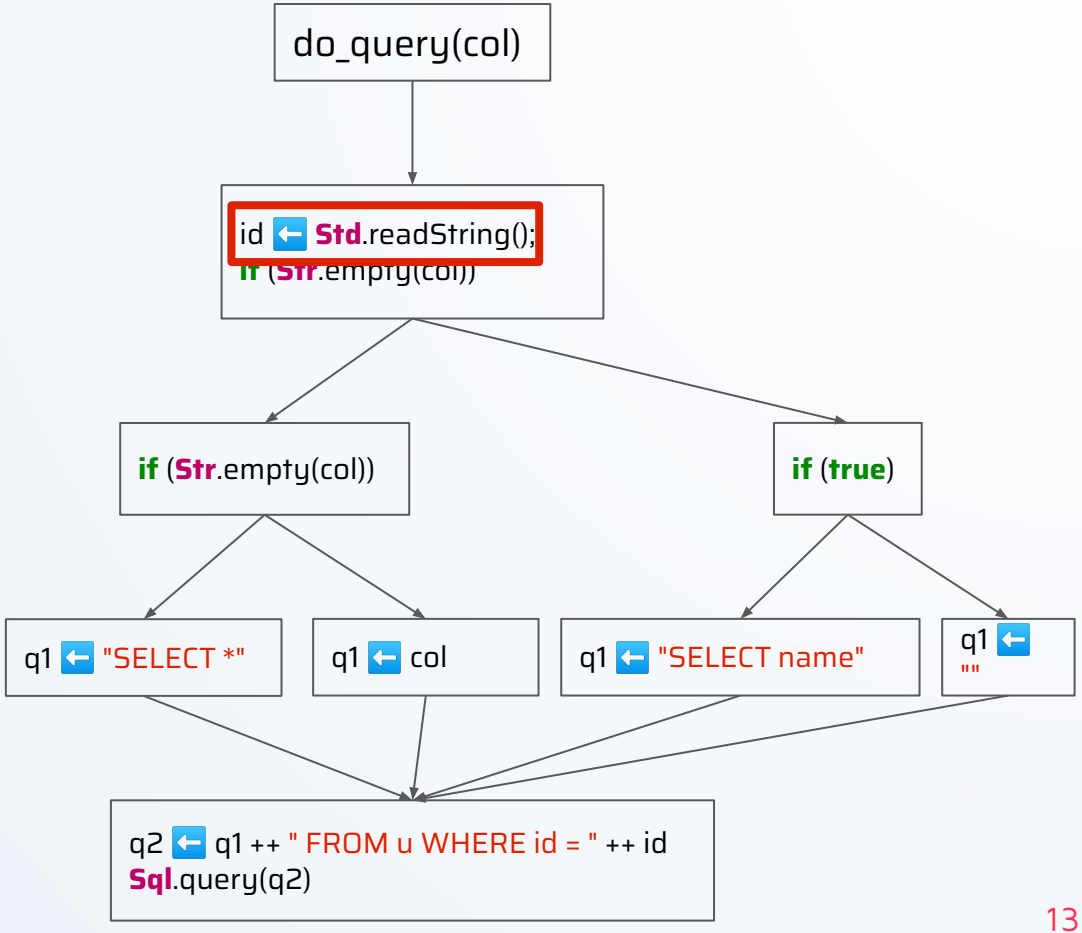


Cross procedural

sonar

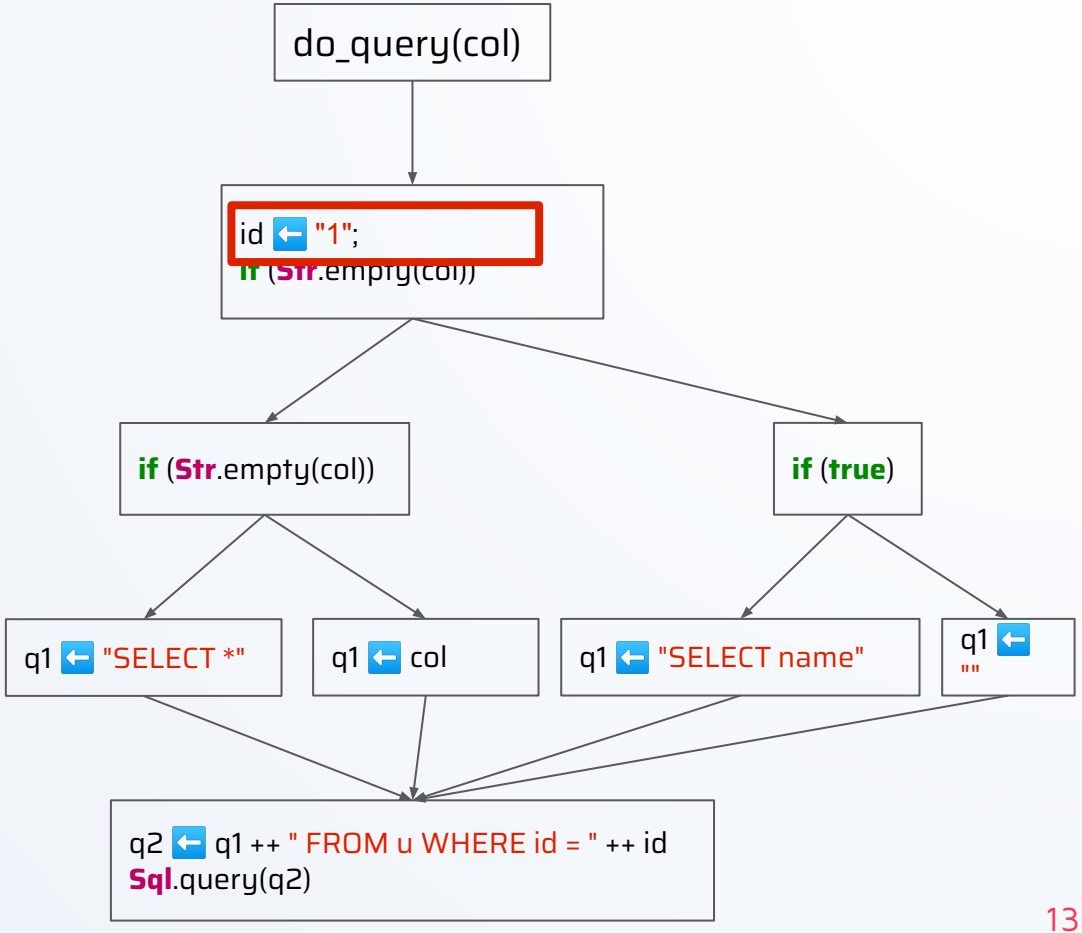
Cross Procedural

```
fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}
```



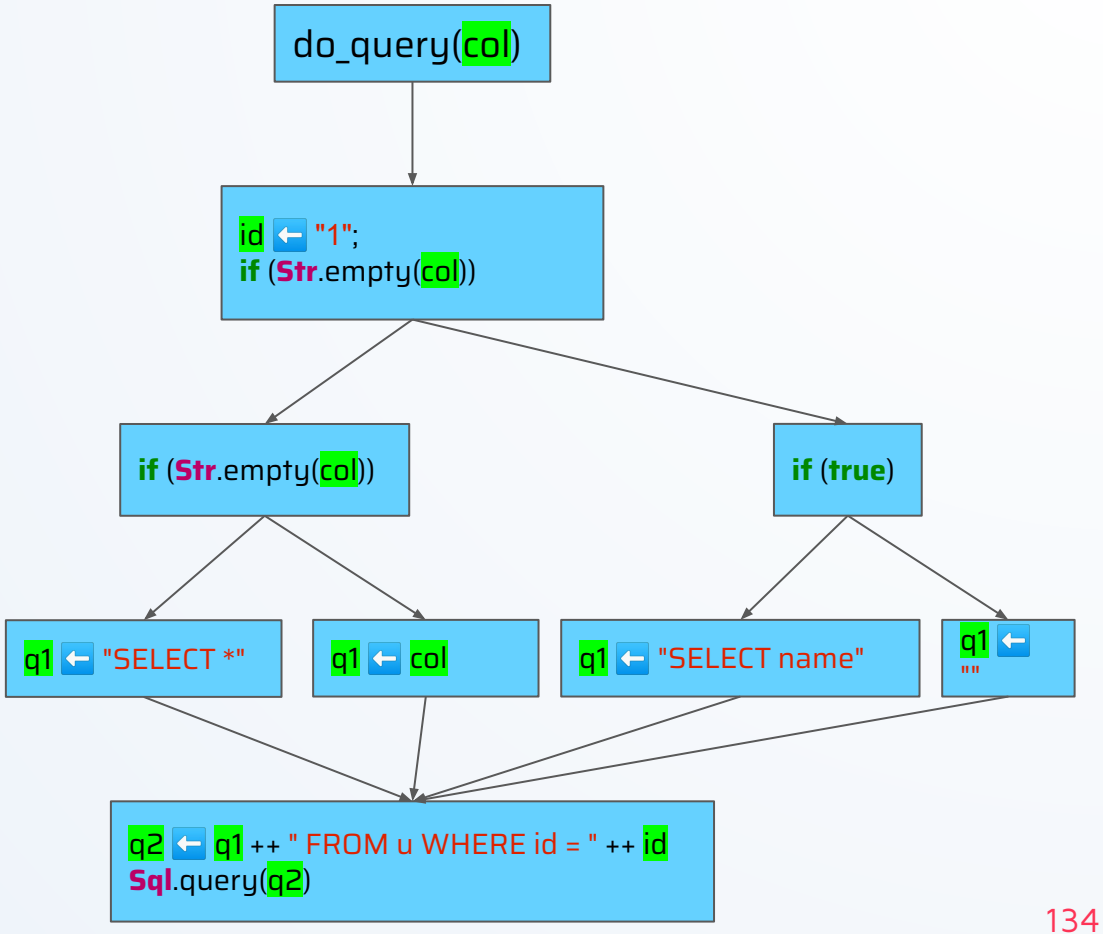
Cross Procedural

```
fn do_query(col: String): String = {
  val id: String = "1";
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}
```



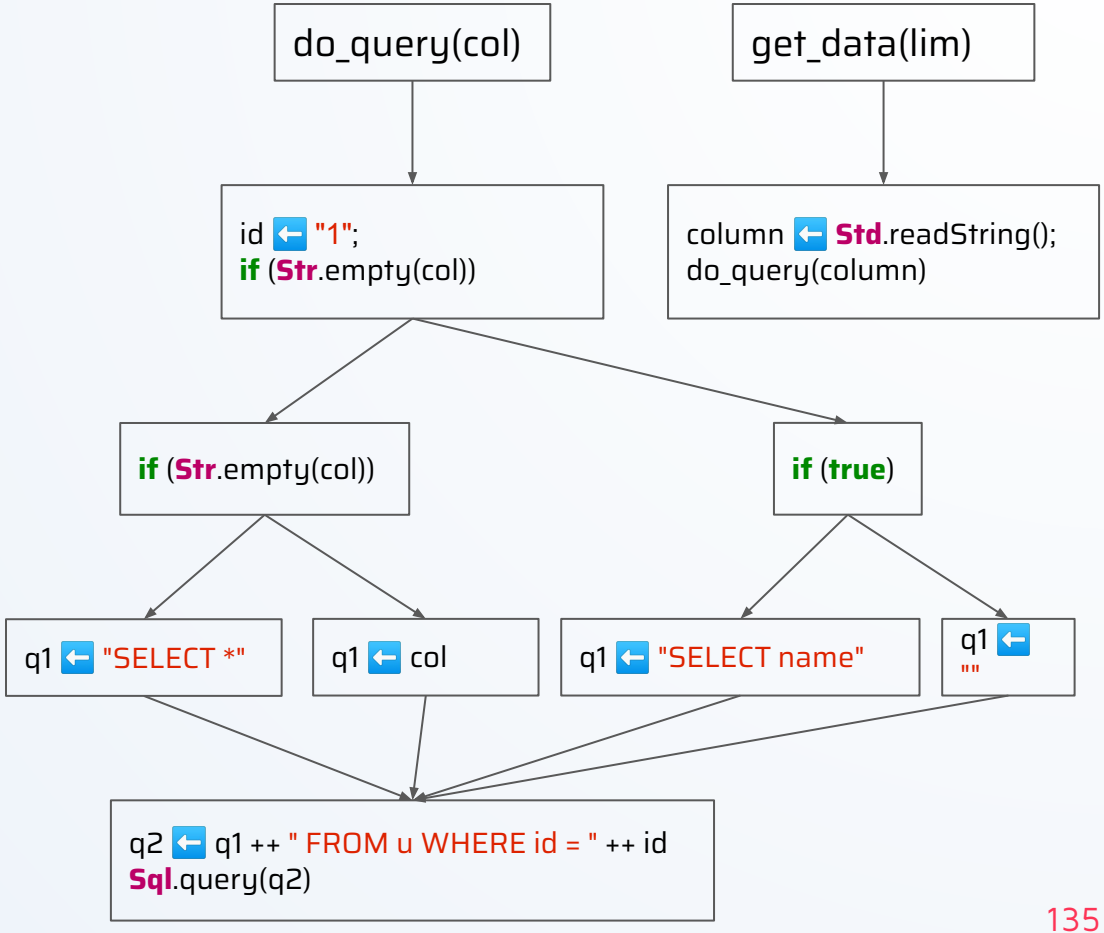
Cross Procedural

```
fn do_query(col: String): String = {
  val id: String = "1";
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}
```



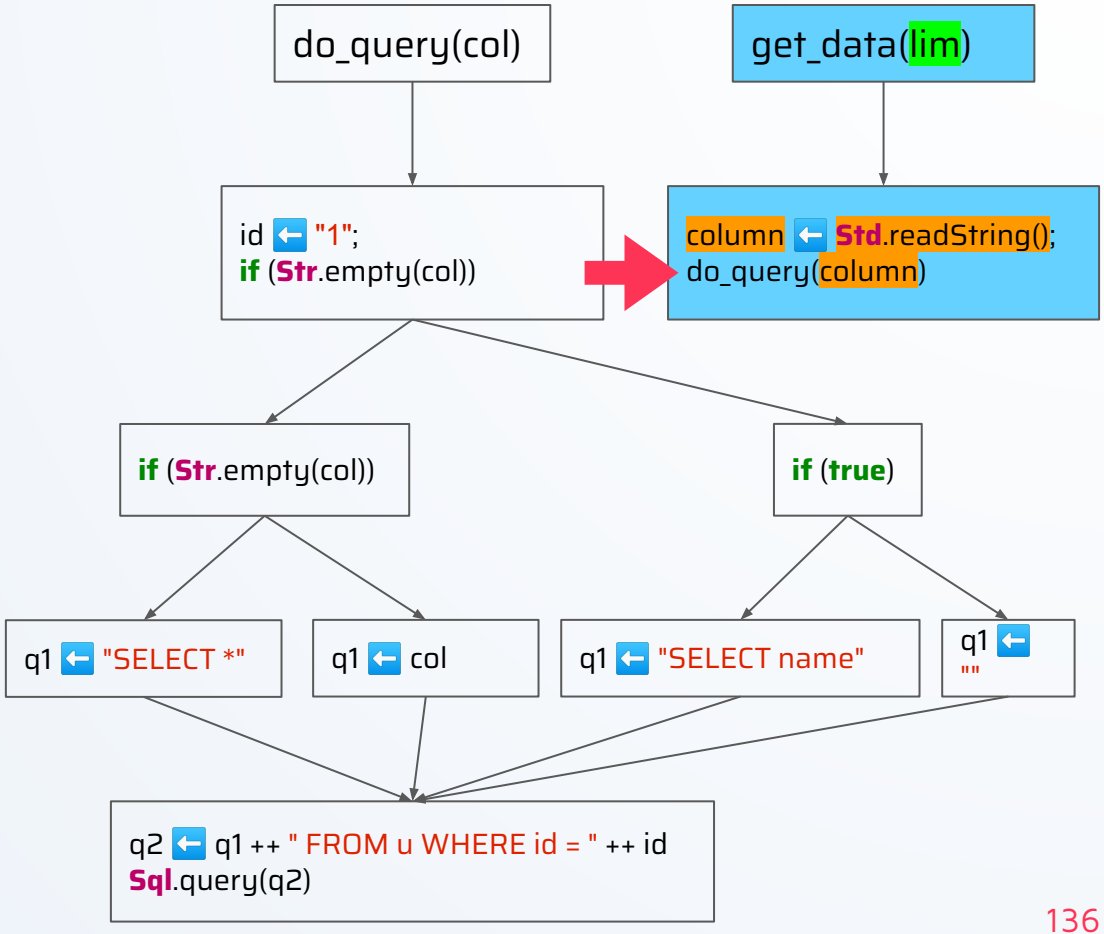
Cross Procedural

```
fn do_query(col: String): String = {
  val id: String = "1";
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}
```



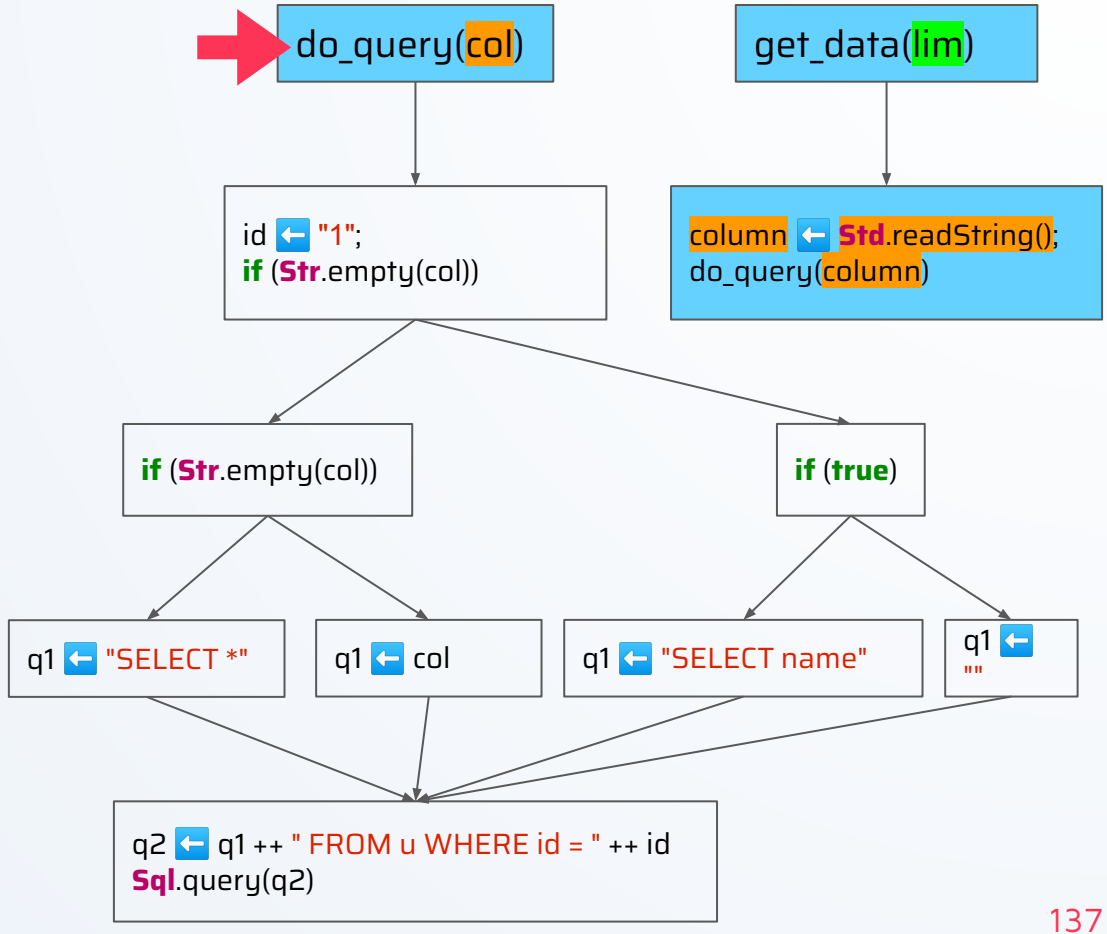
Cross Procedural

```
fn do_query(col: String): String = {
  val id: String = "1";
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}
```



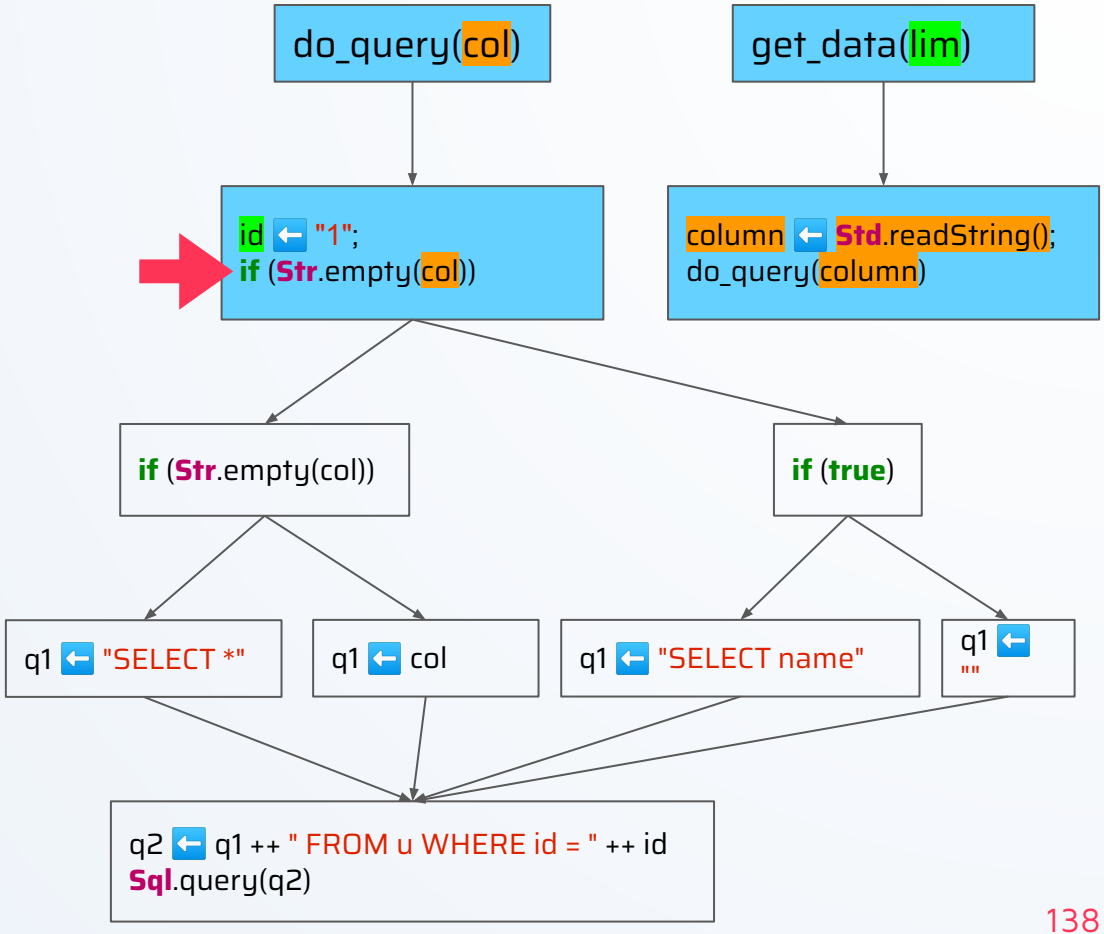
Cross Procedural

```
fn do_query(col: String): String = {
  val id: String = "1";
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}
```



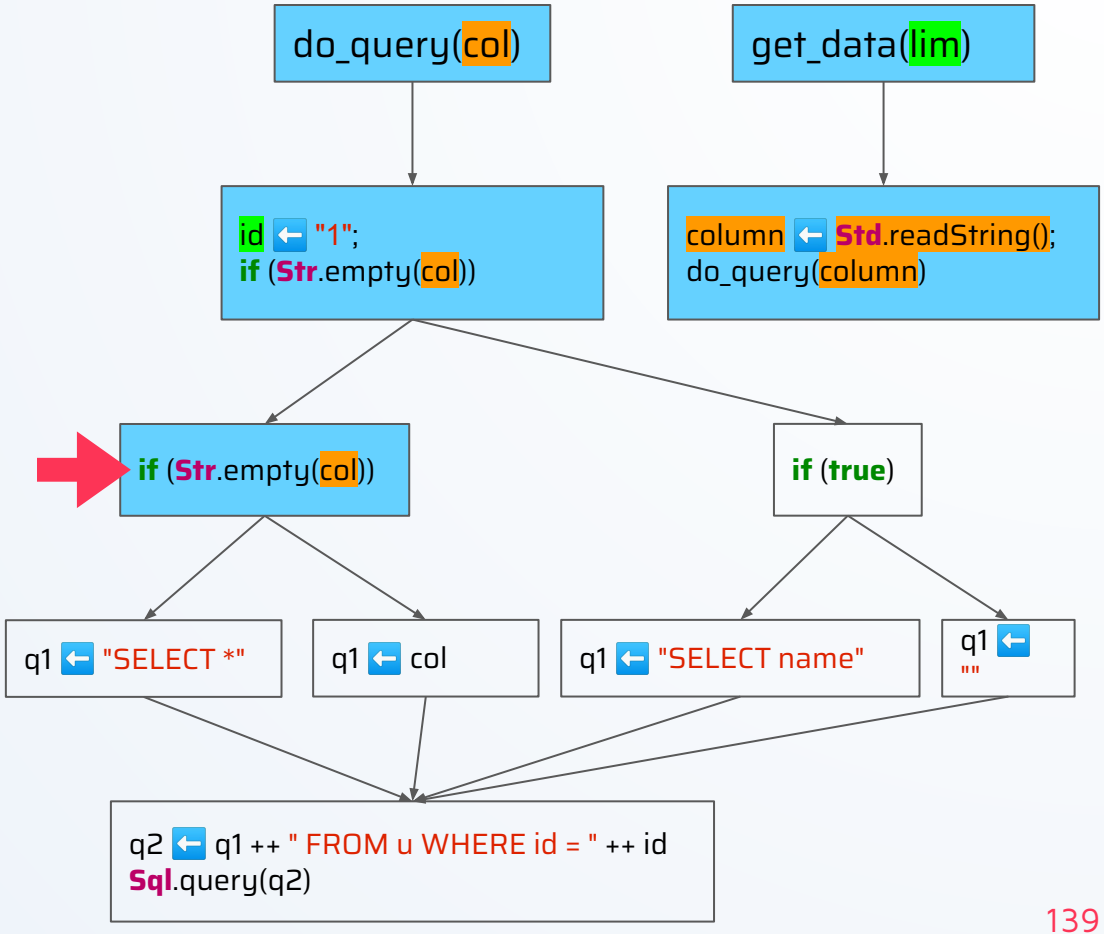
Cross Procedural

```
fn do_query(col: String): String = {
  val id: String = "1";
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}
```



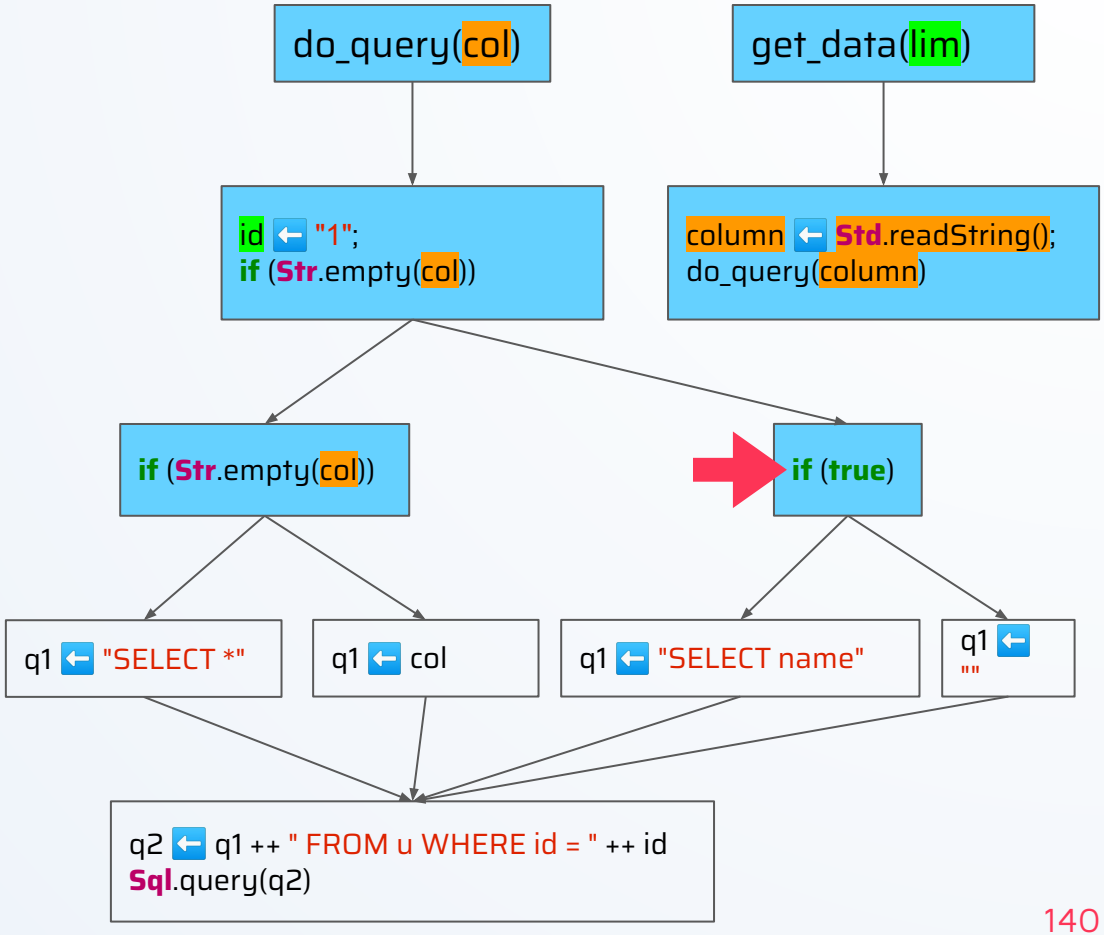
Cross Procedural

```
fn do_query(col: String): String = {
  val id: String = "1";
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}
```



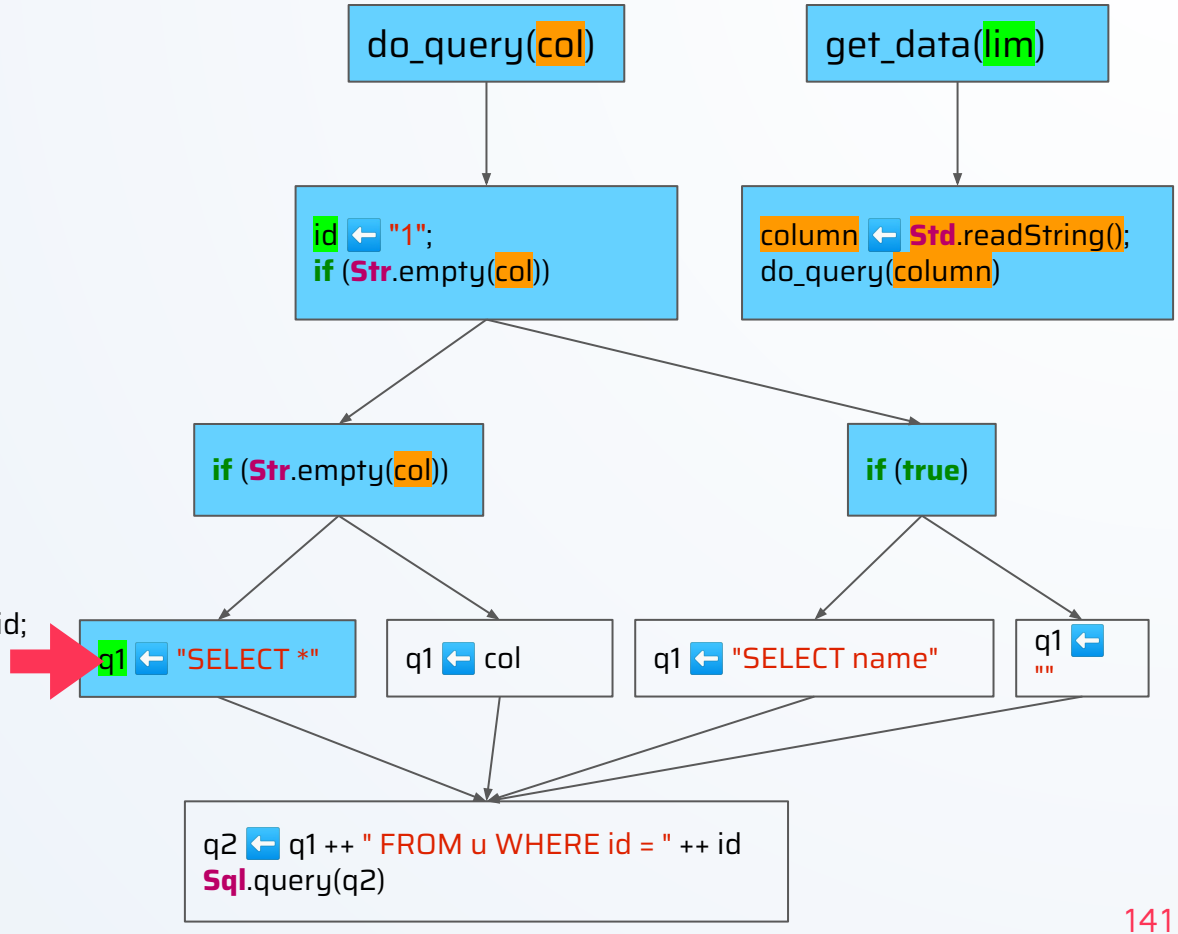
Cross Procedural

```
fn do_query(col: String): String = {
  val id: String = "1";
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}
```



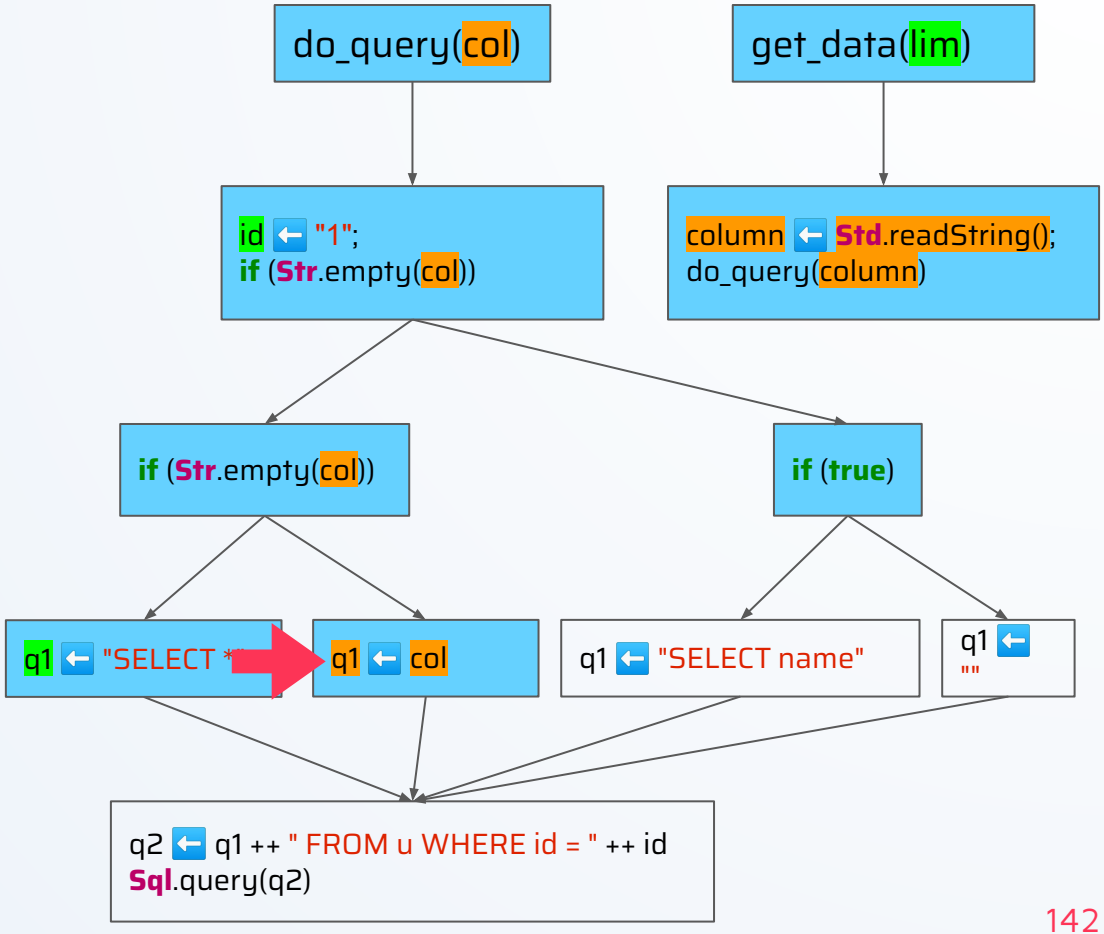
Cross Procedural

```
fn do_query(col: String): String = {
  val id: String = "1";
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}
```



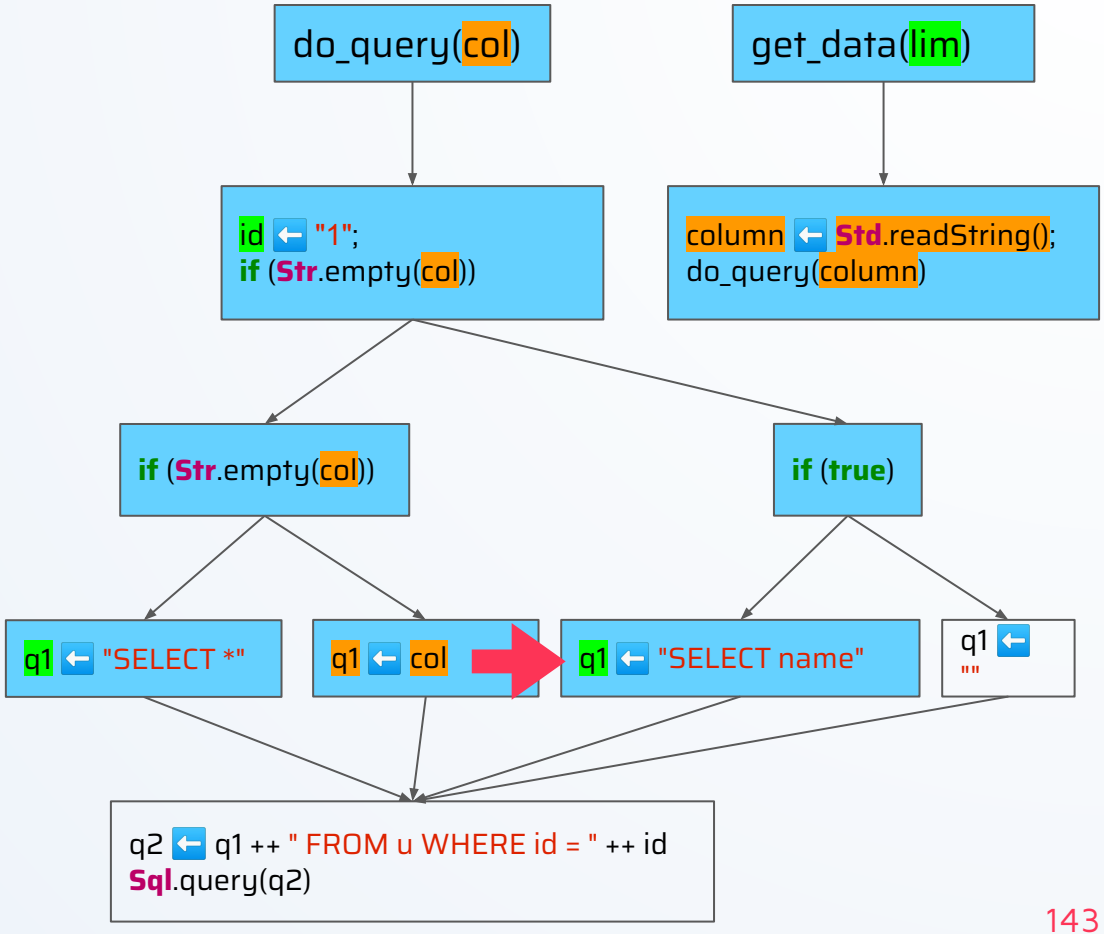
Cross Procedural

```
fn do_query(col: String): String = {
  val id: String = "1";
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}
```



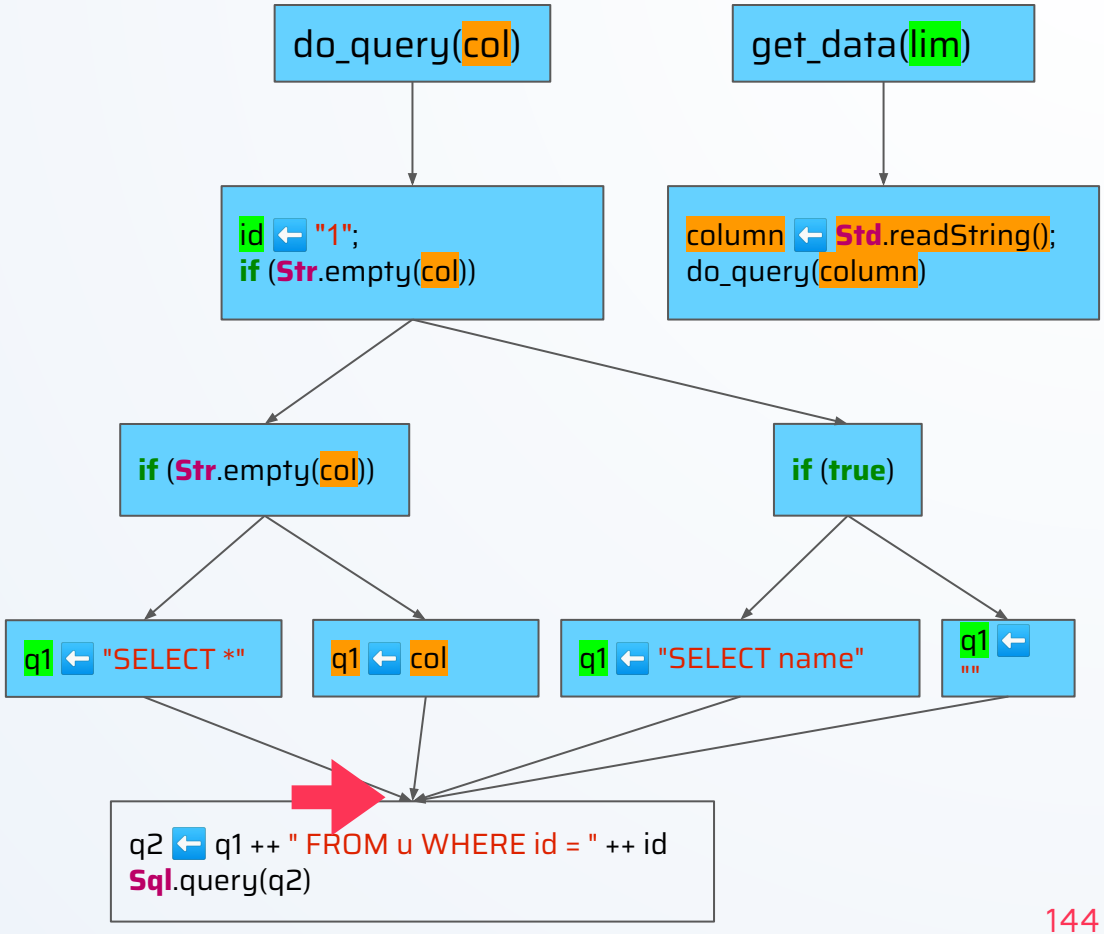
Cross Procedural

```
fn do_query(col: String): String = {
  val id: String = "1";
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}
```



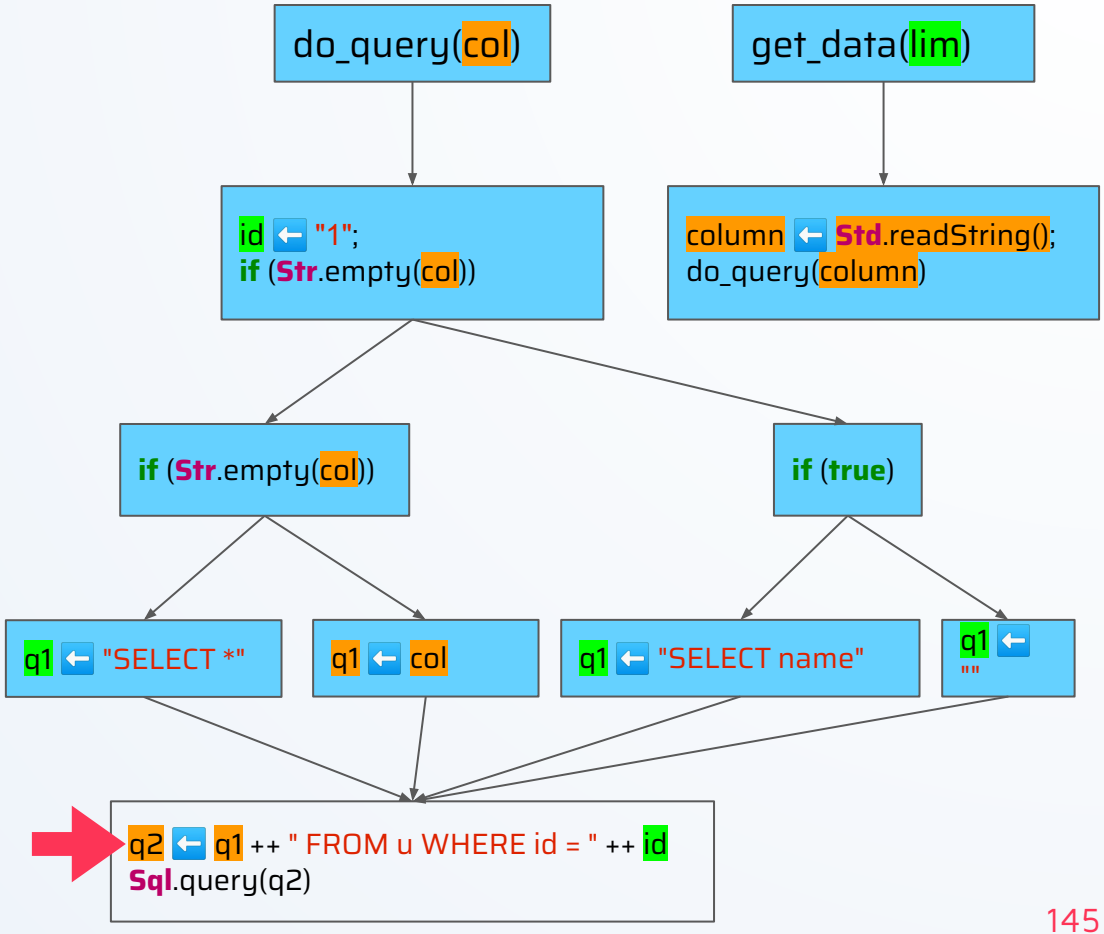
Cross Procedural

```
fn do_query(col: String): String = {
  val id: String = "1";
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}
```



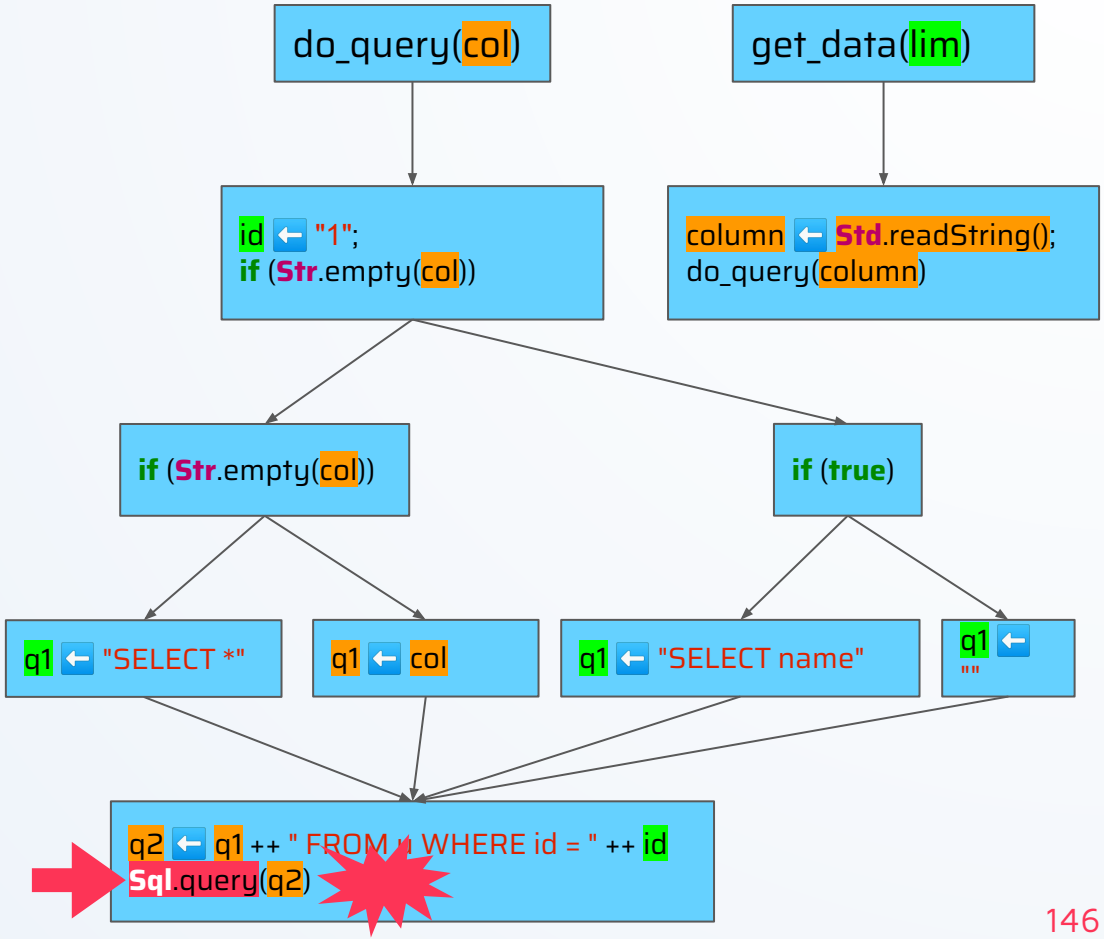
Cross Procedural

```
fn do_query(col: String): String = {
  val id: String = "1";
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}
```



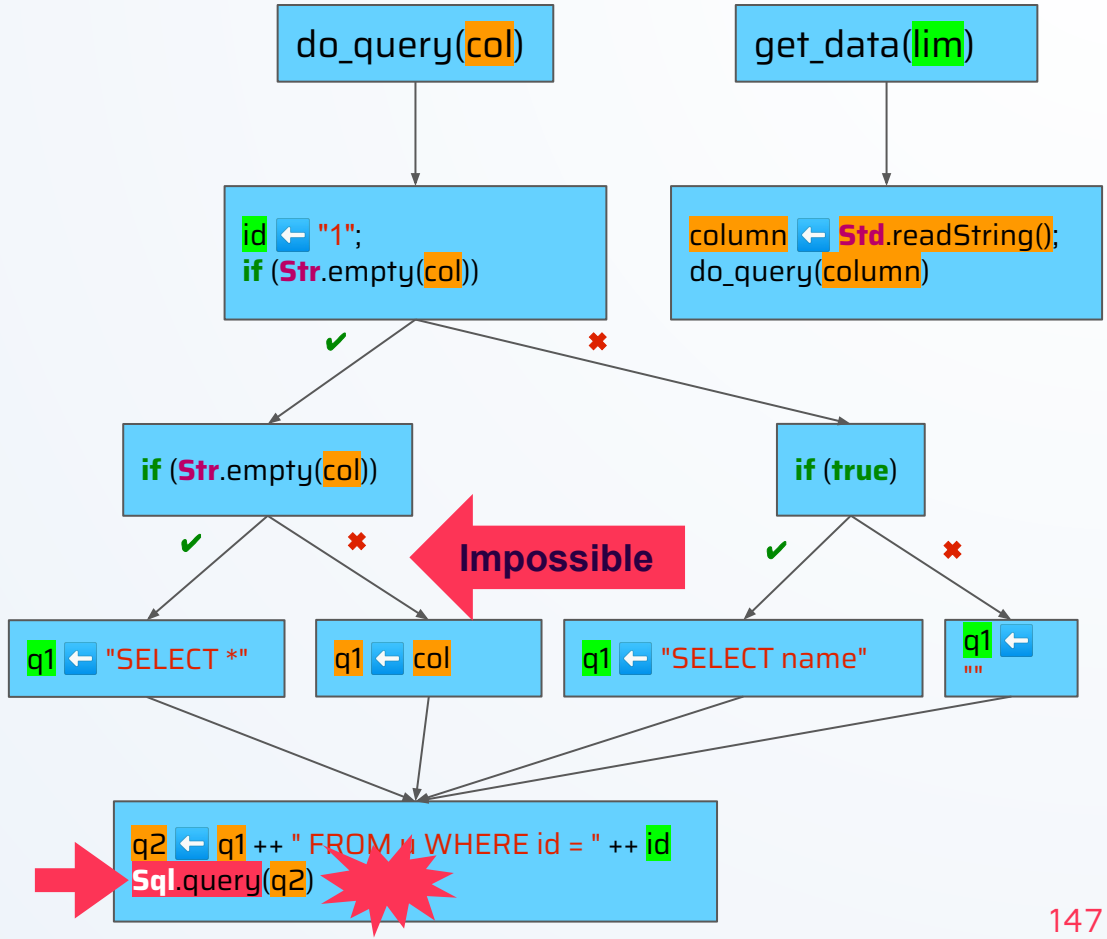
Cross Procedural

```
fn do_query(col: String): String = {
  val id: String = "1";
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}
```



Limitation: Precision

```
fn do_query(col: String): String = {
  val id: String = "1";
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}
```



Outline

First hour

Intro to static analysis

Place for static analysis

AST-based analysis

Visitors & Matchers

Second hour

Taint Analysis


Symbolic Execution

Static Analysis Trade-off

Demo

Symbolic Execution

More precise

More expensive

More rules

sonar

Symbolic Execution

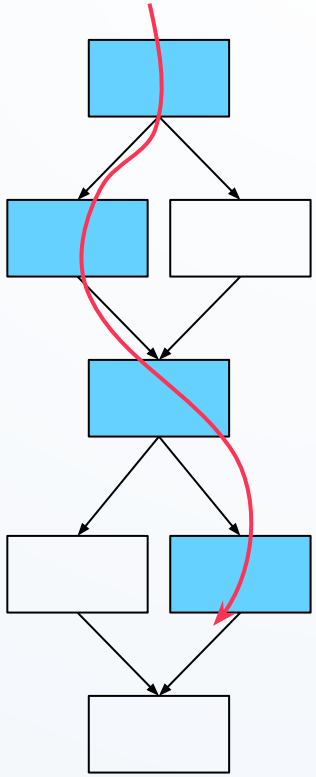
Simulate Execution

Keep unknown values as “symbols”

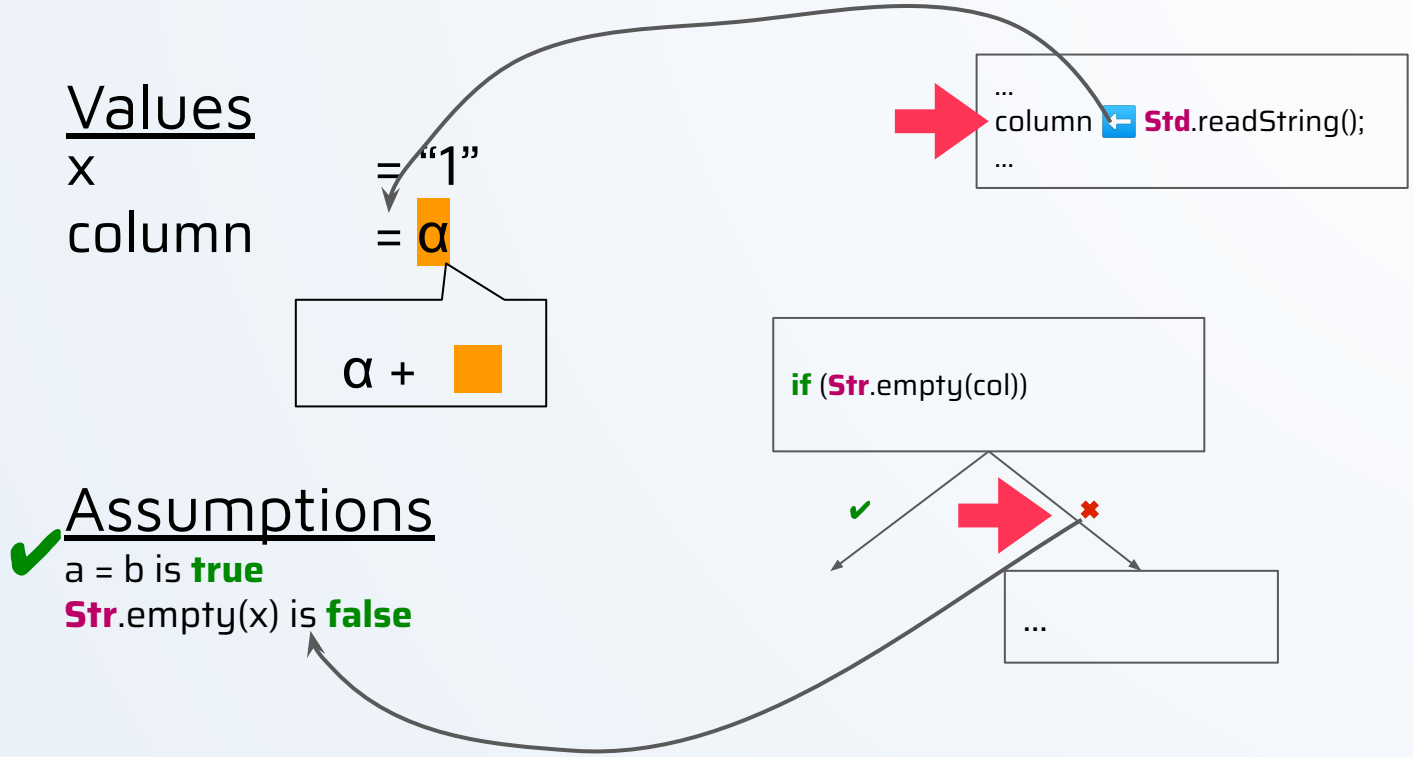
Explore paths taking different branches

Collect branch conditions

Check satisfiability



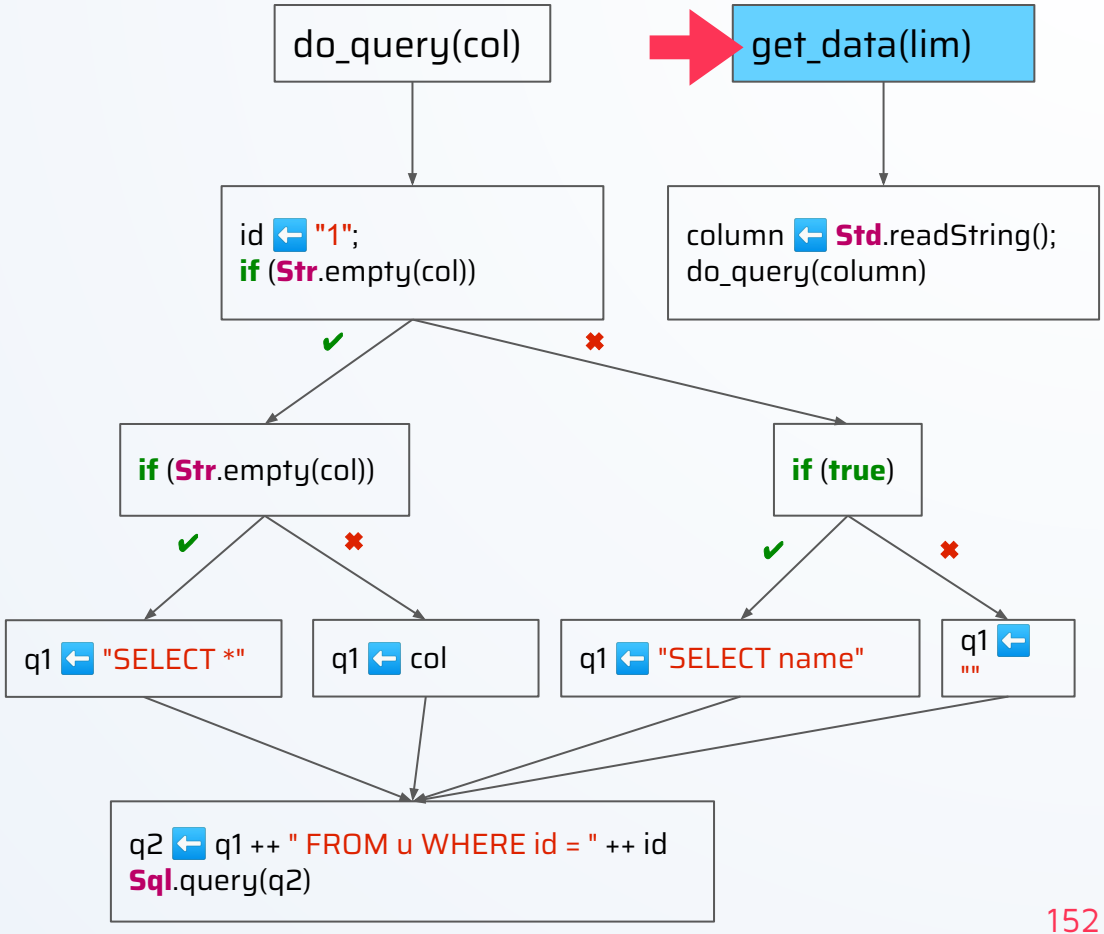
Symbolic Execution



Symbolic Execution

Values
lim = α

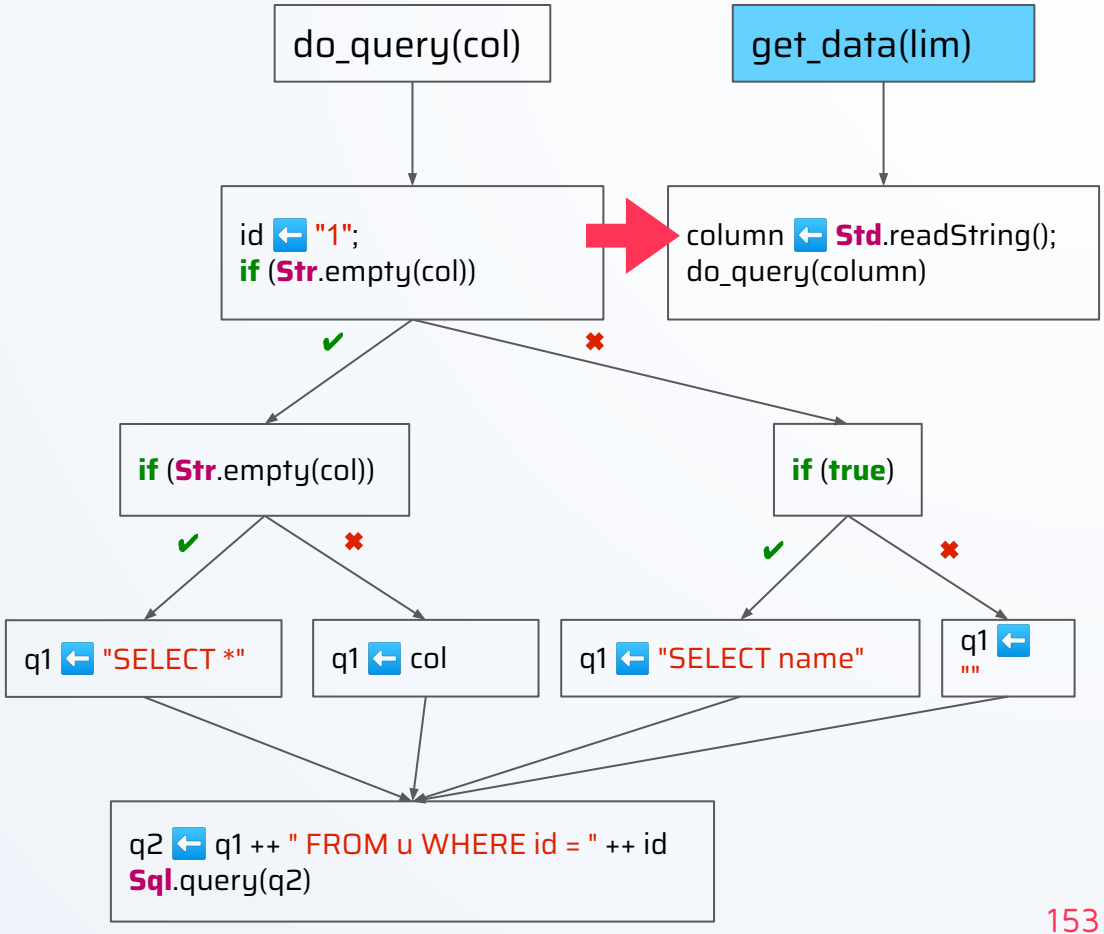
```
fn do_query(col: String): String = {
  val id: String = "1";
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}
```



Symbolic Execution

Values
lim = α
column = ß

```
fn do_query(col: String): String = {
  val id: String = "1";
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}
```

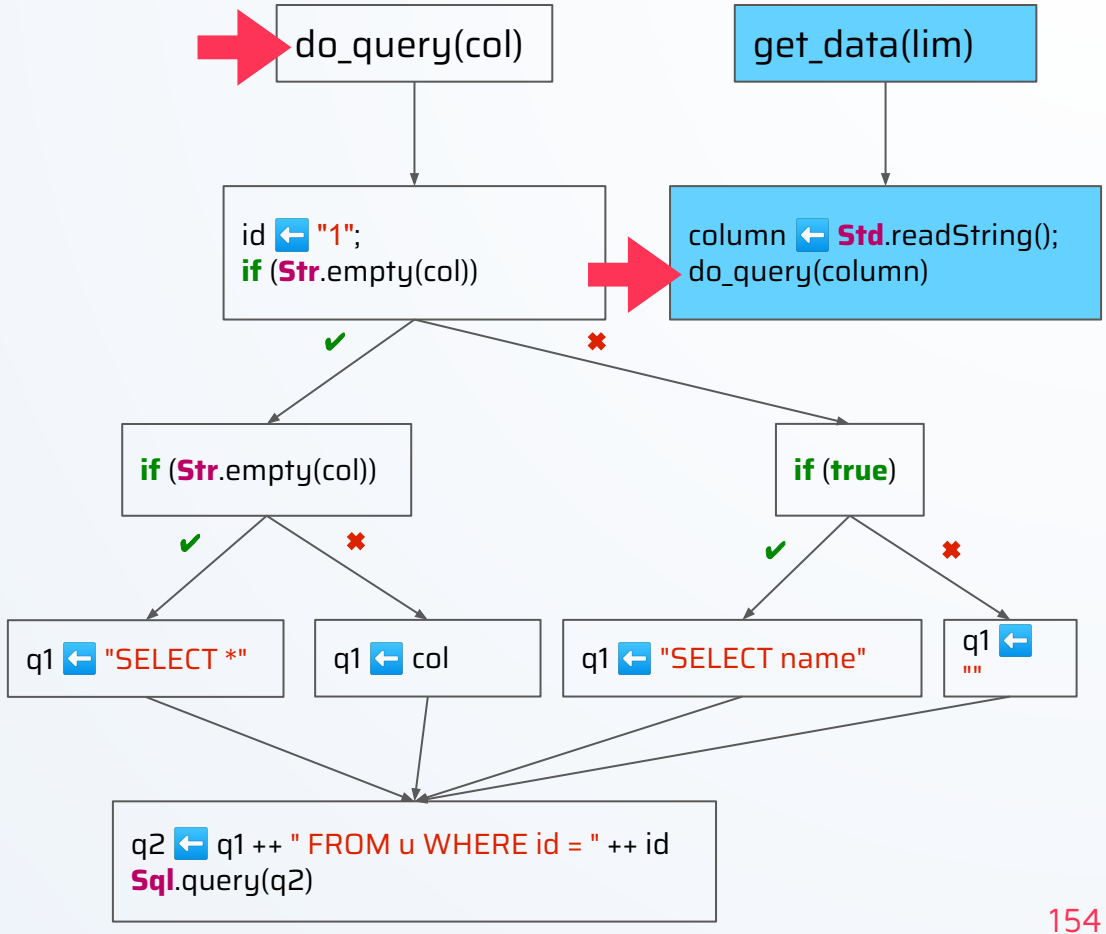


Symbolic Execution

Values

lim = α
column = β
col = β

```
fn do_query(col: String): String = {
  val id: String = "1";
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}
```

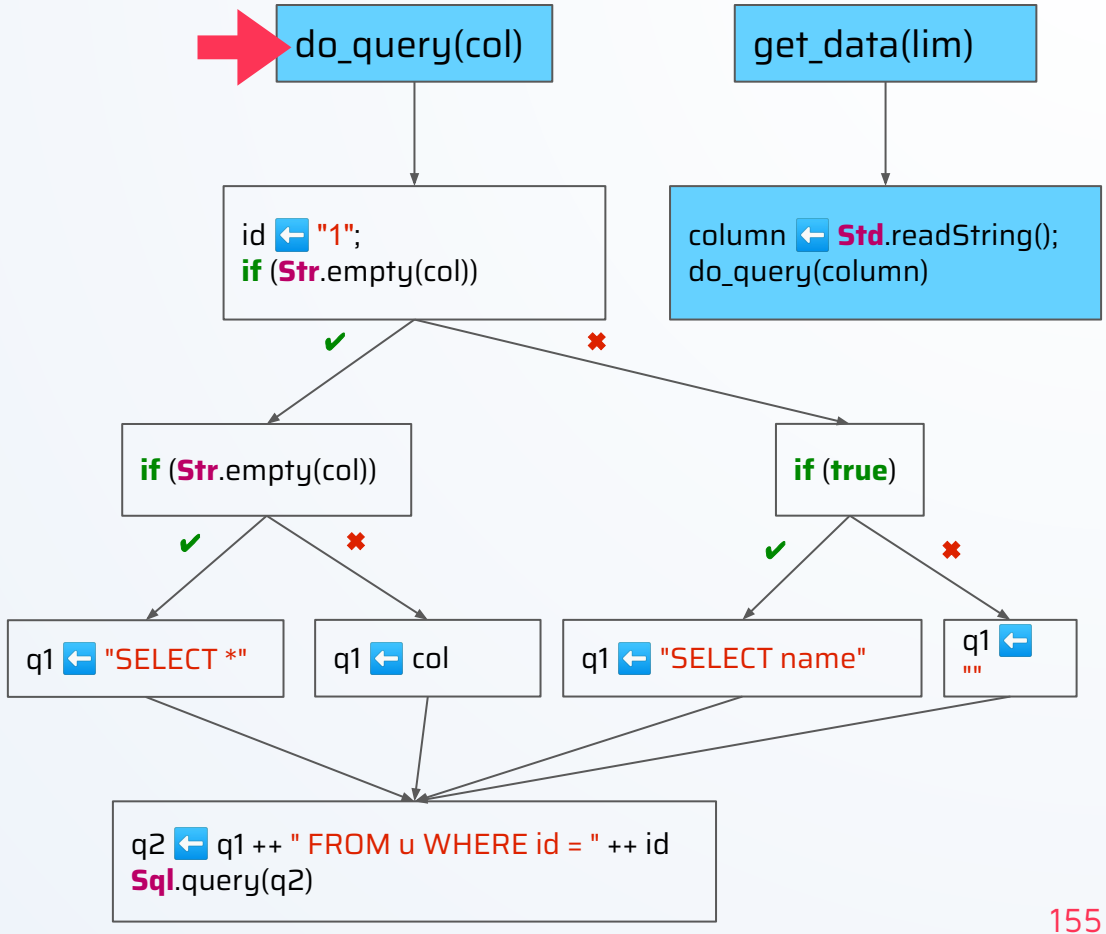


Symbolic Execution

Values

col = ß

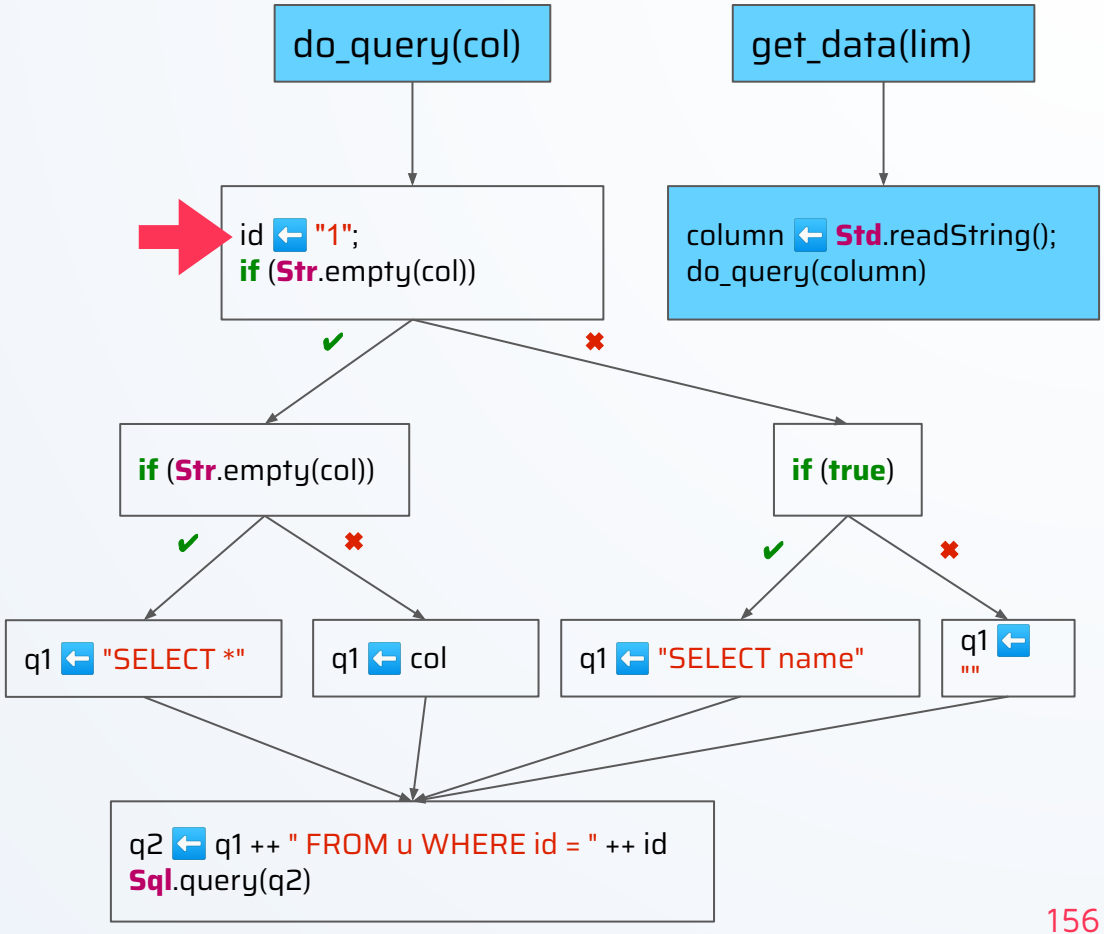
```
fn do_query(col: String): String = {
  val id: String = "1";
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}
```



Symbolic Execution

Values
col = ß
id = “1”

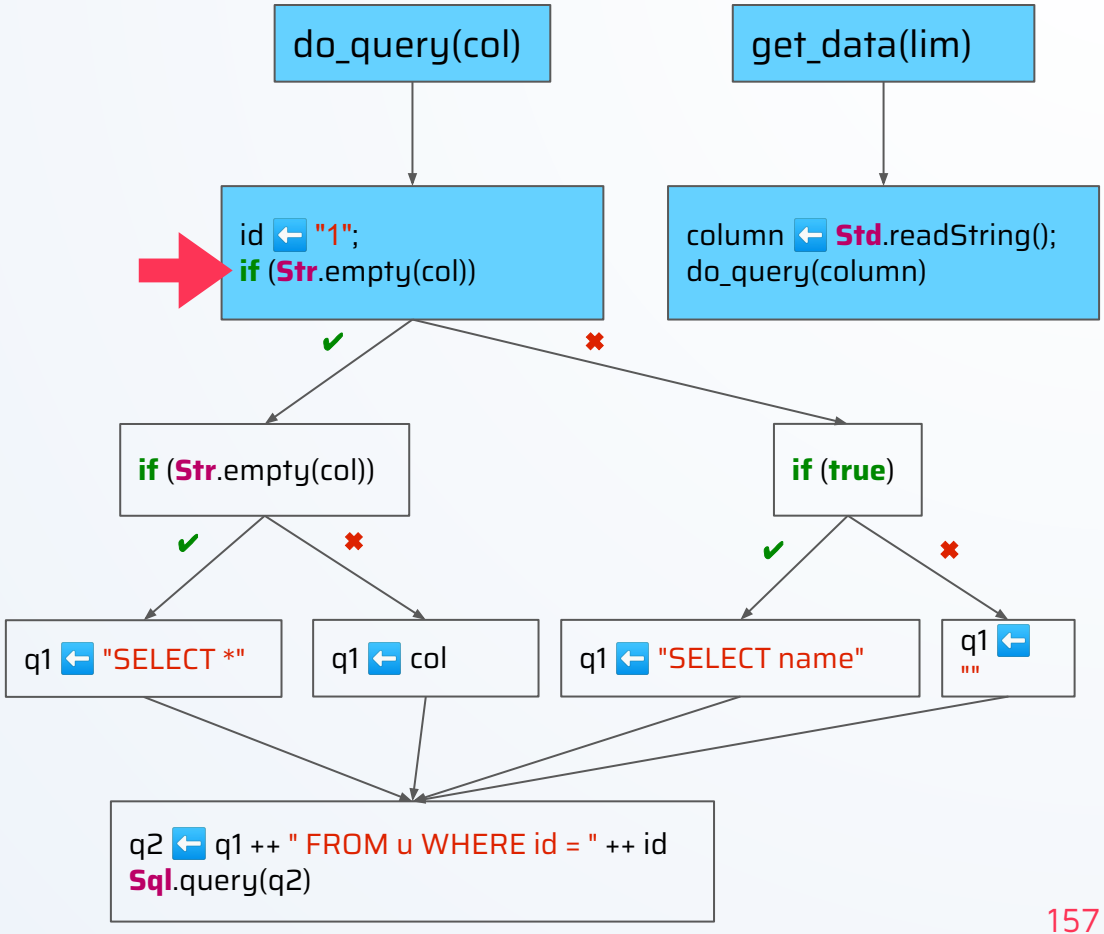
```
fn do_query(col: String): String = {
  val id: String = "1";
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}
```



Symbolic Execution

Values
col = ß
id = “1”

```
fn do_query(col: String): String = {
  val id: String = "1";
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}
```

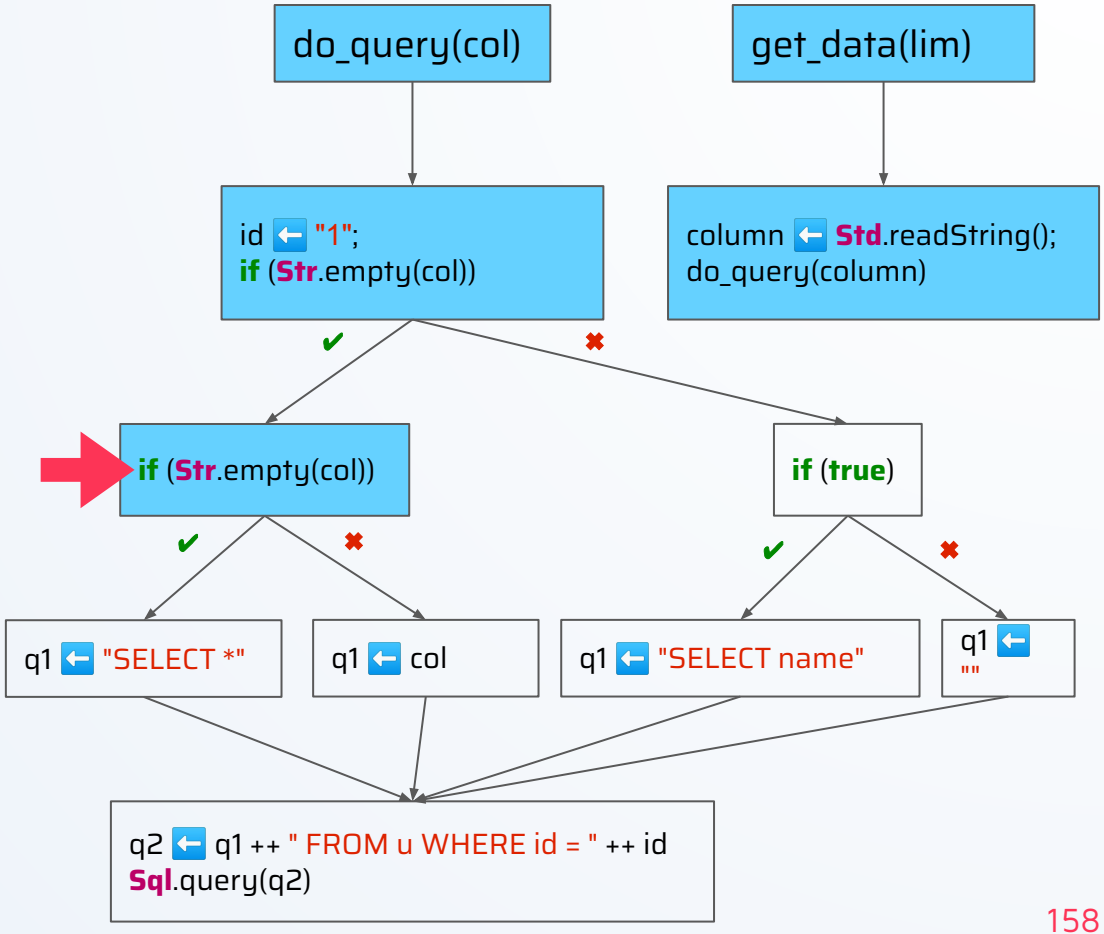


Symbolic Execution

✔ **Assumptions**
Str.empty(ß) is **true**

Values
col = ß
id = “1”

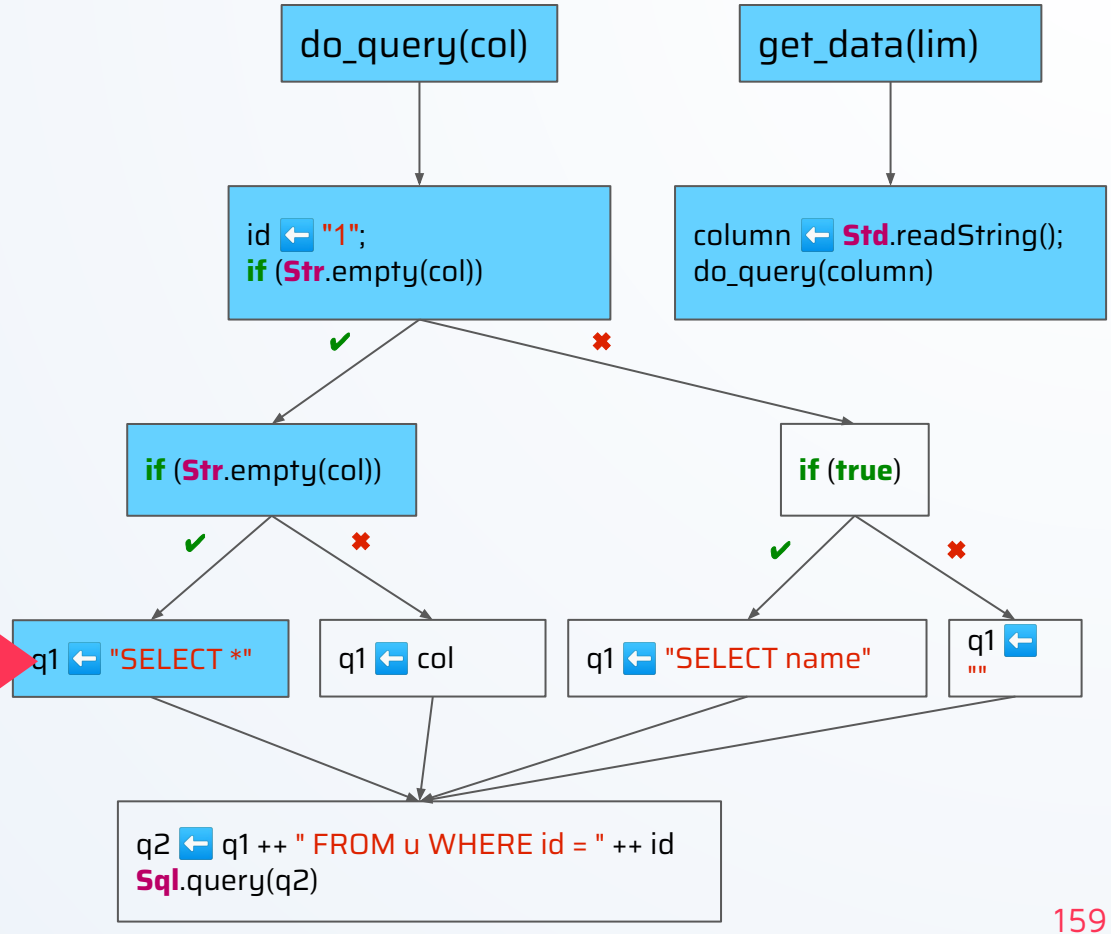
```
fn do_query(col: String): String = {
  val id: String = "1";
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}
```



Symbolic Execution

Assumptions	Values	
✔ Str.empty(ß) is true	col	= ß
Str.empty(ß) is true	id	= “1”
	q1	= “SELECT *”

```
fn do_query(col: String): String = {
  val id: String = "1";
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}
```

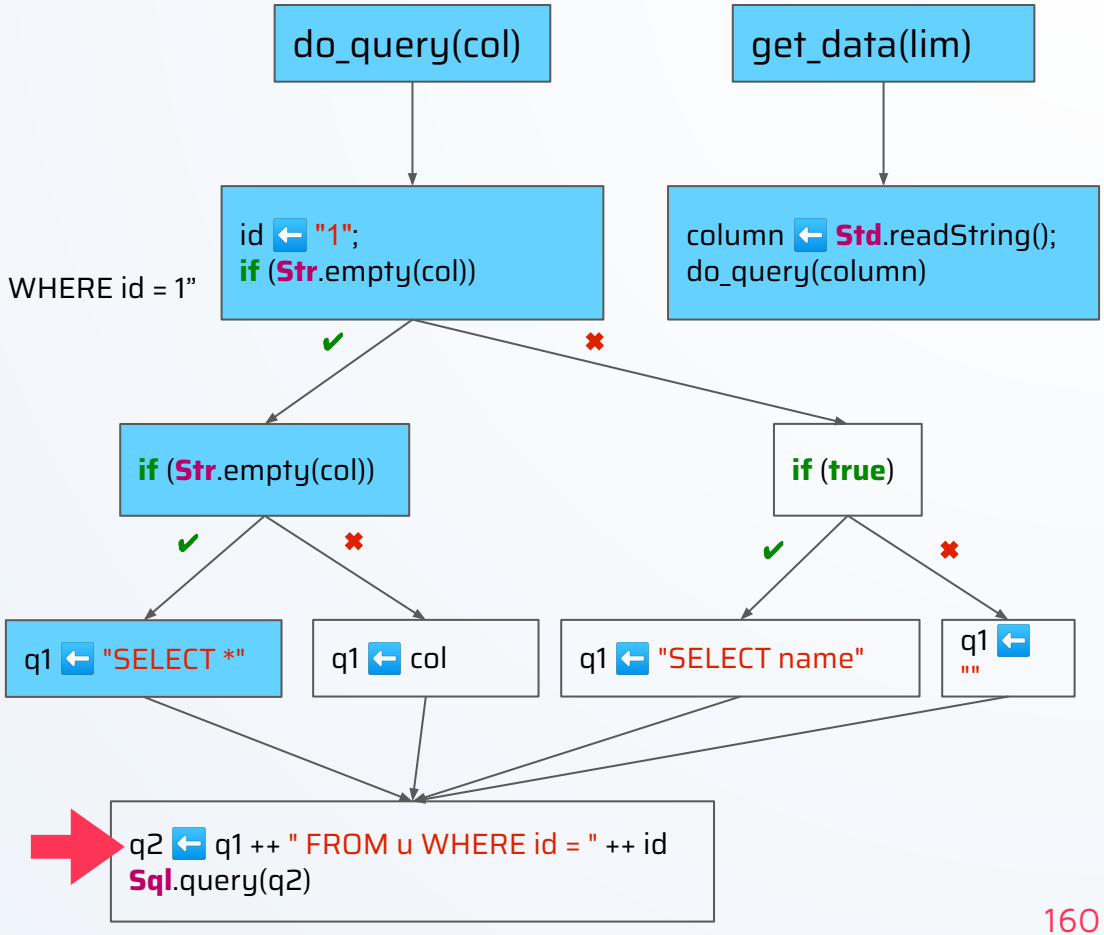


Symbolic Execution

Assumptions
Str.empty(ß) is true
Str.empty(ß) is true

Values
col = ß
id = “1”
q1 = “SELECT *”
q2 = “SELECT * FROM u WHERE id = 1”

```
fn do_query(col: String): String = {
  val id: String = "1";
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}
```

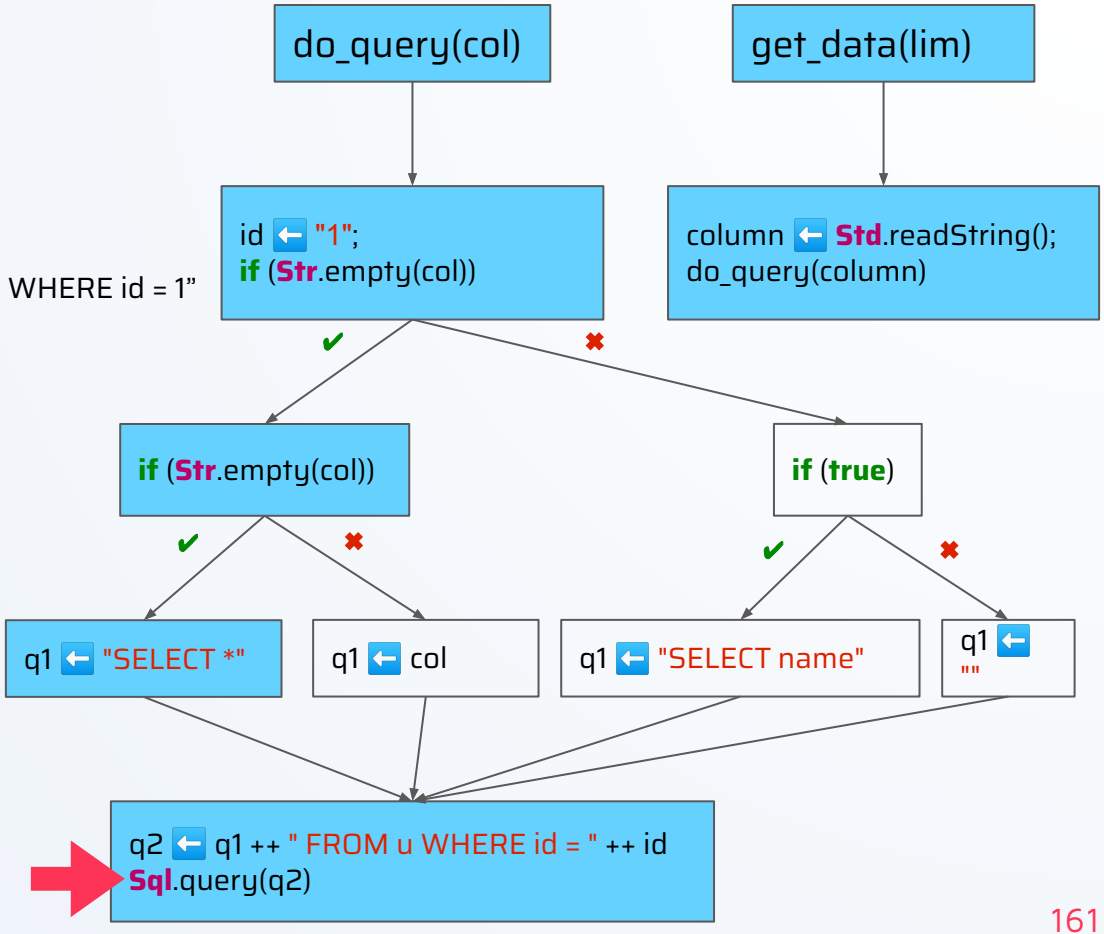


Symbolic Execution

Assumptions
Str.empty(ß) is true
Str.empty(ß) is true

Values
col = ß
id = “1”
q1 = “SELECT *”
q2 = “SELECT * FROM u WHERE id = 1”

```
fn do_query(col: String): String = {
  val id: String = "1";
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}
```

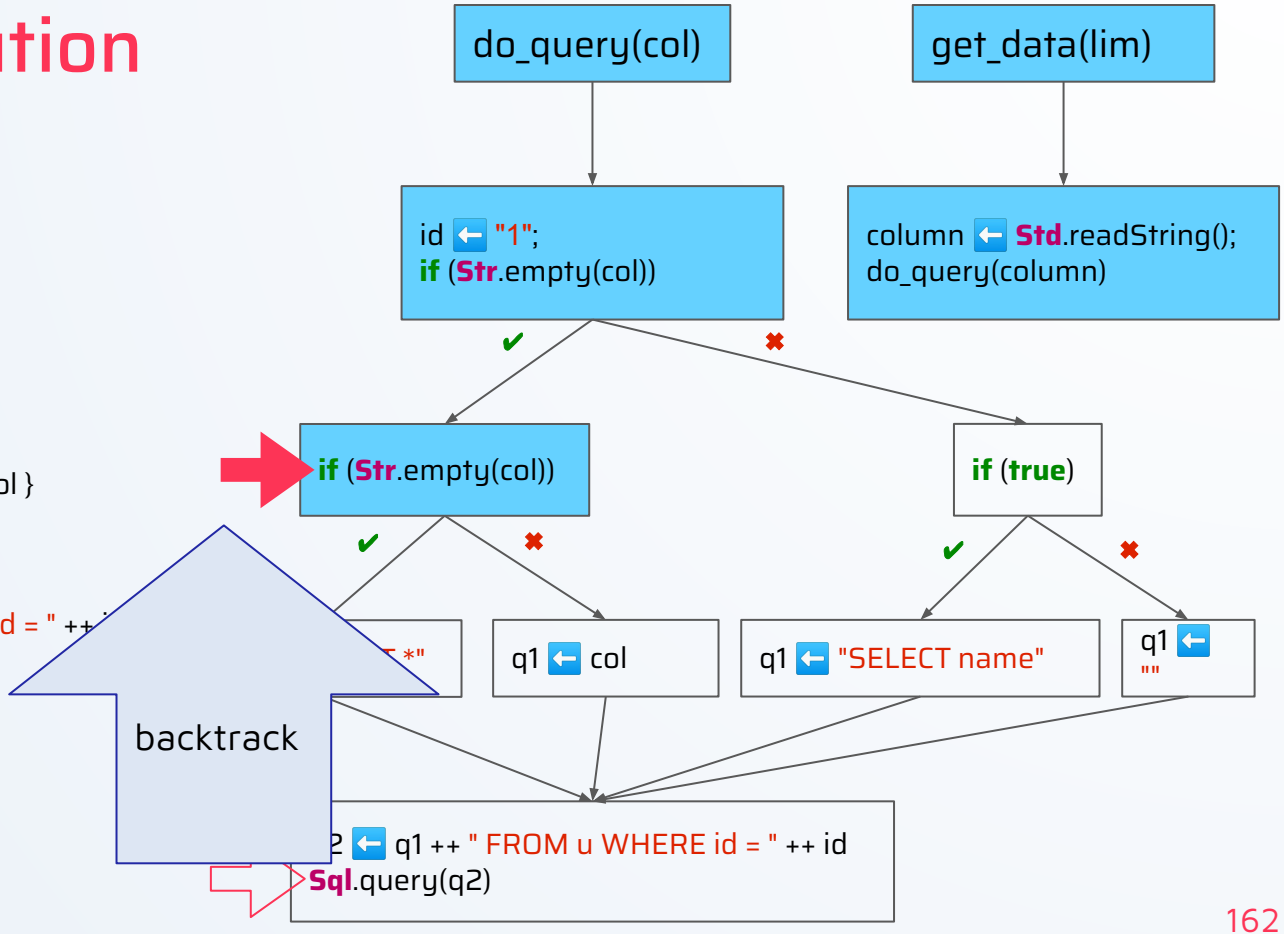


Symbolic Execution

Assumptions
Str.empty(ß) is **true**

Values
col = ß
id = “1”

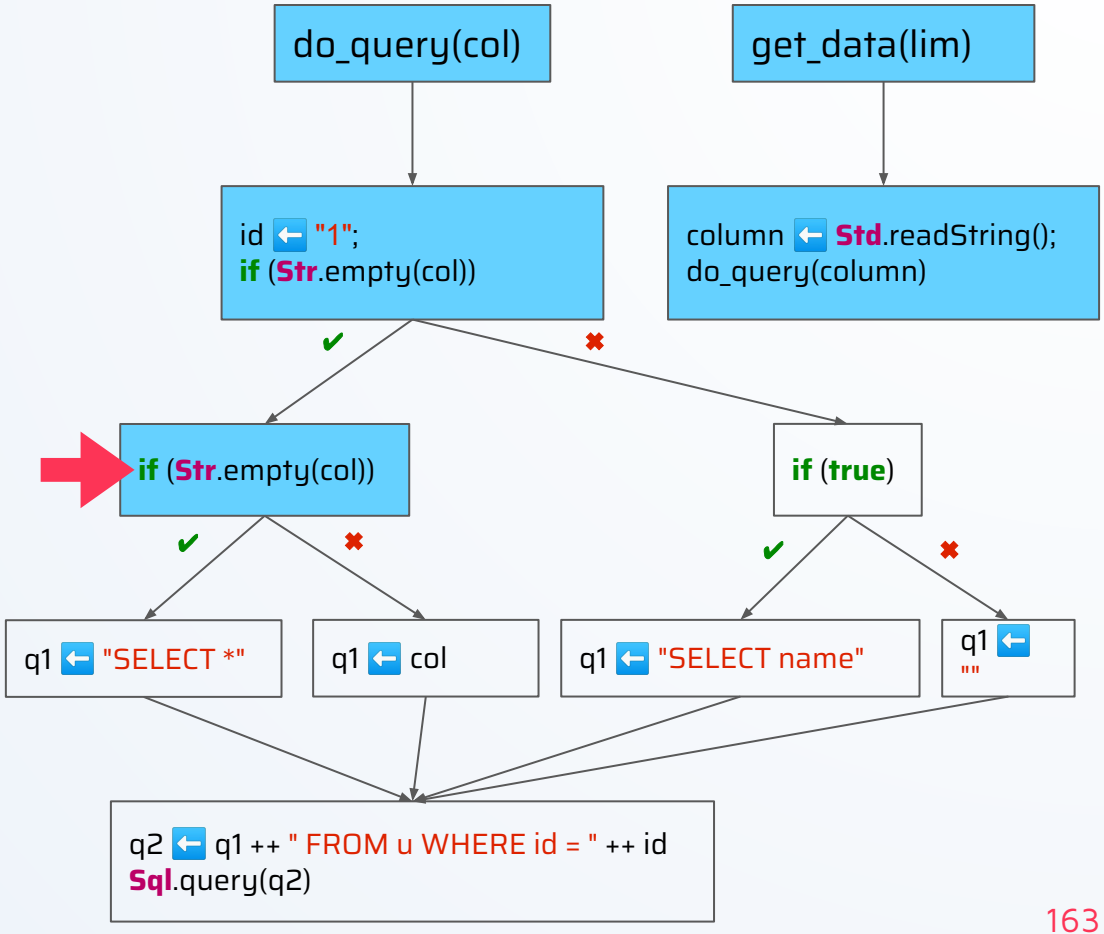
```
fn do_query(col: String): String = {
  val id: String = "1";
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}
```



Symbolic Execution

Assumptions	Values
Str.empty(ß) is **true**	col = ß
	id = “1”

```
fn do_query(col: String): String = {
  val id: String = "1";
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}
```

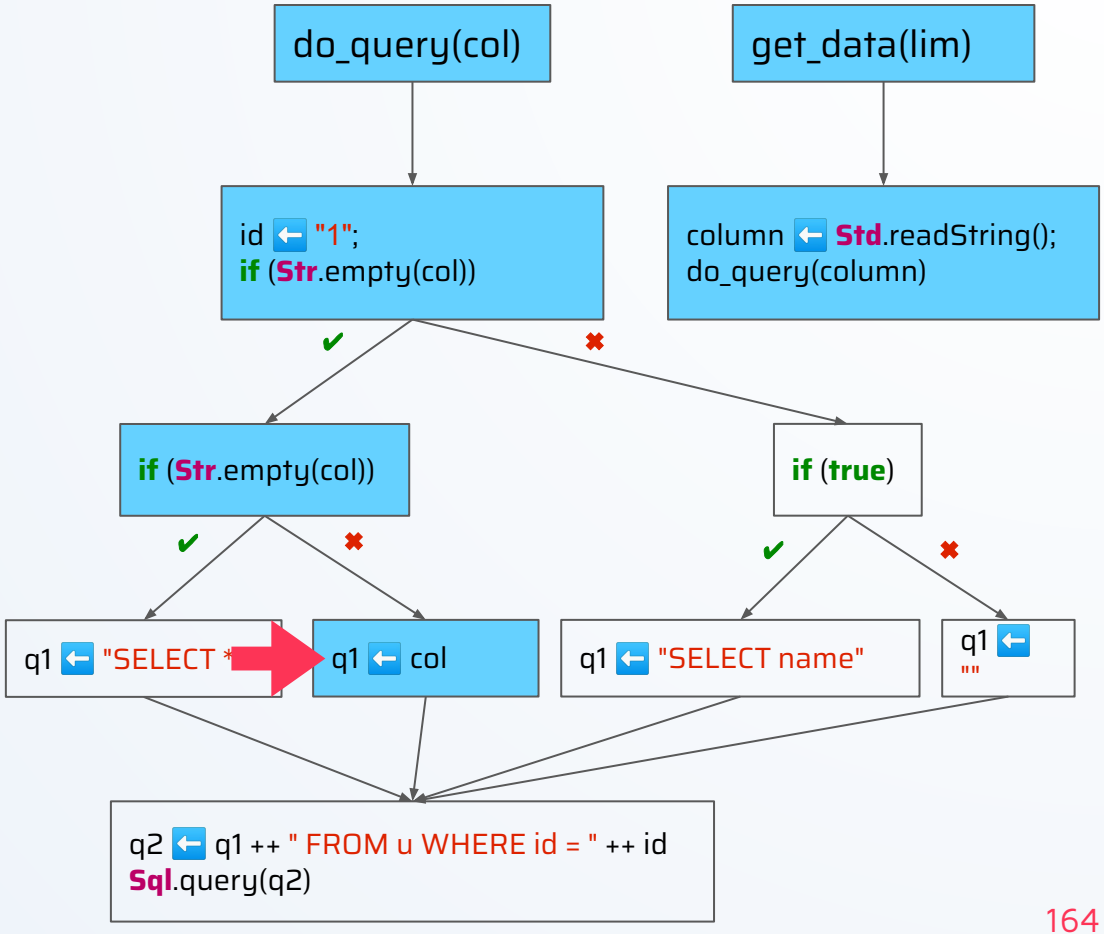


Symbolic Execution

Assumptions
- ✖ Str.empty(ß) is true
- Str.empty(ß) is false

Values
- col = ß
- id = "1"
- q1 = ß

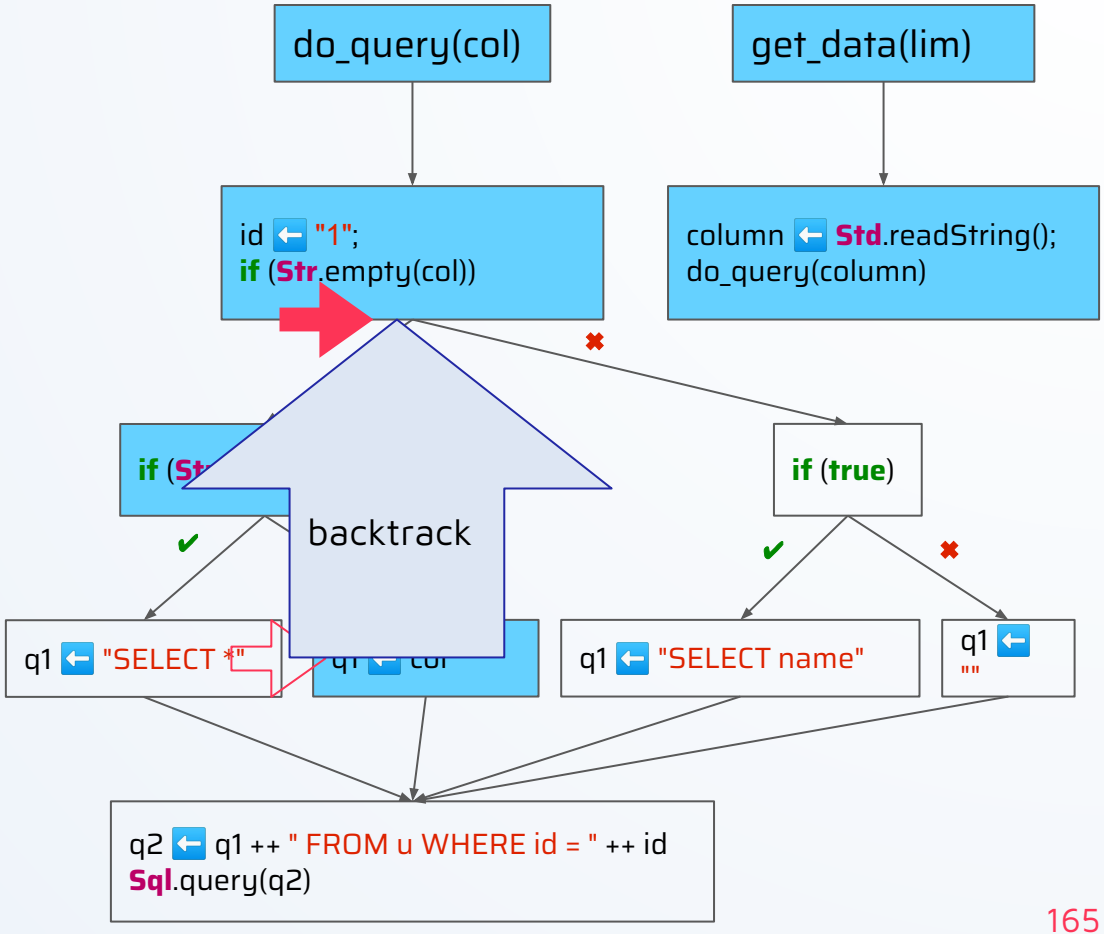
```
fn do_query(col: String): String = {
  val id: String = "1";
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}
```



Symbolic Execution

Assumptions	Values
✖ Str.empty(ß) is true	col = ß
Str.empty(ß) is false	id = "1"
	q1 = ß

```
fn do_query(col: String): String = {
  val id: String = "1";
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}
```

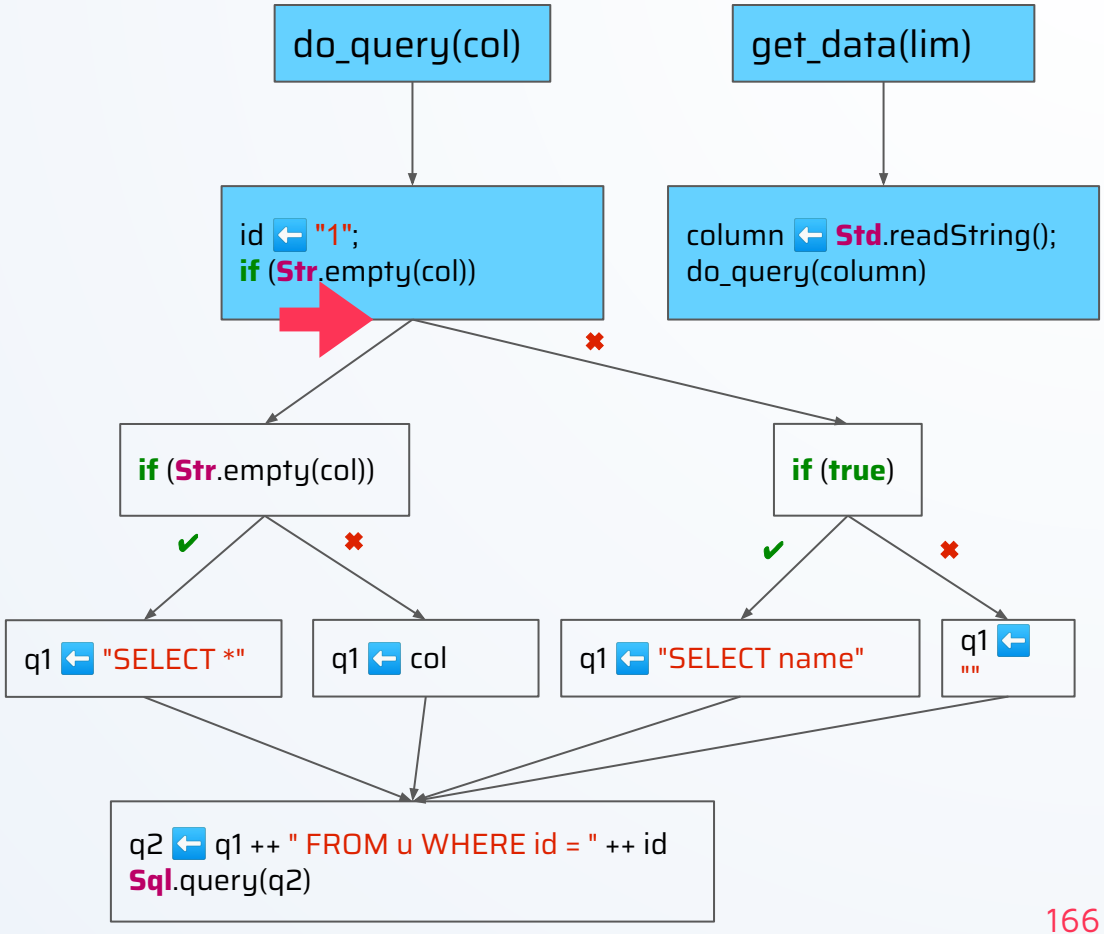


Symbolic Execution

Assumptions

Values
col = β
id = “1”

```
fn do_query(col: String): String = {
  val id: String = "1";
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}
```

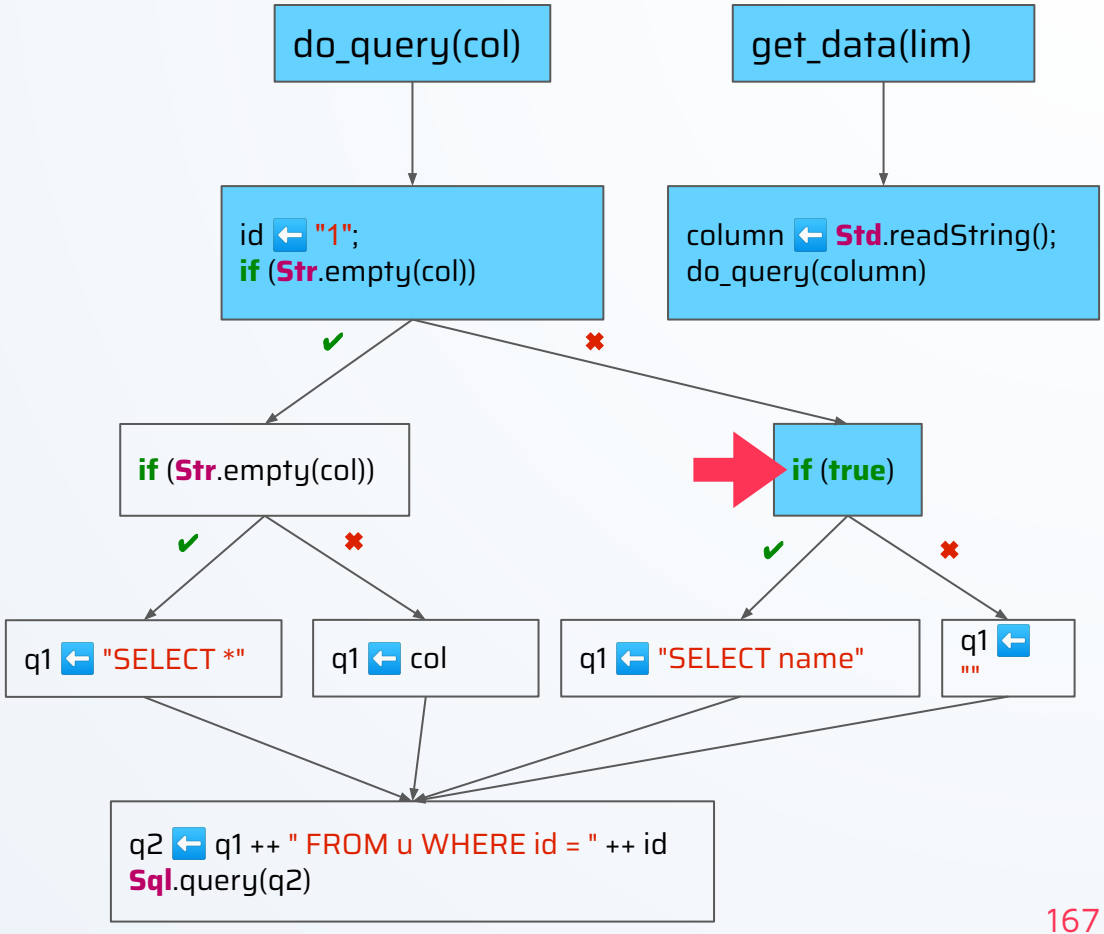


Symbolic Execution

✔ Assumptions
Str.empty(ß) is **false**

Values
col = ß
id = “1”

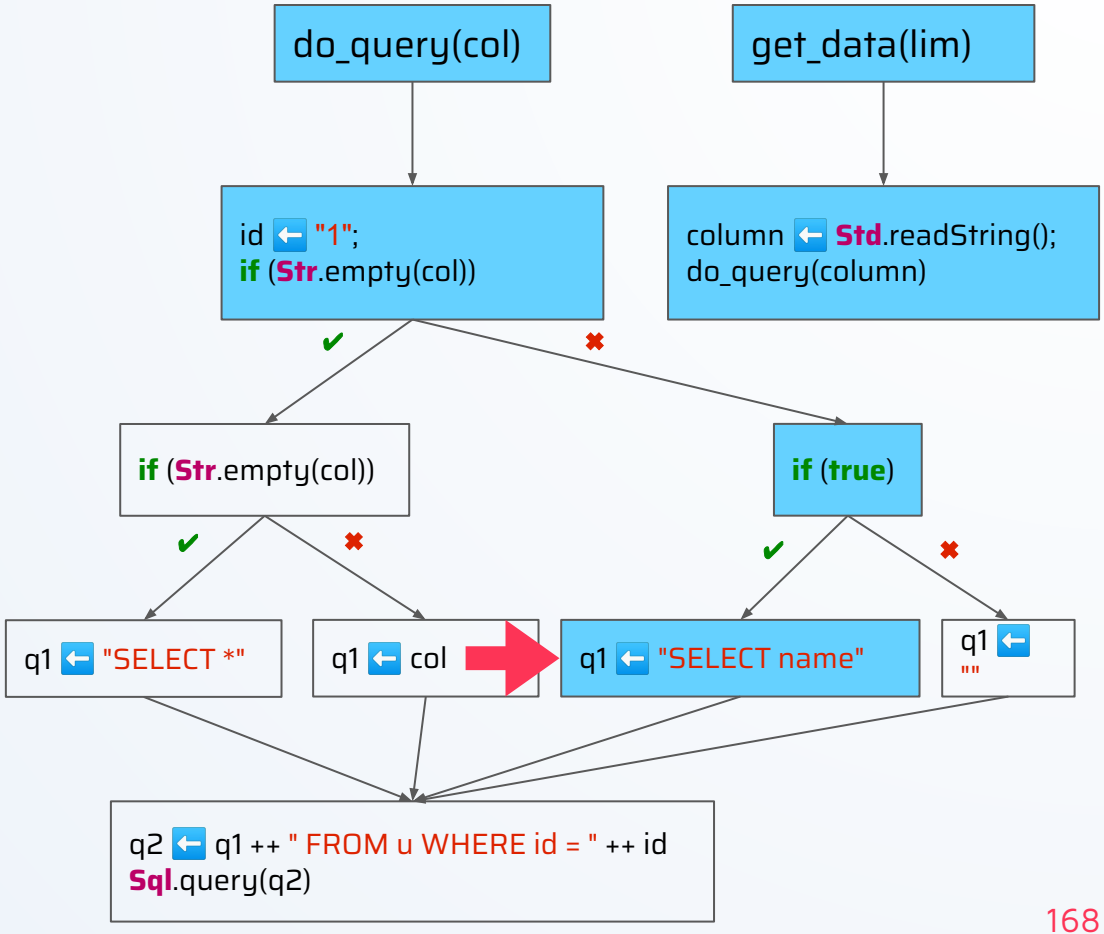
```
fn do_query(col: String): String = {
  val id: String = "1";
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}
```



Symbolic Execution

Assumptions	Values
✔ **Str**.empty(ß) is **false**	col = ß
true is **true**	id = “1”
	q1 = “SELECT name”

```
fn do_query(col: String): String = {
  val id: String = "1";
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}
```

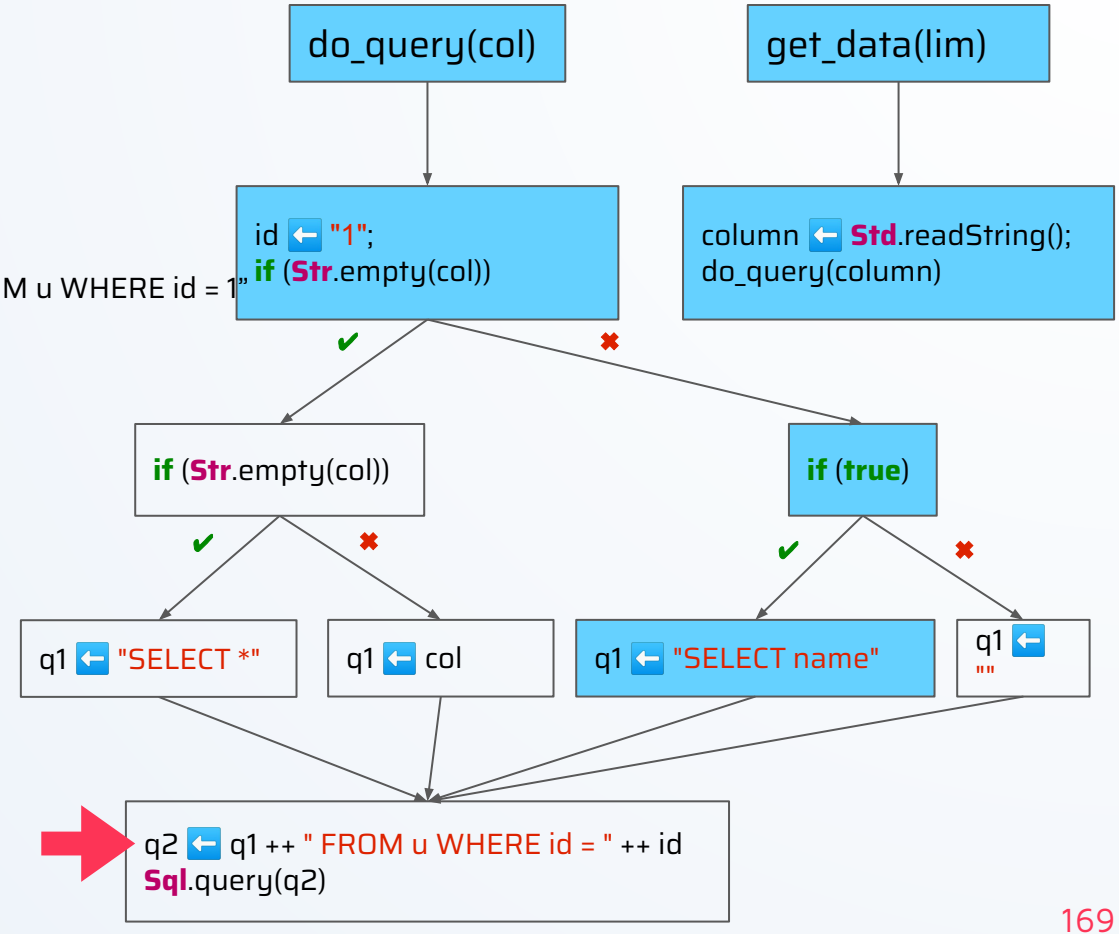


Symbolic Execution

Assumptions
Str.empty(ß) is **false**
true is **true**

Values
col = ß
id = “1”
q1 = “SELECT name”
q2 = “SELECT name FROM u WHERE id = 1”

```
fn do_query(col: String): String = {
  val id: String = "1";
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}
```

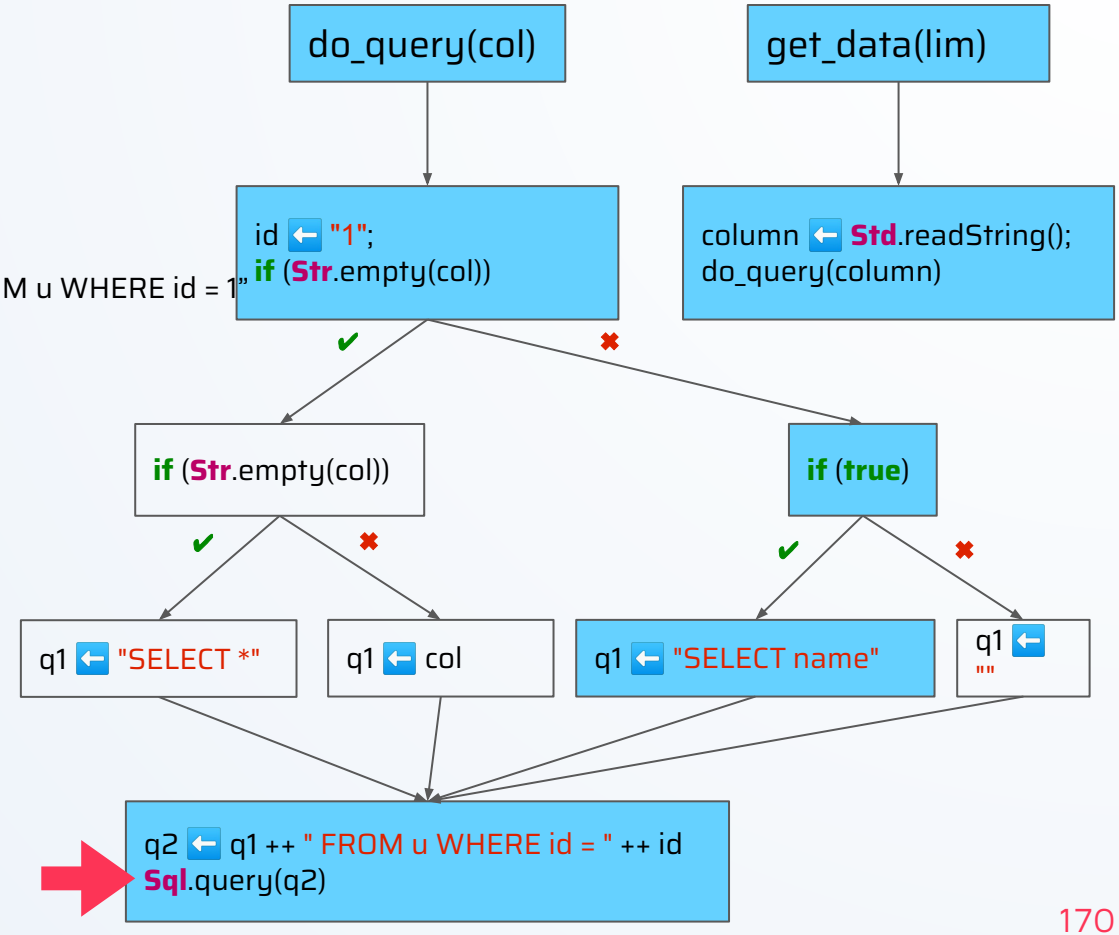


Symbolic Execution

Assumptions
Str.empty(ß) is **false**
true is **true**

Values
col = ß
id = “1”
q1 = “SELECT name”
q2 = “SELECT name FROM u WHERE id = 1”

```
fn do_query(col: String): String = {
  val id: String = "1";
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}
```

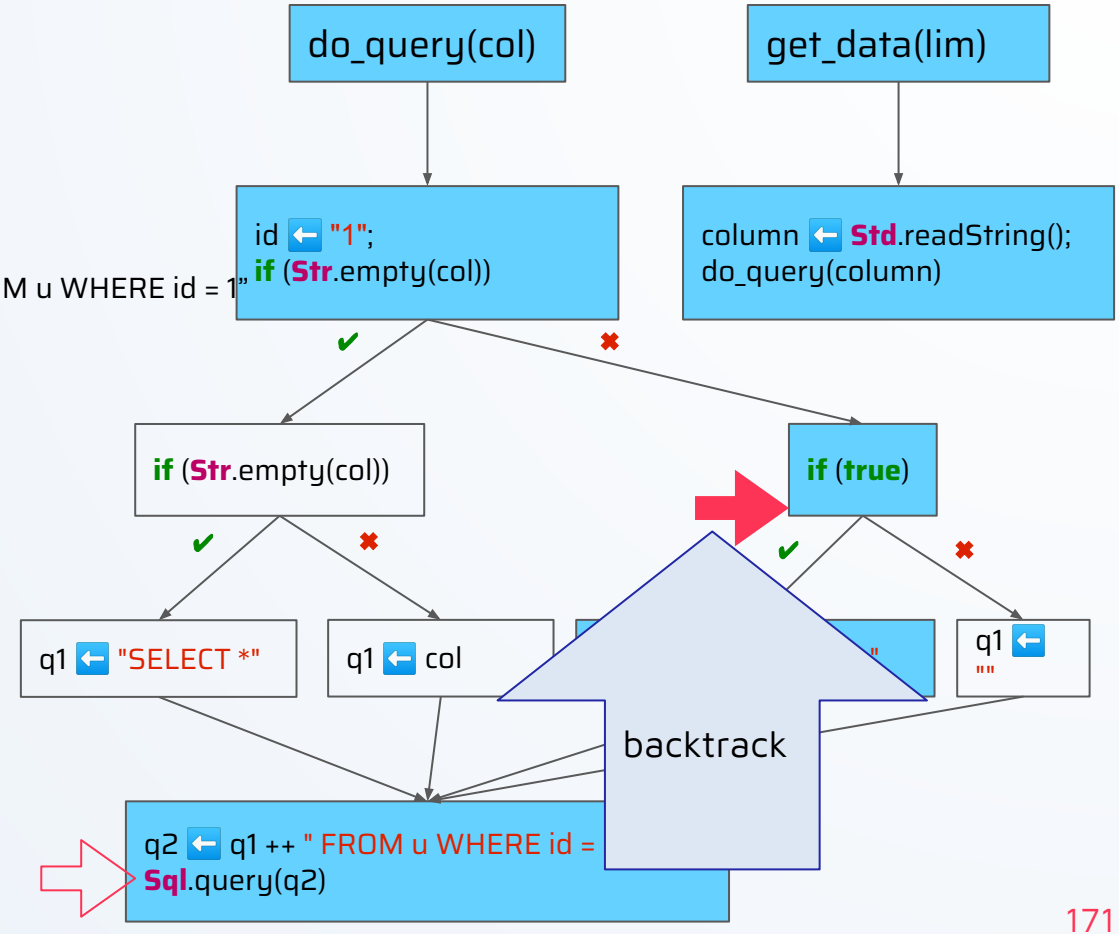


Symbolic Execution

Assumptions
Str.empty(ß) is **false**
true is **true**

Values
col = ß
id = “1”
q1 = “SELECT name”
q2 = “SELECT name FROM u WHERE id = 1”

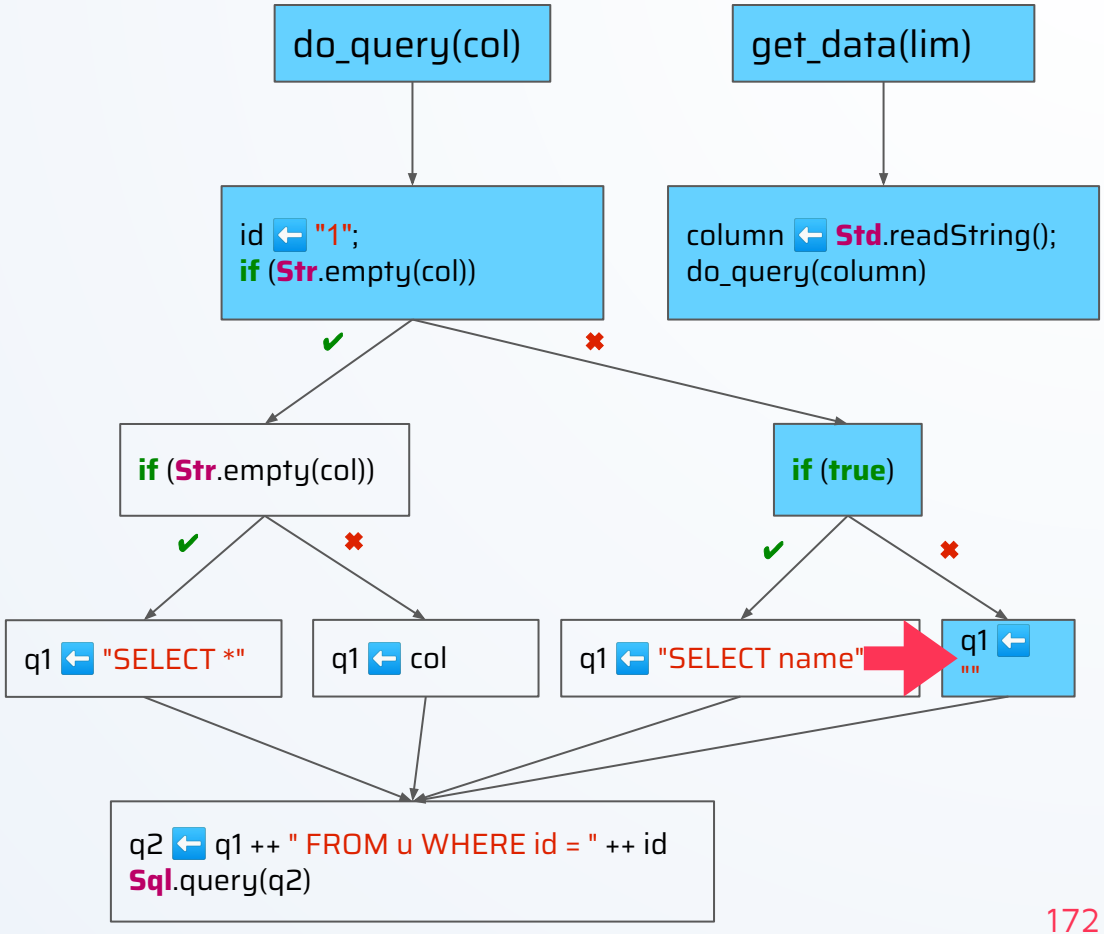
```
fn do_query(col: String): String = {
  val id: String = "1";
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}
```



Symbolic Execution

Assumptions	Values
✖ Str.empty(ß) is false	col = ß
true is false	id = "1"
	q1 = ""

```
fn do_query(col: String): String = {
  val id: String = "1";
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}

fn get_data(lim: Int(32)): String = {
  val column: String = Std.readString();
  do_query(column)
}
```



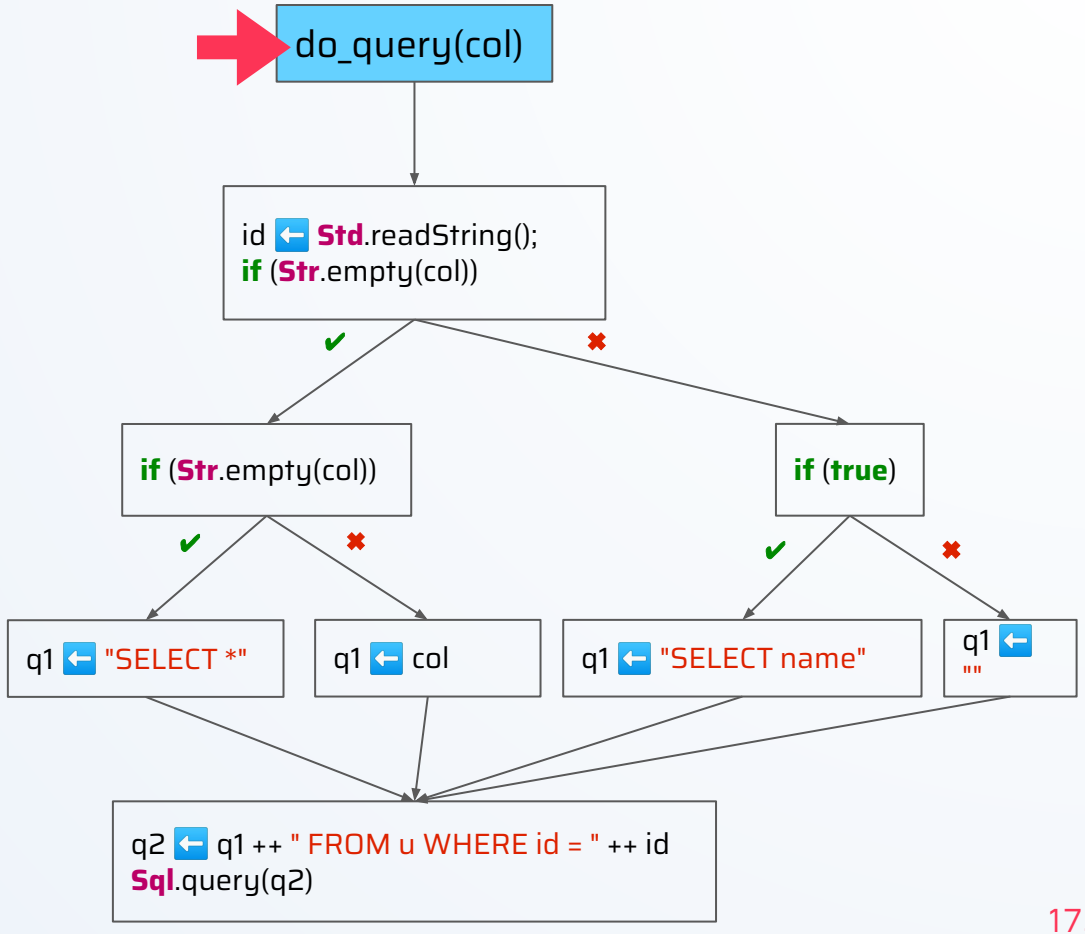
Symbolic Execution

Assumptions

Values

col = α

```
fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}
```



Symbolic Execution

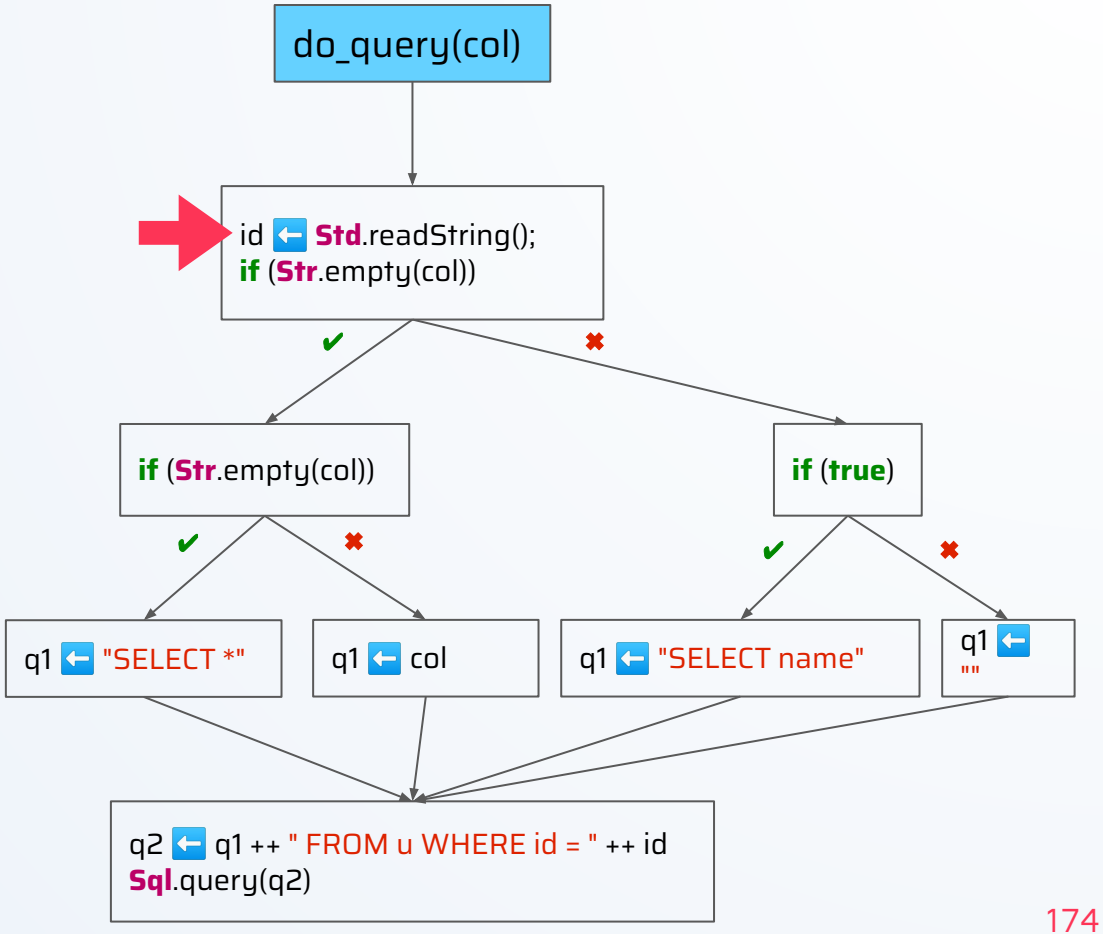
Assumptions

Values

col = α

id = β

```
fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}
```

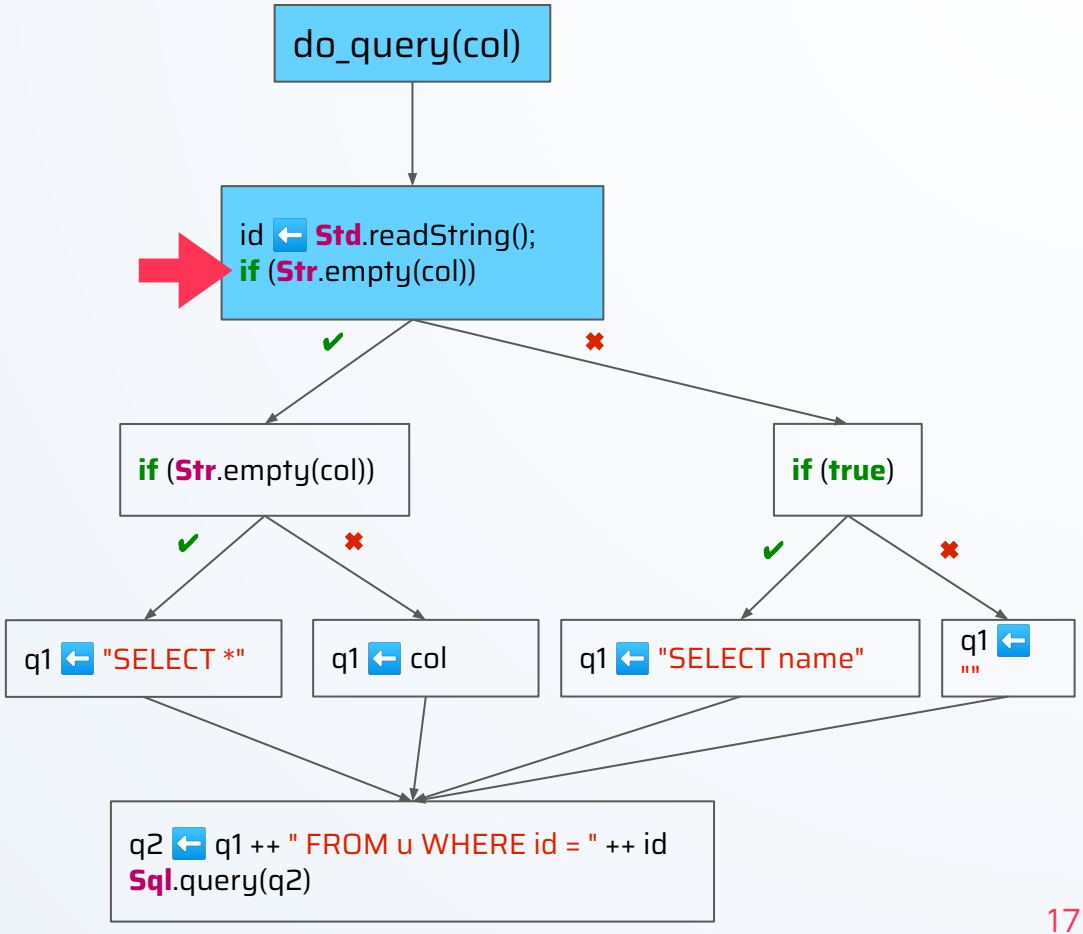


Symbolic Execution

Assumptions

Values
col = α
id = β

```
fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}
```

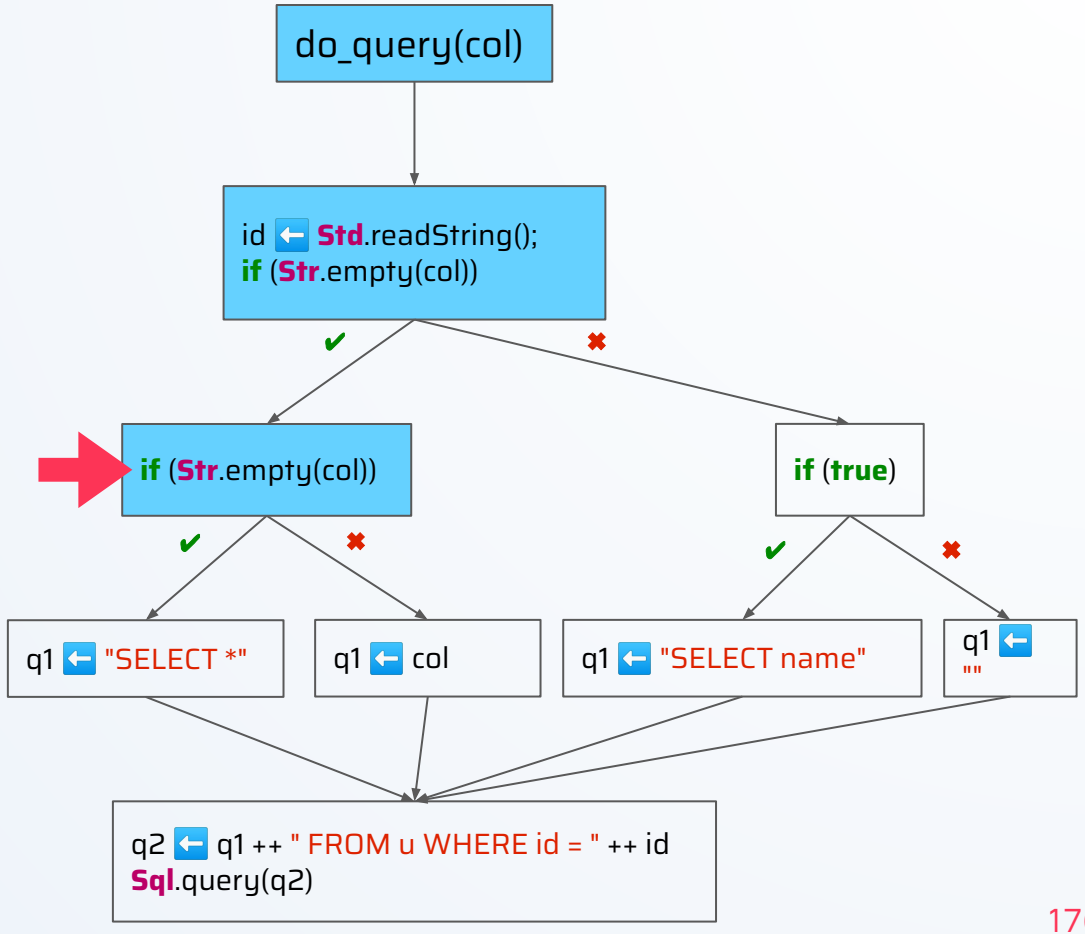


Symbolic Execution

✔ Assumptions
Str.empty(ß) is **true**

Values
col = α
id = ß

```
fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}
```



Symbolic Execution

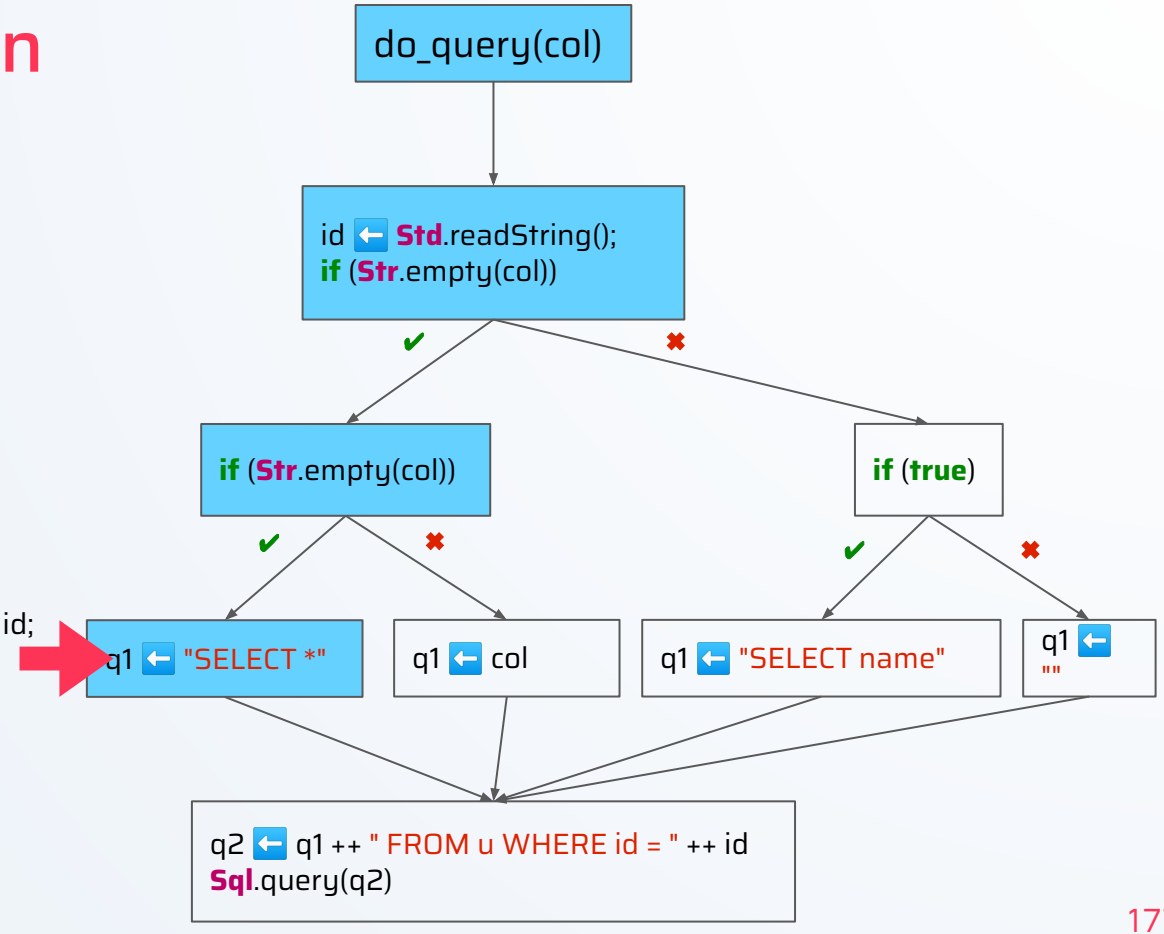
Assumptions

✔ Str.empty(ß) is true
Str.empty(ß) is true

Values

col = α
id = ß
q1 = “SELECT *”

```
fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}
```

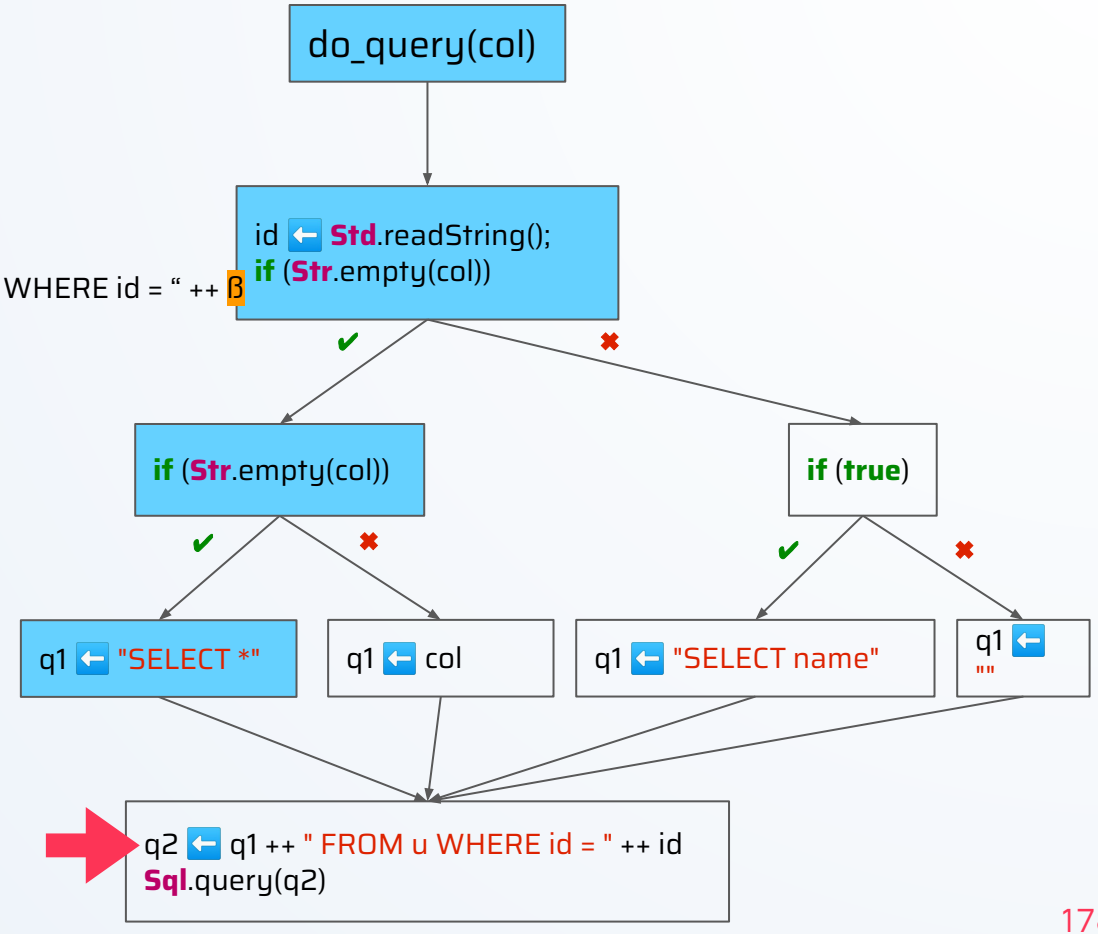


Symbolic Execution

Assumptions
Str.empty(ß) is true
Str.empty(ß) is true

Values
col = α
id = ß
q1 = “SELECT *”
q2 = “SELECT * FROM u WHERE id = “ ++ ß

```
fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}
```



Symbolic Execution

Assumptions

Str.empty(ß) is true

Str.empty(ß) is true

Values

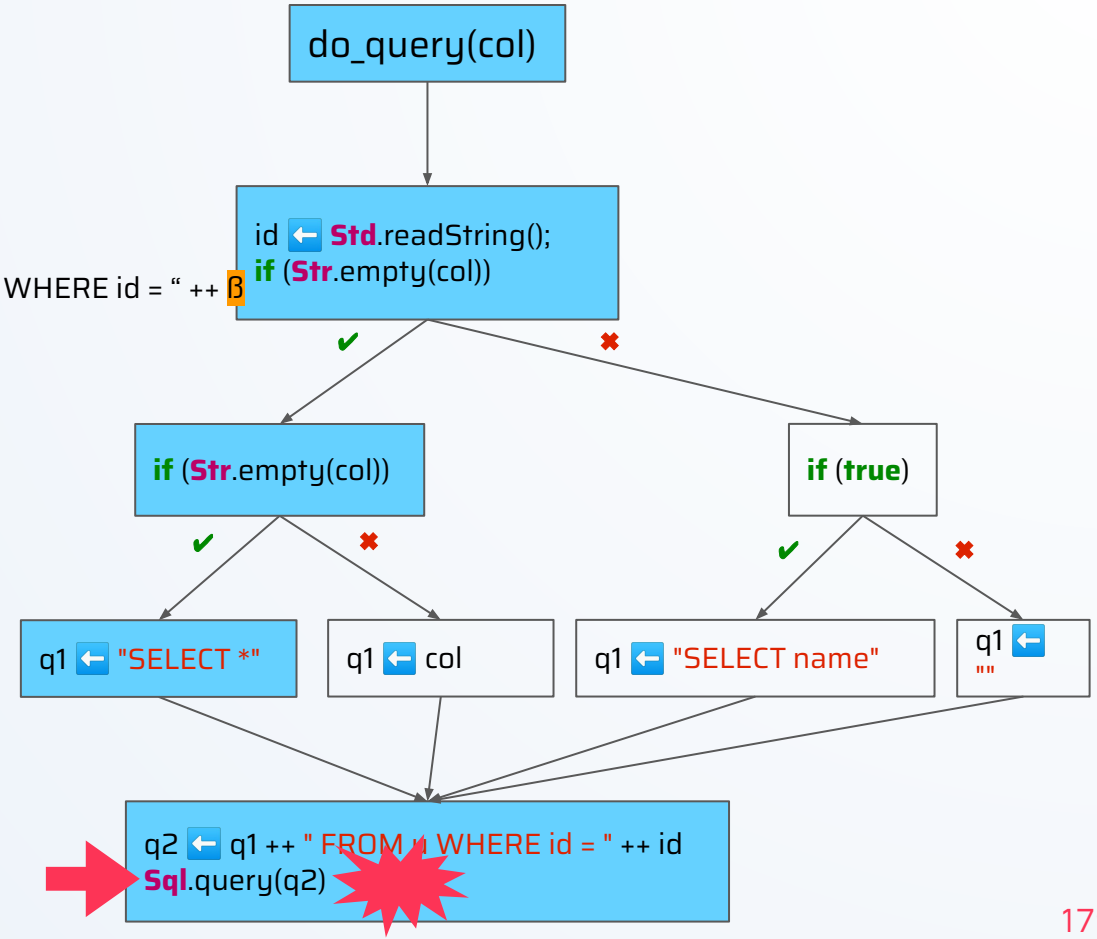
col = α

id = ß

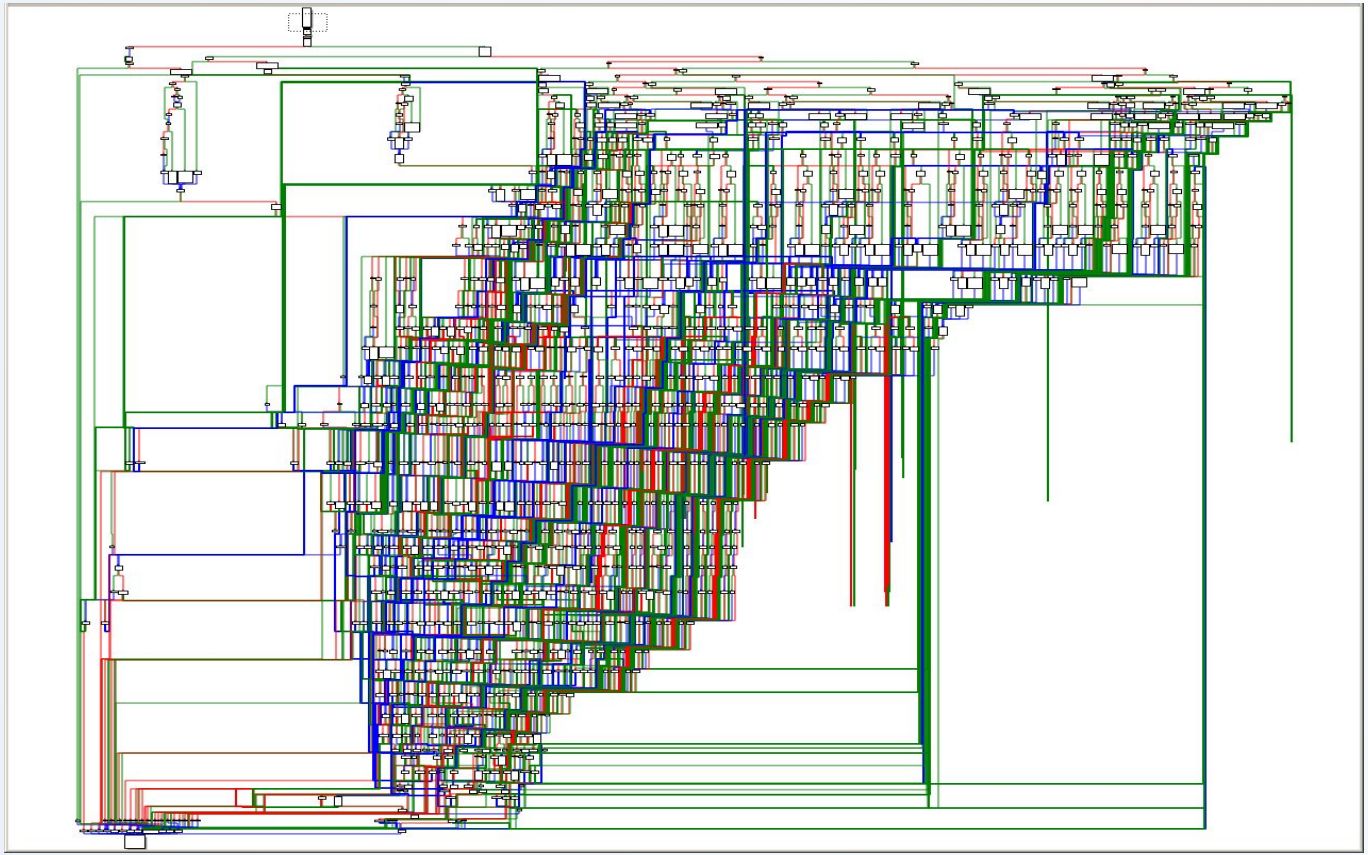
q1 = “SELECT *”

q2 = “SELECT * FROM u WHERE id = “ ++ ß

```
fn do_query(col: String): String = {
  val id: String = Std.readString();
  val q1: String = if (Str.empty(col)) {
    if (Str.empty(col)) { "SELECT *" } else { col }
  } else {
    if (true) { "SELECT name" } else { "" }
  };
  val q2: String = q1 ++ " FROM u WHERE id = " ++ id;
  Sql.query(q2)
}
```



Symbolic Execution: Limitation



Outline

First hour

Intro to static analysis

Place for static analysis

AST-based analysis

- Visitors & Matchers

Second hour

Taint Analysis

Symbolic Execution

→ Static Analysis Trade-off

Demo

Static Analysis Trade-Off

Outline

First hour

Intro to static analysis

Place for static analysis

AST-based analysis

- Visitors & Matchers

Second hour

Taint Analysis

Symbolic Execution

Static Analysis Trade-off

→ Demo

Questions?

arseniy.zaostrovnykh@sonarsource.com
quentin.jaquier@sonarsource.com
@sonarsource

https://sonarsource.com

sonar