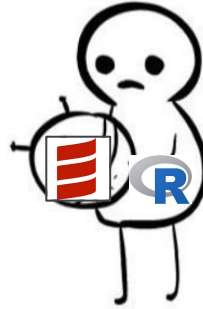# Catalytic Compilation

A modern take on safe Spark

Spark SQL

- Distributed relational engine since 2014

- Scala's Killer App

- Large ecosystem : data sources, languages

```sql
SELECT s.name, r.timestamp, r.temperature
FROM weather_stations s
JOIN weather_readings r ON s.id = r.station_id
WHERE r.temperature > 30.0
ORDER BY r.temperature DESC
```

SQL String

```scala
val df = spark.sql:
  """

  SELECT s.name, r.timestamp, r.temperature
  FROM weather_stations s
  JOIN weather_readings r ON s.id = r.station_id
  WHERE r.temperature > 30.0
  ORDER BY r.temperature DESC
  """
```
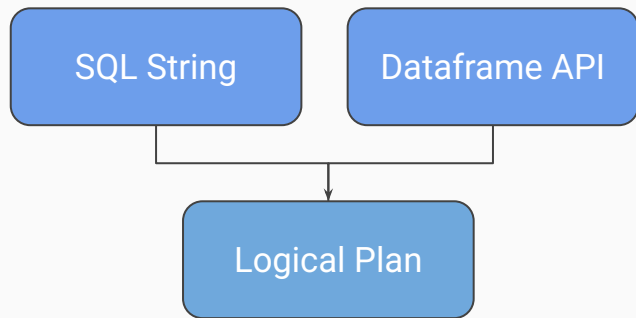
Dataframe API

```
stations
    .as("s")
    .join(readings.as("r"), stations.col("id") === readings.col("station_id"), "inner")
    .where(col("r.temperature") > 30.0)
    .select(
      col("s.name").as("name"),
      col("r.timestamp").as("timestamp"),
      col("r.temperature").as("temperature")
    )
    .orderBy(col("r.temperature").desc)
```

```
'Sort ['r.temperature DESC NULLS LAST], true
+- 'Project ['s.name, 'r.timestamp, 'r.temperature]
  +- 'Filter ('r.temperature > 30.0)
    +- 'Join Inner, ('s.id = 'r.station_id)
        :- 'SubqueryAlias s
        :  +- 'UnresolvedRelation [weather_stations], [], false
        +- 'SubqueryAlias r
            +- 'UnresolvedRelation [weather_readings], [], false
```
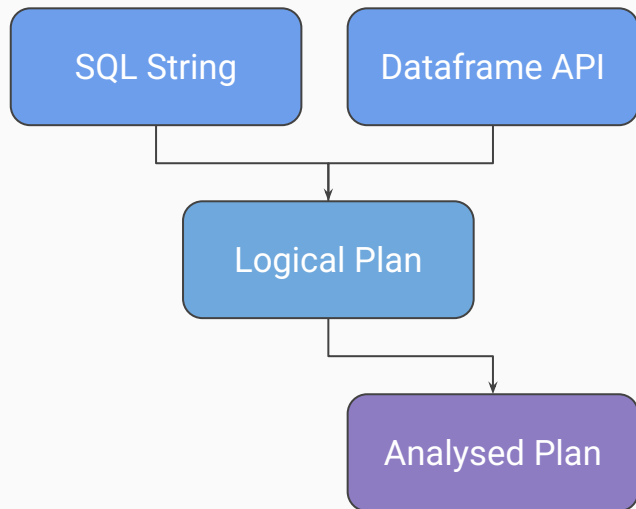
```
Sort [temperature#6 DESC NULLS LAST], true
+- Project [name#1, timestamp#5, temperature#6]
   +- Filter (temperature#6 > cast(30.0 as double))
      +- Join Inner, (id#0 = station_id#4)
         :- SubqueryAlias s
         :  +- SubqueryAlias compiletime.default.weather_stations
         :     +- RelationV2[id#0, name#1, latitude#2, longitude#3] wea
         +- SubqueryAlias r
            +- SubqueryAlias compiletime.default.weather_readings
               +- RelationV2[station_id#4, timestamp#5, ...] weather_re
```
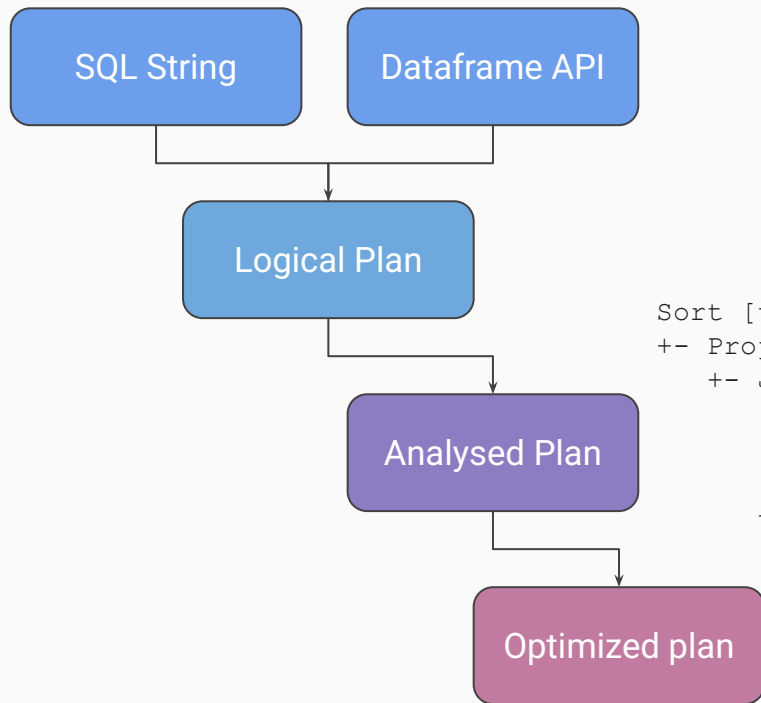
```
Sort [temperature#27 DESC NULLS LAST], true
+- Project [name#11, timestamp#28, temperature#27]
   +- Join Inner, (id#8L = station_id#26L)
      :- Project [id#8L, name#11]
      :  +- Filter isnotnull(id#8L)
      :     +- Relation [id#8L,latitude#9,longitude#10,name#11] json
      +- Project [station_id#26L, temperature#27, timestamp#28]
         +- Filter ((isnotnull(temperature#27) AND (temperature#27 > 30
                    AND isnotnull(station_id#26L))
            +- Relation [humidity#24,pressure#25,station_id#26L,tempera
                       timestamp#28] json
```

```
┌──────────────┐  ┌──────────────┐
│  SQL String  │  │ Dataframe API│
└──────┬───────┘  └──────┬───────┘
       └──────────┬──────┘
                  ▼
         ┌──────────────┐
         │ Logical Plan │
         └──────┬───────┘
                │
                ▼
         ┌──────────────┐
         │ Analysed Plan│
         └──────┬───────┘
                │
                ▼
         ┌──────────────┐
         │ Optimized plan│
         └──────┬───────┘
                │
                ▼
         ┌──────────────┐
         │ Physical Plan│
         └──────────────┘
```

BRACE YOURSELF

WALL OF TEXT IS COMMING

imgflip.com
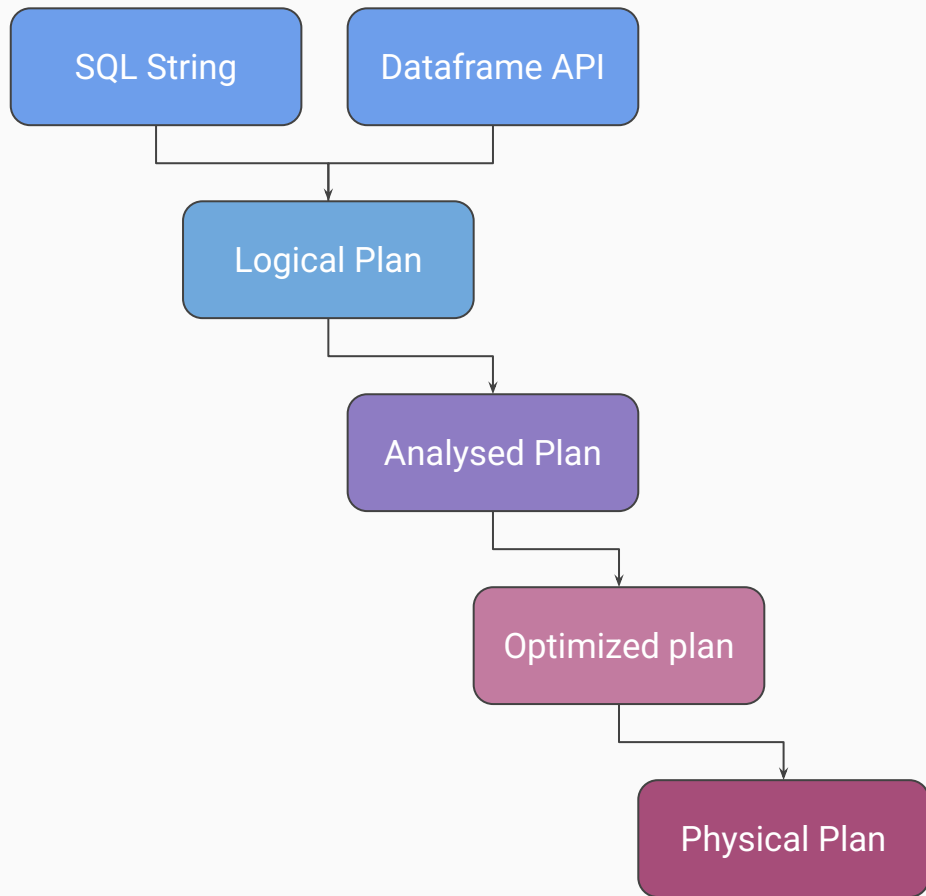
```
AdaptiveSparkPlan isFinalPlan=false
+- Sort [temperature#27 DESC NULLS LAST], true, 0
   +- Exchange rangepartitioning(temperature#27 DESC NULLS LAST, 200), ...
      +- Project [name#11, timestamp#28, temperature#27]
         +- BroadcastHashJoin [id#8L], [station_id#26L], Inner, BuildLeft, false
            :- BroadcastExchange [params…]
            :  +- Filter isnotnull(id#8L)
            :     +- FileScan json [id#8L,name#11]
            :                 Batched: false,
            :                 DataFilters: [isnotnull(id#8L)],
            :                 Format: JSON, Location: InMemoryFileIndex(1 paths)[file],
            :                 PartitionFilters: [],
            :                 PushedFilters: [IsNotNull(id)],
            :                 ReadSchema: struct<id:bigint,name:string>
            +- Filter (filters...)
               +- FileScan json [station_id#26L,temperature#27,timestamp#28]
                           Batched: false,
                           DataFilters: [filters],
                           Format: JSON,
                           Location: InMemoryFileIndex(1 paths)[file],
                           PartitionFilters: [],
                           PushedFilters: [filters],
                           ReadSchema: struct<station_id:bigint,temperature:double,timestamp:string>
```
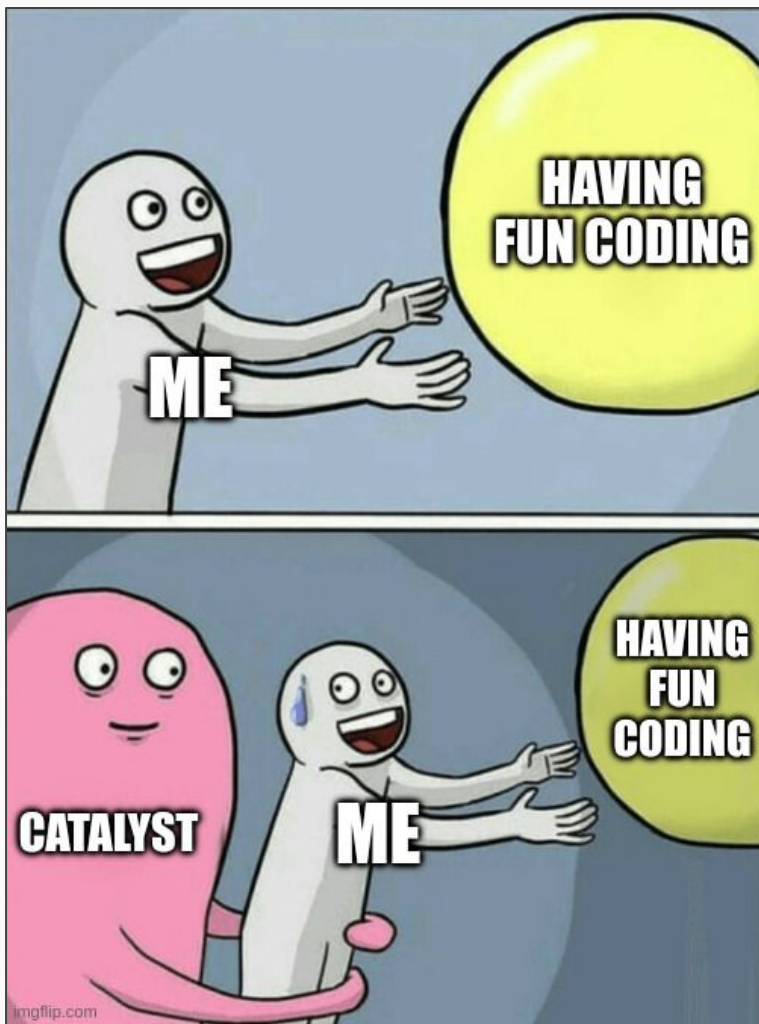
Spark SQL

Catalyst is just a compiler

Spark SQL

Catalyst is just a compiler *in a runtime*

```scala
val df = spark.sql:
  """
  SELECT s.name, r.timestamp r.temperature
  FROM weather_stations s
  JOIN weather_readings r ON s.id = r.station_id
  WHERE r.temperature > 30.0
  ORDER BY r.temperature DESC
  """
```

Strongly typed pipelines ?

```
[PARSE_SYNTAX_ERROR] Syntax error at or near '.'.(line 2, pos 30)


== SQL ==


 SELECT s.name, r.timestamp r.temperature
-----------------------------^^^
 FROM weather_stations s
 JOIN weather_readings r ON s.id = r.station_id
 WHERE r.temperature > 30.0
 ORDER BY r.temperature DESC
```

Strongly typed pipelines ?

```
stations
    .as("s")
    .join(readings.as("r"), stations.col("id") === readings.col("station_id"), "inner")
    .where(col("r.temperature") > 30.0)
    .select(
      col("s.name").as("name"),
      col("r.timestamp").as("timestamp"),
      col("r.temprature").as("temperature")
    )
    .orderBy(col("r.temperature").desc)
```

Strongly typed pipelines ?

```
[UNRESOLVED_COLUMN.WITH_SUGGESTION] A column or function parameter with name `r`.`temprature` cannot be
resolved. Did you mean one of the following? [`r`.`temperature`, `r`.`pressure`, `r`.`humidity`,
`s`.`latitude`, `s`.`longitude`].;
'Project [name#11 AS name#61, timestamp#28 AS timestamp#62, 'r.temprature AS temperature#63]
 +- Filter (temperature#27 > 30.0)
    +- Join Inner, (id#8L = station_id#26L)
       :- SubqueryAlias s
       :  +- SubqueryAlias weather_stations
       :     +- View (`weather_stations`, [id#8L,latitude#9,longitude#10,name#11])
       :        +- Relation [id#8L,latitude#9,longitude#10,name#11] json
       +- SubqueryAlias r
          +- SubqueryAlias weather_readings
             +- View (`weather_readings`, [...])
                +- Relation [humidity#24,pressure#25,station_id#26L,temperature#27,timestamp#28] json
```

Stringly typed pipelines !

The issue with Spark SQL

- Scala add **little to no value** in term of correctness

- Scala compiler do not understand table and rows

- Catalyst is fantastic but deferred until runtime (or tests)

Why did pyspark succeed ?

```python
from pyspark.sql.functions import col

stations = spark.table("weather_stations")
readings = spark.table("weather_readings")

df = (
    stations.alias("s")
    .join(
        readings.alias("r"),
        col("s.id") == col("r.station_id"),
        "inner"
    )
    .where(col("r.temperature") > 30.0)
    .select(
        col("s.name").alias("name"),
        col("r.timestamp").alias("timestamp"),
        col("r.temperature").alias("temperature")
    )
    .orderBy(col("r.temperature").desc())
)
```

```scala
import org.apache.spark.sql.functions.*

val stations = spark.table("weather_stations")
val readings = spark.table("weather_readings")

val df =
  stations.as("s")
  .join(
      readings.as("r"),
      stations.col("id") === readings.col("station_id"),
      "inner"
  )
  .where(col("r.temperature") > 30.0)
  .select(
    col("s.name").as("name"),
    col("r.timestamp").as("timestamp"),
    col("r.temprature").as("temperature")
  )
  .orderBy(col("r.temperature").desc)
```

What if there was a better way ?

Strongly typed pipelines ?

```scala
object Stations extends Table:
  val id        = col("id")
  val name      = col("name")
  val latitude  = col("latitude")
  val longitude = col("longitude")

object Readings extends Table:
  val station_id  = col("station_id")
  val timestamp   = col("timestamp")
  val temperature = col("temprature")
  val humidity    = col("humidity")
  val pressure    = col("pressure")
```

```scala
Stations
  .join(Readings, Stations.id === Readings.station_id, "inner")
  .where(Readings.temperature > 30.0)
  .select(
    Stations.name,
    Readings.timestamp,
    Readings.temperature
  )
  .orderBy(Readings.temperature.desc)
```

Strongly typed pipelines ?

```
[UNRESOLVED_COLUMN.WITH_SUGGESTION] A column or function parameter with name `r`.`temprature` cannot be
resolved. Did you mean one of the following? [`r`.`temperature`, `r`.`pressure`, `r`.`humidity`,
`s`.`latitude`, `s`.`longitude`].;
'Project [name#11 AS name#61, timestamp#28 AS timestamp#62, 'r.temprature AS temperature#63]
 +- Filter (temperature#27 > 30.0)
    +- Join Inner, (id#8L = station_id#26L)
       :- SubqueryAlias s
       :  +- SubqueryAlias weather_stations
       :     +- View (`weather_stations`, [id#8L,latitude#9,longitude#10,name#11])
       :        +- Relation [id#8L,latitude#9,longitude#10,name#11] json
       +- SubqueryAlias r
          +- SubqueryAlias weather_readings
             +- View (`weather_readings`, [...])
                +- Relation [humidity#24,pressure#25,station_id#26L,temperature#27,timestamp#28] json
```

What if there was a better way ? (season 2)

typelevel/**frameless**

Expressive types for Spark.

👥 54
Contributors

⊙ 37
Issues

⭐ 889
Stars

⑂ 137
Forks

Frameless

```scala
case class WeatherStation(id: Int, name: String, latitude: Double, longitude: Double)
case class WeatherReading(station_id: Int, timestamp: java.sql.Timestamp, temperature: Double, humidity: Double, pressure: Double)

implicit val stationEnc: Encoder[WeatherStation] = Encoders.product
implicit val readingEnc: Encoder[WeatherReading] = Encoders.product

spark.read.json("./stations.json").createTempView("weather_stations")
spark.read.json("./readings.json").createTempView("weather_readings")

val stations = TypedDataset.create(spark.table("weather_stations").as[WeatherStation])
val readings = TypedDataset.create(spark.table("weather_readings").as[WeatherReading])

@nowarn
val joined = stations.joinInner(readings)(stations('id) === readings('station_id))

@nowarn
val selected = joined
  .filter(joined('_2).field('temperature) > 30.0)
  .select(
    joined('_1).field('name),
    joined('_2).field('timestamp),
    joined('_2).field('temperature)
  )

@nowarn
val df = selected.orderBy(selected('_3).desc)
```

Frameless

```scala
case class WeatherStation(id: Int, name: String, latitude: Double, longitude: Double)
case class WeatherReading(station_id: Int, timestamp: java.sql.Timestamp, temperature: Double, humidity: Double, pressure: Double)

implicit val stationEnc: Encoder[WeatherStation] = Encoders.product
implicit val readingEnc: Encoder[WeatherReading] = Encoders.product

spark.read.json("./stations.json").createTempView("weather_stations")
spark.read.json("./readings.json").createTempView("weather_readings")

val stations = TypedDataset.create(spark.table("weather_stations").as[WeatherStation])
val readings = TypedDataset.create(spark.table("weather_readings").as[WeatherReading])

@nowarn
val joined = stations.joinInner(readings)(stations('id) === readings('station_id))

@nowarn
val selected = joined
  .filter(joined('_2).field('temperature) > 30.0)
  .select(
    joined('_1).field('name),
    joined('_2).field('timestamp),
    joined('_2).field('temperature)
  )

@nowarn
val df = selected.orderBy(selected('_3).desc)
```

⚠️ Uses Shapeless Witnesses

Frameless

```scala
case class WeatherStation(id: Int, name: String, latitude: Double, longitude: Double)
case class WeatherReading(station_id: Int, timestamp: java.sql.Timestamp, temperature: Double, humidity: Double, pressure: Double)

implicit val stationEnc: Encoder[WeatherStation] = Encoders.product
implicit val readingEnc: Encoder[WeatherReading] = Encoders.product

spark.read.json("./stations.json").createTempView("weather_stations")
spark.read.json("./readings.json").createTempView("weather_readings")

val stations = TypedDataset.create(spark.table("weather_stations").as[WeatherStation])
val readings = TypedDataset.create(spark.table("weather_readings").as[WeatherReading])

@nowarn
val joined = stations.joinInner(readings)(stations('id) === readings('station_id))

@nowarn
val selected = joined
  .filter(joined('_2).field('temperature) > 30.0)
  .select(
    joined('_1).field('name),
    joined('_2).field('timestamp),
    joined('_2).field('temperature)
  )

@nowarn
val df = selected.orderBy(selected('_3).desc)
```

⚠️ Uses Shapeless Witnesses

⚠️ Quoted Symbols Deprecation in 2.13

Frameless

```scala
case class WeatherStation(id: Int, name: String, latitude: Double, longitude: Double)
case class WeatherReading(station_id: Int, timestamp: java.sql.Timestamp, temperature: Double, humidity: Double, pressure: Double)

implicit val stationEnc: Encoder[WeatherStation] = Encoders.product
implicit val readingEnc: Encoder[WeatherReading] = Encoders.product

spark.read.json("./stations.json").createTempView("weather_stations")
spark.read.json("./readings.json").createTempView("weather_readings")

val stations = TypedDataset.create(spark.table("weather_stations").as[WeatherStation])
val readings = TypedDataset.create(spark.table("weather_readings").as[WeatherReading])

@nowarn
val joined = stations.joinInner(readings)(stations('id) === readings('station_id))

@nowarn
val selected = joined
  .filter(joined('_2).field('temperature) > 30.0)
  .select(
    joined('_1).field('name),
    joined('_2).field('timestamp),
    joined('_2).field('temperature)
  )

@nowarn
val df = selected.orderBy(selected('_3).desc)
```

⚠️ Uses Shapeless Witnesses

⚠️ Quoted Symbols Deprecation in 2.13

▌ No column Symbol with shapeless.tag.Tagged[String("temprature")] of type V in Playground.App.WeatherReading

What if there was a better way ? (season 3)

VirtusLab/**iskra**

Typesafe wrapper for Apache Spark DataFrame API

3 Contributors 7 Issues 141 Stars 9 Forks

```scala
// no iska encoder defined for java.sql.Timestamp

 case class WeatherStation(id: Int, name: String, latitude: Double, longitude: Double)
 case class WeatherReading(station_id: Int, timestamp: Long, temperature: Double, humidity: Double, pressure:
Double)

 spark.read.json("./stations.json").createTempView("weather_stations")
 spark.read.json("./readings.json").createTempView("weather_readings")

 val stations = spark.table("weather_stations").typed[WeatherStation]
 val readings = spark.table("weather_readings").typed[WeatherReading]

 val df = stations
   .innerJoin(readings)
   .on($.stations.id === $.readings.station_id)
   .where($.readings.temperature > lit(30.0))
   .select(
     $.stations.name,
     $.readings.timestamp,
     $.readings.temperature
   )
 // .orderBy($.readings.temperature.desc)
```

Iskra

```scala
// no iska encoder defined for java.sql.Timestamp

 case class WeatherStation(id: Int, name: String, latitude: Double, longitude: Double)
 case class WeatherReading(station_id: Int, timestamp: Long, temperature: Double, humidity: Double, pressure:
Double)

 spark.read.json("./stations.json").createTempView("weather_stations")
 spark.read.json("./readings.json").createTempView("weather_readings")

 val stations = spark.table("weather_stations").typed[WeatherStation]
 val readings = spark.table("weather_readings").typed[WeatherReading]

 val df = stations
   .innerJoin(readings)
   .on($.stations.id === $.readings.station_id)
   .where($.readings.temperature > lit(30.0))
   .select(
     $.stations.name,
     $.readings.timestamp,
     $.readings.temperature
   )
 // .orderBy($.readings.temperature.desc)
```

⚠️ Some rough edges

Iskra

```scala
// no iska encoder defined for java.sql.Timestamp

 case class WeatherStation(id: Int, name: String, latitude: Double, longitude: Double)
 case class WeatherReading(station_id: Int, timestamp: Long, temperature: Double, humidity: Double, pressure:
Double)

 spark.read.json("./stations.json").createTempView("weather_stations")
 spark.read.json("./readings.json").createTempView("weather_readings")

 val stations = spark.table("weather_stations").typed[WeatherStation]
 val readings = spark.table("weather_readings").typed[WeatherReading]

 val df = stations
   .innerJoin(readings)
   .on($.stations.id === $.readings.station_id)
   .where($.readings.temperature > lit(30.0))
   .select(
     $.stations.name,
     $.readings.timestamp,
     $.readings.temperature
   )
 // .orderBy($.readings.temperature.desc)
```
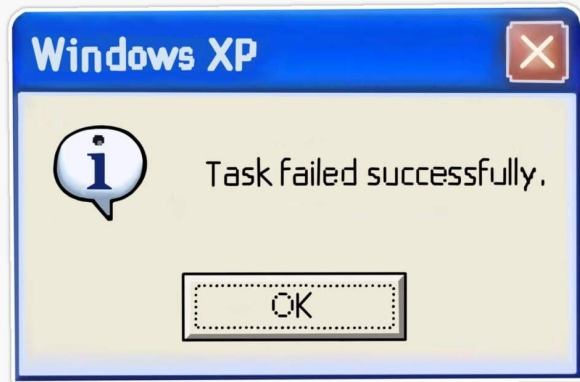
⚠ Some rough edges

⚠ Use its own Codecs

Brute forcing the problem

- High migration costs

- Everything needs to be modeled

- The ORM syndrome : another tool to learn

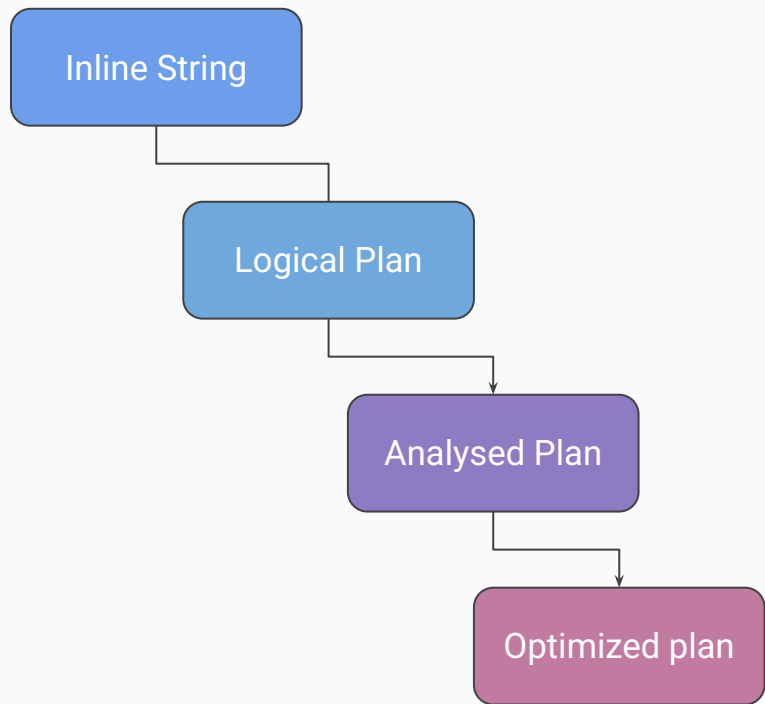What if there was a better way ? (season 4)

Let's put spark into the compiler !

And then ?

Live demo

Catalyst *itself* is parsing and optimizing the query

How does it work ?

```
┌─────────────────┐
│  Inline String  │
└─────────────────┘
        │
        ▼
┌─────────────────┐
│  Logical Plan   │
└─────────────────┘
        │
        ▼
┌─────────────────┐
│  Analysed Plan  │
└─────────────────┘
        │
        ▼
┌─────────────────┐
│ Optimized plan  │
└─────────────────┘
```

- Type level catalog using mirrors

- Table mirrors are derived from their creation query

- Macros runs lightweight catalog (no SparkSession)

A Typelevel Catalog

```scala
val stations: TableMirror {
  type DB = "default"
  type Name = "weather_stations"
  type Schema = "id STRING,name STRING,lat DOUBLE,lon DOUBLE"
  type Query = "<query>"
}

val readings: TableMirror {
  type DB = "default";
  type Name = "weather_readings";
  type Schema = "station_id STRING,timestamp STRING, [...]";
  type Query = "<query>"
}

val weatherDB: CatalogMirror {type Tables = (stations.type, readings.type) }
```

```scala
trait Query


trait Parsed


inline def parse(inline query: Query): Parsed = new Parsed {}


def run(parsed: Parsed): Unit = ()
```

Why tables mirrors

```scala
inline def query(inline str: String): Query = new Query {}

inline def queryButErrored(inline str: String): Query =
  compiletime.error("This is a compile-time error example.")

inline def success = query("some text")

inline def errored = queryButErrored("some text")
```

```scala
inline def parsedSuccess = parse(success)

inline def parsedErrored = parse(errored)
```

Why tables mirrors

```scala
trait TableMirror:
  type DB <: String
  type Name <: String
  type Schema <: String
  type Query <: String

  inline def db: String          = TableMirror.db[this.type]
  inline def name: String        = TableMirror.name[this.type]
  inline def schema: StructType   = TableMirror.schema[this.type]
  inline def schemaString: String = TableMirror.schemaString[this.type]
  inline def query: String        = TableMirror.query[this.type]
```

```scala
object TableMirror:

  inline def query[T]: String =
    ${ macros.queryImpl[T] }

// other implementations
```

Table mirrors : to type matching

```scala
def queryImpl[T: Type](using Quotes): Expr[String] =
 import quotes.reflect.*
 Type.of[T] match
   case '[TableMirror { type Query = query & String }] =>
     Expr(utils.stringFromType[query])
```

```scala
def stringFromType[T: Type](using Quotes): String =
 import quotes.reflect.*
 TypeRepr.of[T] match
   case ConstantType(StringConstant(label)) => label
   case _                                    =>
     report.errorAndAbort(s"expected a constant string, got ${TypeRepr.of[T]}")
```

```scala
def typeFromString(name: String)(using Quotes): Type[?] =
  import quotes.reflect.*
  ConstantType(StringConstant(name)).asType
```

Table mirrors : materialization

```scala
val dbType     = utils.typeFromString(tableName.namespace().mkString("."))
val nameType   = utils.typeFromString(tableName.name())
val schemaType = utils.typeFromString(tableSchema.toDDL)
val queryType  = utils.typeFromString(sql)


(dbType, nameType, schemaType, queryType) match
  case ('[db], '[name], '[schema], '[query]) =>
    '{
        new TableMirror {
            type DB     = db      & String
            type Name   = name    & String
            type Schema = schema & String
            type Query  = query   & String
        }
    }
  case unreachable                            =>
    report.errorAndAbort(s"Unexpected types: $unreachable")
```

- Table mirrors are only extractable from SQL statement
  - Other sources could be used : avrodl, DB schemas, Hive catalogs

- No concatenation / interpolators

- API is rough while interacting with datasets / dataframes

Conclusion

- Embrace SQL as your relational data manipulation DSL

- Scala macros are super fun to use !

- Scala shines at making compiler extensions
  - ideas: Apache calcite, datalog, or more (CoQ?)

Thank you !

https://github.com/vbergeron/conferences