SEQUEL EXTRACT

LET US EXPRESS YOUR SQL& EXTRACT THE OUTPUT

OUTLINE

- Motivation
- UML Architecture
- Input Example
- Technologies
- Limitations
- Demo
- Overview
- Team Information

MOTIVATION

PROBLEM

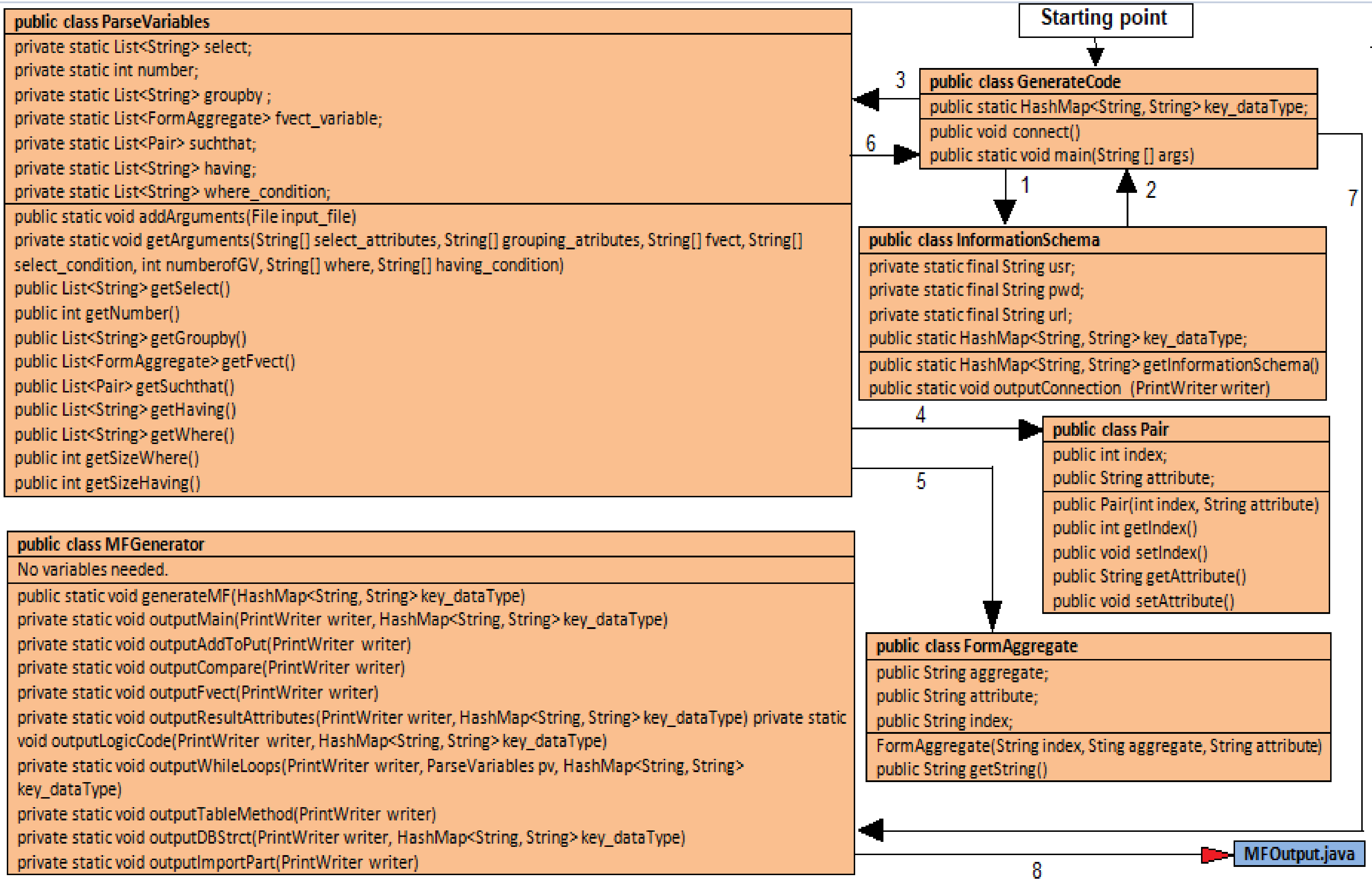
- In SQL, when we try to express Ad-hoc OLAP queries (even the simplest types) also known as multi-dimensional queries, often lead us to complex relational algebraic expressions with multi-joins, group-bys and sub-queries.
- When these queries are optimized by optimizers, they try to optimize a series of joins and group-bys for those queries to make it simpler, leading to poor performance.
- To summarize the problem, we can say that SQL is not potentially an expressive language.

SOLUTION

- If expression is written in succinct manner evaluation can be done efficiently and therefore, we are adding an adjusting feature calling it to be an ESQL, which is more succinct.
- ESQL will provide a syntactic framework by extending the group-by statement and adding the new clause, "***such that***" , and a new operator called ***"phi"*** , and in turn, provide a simple, efficient and scalable algorithm to process the queries, and thereby avoiding multiple sub-queries and joins.
- Also to notice, ESQL is more expressive/succinct than SQL, thus we cannot say its more powerful. The reason behind it is, both of them gives the same output.

UML **ARCHITECTURE**

UML ARCHITECTURE(MF)



```
public class ParseVariables
private static List<String> select;
private static int number;
private static List<String> groupby ;
private static List<FormAggregate> fvect_variable;
private static List<Pair> suchthat;
private static List<String> having;
private static List<String> where_condition;
public static void addArguments(File input_file)
private static void getArguments(String[] select_attributes, String[] grouping_atributes, String[] fvect, String[]
select_condition, int numberofGV, String[] where, String[] having_condition)
public List<String> getSelect()
public int getNumber()
public List<String> getGroupby()
public List<FormAggregate> getFvect()
public List<Pair> getSuchthat()
public List<String> getHaving()
public List<String> getWhere()
public int getSizeWhere()
public int getSizeHaving()
```

```
public class GenerateCode
public static HashMap<String, String> key_dataType;
public void connect()
public static void main(String [] args)
```

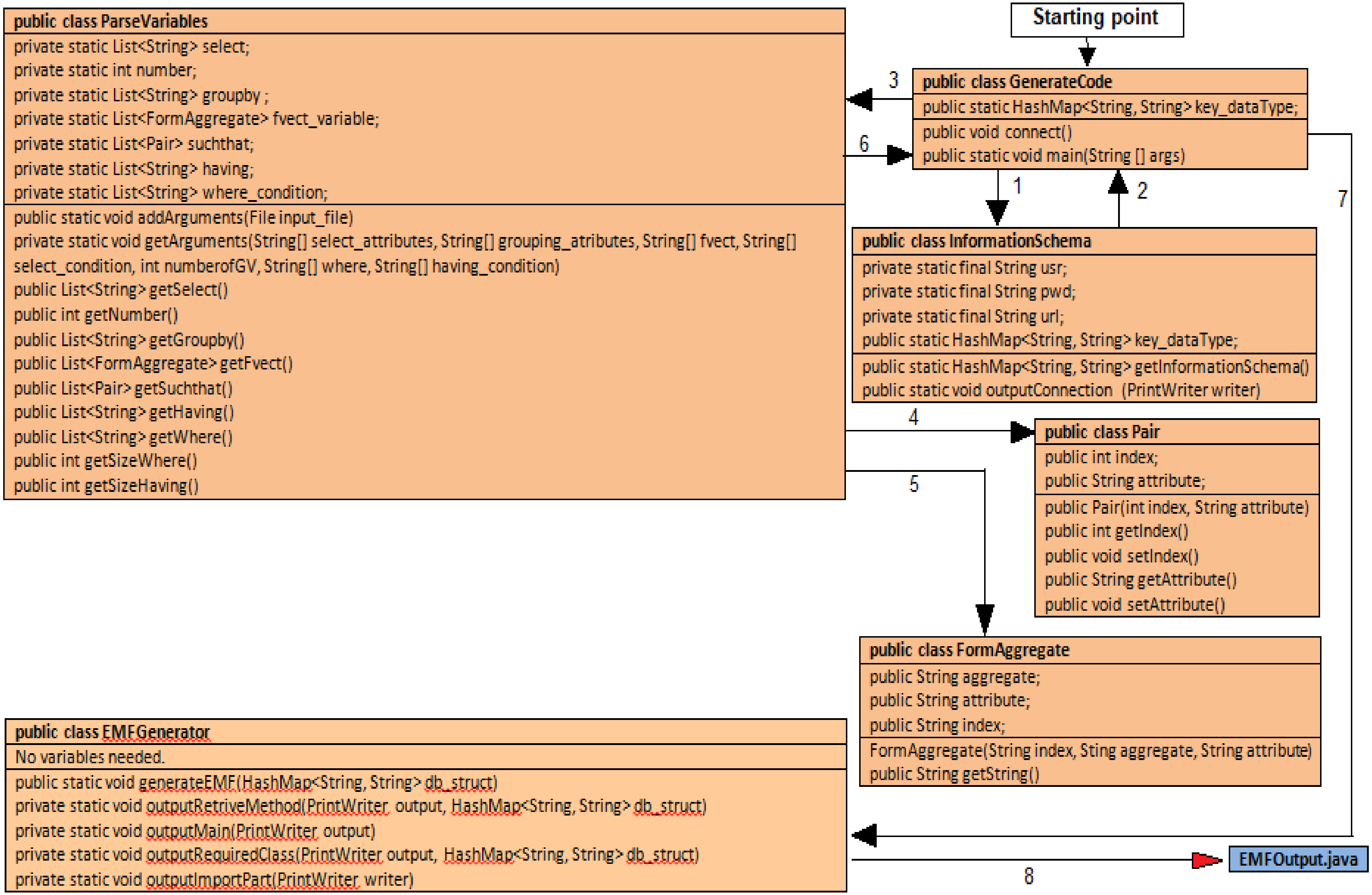
```
public class InformationSchema
private static final String usr;
private static final String pwd;
private static final String url;
public static HashMap<String, String> key_dataType;
public static HashMap<String, String> getInformationSchema()
public static void outputConnection  (PrintWriter writer)
```

```
public class Pair
public int index;
public String attribute;
public Pair(int index, String attribute)
public int getIndex()
public void setIndex()
public String getAttribute()
public void setAttribute()
```

```
public class MFGenerator
No variables needed.
public static void generateMF(HashMap<String, String> key_dataType)
private static void outputMain(PrintWriter writer, HashMap<String, String> key_dataType)
private static void outputAddToPut(PrintWriter  writer)
private static void outputCompare(PrintWriter writer)
private static void outputFvect(PrintWriter writer)
private static void outputResultAttributes(PrintWriter writer, HashMap<String, String> key_dataType) private static
void outputLogicCode(PrintWriter  writer, HashMap<String, String> key_dataType)
private static void outputWhileLoops(PrintWriter writer, ParseVariables pv, HashMap<String, String>
key_dataType)
private static void outputTableMethod(PrintWriter writer)
private static void outputDBStrct(PrintWriter writer, HashMap<String, String> key_dataType)
private static void outputImportPart(PrintWriter  writer)
```

```
public class FormAggregate
public String aggregate;
public String attribute;
public String index;
FormAggregate(String index, Sting aggregate, String attribute)
public String getString()
```

UML ARCHITECTURE(EMF)



```
public class ParseVariables
-----------------------------
private static List<String> select;
private static int number;
private static List<String> groupby ;
private static List<FormAggregate> fvect_variable;
private static List<Pair> suchthat;
private static List<String> having;
private static List<String> where_condition;
-----------------------------
public static void addArguments(File input_file)
private static void getArguments(String[] select_attributes, String[] grouping_atributes, String[] fvect, String[]
select_condition, int numberofGV, String[] where, String[] having_condition)
public List<String> getSelect()
public int getNumber()
public List<String> getGroupby()
public List<FormAggregate> getFvect()
public List<Pair> getSuchthat()
public List<String> getHaving()
public List<String> getWhere()
public int getSizeWhere()
public int getSizeHaving()
```

```
public class GenerateCode
-----------------------------
public static HashMap<String, String> key_dataType;
-----------------------------
public void connect()
public static void main(String [] args)
```

```
public class InformationSchema
-----------------------------
private static final String usr;
private static final String pwd;
private static final String url;
public static HashMap<String, String> key_dataType;
-----------------------------
public static HashMap<String, String> getInformationSchema()
public static void outputConnection  (PrintWriter writer)
```

```
public class Pair
-----------------------------
public int index;
public String attribute;
-----------------------------
public Pair(int index, String attribute)
public int getIndex()
public void setIndex()
public String getAttribute()
public void setAttribute()
```

```
public class FormAggregate
-----------------------------
public String aggregate;
public String attribute;
public String index;
-----------------------------
FormAggregate(String index, Sting aggregate, String attribute)
public String getString()
```

```
public class EMFGenerator
-----------------------------
No variables needed.
-----------------------------
public static void generateEMF(HashMap<String, String> db_struct)
private static void outputRetriveMethod(PrintWriter  output,  HashMap<String, String> db_struct)
private static void outputMain(PrintWriter  output)
private static void outputRequiredClass(PrintWriter output,  HashMap<String, String> db_struct)
private static void outputImportPart(PrintWriter  writer)
```

INPUT EXAMPLE

INPUT EXAMPLE

- The MF/EMF queries for the project will be based on the following schema:

  sales(cust, prod, day, month, year, state, quant)

❖ Let us now have a look at an example:

- **Query statement**: Find for each customer the sum of sales in 'NY', 'NJ' and 'CT' where the average sales for 'NY' is greater than average sales of 'NJ'.
- **ESQL query** : select cust, sum(x.quant), sum(y.quant), sum(z.quant)

```
select cust, sum(x.quant), sum(y.quant), sum(z.quant)
   from sales
   group by cust: x, y, z
   such that  x.state = 'NY'
and  y.state = 'NJ'
and  z.state = 'CT'
   having  avg(x.quant) > avg(y.quant)
```

INPUT EXAMPLE

❖ For our project we are going to use ***"phi"*** operator which will have six parameters. We are also using "**where**" as an operator over here but it is totally optional. All parameters for our query are as under:

1. **SELECT ATTRIBUTES (S)**: cust, 1_sum_quant, 2_sum_quant, 3_sum_quant
2. **Number of grouping variables (n)**: 3
3. **Grouping Attributes (V)**: cust
4. **F-VECT ([F])** : 1_avg_quant,1_sum_quant, 2_avg_quant, 2_sum_quant, 3_sum_quant
5. **SELECT CONDITION-VECT ([σ])** : 1_state="NY", 2_state="NJ", 3_state="CT"
6. **HAVING CONDITION (G)**: avg_quant_1>avg_quant _2
7. **Where (Optional)** : null

TECHNOLOGIES

TECHNOLOGIES

❖ The following technologies are being used in the project:

- **Programming language**: Java
- **DBMS**: PostgreSQL
- **Libraries/Drivers/External Jars** : PostgreSQL JDBC driver
- **Compilers / IDE**: Eclipse, VSCode ,pgadmin 4
- **Information related to JDBC driver**:

1. **URL**: http://jdbc.postgresql.org/download/postgresql-8.3-604.jdbc4.jar
2. **File**: postresql-8.3-604.jdbc.jar
3. **Version**: 8.3-604

LIMITATIONS

LIMITATIONS

- The current program can only take **phi** operator parameters for the moment as input and not the ESQL queries. Also, it reads the parameters from the “file” and not using command-line.
- The current program only works for “and” operators (also called as conjunction operators) and not the “or” (also called as disjunction operators) in having clause, where clause or such- that clause.
- In the having clause we are passing the aggregate functions with the index on the last position and not on the first position like other parameters. For EMF, no having clause have been implemented and some of the inputs uses java.
- No error checking for the existence of tables or columns. The code is not totally dynamic.

DEMO

Let us express your SQL & extract the output!

OVERVIEW

OVERVIEW

❖ We can have a complete overview for the project using following **SWOT** analysis:

Strengths	**Weaknesses**
• Succinct expression and evaluation of queries. • Better optimization and performance.	• Can only read "file" as input. • Not completely dynamic and takes input as "phi" operator parameters. • No dynamic error-checking.
Opportunities	**Threats**
• Can try to improve the input types to query and make the program more dynamic. • Can add more features like disjunction operators.	• Wrong type parameters can break the program. • Absence/unexpected table/tuples or wrong inputs can be a major threat.

TEAM INFORMATION

Name	Work Responsibility
Kunj Desai	• InformationSchema.java (DBMS Connection) • ParseVariables.java (Manipulate phi operators.) • MFGenerator.java (Write into MFOuput file) • Sample queries for MF (MF Input)
Deep Chokshi	• GenerateCode.java (Starting point) • EMFGenerator.java (Write into EMFOutput file). • Sample queries for EMF(EMF Input)

THANK YOU!