

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

# DeeBee

## Implementation of a Relational Database Query-Processing System

Hawk Weisman

Department of Computer Science  
Allegheny College

December 8th, 2014

# Outline

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- 1** Introduction
- 2** Background
  - Relational Databases
  - Query Languages
  - Query Processing
- 3** DeeBee
  - Design
  - Implementation
    - Query Parsing
  - Parsing Demo
  - Query Processing
  - Query Processing Demo

# Introduction

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

## ■ Query processing:

- An application of many concepts from compilers
- Vital to today's world (databases are everywhere)

## ■ DeeBee:

- A very small relational database (<1500 LoC)
- Implements a subset of the Structured Query Language
- For educational purposes only (don't use this in production)
- Written in the Scala programming language

# Introduction

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- Query processing:
  - An application of many concepts from compilers
  - Vital to today's world (databases are everywhere)
- DeeBee:
  - A very small relational database (<1500 LoC)
  - Implements a subset of the Structured Query Language
  - For educational purposes only (don't use this in production)
  - Written in the Scala programming language

# Introduction

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- Query processing:
  - An application of many concepts from compilers
  - Vital to today's world (databases are everywhere)
- DeeBee:
  - A very small relational database (<1500 LoC)
  - Implements a subset of the Structured Query Language
  - For educational purposes only (don't use this in production)
  - Written in the Scala programming language

# Introduction

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- Query processing:
  - An application of many concepts from compilers
  - Vital to today's world (databases are everywhere)
- DeeBee:
  - A very small relational database (<1500 LoC)
  - Implements a subset of the Structured Query Language
  - For educational purposes only (don't use this in production)
  - Written in the Scala programming language

# Introduction

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- Query processing:
  - An application of many concepts from compilers
  - Vital to today's world (databases are everywhere)
- DeeBee:
  - A very small relational database (<1500 LoC)
  - Implements a subset of the Structured Query Language
  - For educational purposes only (don't use this in production)
  - Written in the Scala programming language

# Introduction

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- Query processing:
  - An application of many concepts from compilers
  - Vital to today's world (databases are everywhere)
- DeeBee:
  - A very small relational database (<1500 LoC)
  - Implements a subset of the Structured Query Language
  - For educational purposes only (don't use this in production)
  - Written in the Scala programming language



# Introduction

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- Query processing:
  - An application of many concepts from compilers
  - Vital to today's world (databases are everywhere)
- DeeBee:
  - A very small relational database (<1500 LoC)
  - Implements a subset of the Structured Query Language
  - For educational purposes only (don't use this in production)
  - Written in the Scala programming language

# Introduction

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- Query processing:
  - An application of many concepts from compilers
  - Vital to today's world (databases are everywhere)
- DeeBee:
  - A very small relational database (<1500 LoC)
  - Implements a subset of the Structured Query Language
  - For educational purposes only (don't use this in production)
  - Written in the Scala programming language

# Outline

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

## 1 Introduction

## 2 Background

### ■ Relational Databases

#### ■ Query Languages

#### ■ Query Processing

## 3 DeeBee

### ■ Design

### ■ Implementation

#### ■ Query Parsing

### ■ Parsing Demo

### ■ Query Processing

### ■ Query Processing Demo

# What is a relational database?

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

- “The primary data model for commercial data-processing applications.” [13, page 39]
- A database consists of multiple tables of values, called **relations** [13, 8, 4]
- A relation consists of: [13, 8, 4]
  - a set of rows, or **tuples**
  - a set of columns, or **attributes**
- So how does this relate to compilers?

# What is a relational database?

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

- “The primary data model for commercial data-processing applications.” [13, page 39]
- A database consists of multiple tables of values, called **relations** [13, 8, 4]
- A relation consists of: [13, 8, 4]
  - a set of rows, or **tuples**
  - a set of columns, or **attributes**
- So how does this relate to compilers?

# What is a relational database?

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

- “The primary data model for commercial data-processing applications.” [13, page 39]
- A database consists of multiple tables of values, called **relations** [13, 8, 4]
- A relation consists of: [13, 8, 4]
  - a set of rows, or **tuples**
  - a set of columns, or **attributes**
- So how does this relate to compilers?

# What is a relational database?

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

- “The primary data model for commercial data-processing applications.” [13, page 39]
- A database consists of multiple tables of values, called **relations** [13, 8, 4]
- A relation consists of: [13, 8, 4]
  - a set of rows, or **tuples**
  - a set of columns, or **attributes**
- So how does this relate to compilers?

# What is a relational database?

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

- “The primary data model for commercial data-processing applications.” [13, page 39]
- A database consists of multiple tables of values, called **relations** [13, 8, 4]
- A relation consists of: [13, 8, 4]
  - a set of rows, or **tuples**
  - a set of columns, or **attributes**
- So how does this relate to compilers?



# Outline

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

**Query Languages**

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

## 1 Introduction

## 2 Background

- Relational Databases

- **Query Languages**

- Query Processing

## 3 DeeBee

- Design

- Implementation

  - Query Parsing

- Parsing Demo

- Query Processing

- Query Processing Demo

# What is a query language?

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

- Users and client software interact with databases through **query languages** [13, 8, 4]
- These are **domain-specific languages** for accessing and modifying the database
- Query languages are **declarative** rather than **imperative** [13, 8, 4]
- Just like other programming languages, query languages must be parsed, analyzed, and compiled or interpreted. [13, 8, 4]

# What is a query language?

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

- Users and client software interact with databases through **query languages** [13, 8, 4]
- These are **domain-specific languages** for accessing and modifying the database
- Query languages are **declarative** rather than **imperative** [13, 8, 4]
- Just like other programming languages, query languages must be parsed, analyzed, and compiled or interpreted. [13, 8, 4]

# What is a query language?

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

- Users and client software interact with databases through **query languages** [13, 8, 4]
- These are **domain-specific languages** for accessing and modifying the database
- Query languages are **declarative** rather than **imperative** [13, 8, 4]
- Just like other programming languages, query languages must be parsed, analyzed, and compiled or interpreted. [13, 8, 4]

# What is a query language?

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

- Users and client software interact with databases through **query languages** [13, 8, 4]
- These are **domain-specific languages** for accessing and modifying the database
- Query languages are **declarative** rather than **imperative** [13, 8, 4]
- Just like other programming languages, query languages must be parsed, analyzed, and compiled or interpreted. [13, 8, 4]

# SQL

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

- SQL is the **Structured Query Language**.
- It is the query language used by most modern RDBMSs
- SQL consists of two components:
  - **Data definition language** (DDL): defines the structure of the database [13, 8]
    - creating and deleting tables
    - adding relationships between tables
    - et cetera
  - **Data manipulation language** (DML): accesses and modifies data stored in the database [13, 8]
    - selecting rows
    - adding, deleting, and modifying rows
    - et cetera
- SQL = DDL + DML

# SQL

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

- SQL is the **Structured Query Language**.
- It is the query language used by most modern RDBMSs
- SQL consists of two components:
  - **Data definition language** (DDL): defines the structure of the database [13, 8]
    - creating and deleting tables
    - adding relationships between tables
    - et cetera
  - **Data manipulation language** (DML): accesses and modifies data stored in the database [13, 8]
    - selecting rows
    - adding, deleting, and modifying rows
    - et cetera
- SQL = DDL + DML

# SQL

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

- SQL is the **Structured Query Language**.
- It is the query language used by most modern RDBMSs
- SQL consists of two components:
  - **Data definition language** (DDL): defines the structure of the database [13, 8]
    - creating and deleting tables
    - adding relationships between tables
    - et cetera
  - **Data manipulation language** (DML): accesses and modifies data stored in the database [13, 8]
    - selecting rows
    - adding, deleting, and modifying rows
    - et cetera
- SQL = DDL + DML



# SQL

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

- SQL is the **Structured Query Language**.
- It is the query language used by most modern RDBMSs
- SQL consists of two components:
  - **Data definition language** (DDL): defines the structure of the database [13, 8]
    - creating and deleting tables
    - adding relationships between tables
    - et cetera
  - Data manipulation language (DML): accesses and modifies data stored in the database [13, 8]
    - selecting rows
    - adding, deleting, and modifying rows
    - et cetera
- SQL = DDL + DML

# SQL

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

- SQL is the **Structured Query Language**.
- It is the query language used by most modern RDBMSs
- SQL consists of two components:
  - **Data definition language** (DDL): defines the structure of the database [13, 8]
    - creating and deleting tables
    - adding relationships between tables
    - et cetera
  - **Data manipulation language** (DML): accesses and modifies data stored in the database [13, 8]
    - selecting rows
    - adding, deleting, and modifying rows
    - et cetera
- SQL = DDL + DML

# SQL

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

- SQL is the **Structured Query Language**.
- It is the query language used by most modern RDBMSs
- SQL consists of two components:
  - **Data definition language** (DDL): defines the structure of the database [13, 8]
    - creating and deleting tables
    - adding relationships between tables
    - et cetera
  - **Data manipulation language** (DML): accesses and modifies data stored in the database [13, 8]
    - selecting rows
    - adding, deleting, and modifying rows
    - et cetera

■ SQL = DDL + DML

# SQL

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

- SQL is the **Structured Query Language**.
- It is the query language used by most modern RDBMSs
- SQL consists of two components:
  - **Data definition language** (DDL): defines the structure of the database [13, 8]
    - creating and deleting tables
    - adding relationships between tables
    - et cetera
  - **Data manipulation language** (DML): accesses and modifies data stored in the database [13, 8]
    - selecting rows
    - adding, deleting, and modifying rows
    - et cetera
- SQL = DDL + DML

# SQL Examples

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

## Example (SQL CREATE TABLE statement (schema))

```
CREATE TABLE Writers (  
    id                INTEGER NOT NULL PRIMARY KEY,  
    first_name        VARCHAR(15) NOT NULL,  
    middle_name        VARCHAR(15),  
    last_name         VARCHAR(15) NOT NULL,  
    birth_date        VARCHAR(10) NOT NULL,  
    death_date        VARCHAR(10),  
    country_of_origin VARCHAR(20) NOT NULL  
);
```

# SQL Examples

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

## Example (SQL SELECT statement)

```
SELECT * FROM test;
```

```
SELECT test1, test2 FROM test;
```

```
SELECT * FROM test WHERE test1 = 9 AND test2 = 5;
```

```
SELECT * FROM test LIMIT 5;
```

## Example (SQL DELETE statement)

```
DELETE FROM test WHERE test2 > 3 LIMIT 100;
```

## Example (SQL INSERT statement)

```
INSERT INTO test VALUES (  
    1, 'a string', 2, 'another string'  
);
```

# SQL Examples

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

## Example (SQL SELECT statement)

```
SELECT * FROM test;
```

```
SELECT test1, test2 FROM test;
```

```
SELECT * FROM test WHERE test1 = 9 AND test2 = 5;
```

```
SELECT * FROM test LIMIT 5;
```

## Example (SQL DELETE statement)

```
DELETE FROM test WHERE test2 > 3 LIMIT 100;
```

## Example (SQL INSERT statement)

```
INSERT INTO test VALUES (  
    1, 'a string', 2, 'another string'  
);
```

# SQL Examples

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

## Example (SQL SELECT statement)

```
SELECT * FROM test;
```

```
SELECT test1, test2 FROM test;
```

```
SELECT * FROM test WHERE test1 = 9 AND test2 = 5;
```

```
SELECT * FROM test LIMIT 5;
```

## Example (SQL DELETE statement)

```
DELETE FROM test WHERE test2 > 3 LIMIT 100;
```

## Example (SQL INSERT statement)

```
INSERT INTO test VALUES (  
    1, 'a string', 2, 'another string'  
);
```



# Outline

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

## 1 Introduction

## 2 Background

- Relational Databases
- Query Languages
- Query Processing

## 3 DeeBee

- Design
- Implementation
  - Query Parsing
- Parsing Demo
- Query Processing
- Query Processing Demo

# Steps in Query Evaluation

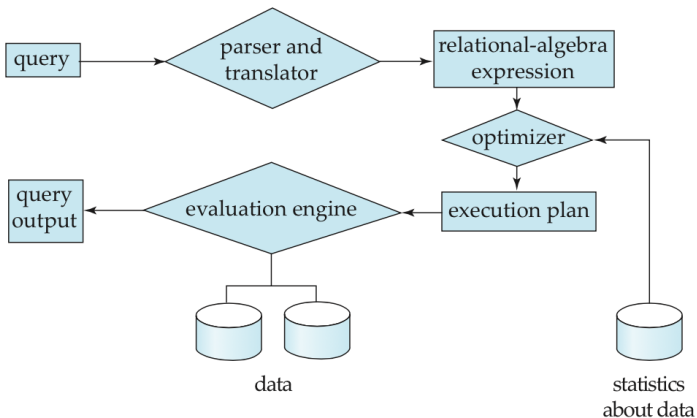


Figure: Steps in query processing [13, 583].

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

# Query Processing

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

- A query processor is essentially a compiler!
- Some stages in the query evaluation process correspond directly to those in compilation:
  - Parsing
  - Semantic analysis
  - IR generation (Relational algebra expression)
  - Optimization

# Query Processing

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

- A query processor is essentially a compiler!
- Some stages in the query evaluation process correspond directly to those in compilation:
  - Parsing
  - Semantic analysis
  - IR generation (Relational algebra expression)
  - Optimization

# Query Processing

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

- A query processor is essentially a compiler!
- Some stages in the query evaluation process correspond directly to those in compilation:
  - Parsing
  - Semantic analysis
  - IR generation (Relational algebra expression)
  - Optimization

# Query Processing

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

- A query processor is essentially a compiler!
- Some stages in the query evaluation process correspond directly to those in compilation:
  - Parsing
  - Semantic analysis
  - IR generation (Relational algebra expression)
  - Optimization

# Query Processing

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

- A query processor is essentially a compiler!
- Some stages in the query evaluation process correspond directly to those in compilation:
  - Parsing
  - Semantic analysis
  - IR generation (Relational algebra expression)
  - Optimization

# Query Processing

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

- A query processor is essentially a compiler!
- Some stages in the query evaluation process correspond directly to those in compilation:
  - Parsing
  - Semantic analysis
  - IR generation (Relational algebra expression)
  - Optimization



# Query Processing

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

- A query processor is essentially a compiler!
- Some stages in the query evaluation process correspond directly to those in compilation:
  - Parsing
  - Semantic analysis
  - IR generation (Relational algebra expression)
  - Optimization

# Outline

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design

Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

## 1 Introduction

## 2 Background

- Relational Databases
- Query Languages
- Query Processing

## 3 DeeBee

- Design
- Implementation
  - Query Parsing
- Parsing Demo
- Query Processing
- Query Processing Demo

# Design Overview

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design

Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- DeeBee implements a subset of SQL
- Chosen to balance functionality with time constraints
  - `SELECT` statements
    - Projections (`SELECT a, b FROM ...`)
    - Filtering by predicates (`SELECT * FROM table WHERE ...`)
    - Nested predicates (`WHERE ... AND ...`)
    - `LIMIT` clauses
    - No `JOINS`
  - `INSERT` statements
  - `DELETE` statements
    - `WHERE` and `LIMIT` clauses
    - Same implementation as `SELECT`
  - `CREATE TABLE` and `DROP TABLE` statements
    - No `CHECK` constraints
    - No `TRIGGERS`

# Design Overview

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design

Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- DeeBee implements a subset of SQL
- Chosen to balance functionality with time constraints
  - `SELECT` statements
    - Projections (`SELECT a, b FROM ...`)
    - Filtering by predicates (`SELECT * FROM table WHERE ...`)
    - Nested predicates (`WHERE ... AND ...`)
    - `LIMIT` clauses
    - No `JOINS`
  - `INSERT` statements
  - `DELETE` statements
    - `WHERE` and `LIMIT` clauses
    - Same implementation as `SELECT`
  - `CREATE TABLE` and `DROP TABLE` statements
    - No `CHECK` constraints
    - No `TRIGGERS`

# Design Overview

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design

Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- DeeBee implements a subset of SQL
- Chosen to balance functionality with time constraints
  - **SELECT statements**
    - Projections (`SELECT a, b FROM ...`)
    - Filtering by predicates (`SELECT * FROM table WHERE ...`)
    - Nested predicates (`WHERE ... AND ...`)
    - `LIMIT` clauses
    - No `JOINS`
  - **INSERT statements**
  - **DELETE statements**
    - `WHERE` and `LIMIT` clauses
    - Same implementation as `SELECT`
  - **CREATE TABLE and DROP TABLE statements**
    - No `CHECK` constraints
    - No `TRIGGERS`

# Design Overview

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design

Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- DeeBee implements a subset of SQL
- Chosen to balance functionality with time constraints
  - **SELECT statements**
    - Projections (`SELECT a, b FROM ...`)
    - Filtering by predicates (`SELECT * FROM table WHERE ...`)
    - Nested predicates (`WHERE ... AND ...`)
    - `LIMIT` clauses
    - No `JOINS`
  - `INSERT` statements
  - `DELETE` statements
    - `WHERE` and `LIMIT` clauses
    - Same implementation as `SELECT`
  - `CREATE TABLE` and `DROP TABLE` statements
    - No `CHECK` constraints
    - No `TRIGGERS`

# Design Overview

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design

Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- DeeBee implements a subset of SQL
- Chosen to balance functionality with time constraints
  - **SELECT statements**
    - Projections (`SELECT a, b FROM ...`)
    - Filtering by predicates (`SELECT * FROM table WHERE ...`)
    - Nested predicates (`WHERE ... AND ...`)
    - `LIMIT` clauses
    - No `JOINS`
  - `INSERT` statements
  - `DELETE` statements
    - `WHERE` and `LIMIT` clauses
    - Same implementation as `SELECT`
  - `CREATE TABLE` and `DROP TABLE` statements
    - No `CHECK` constraints
    - No `TRIGGERS`

# Design Overview

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design

Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- DeeBee implements a subset of SQL
- Chosen to balance functionality with time constraints
  - **SELECT statements**
    - Projections (`SELECT a, b FROM ...`)
    - Filtering by predicates (`SELECT * FROM table WHERE ...`)
    - Nested predicates (`WHERE ... AND ...`)
    - `LIMIT` clauses
    - No JOINS
  - `INSERT` statements
  - `DELETE` statements
    - `WHERE` and `LIMIT` clauses
    - Same implementation as `SELECT`
  - `CREATE TABLE` and `DROP TABLE` statements
    - No CHECK constraints
    - No TRIGGERS



# Design Overview

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- DeeBee implements a subset of SQL
- Chosen to balance functionality with time constraints
  - **SELECT statements**
    - Projections (`SELECT a, b FROM ...`)
    - Filtering by predicates (`SELECT * FROM table WHERE ...`)
    - Nested predicates (`WHERE ... AND ...`)
    - **LIMIT clauses**
    - No JOINS
  - INSERT statements
  - DELETE statements
    - WHERE and LIMIT clauses
    - Same implementation as SELECT
  - CREATE TABLE and DROP TABLE statements
    - No CHECK constraints
    - No TRIGGERS

# Design Overview

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- DeeBee implements a subset of SQL
- Chosen to balance functionality with time constraints
  - **SELECT statements**
    - Projections (`SELECT a, b FROM ...`)
    - Filtering by predicates (`SELECT * FROM table WHERE ...`)
    - Nested predicates (`WHERE ... AND ...`)
    - **LIMIT clauses**
    - **No JOINS**
  - **INSERT statements**
  - **DELETE statements**
    - **WHERE and LIMIT clauses**
    - Same implementation as SELECT
  - **CREATE TABLE and DROP TABLE statements**
    - **No CHECK constraints**
    - **No TRIGGERS**

# Design Overview

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- DeeBee implements a subset of SQL
- Chosen to balance functionality with time constraints
  - **SELECT statements**
    - Projections (`SELECT a, b FROM ...`)
    - Filtering by predicates (`SELECT * FROM table WHERE ...`)
    - Nested predicates (`WHERE ... AND ...`)
    - **LIMIT clauses**
    - **No JOINS**
  - **INSERT statements**
  - **DELETE statements**
    - **WHERE and LIMIT clauses**
    - Same implementation as SELECT
  - **CREATE TABLE and DROP TABLE statements**
    - **No CHECK constraints**
    - **No TRIGGERS**

# Design Overview

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design

Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- DeeBee implements a subset of SQL
- Chosen to balance functionality with time constraints
  - **SELECT statements**
    - Projections (`SELECT a, b FROM ...`)
    - Filtering by predicates (`SELECT * FROM table WHERE ...`)
    - Nested predicates (`WHERE ... AND ...`)
    - **LIMIT clauses**
    - **No JOINS**
  - **INSERT statements**
  - **DELETE statements**
    - WHERE and LIMIT clauses
    - Same implementation as SELECT
  - **CREATE TABLE and DROP TABLE statements**
    - No CHECK constraints
    - No TRIGGERS

# Design Overview

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- DeeBee implements a subset of SQL
- Chosen to balance functionality with time constraints
  - **SELECT statements**
    - Projections (`SELECT a, b FROM ...`)
    - Filtering by predicates (`SELECT * FROM table WHERE ...`)
    - Nested predicates (`WHERE ... AND ...`)
    - **LIMIT clauses**
    - **No JOINS**
  - **INSERT statements**
  - **DELETE statements**
    - **WHERE and LIMIT clauses**
      - Same implementation as SELECT
  - **CREATE TABLE and DROP TABLE statements**
    - No CHECK constraints
    - No TRIGGERS

# Design Overview

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- DeeBee implements a subset of SQL
- Chosen to balance functionality with time constraints
  - **SELECT statements**
    - Projections (`SELECT a, b FROM ...`)
    - Filtering by predicates (`SELECT * FROM table WHERE ...`)
    - Nested predicates (`WHERE ... AND ...`)
    - **LIMIT clauses**
    - **No JOINS**
  - **INSERT statements**
  - **DELETE statements**
    - **WHERE and LIMIT clauses**
    - **Same implementation as SELECT**
  - **CREATE TABLE and DROP TABLE statements**
    - **No CHECK constraints**
    - **No TRIGGERS**

# Design Overview

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- DeeBee implements a subset of SQL
- Chosen to balance functionality with time constraints
  - **SELECT statements**
    - Projections (`SELECT a, b FROM ...`)
    - Filtering by predicates (`SELECT * FROM table WHERE ...`)
    - Nested predicates (`WHERE ... AND ...`)
    - **LIMIT clauses**
    - **No JOINS**
  - **INSERT statements**
  - **DELETE statements**
    - **WHERE and LIMIT clauses**
    - **Same implementation as SELECT**
  - **CREATE TABLE and DROP TABLE statements**
    - **No CHECK constraints**
    - **No TRIGGERS**

# Design Overview

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- DeeBee implements a subset of SQL
- Chosen to balance functionality with time constraints
  - **SELECT statements**
    - Projections (`SELECT a, b FROM ...`)
    - Filtering by predicates (`SELECT * FROM table WHERE ...`)
    - Nested predicates (`WHERE ... AND ...`)
    - **LIMIT clauses**
    - **No JOINS**
  - **INSERT statements**
  - **DELETE statements**
    - **WHERE and LIMIT clauses**
    - **Same implementation as SELECT**
  - **CREATE TABLE and DROP TABLE statements**
    - **No CHECK constraints**
    - **No TRIGGERS**



# Design Overview

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- DeeBee implements a subset of SQL
- Chosen to balance functionality with time constraints
  - **SELECT statements**
    - Projections (`SELECT a, b FROM ...`)
    - Filtering by predicates (`SELECT * FROM table WHERE ...`)
    - Nested predicates (`WHERE ... AND ...`)
    - **LIMIT clauses**
    - **No JOINS**
  - **INSERT statements**
  - **DELETE statements**
    - **WHERE and LIMIT clauses**
    - Same implementation as SELECT
  - **CREATE TABLE and DROP TABLE statements**
    - **No CHECK constraints**
    - **No TRIGGERS**

# Architecture

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design

Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

## ■ The **actors** model [6, 7, 1]

- A construct for concurrent programming
- Actors communicate through **message passing**
- Messages are:
  - Immutable
  - Asynchronous
  - Anonymous (decoupled)
- Actors enqueue recieved messages and respond to them in order

## ■ Essentially, an actor is a **state machine** with a **mailbox**

## ■ Advantages:

- Fault tolerance (loose coupling)
- Scalability
- Concurrency
- Event-driven (good for databases)

## ■ In Scala, the Actors model is provided by the **Akka** framework

# Architecture

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- The **actors** model [6, 7, 1]
  - A construct for concurrent programming
  - Actors communicate through **message passing**
  - Messages are:
    - Immutable
    - Asynchronous
    - Anonymous (decoupled)
  - Actors enqueue recieved messages and respond to them in order
- Essentially, an actor is a **state machine** with a **mailbox**
- Advantages:
  - Fault tolerance (loose coupling)
  - Scalability
  - Concurrency
  - Event-driven (good for databases)
- In Scala, the Actors model is provided by the **Akka** framework

# Architecture

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- The **actors** model [6, 7, 1]
  - A construct for concurrent programming
  - Actors communicate through **message passing**
  - Messages are:
    - Immutable
    - Asynchronous
    - Anonymous (decoupled)
  - Actors enqueue recieved messages and respond to them in order
- Essentially, an actor is a **state machine** with a **mailbox**
- Advantages:
  - Fault tolerance (loose coupling)
  - Scalability
  - Concurrency
  - Event-driven (good for databases)
- In Scala, the Actors model is provided by the **Akka** framework

# Architecture

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- The **actors** model [6, 7, 1]
  - A construct for concurrent programming
  - Actors communicate through **message passing**
  - Messages are:
    - Immutable
    - Asynchronous
    - Anonymous (decoupled)
    - Actors enqueue recieved messages and respond to them in order
- Essentially, an actor is a **state machine** with a **mailbox**
- Advantages:
  - Fault tolerance (loose coupling)
  - Scalability
  - Concurrency
  - Event-driven (good for databases)
- In Scala, the Actors model is provided by the **Akka** framework

# Architecture

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- The **actors** model [6, 7, 1]
  - A construct for concurrent programming
  - Actors communicate through **message passing**
  - Messages are:
    - Immutable
    - Asynchronous
    - Anonymous (decoupled)
    - Actors enqueue recieved messages and respond to them in order
- Essentially, an actor is a **state machine** with a **mailbox**
- Advantages:
  - Fault tolerance (loose coupling)
  - Scalability
  - Concurrency
  - Event-driven (good for databases)
- In Scala, the Actors model is provided by the **Akka** framework

# Architecture

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- The **actors** model [6, 7, 1]
  - A construct for concurrent programming
  - Actors communicate through **message passing**
  - Messages are:
    - Immutable
    - Asynchronous
    - Anonymous (decoupled)
  - Actors enqueue recieved messages and respond to them in order
- Essentially, an actor is a **state machine** with a **mailbox**
- Advantages:
  - Fault tolerance (loose coupling)
  - Scalability
  - Concurrency
  - Event-driven (good for databases)
- In Scala, the Actors model is provided by the **Akka** framework

# Architecture

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- The **actors** model [6, 7, 1]
  - A construct for concurrent programming
  - Actors communicate through **message passing**
  - Messages are:
    - Immutable
    - Asynchronous
    - Anonymous (decoupled)
  - Actors enqueue recieved messages and respond to them in order
- Essentially, an actor is a **state machine** with a **mailbox**
- Advantages:
  - Fault tolerance (loose coupling)
  - Scalability
  - Concurrency
  - Event-driven (good for databases)
- In Scala, the Actors model is provided by the **Akka** framework



# Architecture

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- The **actors** model [6, 7, 1]
  - A construct for concurrent programming
  - Actors communicate through **message passing**
  - Messages are:
    - Immutable
    - Asynchronous
    - Anonymous (decoupled)
  - Actors enqueue recieved messages and respond to them in order
- Essentially, an actor is a **state machine** with a **mailbox**
- Advantages:
  - Fault tolerance (loose coupling)
  - Scalability
  - Concurrency
  - Event-driven (good for databases)
- In Scala, the Actors model is provided by the **Akka** framework

# Architecture

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- The **actors** model [6, 7, 1]
  - A construct for concurrent programming
  - Actors communicate through **message passing**
  - Messages are:
    - Immutable
    - Asynchronous
    - Anonymous (decoupled)
  - Actors enqueue recieved messages and respond to them in order
- Essentially, an actor is a **state machine** with a **mailbox**
- Advantages:
  - Fault tolerance (loose coupling)
  - Scalability
  - Concurrency
  - Event-driven (good for databases)
- In Scala, the Actors model is provided by the **Akka** framework

# Architecture

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- The **actors** model [6, 7, 1]
  - A construct for concurrent programming
  - Actors communicate through **message passing**
  - Messages are:
    - Immutable
    - Asynchronous
    - Anonymous (decoupled)
  - Actors enqueue recieved messages and respond to them in order
- Essentially, an actor is a **state machine** with a **mailbox**
- Advantages:
  - Fault tolerance (loose coupling)
  - Scalability
  - Concurrency
  - Event-driven (good for databases)
- In Scala, the Actors model is provided by the **Akka** framework

# Architecture

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- The **actors** model [6, 7, 1]
  - A construct for concurrent programming
  - Actors communicate through **message passing**
  - Messages are:
    - Immutable
    - Asynchronous
    - Anonymous (decoupled)
  - Actors enqueue recieved messages and respond to them in order
- Essentially, an actor is a **state machine** with a **mailbox**
- Advantages:
  - Fault tolerance (loose coupling)
  - Scalability
  - Concurrency
  - Event-driven (good for databases)
- In Scala, the Actors model is provided by the **Akka** framework

# Architecture

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- The **actors** model [6, 7, 1]
  - A construct for concurrent programming
  - Actors communicate through **message passing**
  - Messages are:
    - Immutable
    - Asynchronous
    - Anonymous (decoupled)
  - Actors enqueue recieved messages and respond to them in order
- Essentially, an actor is a **state machine** with a **mailbox**
- Advantages:
  - Fault tolerance (loose coupling)
  - Scalability
  - Concurrency
  - Event-driven (good for databases)
- In Scala, the Actors model is provided by the **Akka** framework

# Architecture

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- The **actors** model [6, 7, 1]
  - A construct for concurrent programming
  - Actors communicate through **message passing**
  - Messages are:
    - Immutable
    - Asynchronous
    - Anonymous (decoupled)
  - Actors enqueue recieved messages and respond to them in order
- Essentially, an actor is a **state machine** with a **mailbox**
- Advantages:
  - Fault tolerance (loose coupling)
  - Scalability
  - Concurrency
  - Event-driven (good for databases)
- In Scala, the Actors model is provided by the **Akka** framework

# Architecture

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- The **actors** model [6, 7, 1]
  - A construct for concurrent programming
  - Actors communicate through **message passing**
  - Messages are:
    - Immutable
    - Asynchronous
    - Anonymous (decoupled)
  - Actors enqueue recieved messages and respond to them in order
- Essentially, an actor is a **state machine** with a **mailbox**
- Advantages:
  - Fault tolerance (loose coupling)
  - Scalability
  - Concurrency
  - Event-driven (good for databases)
- In Scala, the Actors model is provided by the **Akka** framework

# Architecture

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- The **actors** model [6, 7, 1]
  - A construct for concurrent programming
  - Actors communicate through **message passing**
  - Messages are:
    - Immutable
    - Asynchronous
    - Anonymous (decoupled)
  - Actors enqueue recieved messages and respond to them in order
- Essentially, an actor is a **state machine** with a **mailbox**
- Advantages:
  - Fault tolerance (loose coupling)
  - Scalability
  - Concurrency
  - Event-driven (good for databases)
- In Scala, the Actors model is provided by the **Akka** framework



# Architecture

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

## ■ In DeeBee:

- tables
- databases
- frontends (connections into the database)
- ...are all represented by actors
- A database actor is responsible for:
  - dispatching queries to its' tables
  - sending query results to the querying entity
  - creating and deleting table actors
- A table actor is responsible for:
  - receiving queries
  - (possibly) updating its' state
  - responding with query results or errors

# Architecture

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

- In DeeBee:
  - tables
  - databases
  - frontends (connections into the database)
- ...are all represented by actors
- A database actor is responsible for:
  - dispatching queries to its' tables
  - sending query results to the querying entity
  - creating and deleting table actors
- A table actor is responsible for:
  - receiving queries
  - (possibly) updating its' state
  - responding with query results or errors

# Architecture

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

- In DeeBee:
  - tables
  - databases
  - frontends (connections into the database)
- ...are all represented by actors
- A database actor is responsible for:
  - dispatching queries to its' tables
  - sending query results to the querying entity
  - creating and deleting table actors
- A table actor is responsible for:
  - receiving queries
  - (possibly) updating its' state
  - responding with query results or errors

# Architecture

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

- In DeeBee:
  - tables
  - databases
  - frontends (connections into the database)
- ...are all represented by actors
- A database actor is responsible for:
  - dispatching queries to its' tables
  - sending query results to the querying entity
  - creating and deleting table actors
- A table actor is responsible for:
  - receiving queries
  - (possibly) updating its' state
  - responding with query results or errors

# Architecture

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

- In DeeBee:
  - tables
  - databases
  - frontends (connections into the database)
- ...are all represented by actors
- A database actor is responsible for:
  - dispatching queries to its' tables
  - sending query results to the querying entity
  - creating and deleting table actors
- A table actor is responsible for:
  - receiving queries
  - (possibly) updating its' state
  - responding with query results or errors

# Architecture

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

- In DeeBee:
  - tables
  - databases
  - frontends (connections into the database)
- ...are all represented by actors
- A database actor is responsible for:
  - dispatching queries to its' tables
  - sending query results to the querying entity
  - creating and deleting table actors
- A table actor is responsible for:
  - receiving queries
  - (possibly) updating its' state
  - responding with query results or errors

# Architecture

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

- In DeeBee:
  - tables
  - databases
  - frontends (connections into the database)
- ...are all represented by actors
- A database actor is responsible for:
  - dispatching queries to its' tables
  - sending query results to the querying entity
  - creating and deleting table actors
- A table actor is responsible for:
  - receiving queries
  - (possibly) updating its' state
  - responding with query results or errors

# Architecture

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

- In DeeBee:
  - tables
  - databases
  - frontends (connections into the database)
- ...are all represented by actors
- A database actor is responsible for:
  - dispatching queries to its' tables
  - sending query results to the querying entity
  - creating and deleting table actors
- A table actor is responsible for:
  - receiving queries
  - (possibly) updating its' state
  - responding with query results or errors



# Architecture

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

- In DeeBee:
  - tables
  - databases
  - frontends (connections into the database)
- ...are all represented by actors
- A database actor is responsible for:
  - dispatching queries to its' tables
  - sending query results to the querying entity
  - creating and deleting table actors
- A table actor is responsible for:
  - receiving queries
  - (possibly) updating its' state
  - responding with query results or errors

# Architecture

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

- In DeeBee:
  - tables
  - databases
  - frontends (connections into the database)
- ...are all represented by actors
- A database actor is responsible for:
  - dispatching queries to its' tables
  - sending query results to the querying entity
  - creating and deleting table actors
- A table actor is responsible for:
  - receiving queries
  - (possibly) updating its' state
  - responding with query results or errors

# Architecture

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

- In DeeBee:
  - tables
  - databases
  - frontends (connections into the database)
- ...are all represented by actors
- A database actor is responsible for:
  - dispatching queries to its' tables
  - sending query results to the querying entity
  - creating and deleting table actors
- A table actor is responsible for:
  - receiving queries
  - (possibly) updating its' state
  - responding with query results or errors

# Query Processing

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- SQL queries are internally represented using an **abstract syntax tree** (AST)
- Connection actors receive **query strings**, parse them, and send the AST to the database actor
- Database actor either:
  - processes DDL queries by creating/deleting tables
  - dispatches DML queries to the target child table
- Queries are **interpreted** (not compiled) against a context

# Query Processing

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- SQL queries are internally represented using an **abstract syntax tree** (AST)
- Connection actors receive **query strings**, parse them, and send the AST to the database actor
- Database actor either:
  - processes DDL queries by creating/deleting tables
  - dispatches DML queries to the target child table
- Queries are **interpreted** (not compiled) against a context

# Query Processing

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages  
Query Processing

DeeBee

Design

Implementation

Query Parsing  
Parsing Demo

Query Processing

Query Processing  
Demo

- SQL queries are internally represented using an **abstract syntax tree** (AST)
- Connection actors receive **query strings**, parse them, and send the AST to the database actor
- Database actor either:
  - processes DDL queries by creating/deleting tables
  - dispatches DML queries to the target child table
- Queries are **interpreted** (not compiled) against a context

# Query Processing

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages  
Query Processing

DeeBee

Design

Implementation

Query Parsing  
Parsing Demo

Query Processing

Query Processing  
Demo

- SQL queries are internally represented using an **abstract syntax tree** (AST)
- Connection actors receive **query strings**, parse them, and send the AST to the database actor
- Database actor either:
  - processes DDL queries by creating/deleting tables
  - dispatches DML queries to the target child table
- Queries are **interpreted** (not compiled) against a context

# Query Processing

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages  
Query Processing

DeeBee

Design

Implementation

Query Parsing  
Parsing Demo

Query Processing

Query Processing  
Demo

- SQL queries are internally represented using an **abstract syntax tree** (AST)
- Connection actors receive **query strings**, parse them, and send the AST to the database actor
- Database actor either:
  - processes DDL queries by creating/deleting tables
  - dispatches DML queries to the target child table
- Queries are **interpreted** (not compiled) against a context



# Query Processing

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages  
Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

- SQL queries are internally represented using an **abstract syntax tree** (AST)
- Connection actors receive **query strings**, parse them, and send the AST to the database actor
- Database actor either:
  - processes DDL queries by creating/deleting tables
  - dispatches DML queries to the target child table
- Queries are **interpreted** (not compiled) against a context

# Outline

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

## 1 Introduction

## 2 Background

- Relational Databases
- Query Languages
- Query Processing

## 3 DeeBee

- Design
- Implementation
  - Query Parsing
- Parsing Demo
- Query Processing
- Query Processing Demo

# Parser Combinators

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
**Query Parsing**  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- DeeBee's query processor parses queries using **combinator parsing** [11, 14, 2, 12]
- This is a functional-programming approach to text parsing
  - A **parser** is a function which accepts some strings and rejects others
  - A **parser-combinator** is a higher-order function which takes as input two or more parsers and returns combined parser
  - By repeatedly combining simpler parsers into more complex ones, a **recursive-descent parser** can be created

# Parser Combinators

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
**Query Parsing**  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- DeeBee's query processor parses queries using **combinator parsing** [11, 14, 2, 12]
- This is a functional-programming approach to text parsing
  - A **parser** is a function which accepts some strings and rejects others
  - A **parser-combinator** is a higher-order function which takes as input two or more parsers and returns combined parser
  - By repeatedly combining simpler parsers into more complex ones, a **recursive-descent parser** can be created

# Parser Combinators

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- DeeBee's query processor parses queries using **combinator parsing** [11, 14, 2, 12]
- This is a functional-programming approach to text parsing
  - A **parser** is a function which accepts some strings and rejects others
  - A **parser-combinator** is a higher-order function which takes as input two or more parsers and returns combined parser
  - By repeatedly combining simpler parsers into more complex ones, a **recursive-descent parser** can be created

# Parser Combinators

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- DeeBee's query processor parses queries using **combinator parsing** [11, 14, 2, 12]
- This is a functional-programming approach to text parsing
  - A **parser** is a function which accepts some strings and rejects others
  - A **parser-combinator** is a higher-order function which takes as input two or more parsers and returns combined parser
  - By repeatedly combining simpler parsers into more complex ones, a **recursive-descent parser** can be created

# Parser Combinators

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases

Query Languages

Query Processing

DeeBee

Design

Implementation

Query Parsing

Parsing Demo

Query Processing

Query Processing  
Demo

- DeeBee's query processor parses queries using **combinator parsing** [11, 14, 2, 12]
- This is a functional-programming approach to text parsing
  - A **parser** is a function which accepts some strings and rejects others
  - A **parser-combinator** is a higher-order function which takes as input two or more parsers and returns combined parser
  - By repeatedly combining simpler parsers into more complex ones, a **recursive-descent parser** can be created

# Parser Combinators in Scala

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- Scala's parsing library follows the philosophy of **embedded DSLs** [5, 9, 11, 12]
- It allows parsers to be specified in **BNF-like** syntax

## Example (Combinator Parsing in Scala)

```
def inPlaceConstraint: Parser[Constraint] =  
  ("not" ~ "null") ^^^ Not_Null  
  | ("primary" ~ "key") ^^^ Primary_Key  
  | "unique" ^^^ Unique
```



# Parser Combinators in Scala

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- Scala's parsing library follows the philosophy of **embedded DSLs** [5, 9, 11, 12]
- It allows parsers to be specified in **BNF-like** syntax

## Example (Combinator Parsing in Scala)

```
def inPlaceConstraint: Parser[Constraint] =  
  ("not" ~ "null") ^^^ Not_Null  
  | ("primary" ~ "key") ^^^ Primary_Key  
  | "unique" ^^^ Unique
```

# Parser Combinators in Scala

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Demo

- Scala's parsing library follows the philosophy of **embedded DSLs** [5, 9, 11, 12]
- It allows parsers to be specified in **BNF-like** syntax

## Example (Combinator Parsing in Scala)

```
def inPlaceConstraint: Parser[Constraint] =  
  ("not" ~ "null") ^^^ Not_Null  
  | ("primary" ~ "key") ^^^ Primary_Key  
  | "unique" ^^^ Unique
```

# Packrat Parsing

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- **Packrat parsers** add a memoization facility [10, 3]
  - Guarantees unlimited lookahead and linear parse time
  - Allows parsing of left-recursive grammars
- Parser functions are replaced by lazily-evaluated values

## Example (Combinator Parsing in Scala)

```
lazy val expression: P[Expr[_]] =  
  ("(" ~> comparison <~ ")") ^^ {  
    case c: Comparison => new ParenComparison(c)  
  }  
  | comparison  
  | literal  
  | identifier  
  
lazy val comparison: P[Comparison] =  
  expression ~ operator ~ expression ^^ {  
    case lhs ~ op ~ rhs => Comparison(lhs, op, rhs)  
  }
```

# Packrat Parsing

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- **Packrat parsers** add a memoization facility [10, 3]
  - Guarantees unlimited lookahead and linear parse time
  - Allows parsing of left-recursive grammars
- Parser functions are replaced by lazily-evaluated values

## Example (Combinator Parsing in Scala)

```
lazy val expression: P[Expr[_]] =  
  ("(" ~> comparison <~ ")") ^^ {  
    case c: Comparison => new ParenComparison(c)  
  }  
  | comparison  
  | literal  
  | identifier  
  
lazy val comparison: P[Comparison] =  
  expression ~ operator ~ expression ^^ {  
    case lhs ~ op ~ rhs => Comparison(lhs, op, rhs)  
  }
```

# Packrat Parsing

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- **Packrat parsers** add a memoization facility [10, 3]
  - Guarantees unlimited lookahead and linear parse time
  - Allows parsing of left-recursive grammars
- Parser functions are replaced by lazily-evaluated values

## Example (Combinator Parsing in Scala)

```
lazy val expression: P[Expr[_]] =  
  ("(" ~> comparison <~ ")") ^^ {  
    case c: Comparison => new ParenComparison(c)  
  }  
  | comparison  
  | literal  
  | identifier  
  
lazy val comparison: P[Comparison] =  
  expression ~ operator ~ expression ^^ {  
    case lhs ~ op ~ rhs => Comparison(lhs, op, rhs)  
  }
```

# Packrat Parsing

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- **Packrat parsers** add a memoization facility [10, 3]
  - Guarantees unlimited lookahead and linear parse time
  - Allows parsing of left-recursive grammars
- Parser functions are replaced by lazily-evaluated values

## Example (Combinator Parsing in Scala)

```
lazy val expression: P[Expr[_]] =  
  ("(" ~> comparison <~ ")") ^^ {  
    case c: Comparison => new ParenComparison(c)  
  }  
  | comparison  
  | literal  
  | identifier  
  
lazy val comparison: P[Comparison] =  
  expression ~ operator ~ expression ^^ {  
    case lhs ~ op ~ rhs => Comparison(lhs, op, rhs)  
  }
```

# Packrat Parsing

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- **Packrat parsers** add a memoization facility [10, 3]
  - Guarantees unlimited lookahead and linear parse time
  - Allows parsing of left-recursive grammars
- Parser functions are replaced by lazily-evaluated values

## Example (Combinator Parsing in Scala)

```
lazy val expression: P[Expr[_]] =  
  ("(" ~> comparison <~ ")") ^^ {  
    case c: Comparison => new ParenComparison(c)  
  }  
  | comparison  
  | literal  
  | identifier  
  
lazy val comparison: P[Comparison] =  
  expression ~ operator ~ expression ^^ {  
    case lhs ~ op ~ rhs => Comparison(lhs, op, rhs)  
  }
```

# Outline

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
**Parsing Demo**  
Query Processing  
Query Processing  
Demo

## 1 Introduction

## 2 Background

- Relational Databases
- Query Languages
- Query Processing

## 3 DeeBee

- Design
- Implementation
  - Query Parsing
- **Parsing Demo**
- Query Processing
- Query Processing Demo



# Outline

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
**Query Processing**  
Query Processing  
Demo

## 1 Introduction

## 2 Background

- Relational Databases
- Query Languages
- Query Processing

## 3 DeeBee

- Design
- Implementation
  - Query Parsing
- Parsing Demo
- **Query Processing**
- Query Processing Demo

# Query Processing

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo

Query Processing

Query Processing  
Demo

- DeeBee queries are **interpreted**
- Interpretation is **contextualized** against a database
  - **Type checking**
    - In a compiler, context is preceeding program statements
    - In DBMS, context is the schema of the target table
  - **Predicate interpretation**
    - Convert AST nodes to Scala partial functions
    - Nested predicates are constructed from leaves to roots
  - **Constraints validation**
    - Ensure queries don't violate table constraints
    - Uniqueness
    - Not null
    - Type constraints
    - Eventually, this will be deferrable for transaction processing

# Query Processing

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo

Query Processing

Query Processing  
Demo

- DeeBee queries are **interpreted**
- Interpretation is **contextualized** against a database
  - **Type checking**
    - In a compiler, context is preceeding program statements
    - In DBMS, context is the schema of the target table
  - **Predicate interpretation**
    - Convert AST nodes to Scala partial functions
    - Nested predicates are constructed from leaves to roots
  - **Constraints validation**
    - Ensure queries don't violate table constraints
    - Uniqueness
    - Not null
    - Type constraints
    - Eventually, this will be deferrable for transaction processing

# Query Processing

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo

Query Processing

Query Processing  
Demo

- DeeBee queries are **interpreted**
- Interpretation is **contextualized** against a database
  - **Type checking**
    - In a compiler, context is preceeding program statements
    - In DBMS, context is the schema of the target table
  - **Predicate interpretation**
    - Convert AST nodes to Scala partial functions
    - Nested predicates are constructed from leaves to roots
  - **Constraints validation**
    - Ensure queries don't violate table constraints
    - Uniqueness
    - Not null
    - Type constraints
    - Eventually, this will be deferrable for transaction processing

# Query Processing

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo

Query Processing

Query Processing  
Demo

- DeeBee queries are **interpreted**
- Interpretation is **contextualized** against a database
  - **Type checking**
    - In a compiler, context is preceeding program statements
    - In DBMS, context is the schema of the target table
  - Predicate interpretation
    - Convert AST nodes to Scala partial functions
    - Nested predicates are constructed from leaves to roots
  - **Constraints validation**
    - Ensure queries don't violate table constraints
    - Uniqueness
    - Not null
    - Type constraints
    - Eventually, this will be deferrable for transaction processing

# Query Processing

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- DeeBee queries are **interpreted**
- Interpretation is **contextualized** against a database
  - **Type checking**
    - In a compiler, context is preceeding program statements
    - In DBMS, context is the schema of the target table
  - Predicate interpretation
    - Convert AST nodes to Scala partial functions
    - Nested predicates are constructed from leaves to roots
  - **Constraints validation**
    - Ensure queries don't violate table constraints
    - Uniqueness
    - Not null
    - Type constraints
    - Eventually, this will be deferrable for transaction processing

# Query Processing

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- DeeBee queries are **interpreted**
- Interpretation is **contextualized** against a database
  - **Type checking**
    - In a compiler, context is preceeding program statements
    - In DBMS, context is the schema of the target table
  - **Predicate interpretation**
    - Convert AST nodes to Scala partial functions
    - Nested predicates are constructed from leaves to roots
  - **Constraints validation**
    - Ensure queries don't violate table constraints
    - Uniqueness
    - Not null
    - Type constraints
    - Eventually, this will be deferrable for transaction processing

# Query Processing

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- DeeBee queries are **interpreted**
- Interpretation is **contextualized** against a database
  - **Type checking**
    - In a compiler, context is preceeding program statements
    - In DBMS, context is the schema of the target table
  - **Predicate interpretation**
    - Convert AST nodes to Scala partial functions
      - Nested predicates are constructed from leaves to roots
  - Constraints validation
    - Ensure queries don't violate table constraints
    - Uniqueness
    - Not null
    - Type constraints
    - Eventually, this will be deferrable for transaction processing



# Query Processing

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- DeeBee queries are **interpreted**
- Interpretation is **contextualized** against a database
  - **Type checking**
    - In a compiler, context is preceeding program statements
    - In DBMS, context is the schema of the target table
  - **Predicate interpretation**
    - Convert AST nodes to Scala partial functions
    - Nested predicates are constructed from leaves to roots
  - **Constraints validation**
    - Ensure queries don't violate table constraints
    - Uniqueness
    - Not null
    - Type constraints
    - Eventually, this will be deferrable for transaction processing

# Query Processing

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- DeeBee queries are **interpreted**
- Interpretation is **contextualized** against a database
  - **Type checking**
    - In a compiler, context is preceeding program statements
    - In DBMS, context is the schema of the target table
  - **Predicate interpretation**
    - Convert AST nodes to Scala partial functions
    - Nested predicates are constructed from leaves to roots
  - **Constraints validation**
    - Ensure queries don't violate table constraints
    - Uniqueness
    - Not null
    - Type constraints
    - Eventually, this will be deferrable for transaction processing

# Query Processing

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- DeeBee queries are **interpreted**
- Interpretation is **contextualized** against a database
  - **Type checking**
    - In a compiler, context is preceeding program statements
    - In DBMS, context is the schema of the target table
  - **Predicate interpretation**
    - Convert AST nodes to Scala partial functions
    - Nested predicates are constructed from leaves to roots
  - **Constraints validation**
    - Ensure queries don't violate table constraints
      - Uniqueness
      - Not null
      - Type constraints
      - Eventually, this will be deferrable for transaction processing

# Query Processing

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- DeeBee queries are **interpreted**
- Interpretation is **contextualized** against a database
  - **Type checking**
    - In a compiler, context is preceeding program statements
    - In DBMS, context is the schema of the target table
  - **Predicate interpretation**
    - Convert AST nodes to Scala partial functions
    - Nested predicates are constructed from leaves to roots
  - **Constraints validation**
    - Ensure queries don't violate table constraints
    - Uniqueness
      - Not null
      - Type constraints
      - Eventually, this will be deferrable for transaction processing

# Query Processing

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- DeeBee queries are **interpreted**
- Interpretation is **contextualized** against a database
  - **Type checking**
    - In a compiler, context is preceeding program statements
    - In DBMS, context is the schema of the target table
  - **Predicate interpretation**
    - Convert AST nodes to Scala partial functions
    - Nested predicates are constructed from leaves to roots
  - **Constraints validation**
    - Ensure queries don't violate table constraints
    - Uniqueness
    - Not null
    - Type constraints
    - Eventually, this will be deferrable for transaction processing

# Query Processing

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- DeeBee queries are **interpreted**
- Interpretation is **contextualized** against a database
  - **Type checking**
    - In a compiler, context is preceeding program statements
    - In DBMS, context is the schema of the target table
  - **Predicate interpretation**
    - Convert AST nodes to Scala partial functions
    - Nested predicates are constructed from leaves to roots
  - **Constraints validation**
    - Ensure queries don't violate table constraints
    - Uniqueness
    - Not null
    - Type constraints
    - Eventually, this will be deferrable for transaction processing

# Query Processing

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

- DeeBee queries are **interpreted**
- Interpretation is **contextualized** against a database
  - **Type checking**
    - In a compiler, context is preceeding program statements
    - In DBMS, context is the schema of the target table
  - **Predicate interpretation**
    - Convert AST nodes to Scala partial functions
    - Nested predicates are constructed from leaves to roots
  - **Constraints validation**
    - Ensure queries don't violate table constraints
    - Uniqueness
    - Not null
    - Type constraints
    - Eventually, this will be deferrable for transaction processing

# Outline

DeeBee

Hawk  
Weisman

Introduction

Background

Relational  
Databases  
Query Languages  
Query Processing

DeeBee

Design  
Implementation  
Query Parsing  
Parsing Demo  
Query Processing  
Query Processing  
Demo

## 1 Introduction

## 2 Background

- Relational Databases
- Query Languages
- Query Processing

## 3 DeeBee

- Design
- Implementation
  - Query Parsing
- Parsing Demo
- Query Processing
- Query Processing Demo



# References I

DeeBee

Hawk  
Weisman

Appendix

References

References

- [1] Gul Abdulnabi Agha. Actors: a model of concurrent computation in distributed systems. 1985.
- [2] Jeroen Fokker. Functional parsers. In *Advanced Functional Programming*, pages 1–23. Springer, 1995.
- [3] Richard A Frost, Rahmatullah Hafiz, and Paul Callaghan. Parser combinators for ambiguous left-recursive grammars. In *Practical Aspects of Declarative Languages*, pages 167–181. Springer, 2008.
- [4] Hector Garcia-Molina, Jeffrey D Ullman, and Jennifer Widom. *Database System Implementation*, volume 654. Prentice Hall Upper Saddle River, NJ:, 2000.
- [5] Debasish Ghosh. *DSLs in action*. Manning Publications Co., 2010.

# References II

DeeBee

Hawk  
Weisman

Appendix

References

References

- [6] Munish Gupta. *Akka Essentials*. Packt Publishing Ltd, 2012.
- [7] Philipp Haller. On the integration of the actor model in mainstream technologies: the scala perspective. In *Proceedings of the 2nd edition on Programming systems, languages and applications based on actors, agents, and decentralized control abstractions*, pages 1–6. ACM, 2012.
- [8] Jan L Harrington. *Relational database design and implementation: clearly explained*. Morgan Kaufmann, 2009.

# References III

DeeBee

Hawk  
Weisman

Appendix  
References

References

- [9] Christian Hofer, Klaus Ostermann, Tillmann Rendel, and Adriaan Moors. Polymorphic embedding of DSLs. In *Proceedings of the 7th International Conference on Generative Programming and Component Engineering*, pages 137–148. ACM, 2008.
- [10] Manohar Jonnalagedda, Martin Odersky, and Tiark Rumpf. Packrat Parsing in Scala. Technical report, Ecole Polytechnique Fédérale de Lausanne, 2009.
- [11] Adriaan Moors, Frank Piessens, and Martin Odersky. Parser combinators in Scala. Technical report, Katholieke Universiteit Leuven, 2008.
- [12] Martin Odersky, Lex Spoon, and Bill Venners. *Programming in Scala*. Artima Inc, 2008.
- [13] A. Silberschatz, H. Korth, and S. Sudarshan. *Database System Concepts*. McGraw-Hill Education, 2010.

# References IV

DeeBee

Hawk  
Weisman

Appendix  
References

References

- [14] S Doaitse Swierstra. Combinator parsers: From toys to tools. *Electronic Notes in Theoretical Computer Science*, 41(1):38–59, 2001.