Practicum in Database Systems

Project 1 intro

Immanuel Trummer
itrummer@cornell.edu
Preliminaries

• Make sure you know how to find
  • CMS
  • Piazza
  • The course website (office hours start next week)
What is the course about?

• Work in groups of 2-3 to develop a simple SQL interpreter
• Process SELECT-FROM-WHERE queries
  • A variety of algorithms for operators (joins etc)
  • Query optimization
Projects

• Projects 1-5: develop your interpreter
  • Start from scratch (empty directory)
  • Each project builds on your previous one
    • No solution code
  • My reference implementation of the whole thing is ~5000 LOC
Projects

- Project 1: SQL interpreter on small data
  - in-memory evaluation
- Project 2: Scaling to bigger files
- Project 3: Indexing
- Project 4: Query optimization
- Project 5: Extensions (Multiple Options)
What you should know

• Mastery of Java and data structures at CS 2110 level
  • Afraid of tree traversals or recursion? This is not the class for you.

• Projects 1 and 2 will help you to determine if this class is a good fit for you
What you should know

• How to work independently and how to debug your code
  • We provide extensive instructions but you need to read them and expect to spend a lot of time on this
  • 10 minute debugging policy
  • Project 1: will be starting from empty directory
  • Projects 2-5: build on your own code from previous projects – no solution code, no "bail-outs"
Grading and logistics

• Based on your project submissions
• No exams
Logistics: due dates and meetings

• Projects due dates in CMS
• Every project has an initial meeting where I give a brief overview of project
  • Look at instructions before then and bring your questions
  • See schedule in CMS
• All meetings optional
Logistics: Eclipse

- Projects will require you to submit an Eclipse project (and a runnable command-line .jar)
- You can develop on whatever platform you like, but must submit Eclipse project for grading
Grading

• Every Project is graded mostly on automated tests
  • See Course Web site for relevant info on these
• Also some points for code style/comments
  • See instructions
• And some must-have requirements
  • If you violate these, we can impose an arbitrary deduction
  • Deduction could exceed number of "code style" points
Project 1 Introduction

• Overall architecture:
  • takes as input DB and SQL queries
  • processes each query and outputs answers

• Samples provided

• Essential to follow our input and output format for grading purposes
The life cycle of a query

• Parsing
  • JSqlParser
• Translation to a (bag) relational algebra query plan
• Evaluation
  • and writing result to a file
About the project overall

• Start from empty directory (no skeleton code)
  • except that a .jar with JSqlParser is provided
• Relatively few hard architectural requirements, see instructions
• Will be building on same codebase for Projects 2-5 (no solution code)
• My reference implementation is ~1100 LOC
• Extensive instructions are provided
• Won't be supporting all of SQL
• Only a limited subset, see Section 2.1 in instructions
• Basically SELECT-FROM-WHERE with optional DISTINCT and/or ORDER BY
Relevant info that is/isn't in 4320

• How to evaluate operators
  • We'll come to that soon in 4320, but get started on this project now – don't wait!
  • Project instructions (and textbook) should be enough

• Iterator model for evaluating operators
  • Every operator extends an abstract Operator class
  • Provides getNextTuple() and reset() methods
  • Discussion on how these work and why we use them
How to do the project

- Suggested step-by-step process in written instructions
- Will only discuss "highlights" now
How to do the project

• Become familiar with JSqlParser
• Implement the scan operator
  • Will probably want a DB catalog and assorted other classes (e.g. Tuple) etc
• Selection
  • Will require evaluating an expression on a tuple
  • JSqlParser provides an ExpressionVisitor interface
• Projection
How to do the project

• Join
  • How to translate a query into a RA plan
  • Don't compute cross products!!
  • Consider pushing selections
• Aliases (Sailors S1, Sailors S2 etc)
• ORDER BY
• DISTINCT (using sorting)
Must-have requirements

• Use Operator model
• Build query plan and evaluate query by calling getNextTuple() on root repeatedly
• Have a method to extract join conditions from the WHERE clause