# Final Project: Build a Probabilistic Database System
# CS267A Fall 2018

In this project, you will be implementing a fully functioning inference system for probabilistic databases in a programming language of your choice. This document contains some necessary background, the classes of queries your program should be able to handle, the format databases will be given to you in, as well as our criteria for evaluating your project. We additionally provide some options for ways to extend your system - you will need to do at least one of these for full credit.

Ask on Piazza or email Steven or Tal if you have a question about the project, or are having trouble finding or forming a group:

- Piazza: `http://piazza.com/ucla/fall2018/cs267a`
- Steven: `sholtzen@cs.ucla.edu`
- Tal: `tal@cs.ucla.edu`

**Academic Honesty** You are encouraged to use open-source code and libraries as long as you respect the licenses of those libraries. In particular, be sure to attribute any code that you use from an open-source library appropriately: *do not copy and paste someone else's code without properly citing its original source*.

Please *do not post your code on a publicly accessible repository*. Please use private repositories for your projects. You can use a free Git or Mercurial repository for free from `http://https://bitbucket.org/`.

## 1 Background

For background on probabilistic database semantics, see Sections 2.1-2.3 (pages 9-20) of Van den Broeck and Suciu [2017]. For a detailed explanation of lifted query evaluation, see Sections 4.1-4.3 (pages 65-78) of Van den Broeck and Suciu [2017]. This material will also be covered in class on Week 3.

## 2 The Project

You will be implementing a system for evaluating safe queries on a probabilistic database. That is, given a collection of tables representing a probabilistic database, as well as a *fully quantified* query, your system should be able to efficiently compute the probability of the query.

All written materials for the project should be submitted to CCLE using the provided LaTeX template at `https://github.com/SHoltzen/cs267a-latex-template`. Late submissions *will not be accepted*.

## 3 Technical Specifications

To efficiently perform probabilistic database inference, you will be implementing the lifted inference rules we discussed in class. The slides for this lecture are available online; if you need further reference or details you should consult Chapter 4 of [Van den Broeck and Suciu, 2017]. Other useful, though more terse, resources include [Dalvi and Suciu, 2012, Gribkoff et al., 2014].

Concretely, we want you to write a program (in any programming language) which takes arguments on the command line. There should be two kinds of arguments:

1. *A query file*, which is a text file which contains the query string;
2. *A table file*, which contains a single table of the database.

So, for example, if your application is called `pdb` then we will invoke your code with an argument like:

```
./pdb --query query.txt
  --table t1.txt
  --table t2.txt
  --table t3.txt
```

We will provide you with example query and table files for you to test your code.

## 3.1 Queries

All queries will be given to you as a fully quantified union of conjunctive queries (UCQ). That is, queries will be given to you as a string of the form:

```
R(x1),S(x1,y1) ||
  S(x2, y2), T(x2)
```

form which you should interpret as

$$(\exists x_1 y_1 R(x_1) \wedge S(x_1, y_1)) \vee (\exists x_2 y_2 S(x_2, y_2) \wedge T(x_2))$$

This string will the be sole contents of the query file; it will be your job to parse this query string. Figure 1 shows the grammar for the query string. The query string consists of the following symbols:

- Tables, labeled `<table>` in Figure 1, are denoted with single capital letters, and contain one or more variables as arguments. For example, `R(x1)` is a table.
- Variables, denoted as a string of letters beginning with a single lower case letter. For example, `x1` is a variable.
- Tables may be separated by commas; these denote conjunctions between tables, and are called *conjunctive queries* (called `<conj>` in Figure 1).
- Queries, denoted `<query>`, are unions of conjunctive queries, i.e. conjunctive queries separated by a "||" symbol. Ultimately, the query file will contain a single query rule.

You will then follow the rules for doing lifted inference to process this query and efficiently do inference. You should assume that the query is safe (that is, that the rules will work), and you can further assume that if you at some point need to perform inclusion-exclusion, all subqueries generated by inclusion exclusion will also be safe. If your database can handle all queries of this type, you will get full marks for correctness.

## 3.2 Table Files

The probabilistic database we are interested in querying will be given a series of CSV files called *table files*. Each file will represent a single table, and each line will represent a single tuple. The first line of the file will be the name of the table. Each column of each tuple will be separated by a comma, and the last column is always the probability that the tuple is in the database. For example, for a table $S(x, y)$, the lines of the file might look like this:

```
S
1,1,0.6
2,1,0.4
1,2,0.8
```

## 3.3 Examples

To help you with your project, we provide a few database/query examples that you can use as a sanity check to see if your system is working correctly. Consider tables $P, Q, R$ given by Tables 1, 2, and 3 respectively.

Then for the following queries

$$Q_1 = R(x, y), Q(x)$$

$$Q_2 = R(x_1, y_1), P(x_1), Q(x_2), R(x_2, y_2)$$

```
<conj> ::= <table>
        | <table> "," <conj>
<query> ::= <conj>
        | <conj> "||" <query>
<table> ::= <tname> "(" <varname> ("," <varname> )* ")"
<tname> ::= Single capital letter
<varname> ::= String beginning with a single lower case letter
start = <query>
```

Figure 1: Query syntax, written in EBNF form. See `http://matt.might.net/articles/grammars-bnf-ebnf` for information about EBNF form. The `start` rule specifies the rule to follow when you begin parsing. A $\star$ character means "zero or more instances"; a rule is denoted with angled brackets; and a required character is quoted.

| $x$ | $p$ |
|---|---|
| 0 | 0.7 |
| 1 | 0.8 |
| 2 | 0.6 |

Table 1: Example table $P(x)$

| $x$ | $p$ |
|---|---|
| 0 | 0.7 |
| 1 | 0.3 |
| 2 | 0.5 |

Table 2: Example table $Q(x)$

You should find that $P(Q_1) = 0.8458$, and that $P(Q_2) = 0.7835$.

| $x$ | $y$ | $p$ |
|---|---|---|
| 0 | 0 | 0.8 |
| 0 | 1 | 0.4 |
| 0 | 2 | 0.5 |
| 1 | 2 | 0.6 |
| 2 | 2 | 0.9 |

Table 3: Example table $R(x, y)$

## 4 Evaluation

Your evaluation will include three components: the first two are check-ins evaluated on a pass/no-pass basis and are each worth 10%. The last component is a final report, which will be worth 80% of the project grade.

### 4.1 Extension Proposal and Check-In (Due Fri Nov. 16)

Provide a document on CCLE with the following information:

- A 1 page summary of the extension you intend to build for your project. This should include references to relevant papers, what the problem is you are attempting to solve, and how you intend to solve it. If the extension is experimental in nature, you should also describe experiments you intend to include in your final report.

- Give a progress update. Have you started programming? What roadblocks have you encountered or anticipate? Which features have you implemented?

This will be graded on a pass / no-pass basis.

## 4.2 Final Report + Code (Due Monday Dec. 10)

**Final Report** This final document is a technical report of at most 4 pages[1] not including references or appendices, giving details of your implementation and extension. This document should include the following:

- A detailed description of your extension, what problem it solves, and the technical details of how it works
- If you did not complete all of your goals, explain where you fell short.
- Any interesting or important design decisions or implementation details in your system.
- Any test cases that you ran on your database to validate its results.
- Provide your feedback on the project. What parts did you like, and which parts would you improve?

**Code** Submit all code you wrote, along with instructions describing how to compile and run your system on example queries **on the seasnet linux servers**. If you need an account, visit `https://www.seasnet.ucla.edu/seasnet-accounts/` for instructions. You can access the servers via ssh or in person at BH2664. *Apply for accounts early*; it can take some time for your account to be activated.

#### 4.2.1 Evaluation

We will evaluate your final submission on the following rubric:

- (10%) *Quality of Writing*: Is the information presented clearly and easy to follow? Are there appropriate usage of figures and mathematical language? Are appropriate references cited? Are there significant grammar or spelling errors that cause confusion?
- (35%) *Correctness of Implementation*: We will run a fixed series of queries on some probabilistic databases, to make sure your inference algorithm returns the correct result, in addition to the test cases that we provide you with. We will also evaluate the quality of your test cases.
- (35%) *Technical Quality*: Is your inference method well-explained, so that we can understand what you did? Can we understand key technical decisions you made? Is your database implemented in a sensible way?
- (20%) *Extension*: Is the extension technically sound? If there are experiments, do they make sense and look reasonable? How does the final product compare to what was proposed?

[1]Use the template given at `https://github.com/SHoltzen/cs267a-latex-template`

### 4.3 Invited Presentations

A subset of projects will be selected to give a 10-minute or 20-minute presentation in class on the final day. This is optional, but highly encouraged: it is a chance for you to sharpen your public speaking skills, for the class to learn about your interesting project, and for the instructors to give additional feedback.

## 5 Extensions

Every probabilistic database needs to be able to handle the basics of processing simple queries. In addition to the basics, we require your project to go above and beyond by extending a simple probabilistic database in a non-trivial way. We detail here a few examples of extensions, although you are also free to propose your own.

- Evaluate your database on a *non-trivial* dataset by storing the dataset in your database and evaluating some interesting queries. Ideally, this dataset will require you to provide some interesting features to your database due to its size or structure. Here is an example collection of datasets, but you are encouraged to find your own:

    - Never Ending Language Learner: `http://rtw.ml.cmu.edu/rtw/`

- Enhance your system with the open world semantics [Ceylan et al., 2016]
- Implement an approximation scheme for hard queries
    - Gibbs sampling
    - Markov-Chain Monte Carlo [Wick et al., 2010]
    - Rao-blackwell sampling [Gribkoff and Suciu, 2016]
- In addition to it being correct, make it fast
    - Use a traditional database querying system like PostgreSQL and SQL to store your database.
    - Use parallelization.
    - Use an interesting high-performance datastructure.

## References

Ismail Ilkan Ceylan, Adnan Darwiche, and Guy Van den Broeck. Open-world probabilistic databases. In *Proceedings of the 15th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, May 2016. URL `http://web.cs.ucla.edu/~guyvdb/papers/CeylanKR16.pdf`.

Nilesh N. Dalvi and Dan Suciu. The dichotomy of probabilistic inference for unions of conjunctive queries. *J. ACM*, 59:30:1–30:87, 2012.

Eric Gribkoff and Dan Suciu. Slimshot: In-database probabilistic inference for knowledge bases. *PVLDB*, 9:552–563, 2016.

Eric Gribkoff, Guy Van den Broeck, and Dan Suciu. Understanding the complexity of lifted inference and asymmetric weighted model counting. In *UAI*, 2014.

Guy Van den Broeck and Dan Suciu. *Query Processing on Probabilistic Data: A Survey*. Foundations and Trends in Databases. Now Publishers, August 2017. doi: 10.1561/1900000052. URL `http://web.cs.ucla.edu/~guyvdb/papers/VdBFTDB17.pdf`.

Michael Wick, Andrew McCallum, and Gerome Miklau. Scalable probabilistic databases with factor graphs and mcmc. *Proc. VLDB Endow.*, 3(1-2): 794–804, September 2010. ISSN 2150-8097. doi: 10.14778/1920841.1920942. URL `http://dx.doi.org/10.14778/1920841.1920942`.