Contents

1 Meeting with Murali's students

Added: *[2021-12-29 Wed 12:38]*

1.1 Planning to finish the article [0/4]

- VLDB Journal

1.1.1 new stuff over previous paper

- new algorithm with variants
- new experiments
  - comparison with PDQ

- optimization time + execution time breakdown

1.1.2 TODO experiments [0/2]

- ☐ TPC-H
  - ☐ translate queries into Datalog (8 / 19?) **(4 hours)**
  - ☐ measure optimization time and runtimes **(4 hours)**
  - ☐ measure PDQ optimization times **(16 hours)**
  - ☐ analyze and plot **(24 hours)**
- ☐ real world data
  - ☐ select datasets
  - ☐ translate all queries into Datalog
  - ☐ measure optimization time and runtimes
  - ☐ measure PDQ optimization times
  - ☐ analyze and plot

1.1.3 TODO missing proofs

1.1.4 TODO replace latex template

1.1.5 TODO writing [0/6]

1. **TODO** introduction [1/4]
   - ☐ add running example
     - ☐ hopefully greedy does not work (if we can keep it simple enough. Otherwise, this example comes later)
   - ☒ prior (new results of IPAW paper)
   - ☐ contributions list
   - ☐ outline of paper
2. **TODO** background [1/4]
   - ☐ Datalog
   - ☐ provenance and capture
   - ☒ dependencies
   - ☐ semantic query optimization with chase & backchase
3. **TODO** related work [0/2]
   - ☐ provenance capture & querying
     - ☐ check for new papers
   - ☐ semantic query optimization
     - ☐ more citations including (provenance-directed) chase & backchase, PDQ-related papers, ...

4. **TODO** restricted provenance capture (capture for one input table) [0/2]
   - ☐ explain basic idea
   - ☐ write rules
5. **TODO** semantic optimization algorithm [0/3]
   - ☐ write algorithm
   - ☐ explain algorithm
   - ☐ proof correctness
6. **TODO** write experiments [0/4]
   - ☐ explain setup and competitors
   - ☐ optimization time comparison with PDQ
   - ☐ runtime comparison with PDQ and unoptimized version
   - ☐ datasets & workloads
     - ☐ TPC-H
     - ☐ real world data

1.1.6 questions

- where would the algorithms show benefit over the old ones
-

1.2 TODO what to introduce everybody before

1.2.1 TODO databases

1. **TODO** SQL
2. **TODO** administrating a database (Postgres)
3. **TODO** datalog
   - Table: **Person**

Name	Age	Salary	Dept
Peter	34	3444	CS
Bob	60	10500	CS
Alice	34	40000	CS
Fred	3444	10	News

   - Table: **Department**

Title	Budget	City
CS	134300	Chicago
News	123	Ann Arbor

   - active domain $adom(D)$ is all values that exist the database

```
Q(name,salary) :- Person(name,_,salary,"CS").

-- name: Peter, age: 60, salary: 40000
Q(Peter,40000) :- Person(Peter,60,40000,"CS").
-- name: Peter, age: 34, salary: 40000
Q(Peter,40000) :- Person(Peter,34,40000,"CS").
-- name: Peter, age: 34, salary: 3444
Q(Peter,3444) :- Person(Peter,34,3444,"CS"). -- works out
-- name: Peter, age: 60, salary: 3444
Q(Peter,3444) :- Person(Peter,_,3444,"CS"). -- works out
-- name: 60, age: 60, salary: 60
```

```
CREATE TABLE person (
  name VARCHAR(15) PRIMARY KEY,
  age INT,
  salary INT,
  dept VARCHAR(10)
  );
```

```
Q(Name) :- Person(Name,X,Y,Z).
```

- get all employee's from the CS department

```
Q(name,salary,dept) :- Person(name,_,salary,dept), dept="CS".
Q(name,salary,dept) :- Person(name,_,salary,"CS").
```

- get all employees that earn more then 100000

```
Q(name,salary,dept) :- Person(name,_,salary,dept), salary > 100000.
```

```
Q(Name,Salary,Dept) :- person(Name,A,Salary,Dept), department(Dept,B,C).
Q(X,Y,Z) :- person(X,_,Y,Z), department(Z,B,C).
```

1.2.2 union in datalog

Person

Name	Dept
Peter	CS
Alice	CS
Bob	HR
Gert	HR

```
Q(Name) :- Person(Name, cs).
Q(Name) :- Person(Name, hr).
```

1. **TODO** equivalence, constraints, semantic query optimization

```
Q(X) :- R(X,Y). -- R is edb
Q2(X) :- Q(X).  -- Q + Q2 is idb
```

- `http://www.cs.iit.edu/~glavic/cs520/2022-spring/exams/`

(a) Set notation $S = \{e_1, \ldots, e_n\}$
$S_1 = \{a, b, c\}$ and $S_2 = \{a, b\}$
$S_1 \supseteq S_2$
$S_1 \supset S_2$
$S_1 = S_2 \Leftrightarrow S_1 \subseteq S_2 \wedge S_1 \supseteq S_2$

(b) Predicate and First-order Logic

- AND $\wedge$, OR $\vee$, NOT $\neg$, implies $\rightarrow$

x	y	x AND y
0	0	0
0	1	0
1	0	0
1	1	1

x	y	x -> y
0	0	1
0	1	1
1	0	0
1	1	1

- variables over domain values $\mathbb{N}$
- predicates $<: \mathbb{N} \times \mathbb{N} \rightarrow \{F, T\}$
  - $x < y$, $x < y \wedge y < z$
- quantification:
  - $\forall x : \phi(x)$ - is true if for all `x` the $\phi(x)$ (universal)
    - $\forall x : bird(x) \rightarrow canfly(x)$

if `x` is bird then `x` canfly? probably not true

$$\forall x : isCSstudent(x) \rightarrow canprogram(x)$$

(c) Equivalence and Containment

- Queries `Q` are function:
  - Input: database `D` (EDB only)
  - Output: database `Q(D)` (EDB + IDB)
- **Query equivalence:**
  - `Q` and `Q'` are equivalent if for every database `D` we have `Q(D) = Q'(D)`

$$Q \equiv Q' \Leftrightarrow \forall D : Q(D) = Q'(D)$$

- **Query containment:**
  - `Q` is contained in `Q'` if for every database `D` we have `Q(D)` subset or equal to `Q'(D)`

$$Q \sqsubseteq Q' \Leftrightarrow \forall D : Q(D) \subseteq Q'(D)$$

- Write query equivalence as containment

$$Q \equiv Q' \Leftrightarrow \forall D : Q(D) \subseteq Q'(D) \wedge Q'(D) \subseteq Q(D) \Leftrightarrow Q \sqsubseteq Q' \wedge Q' \sqsubseteq Q$$

```
Q1(X) :- R(X,Y), R(X,Z).
Q2(X) :- R(X,Y).
Q3(X) :- Q2(X), X < 2. -- Dominick Q3 contained in Q2
Q4(Y) :- R(X,Y). -- not contained in Q2
Q5(X) :- R(X,Y).
Q6(A) :- R(A,GFDFGDFG). -- equivalent to Q2 and Q5
Q7(X) :- R(X,Y), X < 2.
Q8(X) :- R(X,Y), S(Y,Z). -- contained Q2
```

- **Variable names are irrelevant**: only their positions in the body and head matter
- **One body is superset of another body**: it is more restrictive (it returns less results)
- **if two queries do not return the "same" variables (after renaming)**: no containment relationship
- **Containment mapping**
  - Variable mapping `Var(Q) -> Var(Q')` then we rename all variables from `Q` to variables from `Q'`
    - `Q2 -> Q5`: `X -> X`, `Y -> X`, `X -> X`, `Y -> Y`, `X -> Y`, `Y -> Y`, `X -> Y`, `Y -> X`
  - Containment mapping is a variable mapping that fulfills these two conditions:
    - 1) the head is head mapped to the head
    - 2) every atom from the body of `Q` exists after renaming in the body of `Q'`
- **Example**:
  - `Q2 -> Q6`:
    - head to head: `X -> A`
    - body to body: `Y -> GFDFGDFG` we get `R(X,Y)` is mapped to `R(A,GFDFGDFG)`
  - `Q2 -> Q5`:
    - head to head: `X -> X`
    - body to body: `Y -> Y`
  - `Q7 -> Q2`
    - head to head: `X -> X`
    - body to body: `Y -> Y`
  - `Q8 -> Q2`
    - `X -> X`
    - `Y -> Y`
    - =Z -> =
    - head to head: YES
    - body to body: NO
  - `Q2 -> Q8` -> $Q_8 \sqsubseteq Q_2$
    - `X -> X`
    - `Y -> Y`
    - head to head: YES

  * body to body: YES
- `Q1 -> Q2` -> $Q_2 \sqsubseteq Q_1$
  * `X -> X`
  * `Y -> Y`
  * `Z -> Y`
  * head to head: YES
  * body to body: YES
- `Q2 -> Q1` -> $Q_1 \sqsubseteq Q_2$
  * `X -> X`
  * `Y -> Y`
  * head to head: YES
  * body to body: YES

`R`

A	B
a	1
b	2
c	3

`Q1(D)`

X
a
b
c

`Q2(D)`

X
c
a
b

2. **TODO** constraints

- **primary key**: attributes of a table that are unique in a table
- **SSN** as PK for this table

SSN	Name	Salary
111	Peter	30003
222	Peter	12312
333	Bob	12312

(a) functional dependencies

- **functional dependencies**
  - `A -> B` holding over `R`
  - then for any two tuple $t, t' \in R$ if $t.A = t'.A$ then $t.B = t'.B$
  - `SSN -> Name, Salary`

Name	Zip	city
Peter	60616	Chicago
Bob	60616	Chicago
Alice	60657	Chicago
Fred	11111	New York

- evaluate query under the knowledge that `zip -> city` holds for the database

```
Q(C1,C2) :- address(_,Z,C1), address(_,Z,C2), C1 != C2.
```

- result is guaranteed to be empty when know that `zip -> city` holds

(b) fulfilling constraints

Name	zip	city
Peter	60616	Chicago
Bob	60616	New York
Alice	60657	Chicago
Fred	11111	New York

(c) inclusion dependencies

- foreign keys as a special case
- `Person(Name,LiveAt)`, `Address(Id,City,Zip,Street)` with `Id` is PK for address
- `Person(Name,Id,City,Zip,Street)` is also an option

Person

Name	LivesAt
Peter	1
Alice	1
bob	2

Address

Id	city	zip	street
1	Chicago	60614	adsasdas
2	Chicago	60666	adsasdasdasd
3	Chicago	60615	adsasdalosjkdas

- foreign key constraint. For every value of attribute `A` of table `R` there has to exists tuple `s` in table `S` with PK equal to the value of `A`.
  - the set of values in attribute `LivesAt` has to be a subset of the values in attribute `Id`
  - **inclusion dependency**

$$\forall name, livesat : Person(name, livesat) \rightarrow \exists city, zip, street : Address(livesat, city, zip, street)$$

3. **TODO** semantic query optimization

(a) Semantic query optimization problem

- **Inputs:**
  - database `D` and set constraint Σ
  - query `Q`

- **Output:**
  - query `Q'` that is equivalent to `Q` under the Σ
  - "optimal in some way"

(b) Example

- `Person(Name,LiveAt)`, `Address(Id,City,Zip,Street)` with `Id` is PK for address

```
Q1(N) :- Person(N,L), Address(L,C,Z,S).
Q2(N) :- Person(N,L).
```

Person

Name	LivesAt
Peter	3
Alice	1
bob	2

Address

Id	city	zip	street
1	Chicago	60614	adsasdas
2	Chicago	60666	adsasdasdasd
3	Chicago	60615	adsasdalosjkdas

(c) Query minimization

- find smallest query `Q'` such that $body(Q') \subseteq body(Q)$ and that $Q \equiv Q'$
  - size of `Q` is measured as number of atoms in the body of `Q`
  - we have function `equivalent(Q,Q') -> Bool` and have function `unsafe(Q) -> Bool`

```
Q(N) :- Person(N,L), Address(L,C,Z,S).
```

```
Q1(N) :- Address(L,C,Z,S). -- unsafe
Q2(N) :- Person(N,L).
Q3(N) :- . -- unsafe
```

```
Q(X,Y) :- R(X,Y), R(X,Z), R(X,A).
```

- `S = {a,b,c}` , . . . , `{}, {a}, {b}, {c}, {a,b}, {a,c}, {b,c}...`

```
Q1(X,Y) :- R(X,Y), R(X,Z). -- safe, equivalent
Q2(X,Y) :- R(X,Y), R(X,A). -- safe, equivalent
Q3(X,Y) :- R(X,Z), R(X,A). -- unsafe
Q4(X,Y) :- R(X,Y). -- safe, equivalent
Q5(X,Y) :- R(X,A). -- unsafe
Q6(X,Y) :- R(X,Z). -- unsafe
Q7(X,Y) :- . -- unsafe
```

- Equivalence of `Q` and `Q1`, $Q \equiv Q' \Leftrightarrow Q \sqsubseteq Q' \wedge Q' \sqsubseteq Q$

```
Q(X,Y) :- R(X,Y), R(X,Z), R(X,A).
Q1(X,Y) :- R(X,Y), R(X,Z).
```

- `Q -> Q1`:
  - `CM: X -> X, Y -> Y, Z -> Z, A -> Y`
- `Q1 -> Q`

- CM: `X -> X, Y -> Y, Z -> Y`

- revisiting person example

```
Q(N) :- Person(N,L), Address(L,C,Z,S).
Q1(N) :- Person(N,L).
```

- `Q -> Q1`:
  - CM: `N -> N, L -> L, C -> L, Z -> L, S -> L`
    - `CM(Person(N,L)) = Person(N,L)`
    - `CM(Address(L,C,Z,S)) = Address(L,L,L,L)`
- `Q1 -> Q`:
  - CM: `N -> N, L -> L`
    - `CM(Person(N,L)) = Person(N,L)`

(d) Query Optimization with Constraints

- minimization of queries
  - remove body atoms (DL)
- find smallest query `Q'` such that $body(Q') \subseteq body(Q)$ and that $Q \equiv Q'$ given Σ

4. **TODO** more provenance

5. **TODO** GProM (just how to run it)

- source code
- on debussy: `/home/perm/semantic_opt_gprom`
- `./src/command_line/gprom -backend postgres -host 127.0.0.1 -user postgres -passwd test -port 5450 -db gpromtest -frontend dl`
- `-Osemantic_opt TRUE -Oflatten_dl TRUE`
- sqlite on: `./src/command_line/gprom -backend sqlite -db ./examples/test.db -frontend dl`
- time one query: `./src/command_line/gprom -backend postgres -host 127.0.0.1 -user postgres -db semanticopt -port 5433 -passwd test -frontend dl -timing -query 'Q(X) :- "r"(X,Y).'`

(a) compute lineage

```
Q(X) :- R(X,Y), S(Y,Z). ANS: Q. RP(1). FD R: A -> B. LINEAGE FOR R FOR
↪  RESULTS FROM RP.
```

- time it query: `./src/command_line/gprom -backend postgres -host 127.0.0.1 -user postgres -db semanticopt -port 5433 -passwd test -frontend dl -Osemantic_opt TRUE -Oflatten_dl TRUE -timing -query 'Q(X) :- "r"(X,Y), "s"(Y,Z). ANS: Q. RP(1). FD "r": "a" -> "b". LINEAGE FOR "r" FOR RESULTS FROM RP.'`
- run query multiple times and time each execution: `./src/command_line/gprom -backend postgres -host 127.0.0.1 -user postgres -db semanticopt -port 5433 -passwd test -frontend dl -Osemantic_opt TRUE -Oflatten_dl TRUE -timing -time_queries TRUE -repeat_query_count 10 -query 'Q(X) :- "r"(X,Y), "s"(Y,Z). ANS: Q. RP(1). FD "r": "a" -> "b". FD "r": "b" -> "a". LINEAGE FOR "r" FOR RESULTS FROM RP.`

for postgres for now:

```
Q(X) :- "r"(X,Y), "s"(Y,Z). ANS: Q. RP(1). FD "r": "a" -> "b". LINEAGE
↪  FOR "r" FOR RESULTS FROM RP.
```

- just run a query and also time it (using SQLite int his example)
- `./src/command_line/gprom -backend sqlite -db ./examples/test.db -Osemantic_opt TRUE -Oflatten_dl TRUE -timing TRUE -loglevel 3 -frontend dl -query 'Q(X) :- R(X,Y), S(Y,Z). ANS: Q. RP(1).  FD R: A -> B. LINEAGE FOR R FOR RESULTS FROM RP.'`
- **ANS**: the result relation for the query
- **FDs**: `FD table:  columns -> columns.`
- `LINEAGE FOR R` - compute lineage of input table `R`
  - `...  FOR RESULTS FROM RP.` - then only compute lineage for results from `RP`

6. **TODO** how to run PDQ
7. **TODO** git
   - `https://github.com/IITDBGroup/GProM`
8. practice CS520
   (a) Write a Datalog program that returns the lastname and gpa of students that study cs

```
Q(lastname,gpa) :- student(_,_,lastname,major,gpa), major="cs"
```

```
Q(X,Y) :- student(S1, S2, X, "cs", Y)
```

   (b) Surfing or hacking

```
Q(X,Y) :- student(S1, X, Y, S2, S3), interest(S1, "surfing").
```

```
Q(fname,lname) :- student(sid,fname,lname,_,_), interest(sid,"surfing").
```

   - surfing or hacking:

```
Q(fname,lname) :- student(sid,fname,lname,_,_), interest(sid,A), (A =
↪  "surfing" OR A = "hacking").
```

   - surfing and hacking:

```
Q(fname,lname) :- student(sid,fname,lname,_,_), interest(sid,"surfing"),
↪  interest(sid, "hacking").
```

   - surfing or hacking:

```
Q(fname,lname) :- student(sid,fname,lname,_,_), interest(sid,A), A =
↪  "surfing".
Q(fname,lname) :- student(sid,fname,lname,_,_), interest(sid,A), A =
↪  "hacking".
```

$$\forall sid, fname, lname, m, g, A : student(sid, fname, lname, m, g) \wedge interest(sid, A) \wedge A = ”surfing” \rightarrow \exists Q$$

$$\forall fname, lname : Q(fname, lname) \rightarrow \exists sid, m, g, A : student(sid, fname, lname, m, g) \wedge interest(sid, A$$

   (c) Students with same interest

```
Q(S1,L1,S2,L2):- Student(S1,f,L1,m,g), Interest(S1,a1),
↪  Student(S2,f,L1,m,g), Interest(S2,a2), a1 = a2, S1 < S2.
```

1.2.3 DONE terminal basics

1. **DONE** ssh

   - for windows users use `putty` or `WSL`
   - create terminal session on a different machine
     - connect as `USER`
     - to machine `MACHINE`

   ```
   ssh USER@MACHINE
   ```

   - copy files between machines -`scp`

   ```
   scp file otherfile
   ```

2. **DONE** running programs

   - program: run by inputting name
   - options: pass after the program typically start with `-`
   - `find dir options` - searching files in `dir`
   - `man program` - open help for program
     - `SPACE` next page
     - `p` previous page
     - `q` quit
     - `/term` search for `term`
       - /<enter> move to next match
   - `cat file` - print file content
   - `grep` search for content in files
   - `echo msg` print `msg` to stdout

   (a) combining programs

   - input / output streams
     - `stdout`
     - `stderr`

     = `stdin`
   - `p1 | p2` - pass output of `p1` into `p2` (connect `p1` stdout to `p2` stdin
   - redirect `>` redirect stdout, `2>` redirect stderr
   - read `stdin` from file with `< file`

3. **DONE** navigating directories

   (a) directories & files

   i. directories

   - separated by `/`
   - home directory `~`

- list content of current directory `ls`
  - `-a` list hidden files also
  - `-l` list file details
- `pwd` - prints the current directory
- `cd` - move to a different directory
  - `..` means one level up
  - `.` means the current folder
  - starting with / means absolute
  - without prefix / means relative to current directory

ii. permissions

- `r` - reading
- `w` - writing
- `x` - executable (files), can change into for directories
- permission 9 values (3 for user (owner), 3 for group (owning), 3 for public (everybody else))
- `chown user file` - change owner of `file` to `user`
- `chgrp grp file` - change owner group of `file` to `grp`
- `chmod permission file` - change permissions of `file` to `permission`
  - as 3 numbers (user, group, public) `4` means reading, `2` writing, `1` executing. Sum up these numbers
- super power user: `root` can do everything
- temporarily become root: `sudo`

iii. deletion

- `rm file` deletes file

1.2.4 TODO programming languages

1. **TODO** Python
2. **TODO** C
3. **TODO** Java

1.3 setup access to machines

1.3.1 machines

- `debussy.cs.iit.edu`

1.3.2 ssh

1. Mac

   - open terminal
   - run ssh with `user` and machine `machine`

```
ssh user@machine
```

for instance

```
ssh perm@debussy.cs.iit.edu
```

2. windows

   - option 1: install Putty
   - WSL -> like mac

1.3.3 running postgres

- **postgres cluster**: where the data is stored
- **postgres server program**: `postgres` or postmaster
  - `-D` tells postgres where the data will be stored
  - `-c` where to find the configuration file
  - `-p` which network port to run on
- **psql**: `psql` - run queries
  - `-U USER` - connect as user `USER`
  - `-p PORT` - port
  - `-h HOST` - host: `127.0.0.1`
  - `-D DATABASE` - the database to connect to

1. psql

   ```
   psql -h 127.0.0.1 -U postgres -p 5433 -D DBNAME
   ```

   - `\q` - quit psql
   - `\?` - help for all backslash commands
   - `\d OBJECT` - print information about `OBJECT` (e.g., a table)
   - to load data (by running a sql script)

   ```
   psql# \i file
   ```

   - TPC-H loading scripts `/local/perm/tpchdata/scripts/ddl_1.sql` - is 1GB
   - creating database

   ```
   CREATE DATABASE name;
   ```

2. postgres 10

   - check with servers are running

   ```
   ps aux | grep postgres
   ```

   ```
   /usr/lib/postgresql/10/bin/postgres -D /var/lib/postgresql/10/main -c
   ↪  config_file=/etc/postgresql/10/main/postgresql.conf
   ```

- connect to server

```
psql -h 127.0.0.1 -U postgres -p 5433 postgres
```

- connect with gprom

```
./src/command_line/gprom -backend postgres -host 127.0.0.1 -user postgres -db
↪  semanticopt -port 5433 -passwd test
```

3. postgres 11

- connect to server

```
psql -h 127.0.0.1 -U postgres -p 5453 semanticopt
```

1.4 code and what to compare

1.5 experiments

1.5.1 the problem setting

- **input:** user query for provenance wrt. to query `Q`, database `D`, set of constraints Σ, input table `R` to a query result subset `R'<=Q(D)`
- **step 1:** Generate query `QP` that computes provenance of `R'` in `R` for `Q`, `D`
- **step 2:** optimize the query to minimize it's size (to generate `QP'` equivalent to `QP` under a given set of constraints Σ
- **step 3:** run the optimized query `QP'(D)`
- **output:** provenance which is subset of `R`

1.5.2 what parameters to vary?

- **database size** `D`
  - **data distribution / real world or benchmark datasets**
- **structure and size of query** `Q`
  - **how selective is the query in terms of provenance**
- **number of constraints** Σ
- **which input table** `R`
- **what subset of results** (`R'`)

1.5.3 competitors

- **what methods to compare:**
  - **baseline:** do not optimize the query (free step 2, we pay at step 3)
  - **PDQ:** has an expensive step 2, but may be better sometimes in step 3 (complete method)
  - **our approach:** less expensive step 2, but may be worse in step 3 than PDQ

1.5.4 running step 2

```
for x in `seq 100`;
do
    ./src/command_line/gprom -backend postgres -host 127.0.0.1 -user postgres
    ↪  -db semanticopt -port 5453 -passwd test -frontend dl -Osemantic_opt TRUE
    ↪  -Oflatten_dl TRUE -loglevel 0 -Pexecutor sql -timing -queryFile
    ↪  ./umflint/tpcq18/customer.sql;
done \
    | grep 'timer: TOTAL' \
    | awk ' { print $5 }' \
    > q18-opttime-customer.csv

rm q18-opttime-customer.csv; \
for x in `seq 100`;
do
    ./src/command_line/gprom -backend postgres -host 127.0.0.1 -user postgres
    ↪  -db semanticopt -port 5453 -passwd test -frontend dl -Osemantic_opt TRUE
    ↪  -Oflatten_dl TRUE -loglevel 0 -Pexecutor sql -timing -queryFile
    ↪  ./umflint/tpcq18/customer.sql \
    | grep 'timer: TOTAL' \
    | awk ' { print $5 }' \
    >> q18-opttime-customer.csv
done
```

1.5.5 running step 3

1. generate provenance capture SQL queries

    - generate file with optimized SQL query capturing provenance

```
./src/command_line/gprom -backend postgres -host 127.0.0.1 -user postgres -db
↪  semanticopt -port 5453 -passwd test -frontend dl -Osemantic_opt TRUE
↪  -Oflatten_dl TRUE -loglevel 0 -Pexecutor sql -queryFile
↪  ./umflint/tpcq18/customer.sql \
> ./umflint/tpcq18/p_customer.sql
```

    - generate file with unoptimized SQL query capturing provenance

```
./src/command_line/gprom -backend postgres -host 127.0.0.1 -user postgres -db
↪  semanticopt -port 5453 -passwd test -frontend dl -Osemantic_opt FALSE
↪  -Oflatten_dl TRUE -loglevel 0 -Pexecutor sql -queryFile
↪  ./umflint/tpcq18/customer.sql \
> ./umflint/tpcq18/p_customer-unopt.sql
```

2. evaluate provenance capture queries

    - time with psql (optimized)

```
for x in `seq 1 1000`; do \
    psql -h 127.0.0.1 -U postgres -d semanticopt -p 5453 -o /dev/null -c
    ↪  '\timing on' -f ./umflint/tpcq18/p_customer.sql | grep 'Time:' | awk '
    ↪  { print $2 }'; \
done > exp_results/tpcq18/p_customer.csv
```

- time with psql (unoptimized)

```
for x in `seq 1 1000`; do \
    psql -h 127.0.0.1 -U postgres -d semanticopt -p 5453 -o /dev/null -c
    ↪  '\timing on' -f ./umflint/tpcq18/p_customer-unopt.sql | grep 'Time:' |
    ↪  awk ' { print $2 }'; \
done > exp_results/tpcq18/p_customer-unopt.csv
```

3. tmux

   - create a terminal session that continues after you disconnect from your ssh session
   - create tmux

```
tmux
```

   - detach from session `CTRL-b d`
   - attach to existing tmux session (if our session is `5`)

```
tmux list-sessions
tmux a -t 5
```

   - create new window: `CTRL-b c`
   - rename a window: `CTRL-b ,`
   - jump to window numbered n: `CTRL-b n`, e.g., `CTRL-b 0`
   - delete window: `CTRL-b &`

4. check system load (is somebody else utilizing the system for heavy work)

```
htop # show process / CPU / memory utilization
sudo iotop # show disk utilization (read / write)
```

5. generate TPC-H Datalog

   (a) TPC-H Q3

      i. translate query to datalog

```
SELECT l_orderkey, -- select this column
       sum(l_extendedprice*(1-l_discount)) as revenue,
       o_orderdate,
       o_shippriority
FROM customer c, orders o, lineitem l
WHERE o_orderdate < '1995-03-15'
   AND l_shipdate > '1995-03-15'
```

```
   AND c.c_mktsegment = 'BUILDING'
   AND c.c_custkey = o.o_custkey
   AND l.l_orderkey = o.o_orderkey
GROUP BY l_orderkey, o_orderdate, o_shippriority
```

```
SELECT l_orderkey, -- select this column
       sum(l_extendedprice*(1-l_discount)) as revenue,
       o_orderdate,
       o_shippriority
FROM customer c JOIN orders o ON (c.c_custkey = o.o_custkey) JOIN
↪  lineitem l ON (l.l_orderkey = o.o_orderkey)
WHERE o_orderdate < '1995-03-15'
   AND l_shipdate > '1995-03-15'
   AND c.c_mktsegment = 'BUILDING'
GROUP BY l_orderkey, o_orderdate, o_shippriority
```

- equivalent datalog
- if aggregation function in head, then non-aggregated variables are group-by

```
Q(l_ok, sum(l_ep * (1-l_d)), o_od, o_sp) :-
    customer(c_ck,c_n,c_a,c_nk,c_p,c_ab,c_ms,c_ct),
    orders(o_ok,o_ck,o_os,o_t,o_od,o_op,o_c,o_sp,o_ct),
    lineitem(l_ok,l_pk,l_sk,l_ln,l_q,l_ep,l_d,x,y,z,a,b,c,d,e,f),
    o_od < '1995-03-15',
    l_sd > '1995-03-15',
    c_ms = 'BUILDING',
    c_ck = o_ck,
    l_ok = o_ok.
```

- with reusing variables instead of equality comparisons

```
Q(o_ok, sum(l_ep * (1-l_d)), o_od, o_sp) :-
    customer(c_ck,c_n,c_a,c_nk,c_p,c_ab,c_ms,c_ct),
    orders(o_ok,c_ck,o_os,o_t,o_od,o_op,o_c,o_sp,o_ct),
    lineitem(o_ok,l_pk,l_sk,l_ln,l_q,l_ep,l_d,x,y,z,a,b,c,d,e,f),
    o_od < '1995-03-15',
    l_sd > '1995-03-15',
    c_ms = 'BUILDING'.
```

- compute provenance for table `customer`

```
Q(o_ok, sum(l_ep * (1-l_d)), o_od, o_sp) :-
↪  customer(c_ck,c_n,c_a,c_nk,c_p,c_ab,c_ms,c_ct),
↪  orders(o_ok,c_ck,o_os,o_t,o_od,o_op,o_c,o_sp,o_ct),
↪  lineitem(o_ok,l_pk,l_sk,l_ln,l_q,l_ep,l_d,x,y,z,a,b,c,d,e,f), o_od
↪  < '1995-03-15', l_sd > '1995-03-15', c_ms = 'BUILDING'.

ANS : Q.

LINEAGE FOR customer.
```

- compute provenance for subset of results

```
Q(o_ok, sum(l_ep * (1-l_d)), o_od, o_sp) :-
↪  customer(c_ck,c_n,c_a,c_nk,c_p,c_ab,c_ms,c_ct),
↪  orders(o_ok,c_ck,o_os,o_t,o_od,o_op,o_c,o_sp,o_ct),
↪  lineitem(o_ok,l_pk,l_sk,l_ln,l_q,l_ep,l_d,x,y,z,a,b,c,d,e,f), o_od
↪  < '1995-03-15', l_sd > '1995-03-15', c_ms = 'BUILDING'.

ANS : Q.

QP(a,b,c,d) :- Q(a,b,c,d), a = 1231455.

LINEAGE FOR customer FOR RESULTS FROM QP.
```

1.5.6 TODO running things in PDQ

1.5.7 experiment dry run

1.6 TPC-H datalog queries

1.6.1 Q3

1. capture provenance for customer

```
q(l_ok, sum(l_ep*(1-l_d)), o_od, o_sp) :-
customer(c_ck,c_n,c_a,c_nk,c_p,c_ab,'BUILDING',c_ct),
orders(l_ok,c_ck,o_os,o_t,o_od,o_op,o_c,o_sp,o_ct),
lineitem(l_ok,l_pk,l_sk,l_ln,l_q,l_ep,l_d,x,y,z,l_sd,b,c,d,e,f),
o_od < '1995-03-15', l_sd > '1995-03-15'.

ANS: q.

LINEAGE FOR customers FOR RESULTS FROM q.
```

2. capture provenance for lineitems

```
q(l_ok, sum(l_ep*(1-l_d)), o_od, o_sp) :-
customer(c_ck,c_n,c_a,c_nk,c_p,c_ab,'BUILDING',c_ct),
orders(l_ok,c_ck,o_os,o_t,o_od,o_op,o_c,o_sp,o_ct),
lineitem(l_ok,l_pk,l_sk,l_ln,l_q,l_ep,l_d,x,y,z,l_sd,b,c,d,e,f),
o_od < '1995-03-15', l_sd > '1995-03-15'.

ANS: q.

LINEAGE FOR lineitems FOR RESULTS FROM q.
```

3. capture provenance for orders

```
q(l_ok, sum(l_ep*(1-l_d)), o_od, o_sp) :-
customer(c_ck,c_n,c_a,c_nk,c_p,c_ab,'BUILDING',c_ct),
orders(l_ok,c_ck,o_os,o_t,o_od,o_op,o_c,o_sp,o_ct),
lineitem(l_ok,l_pk,l_sk,l_ln,l_q,l_ep,l_d,x,y,z,l_sd,b,c,d,e,f),
o_od < '1995-03-15', l_sd > '1995-03-15'.
```

```
ANS: q.

LINEAGE FOR orders FOR RESULTS FROM q.
```

1.6.2 Q18

2 Meeting Murali

Added: *[2021-11-12 Fri 13:46]*

- schema: `R(A,B)`, `S(C,D)`

```
Q(X) :- R(X,Y), S(Y,Z).
```

$QP \subseteq Q$

```
PROV_R(X,Y) :- R(X,Y), S(Y,Z). QP(X).
```

functional dependency: `A -> B`

```
PROV_R(X,Y) :- R(X,Y). QP(X).
```

A	B	C
1	1	a
2	2	c
2	2	d

$\forall x, y, z, x', z' : address(x, y, z) \wedge address(x', y, z') \rightarrow z = z'$
`zip -> city`

street	zip	city
10	60614	NY
23	60614	NY