Problem 1.
Express the queries from HWK 1, Problem 1 in Datalog. Name this file "xxx_pr1hwk2.P", where xxx is your last name. (You can test your queries on the College Database file we have provided for you, but remember – it should work for all database instances!)

Note: below, duplicates can be removed by using "nodup_query(attr) :- setof(_,Goal,_)."

%% Names of students who took some 4-credit course
%% Duplicates are removed by using "setof(_,Goal,_)".
%% Q1_RA=πname(STUDENTS⋈GRADES⋈σcr=4COURSES)
q1(Name) :- studs(Stno,Name,_,_,_,_), grads(Stno,_,Cno,_,_,_), cours(Cno,_,4).

%% Names of students who took ONLY 4-credit course
%% No need to remove duplicates because deducing from q11(Name)
%% Q2_RA=Q1_RA-πname(STUDENTS⋈GRADES⋈σcr≠4COURSES)
took_non_4_cred_cour(Name) :- studs(Stno,Name,_,_,_,_),
grads(Stno,_,Cno,_,_,_),
cours(Cno,_,Cr),
Cr=\=4.
q2(Name) :- q1(Name), \+took_non_4_cred_cour(Name).

%% Names of students who did not take both cs310 and cs210
%% Duplicates are removed by using "setof(_,Goal,_)".
%% Q3_RA=πname(STUDENTS)-πname(STUDENTS⋈(σcno='cs210'GRADES ⊕σcno='cs310'GRADES))
took_both(Stno) :- grads(Stno,_,cs210,_,_,_),
grads(Stno,_,cs310,_,_,_),
q3(Name) :- studs(Stno,Name,_,_,_,_), \+took_both).

%% Names of students who took at least two courses
%% Q4_RA=πname(STUDENTS⋈(σT1.cno=T2.cno(ρcno=T1.cnoGRADES ⊕ρcno=T2.cnoGRADES))
took_two_diff_cour(Stno) :- grads(Stno,_,Cno1,_,_,_),
grads(Stno,_,Cno2,_,_,_),
Cno1=\=Cno2.
q4(Name) :- studs(Stno,Name,_,_,_,_), took_two_diff_cour(Stno).
%% Names of students who do not have an advisor
%% Q5_RA=π\text{name}\text{STUDENTS}-π\text{name}(\text{STUDENTS}⋈\text{ADVISING})
q5(\text{Name}) :- \text{studs}(\text{Stno},\text{Name},\_,\_,\_,\_), \+\text{advis}(\text{Stno},\_).

%% Student and Instructor names where this student never took any course
%% with this instructor
%% Q6_RA=π\text{s\_name, i\_name, (}
%%
%% (π\text{stno}\text{STUDENTS}×π\text{empno}\text{INSTRUCTORS}
%%
%% -π\text{stno}\text{STUDENTS}×π\text{empno}\text{INSTRUCTORS}⋈\text{GRADES})
%%
%% ⟨π\text{name}→\text{s\_name}\text{STUDENTS}
%%
%% ⟨π\text{name}→\text{i\_name}\text{INSTRUCTORS}
%%
%% )

q6(S\text{Name},I\text{Name}) :- \text{studs}(\text{Stno},S\text{Name},\_,\_,\_,\_),
\text{insts}(\text{Empno},I\text{Name},\_,\_,\_,\_),
\+\text{grads}(\text{Stno},\text{Empno},\_,\_,\_,\_).
Problem 2.
(ii) Write a stratified Datalog program to answer the following queries, creating a file “xxx_pr2hwk2.P”. You are encouraged to reuse earlier predicates when defining later ones.

Note: I will assume that we always want the origin and destination to be different.

(a) train_only(a,b): Find the pairs of stations (a,b) such that one can go from a to b by train but not by plane.

\[
\begin{align*}
\text{train\_path}(A,B) & : = \text{train}(A,B). \\
\text{train\_path}(A,B) & : = \text{train}(A,X), \text{train\_path}(X,B), A \neq B.
\end{align*}
\]

\[
\begin{align*}
\text{plane\_path}(A,B) & : = \text{plane}(A,B). \\
\text{plane\_path}(A,B) & : = \text{plane}(A,X), \text{plane\_path}(X,B), A \neq B.
\end{align*}
\]

\[
\begin{align*}
\text{train\_only}(A,B) & : = \text{train\_path}(A,B), \neg \text{plane\_path}(A,B).
\end{align*}
\]

(b) pure_plane_path(a,b): a pure plane path from a to b is a plane itinerary from a to b such that for all consecutive stops c,d along the way, one cannot go from c to d by train. Find the pairs of stations (a,b) such that there is a pure plane path from a to b.

\[
\begin{align*}
\text{pure\_plane\_hop}(A,B) & : = \text{plane}(A,B), \neg \text{train\_path}(A,B). \\
\text{pure\_plane\_path}(A,B) & : = \text{pure\_plane\_hop}(A,B). \\
\text{pure\_plane\_path}(A,B) & : = \text{pure\_plane\_hop}(A,X), \text{pure\_plane\_path}(X,B), A \neq B.
\end{align*}
\]

(c) neither_alone(a,b): Find the pairs of stations (a,b) such that b can be reached from a by some combination of plane or train, but not by either train or plane alone.

\[
\begin{align*}
\text{combined\_hop}(A,B) & : = \text{train}(A,B). \\
\text{combined\_hop}(A,B) & : = \text{plane}(A,B). \\
\text{combined\_path}(A,B) & : = \text{combined\_hop}(A,X). \\
\text{combined\_path}(A,B) & : = \text{combined\_hop}(A,X), \text{combined\_path}(X,B), A \neq B.
\end{align*}
\]

\[
\begin{align*}
\text{neither\_alone}(A,B) & : = \text{combined\_path}(A,B), \neg \text{train\_path}(A,B), \neg \text{plane\_path}(A,B).
\end{align*}
\]
(iii) Show the predicate dependency graph for part (ii)
Problem 3.
Redo Problem 2(ii) in Recursive SQL (as proposed in the handouts we studied)

Note: I will assume that we always want the origin and destination to be different.

(a) train_only(a,b): Find the pairs of stations (a,b) such that one can go from a to b by train but not by plane.

CREATE VIEW train_path(a,b) AS
SELECT a, b FROM train
UNION ALL
SELECT in.a, out.b FROM train in, train_path out
WHERE in.b=out.a AND in.a≠out.b

CREATE VIEW plane_path(a,b) AS
SELECT a, b FROM plane
UNION ALL
SELECT in.a, out.b FROM plane in, plane_path out
WHERE in.b=out.a AND in.a≠out.b

CREATE VIEW train_only(a,b) AS
SELECT a, b FROM train_path
EXCEPT
SELECT a, b FROM plane_path

(b) pure_plane_path(a,b): a pure plane path from a to b is a plane itinerary from a to b such that for all consecutive stops c,d along the way, one cannot go from c to d by train. Find the pairs of stations (a,b) such that there is a pure plane path from a to b.

CREATE VIEW pure_plane_hop(a,b) AS
SELECT a, b FROM plane
EXCEPT
SELECT a, b FROM train_path

CREATE VIEW pure_plane_path(a,b) AS
SELECT a, b FROM pure_plane_hop
UNION ALL
SELECT in.a, out.b FROM pure_plane_hop in, pure_plane_path out
WHERE in.b=out.a AND in.a=out.b
(c) neither_alone(a,b): Find the pairs of stations (a,b) such that b can be reached from a by some combination of plane or train, but not by either train or plane alone.

CREATE VIEW combined_hop(a,b) AS
    SELECT a, b FROM train
    UNION
    SELECT a, b FROM plane

CREATE VIEW combined_path(a,b) AS
    SELECT a, b FROM combined_hop
    UNION ALL
    SELECT in.a, out.b FROM combined_hop in, combined_path out
        WHERE in.b=out.a AND in.a=out.b

CREATE VIEW neither_alone(a,b) AS
    SELECT a, b FROM combined_path
    EXCEPT
    (SELECT a, b FROM train_path UNION SELECT a, b FROM plane_path)
Problem 4.
Model intersection property: Let P be a positive program, and M1 and M2 be two models for P. Then M1 ∩ M2 is also a model for P.

Proof:
Assume M = M1 ∩ M2 is not a model for P. Then M does not satisfy some rule r ∈ P:

A ← A1, A2, ..., An (n ≥ 0)

Since the above formula is false, it implies that:

A ∉ M and A1, A2, ..., An ∈ M

Since M ⊆ M1 and M ⊆ M2, we have:

A1, A2, ..., An ∈ M1 and A1, A2, ..., An ∈ M2

Since both M1 and M2 are models, they satisfy this rule (with A1, A2, ..., An as the body of the rule), hence there must be an A’ in M1 and M2, which is the head of the rule:

A’ ← A1, A2, ..., An

Since A’ exists in both M1 and M2, A’ is also in M. Hence M also satisfies the rule:

A’ ← A1, A2, ..., An

This is a contradiction to the assumption that M does not satisfy this rule.
Problem 5.

This problem deals with topic covered in the guest lecture by Prof. Shvartsman, on Sep. 16. The topic deals with replicated consistent object implementation (he handed out several copies of a paper and gave web search keywords.)

In the lecture, 2-phase algorithms were given for read and write operations on atomic (aka linearizable) replicated objects. Each phase access a majority (or a quorum) of replicas.

Consider what happens if all agents have a GPS (global positioning system) device that provides global time to the agents (same time for all agents).

(a) Does this allow us to simplify the algorithm for write? If yes, how and why? If not, why not?

Yes. In the original design, the first (query) phase of write obtains timestamps from a majority of processes, and creates a new timestamp with value $V_{s+1} = \langle \text{max\_ts}+1, \text{PID} \rangle$, where max\_ts is the maximum of the timestamps it received, and PID is its own ID. In the second (propagation) phase, the process writes $V_{s+1}$ back to the majority of replicas. This algorithm ensures that all timestamps are different, globally ordered and respect real-time order.

With the new design of GPS, all processes may simply use the $V_{s+1} = \langle \text{global time}, \text{PID} \rangle$, and avoid the first phase of write. The reason is because GPS itself is the real-time order, different at each step, and global.

(b) What about the algorithm for read? If yes, how and why? If not, why not?

No. The query phase of the read algorithm obtains both the value and the timestamp. The timestamp is needed from each process in order to determine whether the process’ value is the latest one, before we can propagate it to the majority. We therefore cannot simplify the algorithm by avoiding the query phase.