QUESTION 1:

(1)

- ***Year:***

The relational model was originally proposed in **1970**.

- ***Major features:***

→**Data independence** : Relational model introduced the notions of physical and logical data independence that were not present in earlier systems where a change in the data had a direct impact in the applications.

→**Declarative query language**: With relational model, it was now possible for the final user to simply describe what he wants and the system will have to compute it by itself (the best way). In the earlier systems users has to do manual navigation to find what they want.

→**Mathematical foundation** : The queries are now expressed in relational algebra/calculus that can help for optimization

(2)

- ***Relational algebra***

$$\rho(Sneg, \pi_{sid}(\pi_{sid}(Registered)\, X\, \pi_{cid}(\sigma_{profname='Franklin'}(Courses)) - \pi_{sid,cid}(Registered)))$$

$$\pi_{sname}((\pi_{sid}(Regitered) - Sneg)\, bowtie\, Students)$$

- ***SQL:***

```
SELECT DISTINCT S.s_name,
FROM Students S
WHERE NOT EXISTS (
        SELECT C.cid
        FROM Courses C
        WHERE C.prof_name like '%Franklin%' AND NOT EXISTS (
                SELECT R.cid
                FROM Registerd R
                WHERE R.cid = C.cid AND R.sid = S.sid))
```

QUESTION 2:

(1) Access methods

- ***File scan:***

Here the number of I/o needed will be **500**.

- ***Index on city and select age on the fly:***

If we assume a uniform distribution of the 200 cities, we then hope to find 10,000/200=50 tuples qualified (in average), those records will fit in 50/20 = 3 pages.

To find them in the 3, we first need a top down search. With H = 3 the cost is equal to 2

(taken from the A3 solution).

Also a fraction of the leaf pages are qualified and if we keep the hypothesis of uniform distribution, the number of pages is 250/200 = 2.

The total I/O cost is then 2+2+3 =7

- ***Index on age and select city on the fly:***

Keeping the same logic to find the records in the tree, we first need a top down search. With H = 3 the cost is equal to 2 (taken from the A3 solution).

Also a fraction of the leaf pages are qualified and if we keep the hypothesis of uniform distribution, the number of pages is 250/50 = 5.

The difference now is the fact that we now need to pay 1 I/O for each record. If we keep the hypothesis of uniform distribution, we will have in average 10000/50 = 200 records to get.

The total I/O cost is then 2+5+200 = 207

The most efficient access method will then be ***Index on city and select age on the fly.***

(2) Join cost:

- ***block nested loops:***

Cost: Scan of outer + #outer blocks * scan of inner (outer is the smaller relation)

Cost = 500 + [500/32-2]*800 = 13,833 I/Os

- ***sort-merge join:***

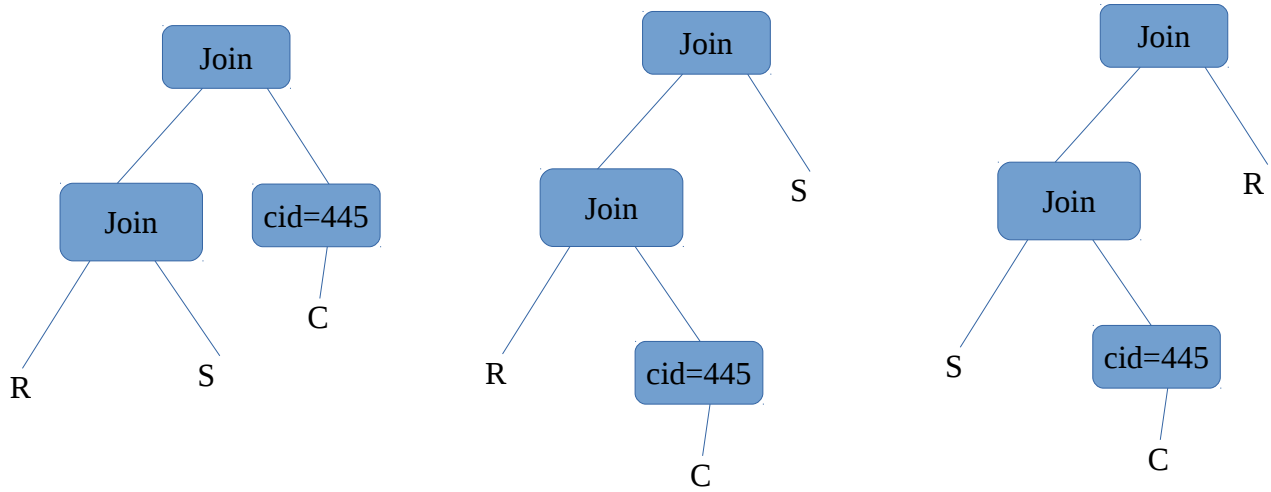
U = max(M, N) = 800 and we have B=32>sqrt(U).

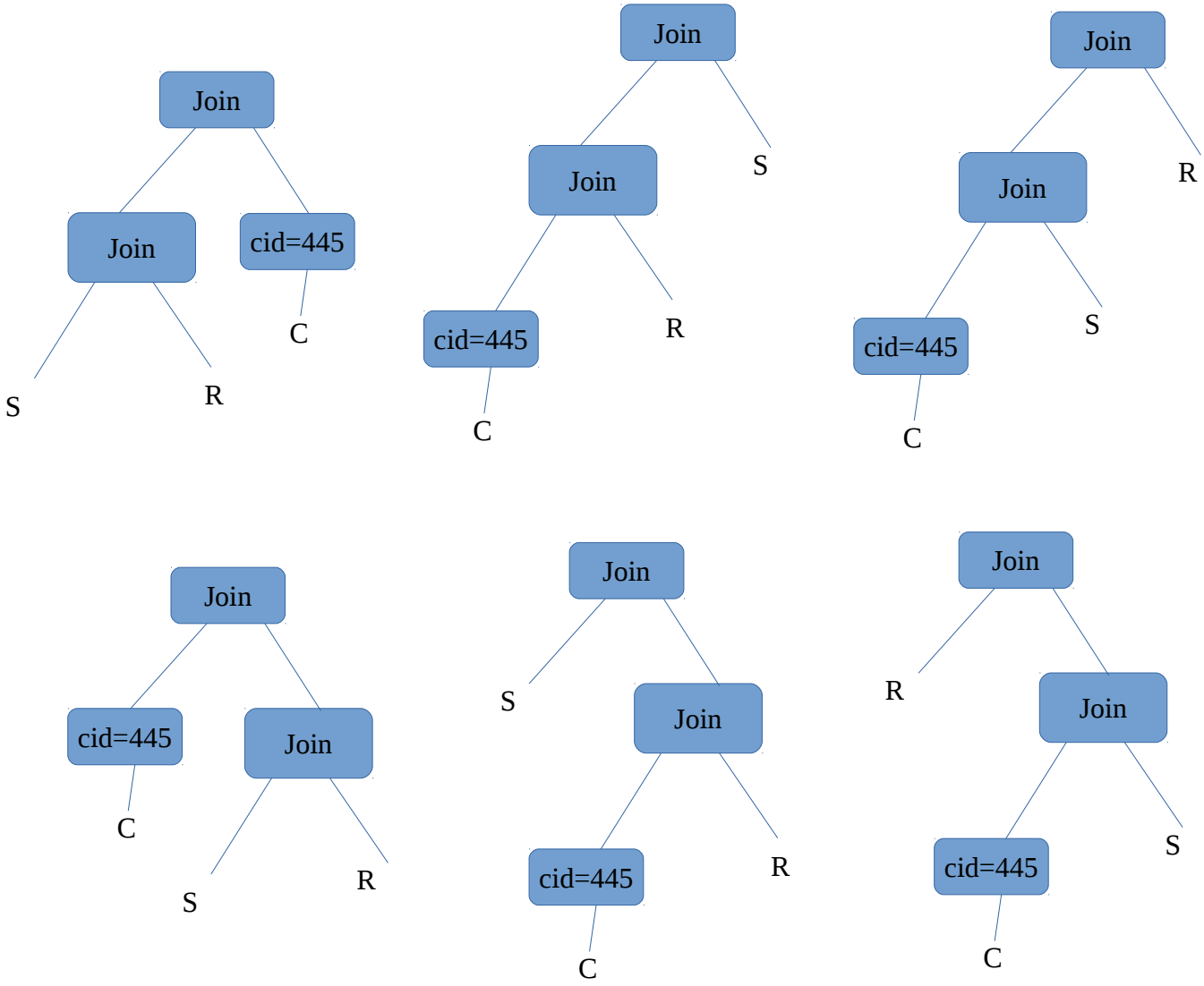
→ Cost = 3*(800+500) = 3,900 I/Os

- ***hash join:***

For the given memory the I/O cost will also be less than 3*(800+500) = 3,900 I/Os

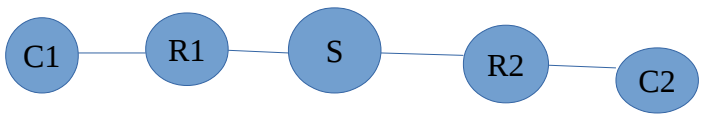
(3) left-deep query plans:





Those are the plans for different order. To those plans we have to add combinations of different access methods (sequential, use of index, …) and the different types of joints possible (has,h, sort-merge, ...)

(3) Query:



- ***k=2:***

find the best plans for C1-R1, R1-C1, R1-S, S-R1, S-R2, R2-S, R2-C2, C2-R2 = **8 plans considered.**

Store (C1-R1), (R1-S), (S-R2), (R2-C2) = 4

- ***k=3:***

find the best: (C1-R1)-S, S-(C1-R1), C1-(R1-S), (R1-S)-C1, (R1-S)-R2, R2-(R1-S), (S-R2)-C2, C2-(S-R2), (S-R2)-R1, R1-(S-R2), (R2-C2)-S, S-(R2-C2) = 12 **plans considered.**

Store : (C1-R1-S), (R1-S-R2), (S-R2-C2) = 3

- ***k=4:***

find the best plans for (C1-R1-S)-R2, R2-(C1-R1-S), (R1-S-R2)-C2, C2-(R1-S-R2), (R1-S-R2)-C1, C1-(R1-S-R2) = 6 **plans considered**

store (C1-R1-S-R2), (R1-S-R2-C2) = 2

- ***k=5:***

find the best plans for (C1-R1-S-R2)-C2, C2-(C1-R1-S-R2), (R1-S-R2-C2)-C1, C1-(R1-S-R2-C2) = 4 **plans considered**

Store (C1-R1-S-R2-C2) =1

QUESTION 3:

(1) V usable:

V is not enough to answer Q. But we can join it with the table advise to answer Q.
Q can be rewrite as
Select Advises.prof, Advises.student, V.quarter
From V, Advises
Where
Advises.prof=V.prof and
Advises.student=V.student

(2)

(a) I don't think that V is usable to answer Q because in V we have the constraint that the teacher has thougth the student (has thougth a course undertook by the student)

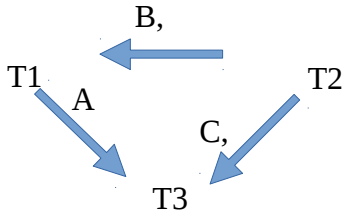
(b) For Q" I will just add the constraint Advises.prof=Teaches.prof

(c) With those index, computing V can be very efficient (joins algorithm). We just have to write tuples on disk sorted by that index to make sure it is clustered.
To compute Q" the join with the view V will be more efficient.

QUESTION 4:

(1)

- ***Precedence graph:***



The graph is acyclic so it is conflict serializable.
The equivalent serial schedule is T2 → T1 → T3

(2) Sequence of actions

Time Stamp	Action
1	T1:S(A)
2	T1:R(A)
3	T2:S(B)
4	T2:R(B)
5	T1:X(B)
6	T1: blocked
7	T2: X(C)
8	T2: W(C)
9	T3: S(C)
10	T3: blocked
11	T2 commit and release locks
12	T1: resumed (with lock on B)
13	T1: W(B)
14	T3: resumed (with lock on c)
15	T3: R(C)
16	T3: X(A)
17	T3: blocked
18	T1 commit and release locks
19	T3 resumed with lock on A
20	T3: W(A)
21	T3: commit and releases locks

(3) Crash recovery

We assume we don't have to redo T1 and T2 because their changes have been written to disk

LSN	XACTID	Type	Data
1	T2	Update	C
2	T2	Commit	-
3	T1	Update	B
4	T1	Commit	
5	T3	Update	A
6	T3	Abort	
7	T3	CLR	A

8	T3	end	
9			

QUESTION 4:

(1) Schema:

```
create table user(
        uid int,
        sname varchar(30),
        address varchar(40),
        primary key (uid));

create table friend(
        uid1 int,
        uid2 int,
        date DateTime,
        primary key (uid1, uid2)
        foreign key (uid1) references user
        foreign key (uid2) references user );

create table post(
        pid int,
        type varchar(30),
        date DateTime,
        content longtext,
        primary key (pid));

create table user_post(
        uid int,
        pid int,
        type varchar(30),
        primary key (uid,pid),
        foreign key (uid) references user,
        foreign key (pid) references post);

create table message(
        mid int,
        date DateTime,
        content longtext,
        primary key (mid));

create table msg_participant(
        uid int,
        mid int,
        primary key (uid,mid),
        foreign key (uid) references user,
        foreign key (mid) references post);
```

```
create table page(
        url varchar(30),
        owner int,
        title varchar(20),
        content longtext,
        primary key (pageid),
        foreign key (owner) references user);

create table click(
        uid int,
        url varchar(30),
        date DateTime,
        primary key (uid,mid,date),
        foreign key (uid) references user,
        foreign key (url) references page);
```

(2) OLTP/OLAP:

The first task is typically an OLTP because it consists of frequents updates of the tables.
The second however needs an OLTP part (frequent update/retrieval of the pages), and an OLAP for the recommendation part.

(3) DBMS:

If I was to design that database I will try to use a mix of all those DBMS types.
- A distributed OLTP (relational DBMS) : For frequents updates (add a user, delete friendship, …). It will implement the relational schema above
- Parallel sytems and map-reduce : Manipulate all those information (Volume and Velocity) can not be handle by a single server. Tons of servers have to run in parallel to be able to deliver a good service (without latency). Also the users are probably located around the world. So it is better to have different servers located in different regions.
The use of map-reduce will necessary for certain tasks because it will help to process this large amount of data and use the parallel servers efficiently.
- OLAP : For decision-making (recommendation, link prediction, users segmentation, ...)

(4) Optimizer:

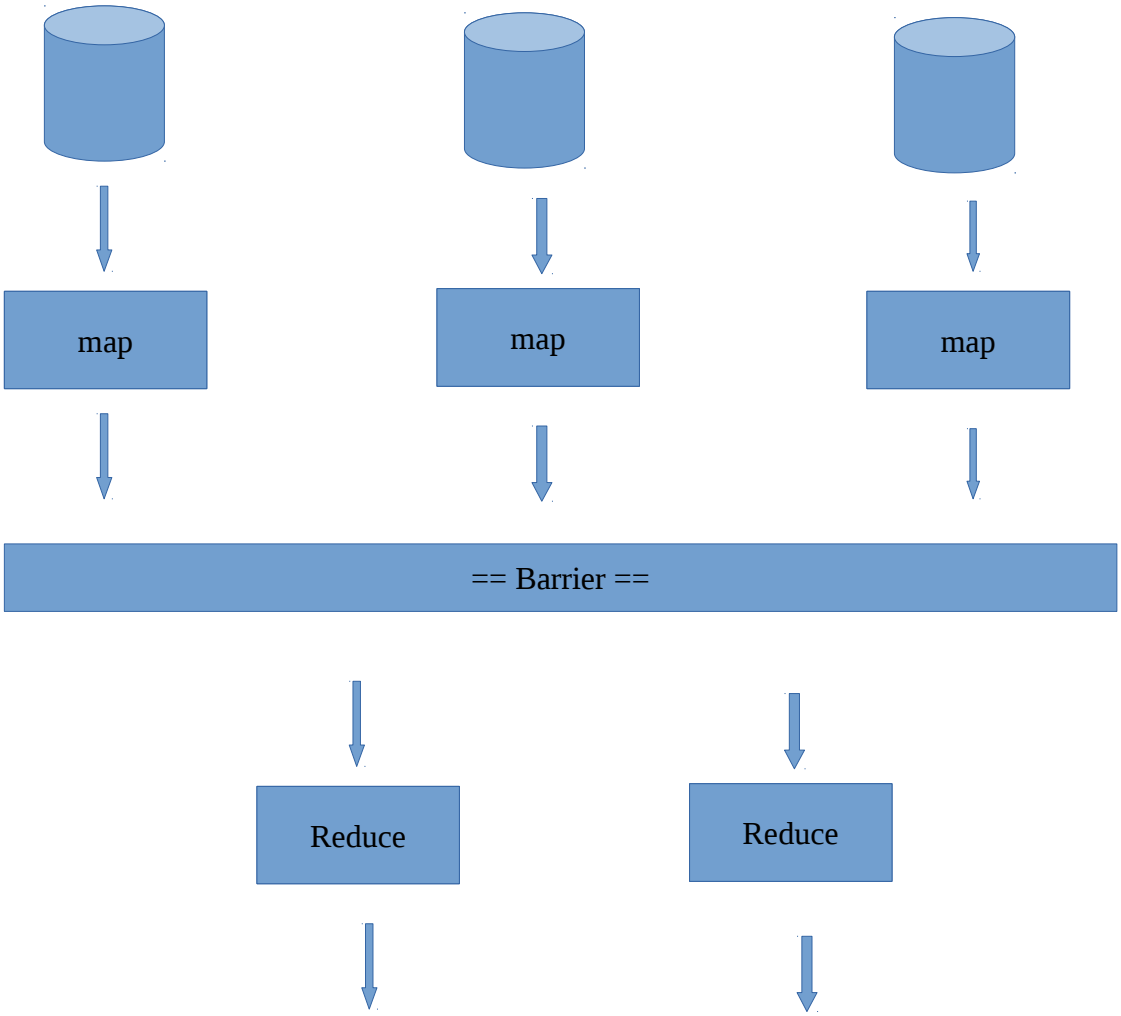
I don't think that system R style optimizer is optimal for parallel programming because it does not write intermediate results to temporary files. This is bad if one of the nodes fails, for recovery we will have to start again.
Also, in parallel processing the number of plans to compare will quickly become impracticable.

(4) First try using map reduce:

Input: Each node contains tuples of the form <key, value> where key = <user_id, session_id> and value = {list of page_id's in the session}

Map: Emit <key,value> where key = all subsets of page_id's for a given line, Value = 1

Reduce : sum value for given <key, value> tuples. Write if the sum is greater than 1 million


map
map
map
== Barrier ==
Reduce
Reduce