BCNF Decomposition

Example: Decompose into BCNF - **Restaurant(id, name, rating, popularity, rec)**

1. **id → name, rating**
2. **rating → popularity**
3. **popularity → rec**

This example is covered in both the section slides and the lecture slides.

Given R(A, B, C, D, E), and functional dependencies: A → C, BD → A, D → E

1. Find the following closures: {A}+, {B}+, {D}+, and {BD}+
   A^+ = {A, C}
   B^+ = {B}
   D^+ = {D, E}
   BD^+ = {A, B, C, D, E}

2. Decompose R into BCNF. In each step, explain which functional dependency you used to decompose and explain why further decomposition is needed. Your answer should consist of a list of table names and attributes. Make sure you indicate the keys for each relation.

   There are multiple ways to break down {ABCDE} depending on what FD you do first.

   R1{ABCDE}

   D -> E
   R2{D, E} R3{A, B, C, D}

   A -> C
   R2{D, E} R4{A, C} R5{A, B, D}

   —----------

R1{ABCDE}

A -> C
R2{A, C} R3{A, B, D, E}

D -> E
R2{A, C} R4{D, E} R5{A, B, D}

—----------

R1{A, B, C, D, E}

BD -> ABCDE
This gives us back the original table. But because of our other FD's, this is not in BCNF. Use the other FDs to break this down further.

Relational Algebra

RA Operators:

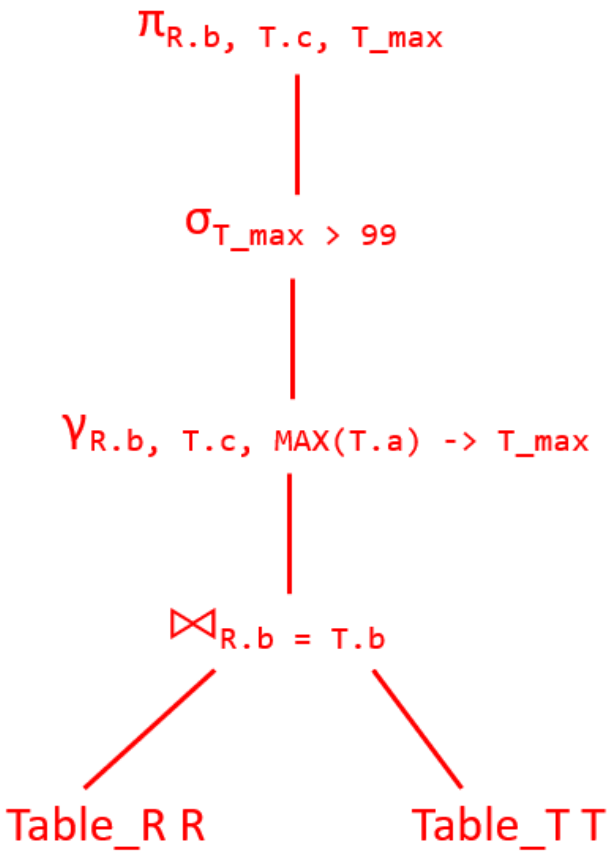
σ = Select (WHERE/HAVING) π = Project (SELECT)

⋈ = Natural Join γ = Group/Aggregation

δ = Duplicate Elimination (DISTINCT)

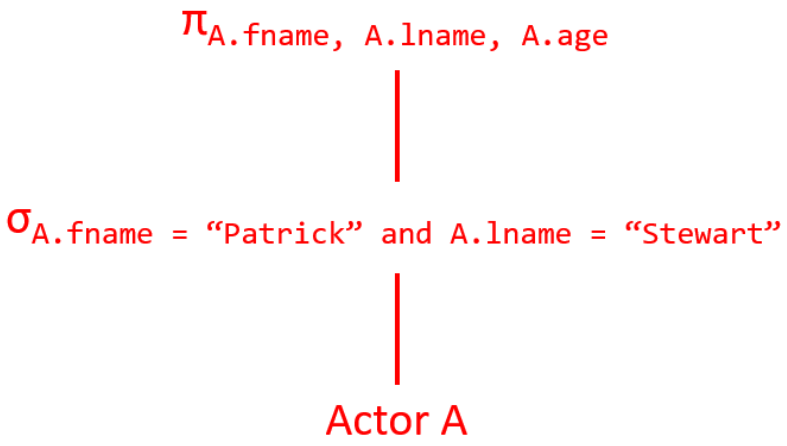
Example: Make this SQL query into RA (remember FJWGHOS)

```
SELECT R.b, T.c, max(T.a) AS T_max
  FROM Table_R AS R, Table_T AS T
 WHERE R.b = T.b
 GROUP BY R.b, T.c
HAVING max(T.a) > 99
```

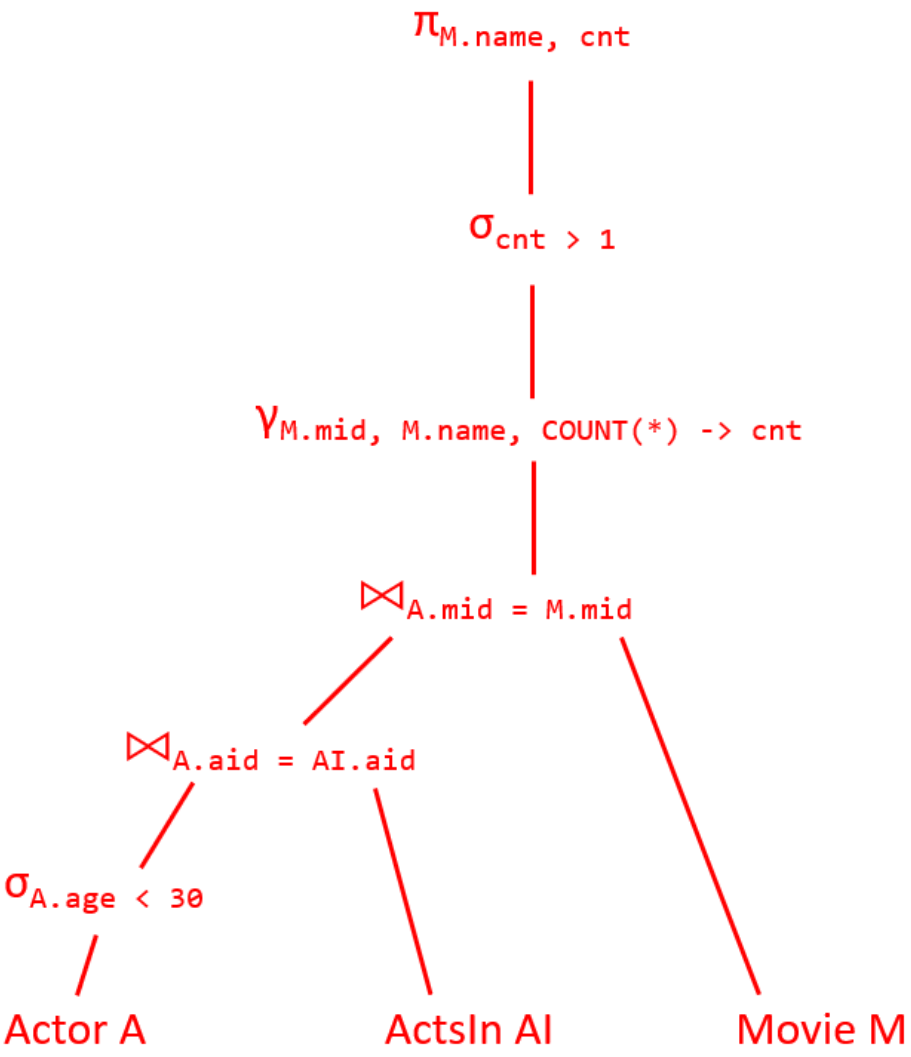


Convert the following SQL queries into logical RA plans, given the following schemas:
Actor(aid, fname, lname, age)
ActsIn(aid, mid)
Movie(mid, name, budget, gross)

1.
```
SELECT A.fname, A.lname, A.age
FROM Actor AS A
WHERE A.fname = "Patrick"
      AND A.lname = "Stewart";
```



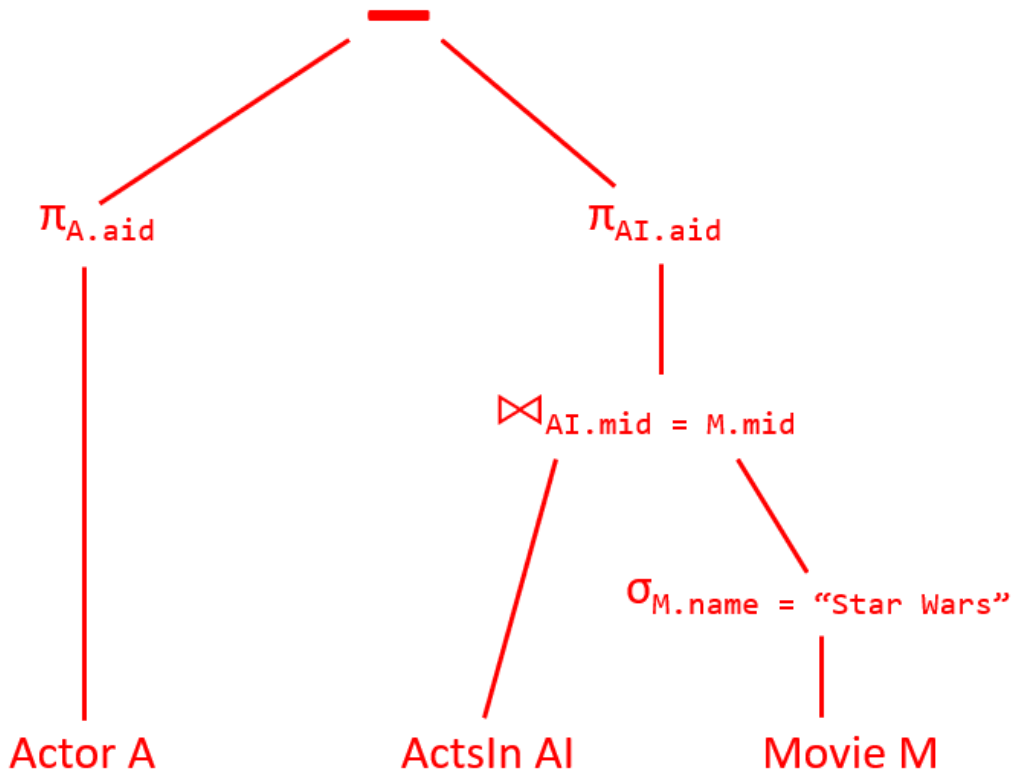
2.
```
SELECT M.name, COUNT(*) AS cnt
FROM Actor AS A, ActsIn In AS AI, Movie AS M
WHERE A.aid = AI.aid AND M.mid = AI.mid
      AND A.age < 30
GROUP BY M.mid, M.name
HAVING COUNT(*) > 1;
```



3.
```
SELECT A.aid
FROM Actor AS A
WHERE NOT EXISTS (
      SELECT *
      FROM ActsIn AS AI, Movie AS M
      WHERE AI.mid = M.mid AND AI.aid = A.id AND M.name = “Star Wars”
);
```

We will begin by rewriting this query to an equivalent query that will be easier to make an RA plan for:

```
SELECT A.aid
FROM Actor AS A
WHERE A.aid NOT IN (
      SELECT AI.aid
      FROM ActsIn AS AI, Movie AS M
      WHERE AI.mid = M.mid AND ~~AI.aid = A.id AND~~ M.name = “Star Wars”
);
```



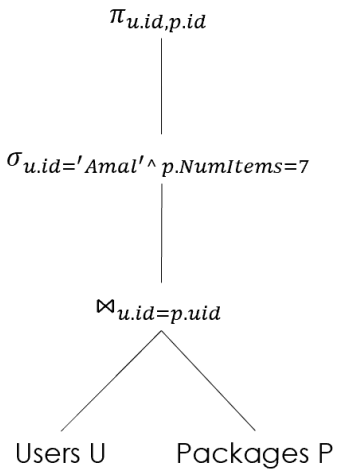
1. RA to RA

Consider the fact that Amazon has shipped several billion packages over the course of its >20y history and that it may surpass 10B packages by 2030. Assume that it tracks its packages and users using the following schema:

Packages(<u>PackageID</u>, UserID, DestAddress, NumItems)

Users(<u>UserID</u>, CreditCardNumber, Languages)

Now, consider the following RA tree:

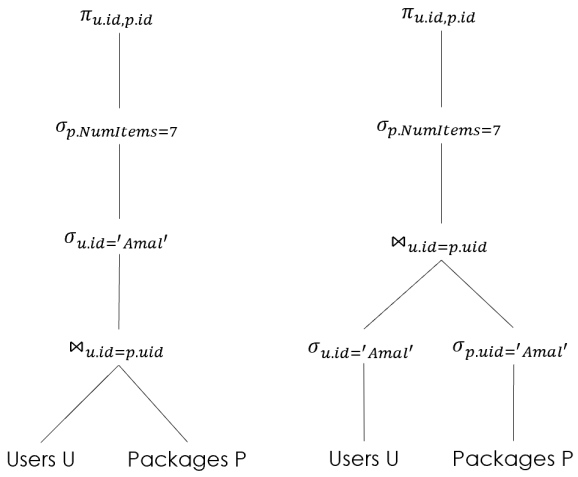


You may notice how, although the PACKAGES table is very very large (10B!!), an individual user may have a very small number of rows. Generate a logically-equivalent tree which, ideally, takes advantage of this fact.

First, use the WHERE-AND rewrite rule to split the conjunction into two sequential operators.

Next, push the u.id selection down into both tables. We don't actually have to push the selection into USERS (since it's the PACKAGES table which has such an advantageous selection) but there's also no harm in reducing the number of tuples we send to the equijoin. Note how we were able to do this because the equijoin predicate used the same attribute as the selection predicate; note also that we need to reference a differently-named attribute in Users.

Lastly, we can optionally push the p.numItems selection down into PACKAGES table and reconstruct the conjunction.


$\pi_{u.id,p.id}$
$\sigma_{p.NumItems=7}$
$\sigma_{u.id='Amal'}$
$\bowtie_{u.id=p.uid}$
Users U
Packages P
$\pi_{u.id,p.id}$
$\sigma_{p.NumItems=7}$
$\bowtie_{u.id=p.uid}$
$\sigma_{u.id='Amal'}$
$\sigma_{p.uid='Amal'}$
Users U
Packages P