CS186 Discussion #9

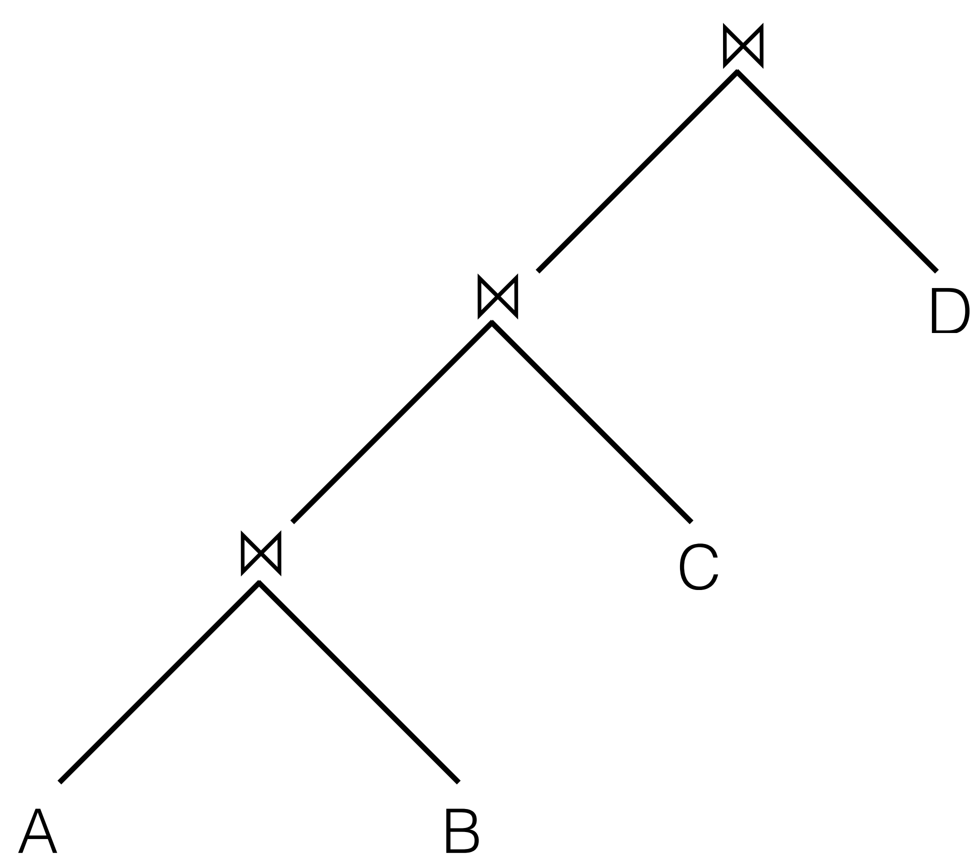
(Query Optimization)

Query Optimization

- What is the best way to run a query?
- Change order and methods of operators for:
  - Faster queries, better resource utilization
  - Smaller # of total I/Os

Plan Space

- Based on relational equivalences
- Only consider left-deep join trees
  - Includes all join orders and join methods



Determinants of Plan Cost

- Access method of base tables
  - Scan, index, range vs. lookup, clustered vs. unclustered

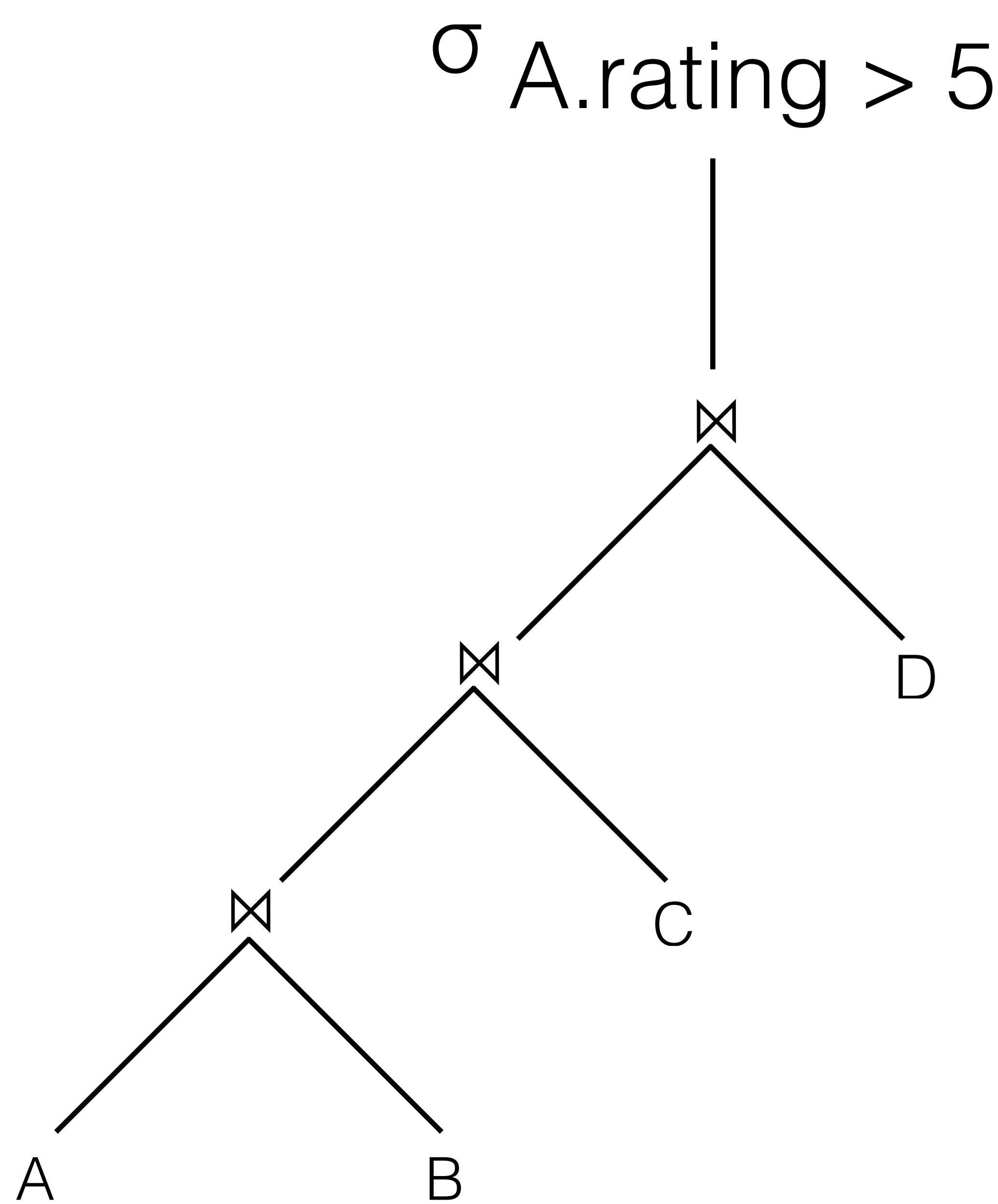
- Join ordering
  - Do we want to keep rereading a big table over and over again?

- Join method
  - Sort-merge? Hash? BNL?

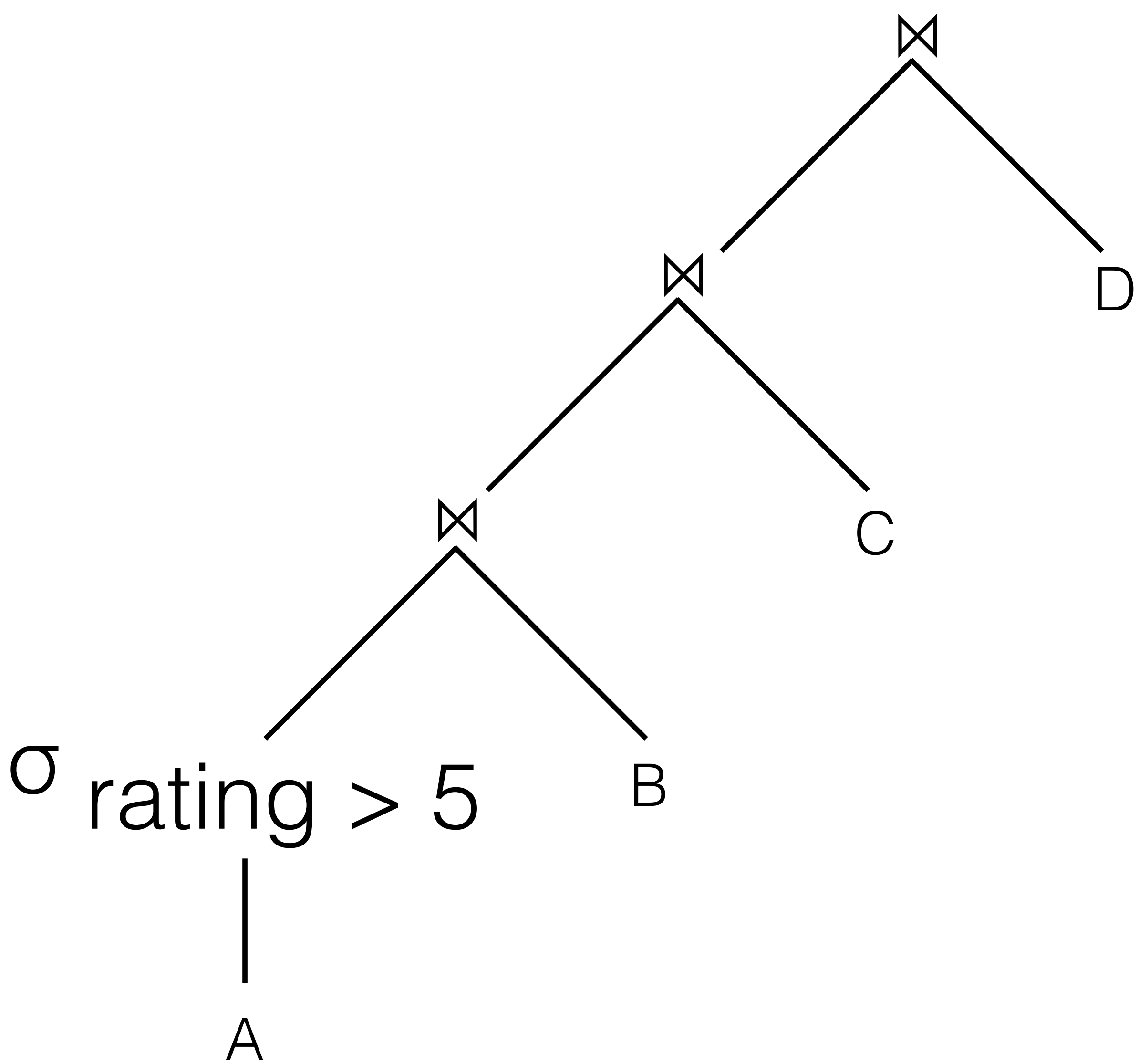
Cost Estimation

- Estimate cost of each operation in plan tree
- Must estimate size of result for each operation using info from system catalog
- For System R, cost: #I/Os + CPU-factor * #tuples
- On-the-fly: The result of one operator is pipelined to another operator without creating a temporary table to hold intermediate result

Push Select



Push Select



Selectivity

- Selectivity represents a predicate's impact on reducing result size
  - |output| / |input|
  - Tuples that contain rating 0 to 100:
    - $\sigma_{\text{rating} > 0}$ has large selectivity
    - $\sigma_{\text{rating} > 99}$ has smaller selectivity
- If missing info to estimate selectivity, assume 1/10!
- Selectivity also known as reduction factor (RF)

Selectivity

- Predicate col=value
  - Selectivity = 1/NKeys(col)
- Predicate col1=col2
  - Selectivity = 1/MAX(NKeys(col1), NKeys(col2))
- Predicate col>value
  - Selectivity= (High(col)-value)/(High(col)-Low(col) + 1)
- Assumes that values and uniformly distributed and independent!
- Result Cardinality: Max # tuples * product of all selectivities

Histograms

- Represent distribution of one or more attributes
- Better estimate of selectivity than assuming uniformity
  - Can be Gaussian (Normal), exponential, etc.
- Think of as lossy compression, more buckets, more accurate

```
Car(license, owner_ssn, year, company, model)
Accident(license, accident_date, damage_amount,
                 zipcode)
Owner(ssn, license, name, gender, street, city,
                 zipcode)
```

- For the query: “SELECT * FROM Accident A, Car C WHERE A.license = C.license AND A.damage_amount > X;” For what types of values of X would selection push-down significantly improve the cost of the query (Car is the inner table of the join)?”

```
Car(license, owner_ssn, year, company, model)
Accident(license, accident_date, damage_amount,
zipcode)
Owner(ssn, license, name, gender, street, city,
zipcode)
```

- For the query: “SELECT * FROM Accident A, Car C WHERE A.license = C.license AND A.damage_amount > X;” For what types of values of X would selection push-down significantly improve the cost of the query (Car is the inner table of the join)?”

- Selection push-down will help with very large values of X, since that would be more selective, and thus result in fewer resulting tuples for the rest of the plan.

```
Car(license, owner_ssn, year, company, model)
Accident(license, accident_date, damage_amount,
zipcode)
Owner(ssn, license, name, gender, street, city,
zipcode)
```

- For the query: "SELECT O.name FROM Car C, Owner O WHERE C.license = O.license AND C.company = 'Volvo';" What is the expected cardinality of the Car relation after the initial selections are applied (before the join)?

NTuples(Car) = 1000 ; NPages(Car) = 100
NTuples(Accident) = 500 ; NPages(Accident) = 20
NTuples(Owner) = 800 ; NPages(Owner) = 50
NDistinct(Car.company) = 50;

```
Car(license, owner_ssn, year, company, model)
Accident(license, accident_date, damage_amount,
zipcode)
Owner(ssn, license, name, gender, street, city,
zipcode)
```

- For the query: “SELECT O.name FROM Car C, Owner O WHERE C.license = O.license AND C.company = ‘Volvo’;” What is the expected cardinality of the Car relation after the initial selections are applied (before the join)?

- You can only push down the Car.company = ‘Volvo’ selection predicate. NDistinct(Car.company) = 50, so we can estimate Selectivity(Car.company) = 1/50. Cardinality(Car.company = ‘Volvo’) = Selectivity(Car.company) * NTuples(Car) = 1000 / 50 = 20

Selinger

- System R Optimizer
- Prunes search space by looking at left-deep plans only
- Minimize cost of plans: I/Os + f*CPUs
- Using selectivity estimates, estimate cost for each operator - sum them up
- “Interesting Orders” change plan cost
- Dynamic programming to search

Interesting Orders

- Operator returns an “interesting order” if its result is in order of:
  - some *ORDER BY* attribute
    - means we don’t have to sort later!
  - some *GROUP BY* attribute
    - means we can use a nice scan method for our group-by later!
  - some Join attribute of other joins
    - Means we can use sort-merge far cheaper!

Search Algorithm

- Find the best 1-table access method.
- Given the best 1-table method as the outer, find the best 2-table.
- …
- Given the best (N-1)-table method as the outer, find the best N-table.
- ** Instead of “strictly the best” we return the best for each interesting order of the tuples.
- ** Do cross products last!

Example

```
Select S.sid, COUNT(*) AS number
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid
AND B.color = “red”
GROUP BY S.sid
```

Sailors:

- Hash, B+ on sid

Reserves:

- Clustered B+ tree on bid
- B+ on sid

Boats

- B+ on color

Example

Find the best 1-table access method for each relation.

```
Select S.sid, COUNT(*) AS number
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid
AND B.color = "red"
GROUP BY S.sid
```

Sailors:

- Hash, B+ on sid

Reserves:

- Clustered B+ tree on bid
- B+ on sid

Boats

- B+ on color

Example

Find the best 1-table access method for each relation.

- Sailors, Reserves: File Scan
  - B+ tree on Reserves.bid as interesting order
  - B+ tree on Sailors.sid as interesting order
- Boats: B+ tree on color

```
Select S.sid, COUNT(*) AS number
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid
AND B.color = "red"
GROUP BY S.sid
```

Sailors:

Hash, B+ on sid

Reserves:

Clustered B+ tree on bid

B+ on sid

Boats

B+ on color

Example

Given the best 1-table method as the outer, find the best 2-table.

- Sailors, Reserves: File Scan
    - B+ tree on Reserves.bid as interesting order
    - B+ tree on Sailors.sid as interesting order
- Boats: B+ tree on color

Example

Given the best 1-table method as the outer, find the best 2-table.

- Sailors, Reserves: File Scan
  - B+ tree on Reserves.bid as interesting order
  - B+ tree on Sailors.sid as interesting order
- Boats: B+ tree on color

- File Scan Reserves (outer) with Boats (inner)
- File Scan Reserves (outer) with Sailors (inner)

Example

Given the best 1-table method as the outer, find the best 2-table.

- Sailors, Reserves: File Scan
  - B+ tree on Reserves.bid as interesting order
  - B+ tree on Sailors.sid as interesting order
- Boats: B+ tree on color

- Reserves Btree on bid (outer) with Boats (inner)
- Reserves Btree on bid (outer) with Sailors (inner)

Example

Given the best 1-table method as the outer, find the best 2-table.

- Sailors, Reserves: File Scan
  - B+ tree on Reserves.bid as interesting order
  - B+ tree on Sailors.sid as interesting order
- Boats: B+ tree on color

- File Scan Sailors (outer) with Boats (inner)
- File Scan Sailors (outer) with Reserves (inner)

Example

Given the best 1-table method as the outer, find the best 2-table.

- Sailors, Reserves: File Scan
  - B+ tree on Reserves.bid as interesting order
  - B+ tree on Sailors.sid as interesting order
- Boats: B+ tree on color

- B+ tree Sailors (outer) with Boats (inner)
- B+ tree Sailors (outer) with Reserves (inner)

Example

Given the best 1-table method as the outer, find the best 2-table.

- Sailors, Reserves: File Scan
  - B+ tree on Reserves.bid as interesting order
  - B+ tree on Sailors.sid as interesting order
- Boats: B+ tree on color

- Boats Btree on color with Sailors (inner)
- Boats Btree on color with Reserves (inner)

Example

Given the best 1-table method as the outer, find the best 2-table.

- File Scan Reserves (outer) with Boats (inner)
- File Scan Reserves (outer) with Sailors (inner)
- Reserves Btree on bid (outer) with Boats (inner)
- Reserves Btree on bid (outer) with Sailors (inner)
- File Scan Sailors (outer) with Boats (inner)
- File Scan Sailors (outer) with Reserves (inner)
- B+ tree Sailors (outer) with Boats (inner)
- B+ tree Sailors (outer) with Reserves (inner)
- Boats Btree on color with Sailors (inner)
- Boats Btree on color with Reserves (inner)

- Retain cheapest plan for each (pair of relations, order)

Worksheet

What are the best single-table plans?

What are the best single-table plans?

- Kitties: B+ tree on cuteness
  - File scan = 100
  - B+ tree = (5+400)*1/10
- Puppies: B+ tree on yappiness
  - File scan = 50
  - B+ tree = (5+200)*1/10
- Humans: File Scan
  - File scan = 1000
  - B+ tree = (20 + 50,000)*1,200/50,000)

List the pairs of tables the optimizer will consider for 2-way joins

List the pairs of tables the optimizer will consider for 2-way joins

- Kitties[unclustered B+] ⋈ Puppies
- Kitties[unclustered B+] ⋈ Humans
- Puppies[unclustered B+] ⋈ Kitties
- Puppies[unclustered B+] ⋈ Humans
- Humans[file scan] ⋈ Kitties
- Humans[file scan] ⋈ Puppies

Which plans will be avoided?

- Kitties[unclustered B+] ⋈ Puppies
- Kitties[unclustered B+] ⋈ Humans
- Puppies[unclustered B+] ⋈ Kitties
- Puppies[unclustered B+] ⋈ Humans
- Humans[file scan] ⋈ Kitties
- Humans[file scan] ⋈ Puppies

Which plans will be avoided?

- Kitties[unclustered B+] ⋈ Puppies
- ~~Kitties[unclustered B+] x Humans~~
- Puppies[unclustered B+] ⋈ Kitties
- Puppies[unclustered B+] ⋈ Humans
- ~~Humans[file scan] x Kitties~~
- Humans[file scan] ⋈ Puppies

Humans and kitties don't have a join predicate!

What would be the IO cost of doing index nested loops join using Puppies as the outer, with the optimal single table selection methods?

What would be the IO cost of doing index nested loops join using Puppies as the outer, with the optimal single table selection methods?

- Index scan: (5+200)*1/10 = ~21 I/Os to select puppies
- # selected puppies = 200 * 1/10 = 20
- 20*(5+400)*1/10 = 810 I/Os
- 21 + 810 = 831 I/Os

#tuples in puppies * index lookup on K.cuteness = P.yappiness

What would be the IO cost of doing index nested loops join using Kitties as the outer, with the optimal single table selection methods?

What would be the IO cost of doing index nested loops join using Kitties as the outer, with the optimal single table selection methods?

- (5+400)*1/10 = 41 I/Os to select kitties
- Clustered lookup for (owner = val and yappiness = 7)
  - (15+50)*(1/10)*(1/10) = 1 I/O — just an estimate!
- 41 + 40*1 = 81 I/Os

Have a great spring break!