Introduction to Data Management CSE 344

Lecture 12: Relational Calculus

Midterm

- Monday, February 8th in class
- Content
  - Lectures 1 through 13
  - Homework 1 through 4 (due Feb 10)
  - Webquiz 1 through 4 (due Feb 6)

- Closed book. No computers, phones, watches, etc.!

- Can bring one letter-sized piece of paper with notes
  - Can write on both sides

How to Study?

- Lecture slides and section materials
- Homework 1 through 4
- Past midterms posted on website
  - Lots of great examples! With solutions
  - But content changes between quarters
    - So some questions may not apply
    - We may have some new questions not present in past
- Practice Webquiz on gradience

Today's Outline

- Finish cost estimation

- Relational Calculus

- Wednesday: datalog (Laurel)
- Friday: midterm review (Jay)

Page-at-a-time Refinement

```
for each page of tuples r in R do
   for each page of tuples s in S do
         for all pairs of tuples t1 in r, t2 in s
               if t1 and t2 join then output (t1,t2)
```

- Cost: B(R) + B(R)B(S)

Block-Nested-Loop Refinement

```
for each group of M-1 pages r in R do
   for each page of tuples s in S do
         for all pairs of tuples t1 in r, t2 in s
               if t1 and t2 join then output (t1,t2)
```

- Cost: B(R) + B(R)B(S)/(M-1)

Index Nested Loop Join

$R \bowtie S$

- Assume S has an index on the join attribute
- Iterate over R, for each tuple fetch corresponding tuple(s) from S

- Cost:
  - If index on S is clustered: $B(R) + T(R)B(S)/V(S,a)$
  - If index on S is unclustered: $B(R) + T(R)T(S)/V(S,a)$

Sort-Merge Join

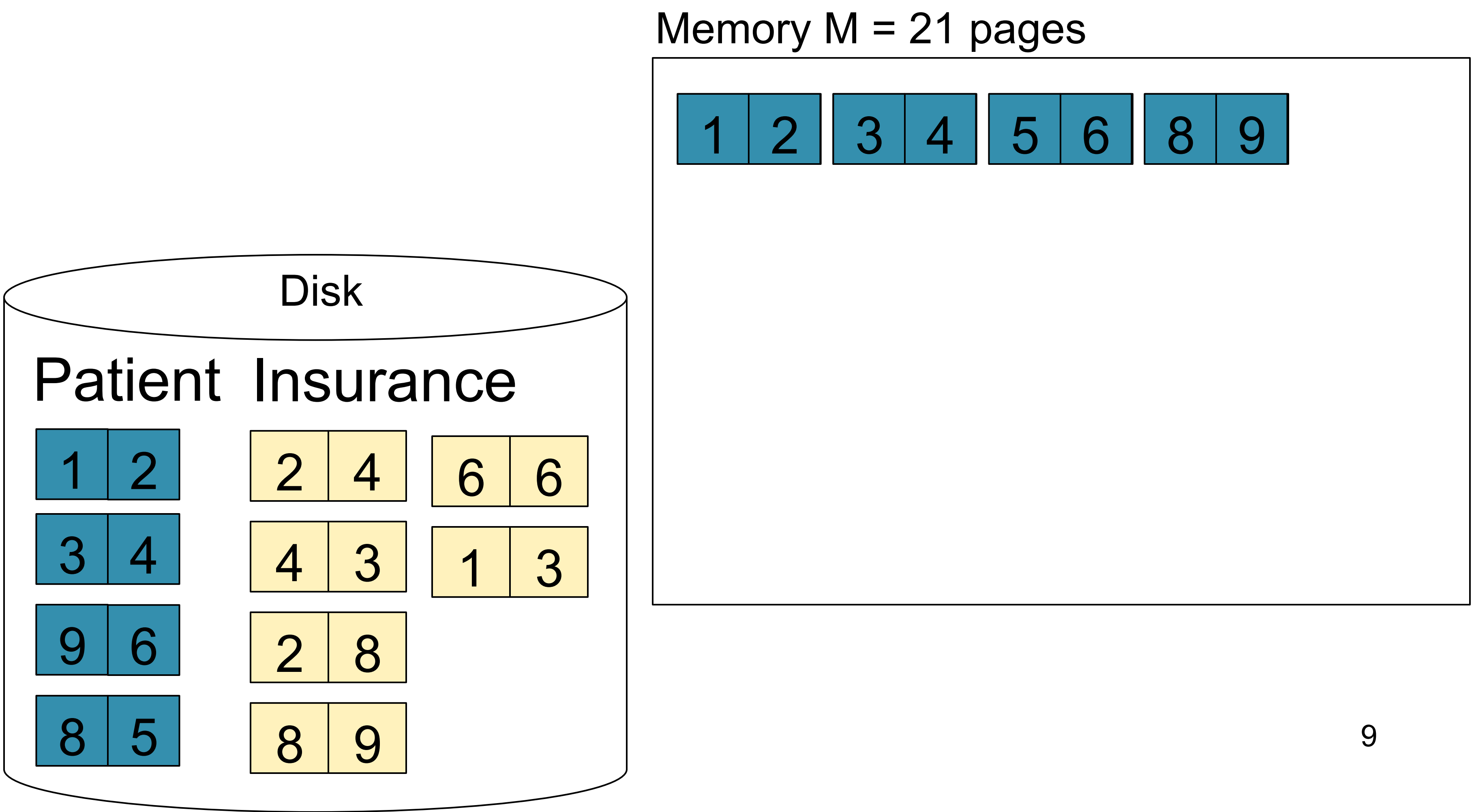
Sort-merge join: R $\bowtie$ S

- Scan R and sort in main memory
- Scan S and sort in main memory
- Merge R and S

- Cost: B(R) + B(S)
- One pass algorithm when B(S) + B(R) <= M
- Typically, this is NOT a one pass algorithm

Sort-Merge Join Example

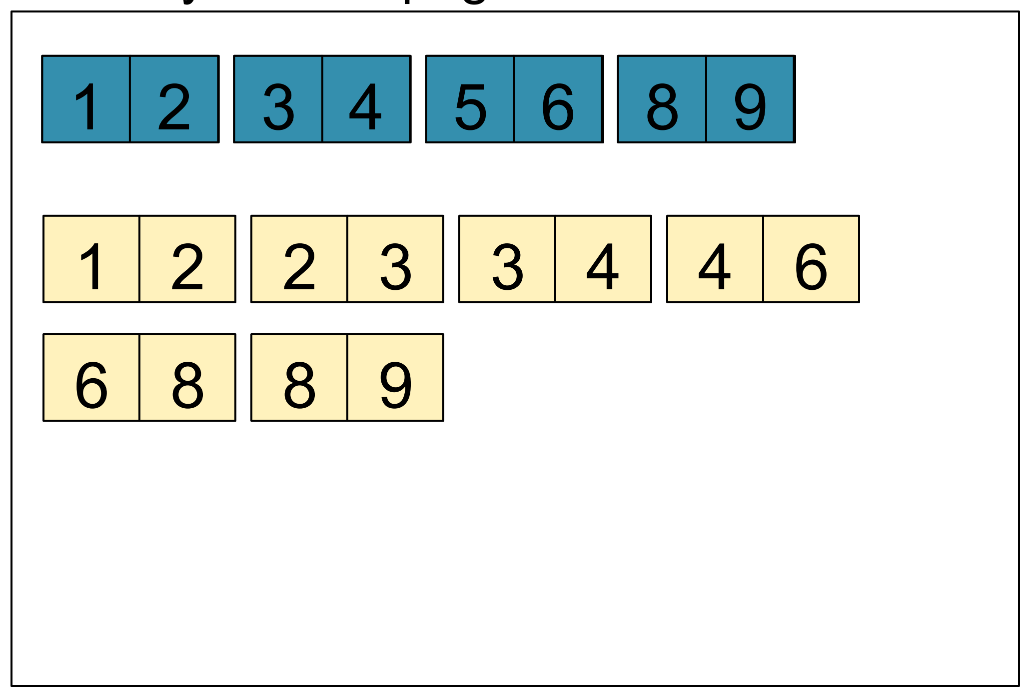
Step 1: Scan Patient and sort in memory

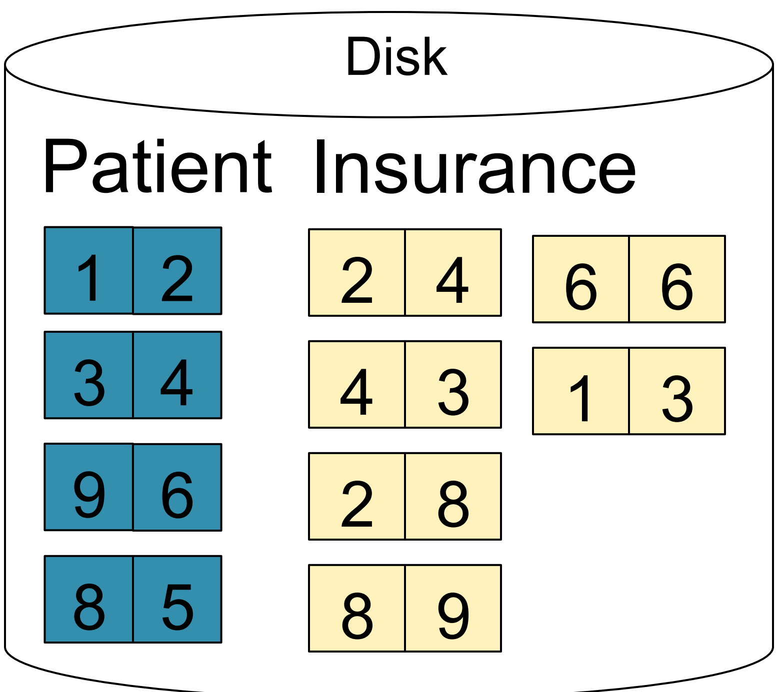


Sort-Merge Join Example

Step 2: Scan Insurance and sort in memory

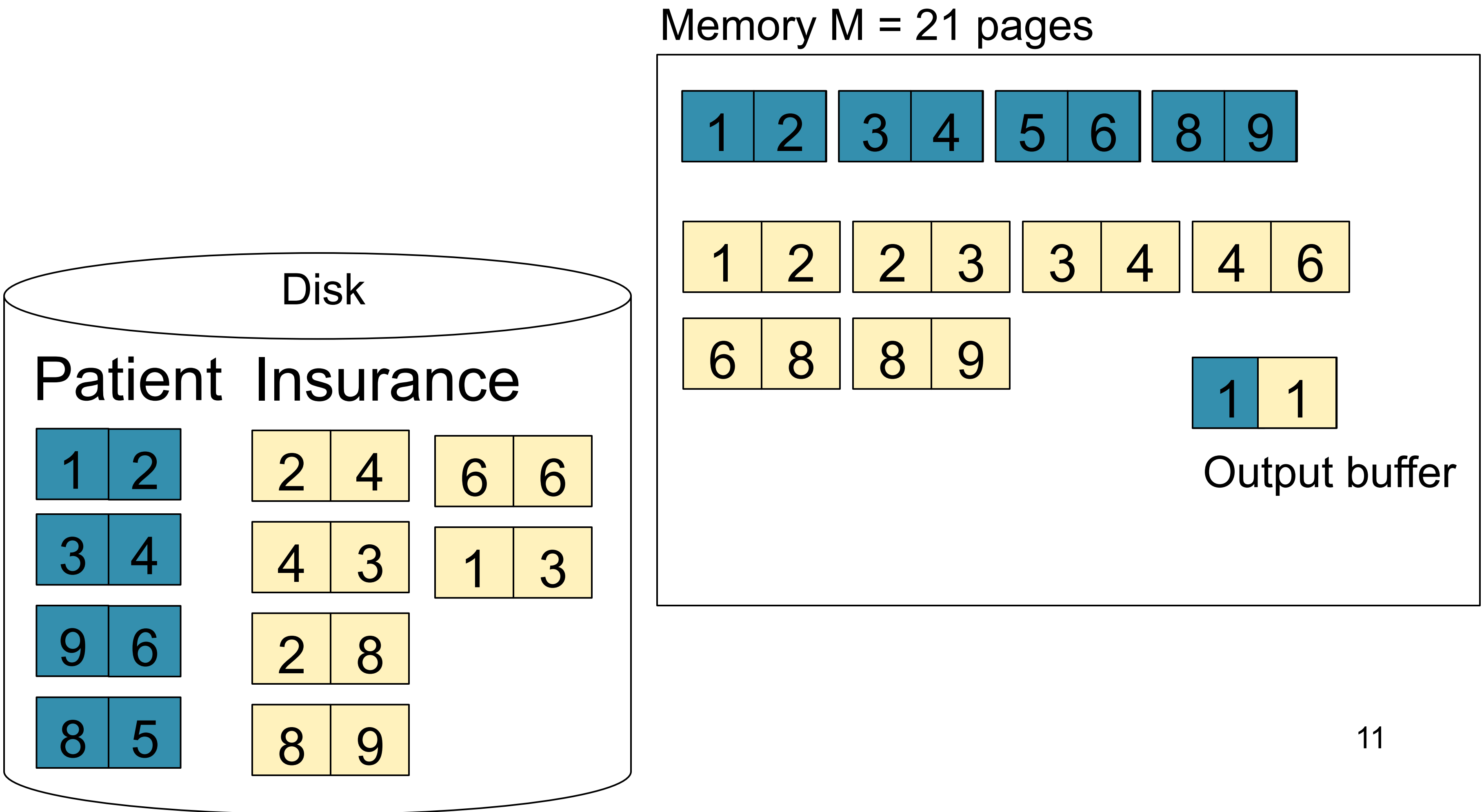
Memory M = 21 pages





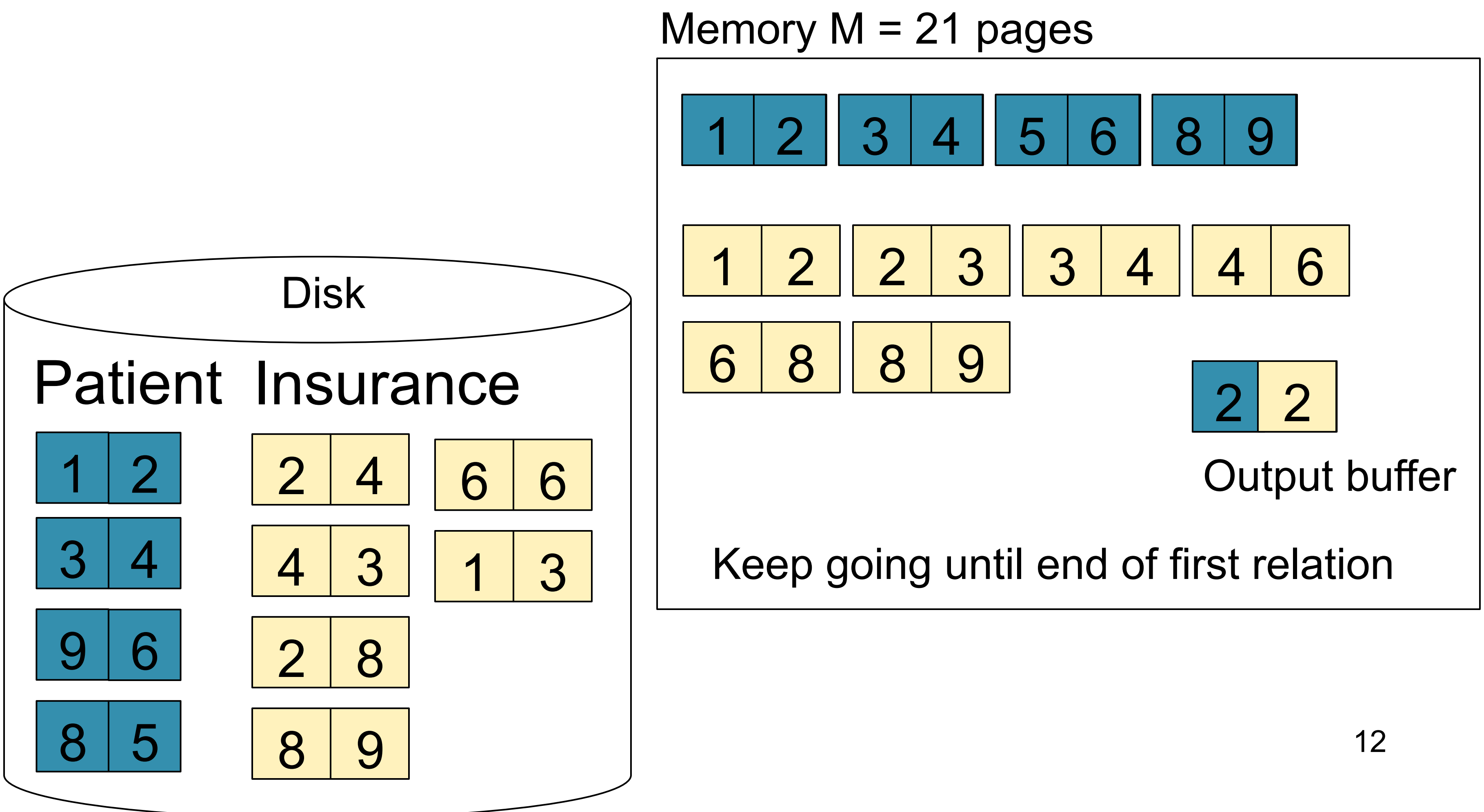
Sort-Merge Join Example

Step 3: Merge Patient and Insurance



Sort-Merge Join Example

Step 3: Merge Patient and Insurance



Cost of Query Plans

T(Supplier) = 1000
T(Supply) = 10,000

B(Supplier) = 100
B(Supply) = 100

V(Supplier,scity) = 20
V(Supplier,state) = 10
V(Supply,pno) = 2,500

M = 11

Physical Query Plan 1

(On the fly) π_{sname}

(On the fly) $\sigma_{\text{scity='Seattle'} \wedge \text{sstate='WA'} \wedge \text{pno=2}}$

(Block Nested loop) $\bowtie_{\text{sno = sno}}$

Supplier (File scan)

Supply (File scan)

Selection and project on-the-fly
-> No additional cost.

Total cost of plan is thus cost of join:
= B(Supplier)+B(Supplier)*B(Supply)/(M-1)
= 100 + 10 * 100
= 1,100 I/Os

T(Supplier) = 1000
T(Supply) = 10,000

B(Supplier) = 100
B(Supply) = 100

V(Supplier,scity) = 20
V(Supplier,state) = 10
V(Supply,pno) = 2,500

M = 11

Physical Query Plan 2

(On the fly) π_{sname} (d)

(Sort-merge join) $\bowtie_{\text{sno = sno}}$ (c)

(Scan write to T1)

(a) $\sigma_{\text{scity='Seattle'} \wedge \text{sstate='WA'}}$

Supplier (File scan)

(Scan write to T2)

(b) $\sigma_{\text{pno=2}}$

Supply (File scan)

Total cost

= 100 + 100 * 1/20 * 1/10 (a)

+ 100 + 100 * 1/2500 (b)

+ 2 (c)

+ 0 (d)

Total cost $\approx$ **204 I/Os**

T(Supplier) = 1000
T(Supply) = 10,000

B(Supplier) = 100
B(Supply) = 100

V(Supplier,scity) = 20
V(Supplier,state) = 10
V(Supply,pno) = 2,500

M = 11

Physical Query Plan 3

(On the fly) (d) π_{sname}

(On the fly) (c) $\sigma_{\text{scity='Seattle'} \wedge \text{sstate='WA'}}$

(b) $\bowtie_{\text{sno = sno}}$ (Index nested loop)

(Use hash index) (a) $\sigma_{\text{pno=2}}$

4 tuples

Supply
(Index on pno)
Assume: clustered

Supplier
(Index on sno)
Clustering does not matter

Total cost
= 1 (a)
+ 4 (b)
+ 0 (c)
+ 0 (d)

Total cost $\approx$ **5 I/Os**

Query Optimizer Overview

- **Input**: A logical query plan
- **Output**: A good physical query plan
- **Basic query optimization algorithm**
  - Enumerate alternative plans (logical and physical)
  - Compute estimated cost of each plan
    - Compute number of I/Os
    - Optionally take into account other resources
  - Choose plan with lowest cost
  - This is called cost-based optimization

Big Picture

- Query languages and data models
  - SQL, SQL, SQL, SQL, …
  - Relational algebra
  - Relational calculus
  - Datalog
- Next week
  - NoSQL, JSon, N1QL

Relational Calculus

- Aka *predicate calculus* or *first order logic*
- TRC = Tuple RC
  - See book
- DRC = Domain RC
  - We study only this one
  - Also see: *Query Language Primer*

Relational Calculus

Relational predicate P is a formula given by this grammar:

$$P ::= \text{atom} \mid P \wedge P \mid P \vee P \mid P \Rightarrow P \mid \text{not}(P) \mid \forall x.P \mid \exists x.P$$

Query Q:

$$Q(x1, \ldots, xk) = P$$

Actor(pid,fName,lName)
Casts(pid,mid)
Movie(pid,title,year)

Relational Calculus

Relational predicate P is a formula given by this grammar:

$$P ::= \text{atom} \mid P \wedge P \mid P \vee P \mid P \Rightarrow P \mid \text{not}(P) \mid \forall x.P \mid \exists x.P$$

Query Q:

$$Q(x1, \ldots, xk) = P$$

Example: find the first/last names of actors who acted in 1940

$$Q(f,l) = \exists x.\ \exists y.\ \exists z.\ (\text{Actor}(z,f,l) \wedge \text{Casts}(z,x) \wedge \text{Movie}(x,y,1940))$$

What does this query return ?

$$Q(f,l) = \exists z.\ (\text{Actor}(z,f,l) \wedge \forall x.(\text{Casts}(z,x) \Rightarrow \exists y.\text{Movie}(x,y,1940)))$$

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Important Observation

Find all bars that serve all beers that Fred likes

$$A(x) = \forall y.\ Likes(\text{"Fred"}, y) \Rightarrow Serves(x,y)$$

- Note: P => Q (read P implies Q) is the same as (not P) OR Q
  In this query: If Fred likes a beer the bar must serve it (P => Q)
  In other words: Either Fred does not like the beer (not P) OR the bar serves that beer (Q).

$$A(x) = \forall y.\ not(Likes(\text{"Fred"}, y))\ OR\ Serves(x,y)$$

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

More Examples

Average Joe

Find drinkers that frequent some bar that serves some beer they like.

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

More Examples

Average Joe

Find drinkers that frequent some bar that serves some beer they like.

$$Q(x) = \exists y.\ \exists z.\ \mathrm{Frequents}(x, y) \wedge \mathrm{Serves}(y,z) \wedge \mathrm{Likes}(x,z)$$

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

More Examples

Average Joe

Find drinkers that frequent <u>some</u> bar that serves <u>some</u> beer they like.

$$Q(x) = \exists y.\ \exists z.\ \text{Frequents}(x, y) \land \text{Serves}(y,z) \land \text{Likes}(x,z)$$

Prudent Peter

Find drinkers that frequent <u>only</u> bars that serves <u>some</u> beer they like.

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

More Examples

Average Joe

Find drinkers that frequent <u>some</u> bar that serves <u>some</u> beer they like.

$$Q(x) = \exists y.\ \exists z.\ \text{Frequents}(x, y) \wedge \text{Serves}(y,z) \wedge \text{Likes}(x,z)$$

Prudent Peter

Find drinkers that frequent <u>only</u> bars that serves <u>some</u> beer they like.

$$Q(x) = \forall y.\ \text{Frequents}(x, y) \Rightarrow (\exists z.\ \text{Serves}(y,z) \wedge \text{Likes}(x,z))$$

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

More Examples

Average Joe

Find drinkers that frequent some bar that serves some beer they like.

$$Q(x) = \exists y.\ \exists z.\ \text{Frequents}(x, y) \land \text{Serves}(y,z) \land \text{Likes}(x,z)$$

Prudent Peter

Find drinkers that frequent only bars that serves some beer they like.

$$Q(x) = \forall y.\ \text{Frequents}(x, y) \Rightarrow (\exists z.\ \text{Serves}(y,z) \land \text{Likes}(x,z))$$

Cautious Carl

Find drinkers that frequent some bar that serves only beers they like.

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

More Examples

Average Joe

Find drinkers that frequent <u>some</u> bar that serves <u>some</u> beer they like.

$$Q(x) = \exists y.\ \exists z.\ Frequents(x, y) \land Serves(y,z) \land Likes(x,z)$$

Prudent Peter

Find drinkers that frequent <u>only</u> bars that serves <u>some</u> beer they like.

$$Q(x) = \forall y.\ Frequents(x, y) \Rightarrow (\exists z.\ Serves(y,z) \land Likes(x,z))$$

Cautious Carl

Find drinkers that frequent <u>some</u> bar that serves <u>only</u> beers they like.

$$Q(x) = \exists y.\ Frequents(x, y) \land \forall z.(Serves(y,z) \Rightarrow Likes(x,z))$$

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

More Examples

Average Joe

Find drinkers that frequent some bar that serves some beer they like.

$$Q(x) = \exists y.\ \exists z.\ \text{Frequents}(x, y) \wedge \text{Serves}(y,z) \wedge \text{Likes}(x,z)$$

Prudent Peter

Find drinkers that frequent only bars that serves some beer they like.

$$Q(x) = \forall y.\ \text{Frequents}(x, y) \Rightarrow (\exists z.\ \text{Serves}(y,z) \wedge \text{Likes}(x,z))$$

Cautious Carl

Find drinkers that frequent some bar that serves only beers they like.

$$Q(x) = \exists y.\ \text{Frequents}(x, y) \wedge \forall z.(\text{Serves}(y,z) \Rightarrow \text{Likes}(x,z))$$

Paranoid Paul

Find drinkers that frequent only bars that serves only beer they like.

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

More Examples

Average Joe

Find drinkers that frequent <u>some</u> bar that serves <u>some</u> beer they like.

$$Q(x) = \exists y.\ \exists z.\ \text{Frequents}(x, y) \wedge \text{Serves}(y,z) \wedge \text{Likes}(x,z)$$

Prudent Peter

Find drinkers that frequent <u>only</u> bars that serves <u>some</u> beer they like.

$$Q(x) = \forall y.\ \text{Frequents}(x, y) \Rightarrow (\exists z.\ \text{Serves}(y,z) \wedge \text{Likes}(x,z))$$

Cautious Carl

Find drinkers that frequent <u>some</u> bar that serves <u>only</u> beers they like.

$$Q(x) = \exists y.\ \text{Frequents}(x, y) \wedge \forall z.(\text{Serves}(y,z) \Rightarrow \text{Likes}(x,z))$$

Paranoid Paul

Find drinkers that frequent <u>only</u> bars that serves <u>only</u> beer they like.

$$Q(x) = \forall y.\ \text{Frequents}(x, y) \Rightarrow \forall z.(\text{Serves}(y,z) \Rightarrow \text{Likes}(x,z))$$

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Domain Independent Relational Calculus

- An *unsafe* RC query, also called *domain dependent,* returns an answer that does not depend just on the database, but on the entire domain

A1(x) = not Likes("Fred", x)

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Domain Independent Relational Calculus

- An *unsafe* RC query, also called *domain dependent,* returns an answer that does not depend just on the database, but on the entire domain

Make sure x is a beer

$\boxed{A1(x) = \text{not Likes("Fred", x)}}$

$A1(x) = \exists y\ \text{Serves}(y,x) \wedge \text{not Likes("Fred", x)}$

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Domain Independent Relational Calculus

- An *unsafe* RC query, also called *domain dependent,* returns an answer that does not depend just on the database, but on the entire domain

Make sure x is a beer

$\boxed{A1(x) = \text{not Likes}(\text{"Fred"}, x)}$

$A1(x) = \exists y\ \text{Serves}(y,x) \wedge \text{not Likes}(\text{"Fred"}, x)$

$\boxed{A2(x,y) = \text{Likes}(\text{"Fred"}, x) \vee \text{Serves}(\text{"Bar"}, y)}$

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Domain Independent Relational Calculus

- An *unsafe* RC query, also called *domain dependent,* returns an answer that does not depend just on the database, but on the entire domain

Make sure x is a beer

$\boxed{A1(x) = \text{not Likes}(\text{"Fred"}, x)}$

$A1(x) = \exists y\ \text{Serves}(y,x) \wedge \text{not Likes}(\text{"Fred"}, x)$

Same here

$\boxed{A2(x,y) = \text{Likes}(\text{"Fred"}, x) \vee \text{Serves}(\text{"Bar"}, y)}$

$A2(x,y) = \exists u\ \text{Serves}(u,x) \wedge \exists w\ \text{Serves}(w,y) \wedge [\text{Likes}(\text{"Fred"}, x) \vee \text{Serves}(\text{"Bar"}, y)]$

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Domain Independent Relational Calculus

- An ***unsafe*** RC query, also called ***domain dependent,*** returns an answer that does not depend just on the database, but on the entire domain

Make sure x is a beer

$A1(x) = \text{not Likes}(\text{"Fred"}, x)$

$A1(x) = \exists y\ \text{Serves}(y,x) \wedge \text{not Likes}(\text{"Fred"}, x)$

Same here

$A2(x,y) = \text{Likes}(\text{"Fred"}, x) \vee \text{Serves}(\text{"Bar"}, y)$

$A2(x,y) = \exists u\ \text{Serves}(u,x) \wedge \exists w\ \text{Serves}(w,y) \wedge [\text{Likes}(\text{"Fred"}, x) \vee \text{Serves}(\text{"Bar"}, y)]$

$A3(x) = \forall y.\ \text{Serves}(x,y)$

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Domain Independent Relational Calculus

- An *unsafe* RC query, also called *domain dependent,* returns an answer that does not depend just on the database, but on the entire domain

Make sure x is a beer

$A1(x)$ = not Likes("Fred", x)

$A1(x) = \exists y\ \text{Serves}(y,x) \wedge \text{not Likes}(\text{"Fred"}, x)$

$A2(x,y) = \text{Likes}(\text{"Fred"}, x) \vee \text{Serves}(\text{"Bar"}, y)$

Same here

$A2(x,y) = \exists u\ \text{Serves}(u,x) \wedge \exists w\ \text{Serves}(w,y) \wedge [\text{Likes}(\text{"Fred"}, x) \vee \text{Serves}(\text{"Bar"}, y)]$

$A3(x) = \forall y.\ \text{Serves}(x,y)$

$A3(x) = \forall y.\ (\exists u\ \text{Serves}(u,y) \rightarrow \text{Serves}(x,y))$

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Domain Independent Relational Calculus

- An *<u>unsafe</u>* RC query, also called *<u>domain dependent,</u>* returns an answer that does not depend just on the database, but on the entire domain

Make sure x is a beer

$A1(x) = \text{not Likes}(\text{"Fred"}, x)$

$A1(x) = \exists y\ \text{Serves}(y,x) \wedge \text{not Likes}(\text{"Fred"}, x)$

Same here

$A2(x,y) = \text{Likes}(\text{"Fred"}, x) \vee \text{Serves}(\text{"Bar"}, y)$

$A2(x,y) = \exists u\ \text{Serves}(u,x) \wedge \exists w\ \text{Serves}(w,y) \wedge [\text{Likes}(\text{"Fred"}, x) \vee \text{Serves}(\text{"Bar"}, y)]$

$A3(x) = \forall y.\ \text{Serves}(x,y)$

$A3(x) = \forall y.\ (\exists u\ \text{Serves}(u,y) \rightarrow \text{Serves}(x,y))$

Lesson: make sure your RC queries are domain independent