Introduction to Data Management
CSE 414

Unit 4: RDBMS Internals
Logical and Physical Plans
Query Execution
Query Optimization

(3 lectures)

Introduction to Database Systems
CSE 414

Lecture 16:
Basics of Data Storage and Indexes

Query Performance

- My database application is too slow… why?
- One of the queries is very slow… why?

- To understand performance, we need to understand:
  - How is data organized on disk
  - How to estimate query costs

  - In this course we will focus on **disk-based** DBMSs

Data Storage

Student

ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
…		

- DBMSs store data in **files**
- Most common organization is row-wise storage
- On disk, a file is split into blocks
- Each block contains a set of tuples

10	Tom	Hanks	block 1
20	Amy	Hanks	
50	…	…	block 2
200	…		
220			block 3
240			
420			
800			

In the example, we have 4 blocks with 2 tuples each

Data File Types

Student

ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
…		

The data file can be one of:

- Heap file
  - Unsorted
- Sequential file
  - Sorted according to some attribute(s) called *key*

Index

- An **additional** file, that allows fast access to records in the data file given a search key

Index

- An **additional** file, that allows fast access to records in the data file given a search key
- The index contains (key, value) pairs:
  - The key = an attribute value (e.g., student ID or name)
  - The value = a pointer to the record

Index

- An **additional** file, that allows fast access to records in the data file given a search key
- The index contains (key, value) pairs:
  - The key = an attribute value (e.g., student ID or name)
  - The value = a pointer to the record
- Could have many indexes for one table

Key = means here search key

This  Is Not A Key

Different keys:

- Primary key – uniquely identifies a tuple
- Key of the sequential file – how the data file is sorted, if at all
- Index key – how the index is organized

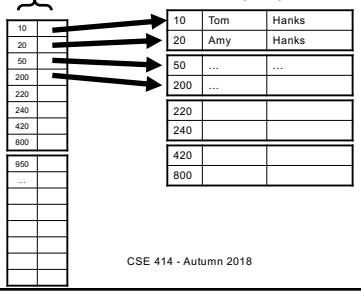
Example 1: Index on ID

Index **Student_ID** on **Student.ID**

Data File **Student**

Student

ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
...		



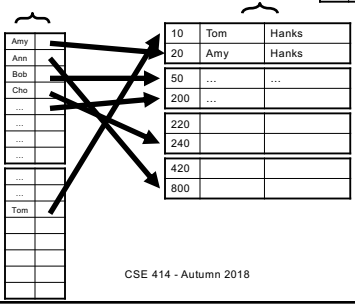
Example 2: Index on fName

Index **Student_fName** on **Student.fName**

Data File **Student**

Student

ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
...		



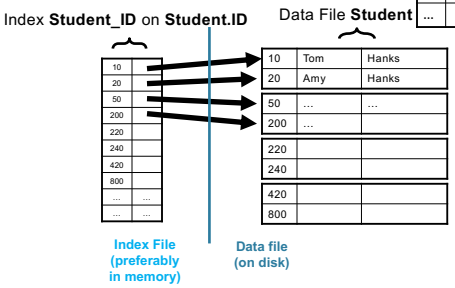
Index Organization

- Hash table
- B+ trees – most common
  - They are search trees, but they are not binary instead have higher fan-out
  - Will discuss them briefly next
- Specialized indexes: bit maps, R-trees, inverted index

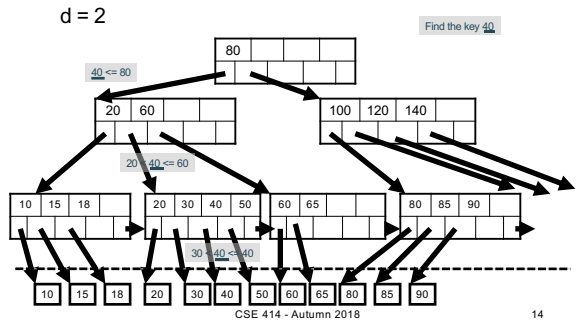
Hash table example

Student

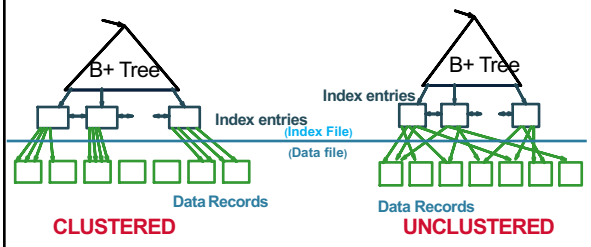
ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
...		



B+ Tree Index by Example



Clustered vs Unclustered



Every table can have **only one** clustered and **many** unclustered indexes
Why?

Index Classification

- **Clustered/unclustered**
  - Clustered = records close in index are close in data
    - Option 1: Data inside data file is sorted on disk
    - Option 2: Store data directly inside the index (no separate files)
  - Unclustered = records close in index may be far in data

Index Classification

- **Clustered/unclustered**
  - Clustered = records close in index are close in data
    - Option 1: Data inside data file is sorted on disk
    - Option 2: Store data directly inside the index (no separate files)
  - Unclustered = records close in index may be far in data
- **Primary/secondary**
  - Meaning 1:
    - Primary = is over attributes that include the primary key
    - Secondary = otherwise
  - Meaning 2: means the same as clustered/unclustered

Index Classification

- **Clustered/unclustered**
  - Clustered = records close in index are close in data
    - Option 1: Data inside data file is sorted on disk
    - Option 2: Store data directly inside the index (no separate files)
  - Unclustered = records close in index may be far in data
- **Primary/secondary**
  - Meaning 1:
    - Primary = is over attributes that include the primary key
    - Secondary = otherwise
  - Meaning 2: means the same as clustered/unclustered
- **Organization** B+ tree or Hash table

Scanning a Data File

- Disks are mechanical devices!
  - Technology from the 60s;
  - Density increases over time
- Read only at the rotation speed!
- Consequence: sequential scan faster than random
  - Good: read blocks 1,2,3,4,5,…
  - Bad: read blocks 2342, 11, 321,9, …
- Rule of thumb:
  - Random read 1-2% of file ≈ sequential scan entire file;
  - 1-2% decreases over time, because of increased density
- Solid state (SSD): still too expensive today

Summary So Far

- Index = a file that enables direct access to records in another data file
  - B+ tree / Hash table
  - Clustered/unclustered
- Data resides on disk
  - Organized in blocks
  - Sequential reads are efficint
  - Random access less efficient
  - Random read 1-2% of data worse than sequential

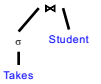
Student(ID, fname, lname)
Takes(studentID, courseID)

```
SELECT *
FROM  Student x, Takes y
WHERE x.ID=y.studentID AND y.courseID > 300
```

Example

Student(ID, fname, lname)
Takes(studentID, courseID)

```
SELECT *
FROM  Student x, Takes y
WHERE x.ID=y.studentID AND y.courseID > 300
```



Example

```
for y in Takes
  if courseID > 300 then
    for x in Student
      if x.ID=y.studentID
        output *
```


Student(ID, fname, lname)
Takes(studentID, courseID)

```
SELECT *
FROM  Student x, Takes y
WHERE x.ID=y.studentID AND y.courseID > 300
```

Example

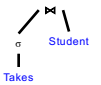
```
for y in Takes
  if courseID > 300 then
    for x in Student
      if x.ID=y.studentID
        output *
```

Assume the database has indexes on these attributes:

- **Takes_courseID** = index on Takes.courseID
- **Student_ID** = index on Student.ID

Student(ID, fname, lname)
Takes(studentID, courseID)

```
SELECT *
FROM  Student x, Takes y
WHERE x.ID=y.studentID AND y.courseID > 300
```



Example

```
for y in Takes
  if courseID > 300 then
    for x in Student
      if x.ID=y.studentID
        output *
```

Assume the database has indexes on these attributes:

- **Takes_courseID** = index on Takes.courseID
- **Student_ID** = index on Student.ID

```
for y' in Takes_courseID where y'.courseID > 300
```

Student(ID, fname, lname)
Takes(studentID, courseID)

```
SELECT *
FROM  Student x, Takes y
WHERE x.ID=y.studentID AND y.courseID > 300
```

Example


```
for y in Takes
  if courseID > 300 then
    for x in Student
      if x.ID=y.studentID
        output *
```

Assume the database has indexes on these attributes:

- **Takes_courseID** = index on Takes.courseID
- **Student_ID** = index on Student.ID

Index selection

```
for y' in Takes_courseID where y'.courseID > 300
  y = fetch the Takes record pointed to by y'
```


Student(ID, fname, lname)
Takes(studentID, courseID)

```
SELECT *
FROM  Student x, Takes y
WHERE x.ID=y.studentID AND y.courseID > 300
```

Example


```
for y in Takes
  if courseID > 300 then
    for x in Student
      if x.ID=y.studentID
        output *
```

Assume the database has indexes on these attributes:

- **Takes_courseID** = index on Takes.courseID
- **Student_ID** = index on Student.ID

Index selection

```
for y' in Takes_courseID where y'.courseID > 300
  y = fetch the Takes record pointed to by y'
  for x' in Student_ID where x'.ID = y.studentID
      x = fetch the Student record pointed to by x'
```

Index join

Student(ID, fname, lname)
Takes(studentID, courseID)

```
SELECT *
FROM  Student x, Takes y
WHERE x.ID=y.studentID AND y.courseID > 300
```

Example


```
for y in Takes
  if courseID > 300 then
    for x in Student
      if x.ID=y.studentID
        output *
```

Assume the database has indexes on these attributes:

- **Takes_courseID** = index on Takes.courseID
- **Student_ID** = index on Student.ID

Index selection

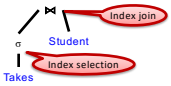
```
for y' in Takes_courseID where y'.courseID > 300
  y = fetch the Takes record pointed to by y'
  for x' in Student_ID where x'.ID = y.studentID
      x = fetch the Student record pointed to by x'
      output *
```

Index join

Student(ID, fname, lname)
Takes(studentID, courseID)

```
SELECT *
FROM  Student x, Takes y
WHERE x.ID=y.studentID AND y.courseID > 300
```

Example




```
for y in Takes
  if courseID > 300 then
    for x in Student
      if x.ID=y.studentID
        output *
```

Assume the database has indexes on these attributes:

- **Takes_courseID** = index on Takes.courseID
- **Student_ID** = index on Student.ID

Index selection

```
for y' in Takes_courseID where y'.courseID > 300
  y = fetch the Takes record pointed to by y'
  for x' in Student_ID where x'.ID = y.studentID
      x = fetch the Student record pointed to by x'
      output *
```

Index join

Getting Practical: Creating Indexes in SQL

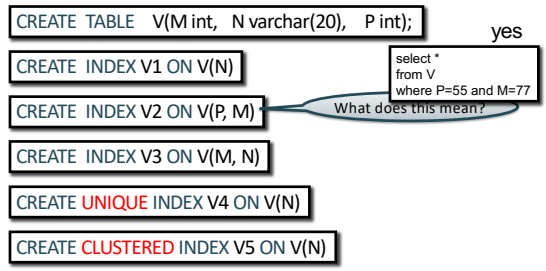
```
CREATE TABLE  V(M int,  N varchar(20),  P int);
CREATE INDEX V1 ON V(N)
CREATE INDEX V2 ON V(P, M)
CREATE INDEX V3 ON V(M, N)
CREATE UNIQUE INDEX V4 ON V(N)
CREATE CLUSTERED INDEX V5 ON V(N)
```


Getting Practical: Creating Indexes in SQL

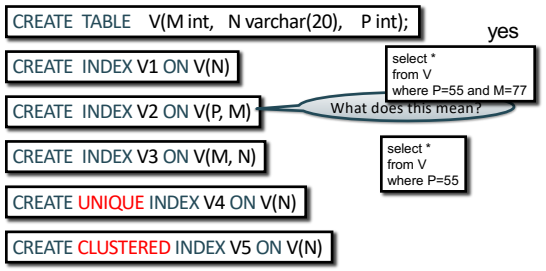
```
CREATE TABLE  V(M int,  N varchar(20),  P int);
CREATE INDEX V1 ON V(N)
CREATE INDEX V2 ON V(P, M)
CREATE INDEX V3 ON V(M, N)
CREATE UNIQUE INDEX V4 ON V(N)
CREATE CLUSTERED INDEX V5 ON V(N)
```

What does this mean?

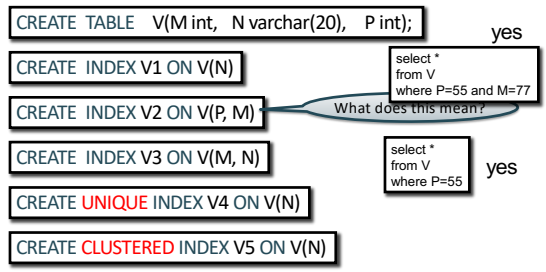
Getting Practical: Creating Indexes in SQL



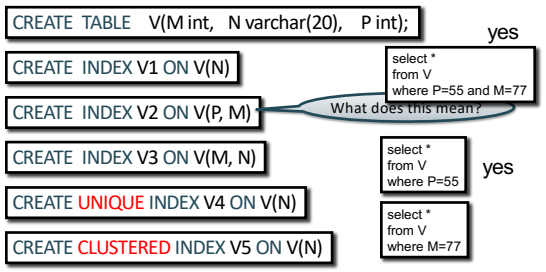
Getting Practical: Creating Indexes in SQL



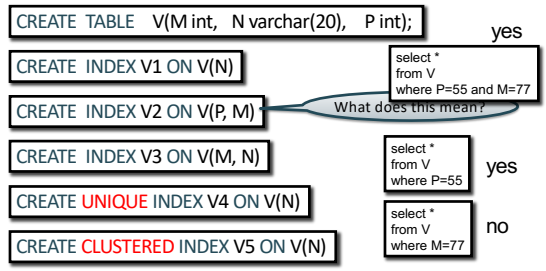
Getting Practical: Creating Indexes in SQL



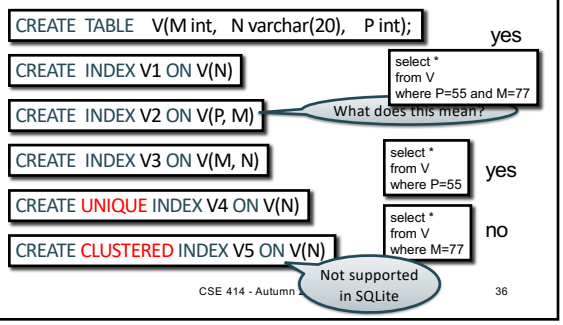
Getting Practical: Creating Indexes in SQL



Getting Practical: Creating Indexes in SQL



Getting Practical: Creating Indexes in SQL



Which Indexes?

Student

ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
…		

- How many indexes could we create?
- Which indexes should we create?

Which Indexes?

Student

ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
…		

- How many indexes could we create?
- Which indexes should we create?

In general this is a very hard problem

Which Indexes?

Student

ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
…		

- The *index selection problem*
  - Given a table, and a "workload" (big Java application with lots of SQL queries), decide which indexes to create (and which ones NOT to create!)
- Who does index selection:
  - The database administrator DBA
  - Semi-automatically, using a database administration tool

Which Indexes?

Student

ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
…		

- The *index selection problem*
  - Given a table, and a "workload" (big Java application with lots of SQL queries), decide which indexes to create (and which ones NOT to create!)
- Who does index selection:
  - The database administrator DBA

  - Semi-automatically, using a database administration tool

Index Selection: Which Search Key

- Make some attribute K a search key if the WHERE clause contains:
  - An exact match on K
  - A range predicate on K
  - A join on K

The Index Selection Problem 1

```
V(M, N, P);
```

Your workload is this

100000 queries:

```
SELECT *
FROM V
WHERE N=?
```

100 queries:

```
SELECT *
FROM V
WHERE P=?
```

The Index Selection Problem 1

V(M, N, P);

Your workload is this

100000 queries:

```
SELECT *
FROM V
WHERE N=?
```

100 queries:

```
SELECT *
FROM V
WHERE P=?
```

What indexes ?

The Index Selection Problem 1

V(M, N, P);

Your workload is this

100000 queries:

```
SELECT *
FROM V
WHERE N=?
```

100 queries:

```
SELECT *
FROM V
WHERE P=?
```

A: V(N) and V(P) (hash tables or B-trees)

The Index Selection Problem 2

V(M, N, P);

Your workload is this

100000 queries:

```
SELECT *
FROM V
WHERE N>? and N<?
```

100 queries:

```
SELECT *
FROM V
WHERE P=?
```

100000 queries:

```
INSERT INTO V
VALUES (?, ?, ?)
```

What indexes ?

The Index Selection Problem 2

V(M, N, P);

Your workload is this

100000 queries:

```
SELECT *
FROM V
WHERE N>? and N<?
```

100 queries:

```
SELECT *
FROM V
WHERE P=?
```

100000 queries:

```
INSERT INTO V
VALUES (?, ?, ?)
```

A: definitely V(N) (must B-tree); unsure about V(P)

The Index Selection Problem 3

V(M, N, P);

Your workload is this

100000 queries:

```
SELECT *
FROM V
WHERE N=?
```

1000000 queries:

```
SELECT *
FROM V
WHERE N=? and P>?
```

100000 queries:

```
INSERT INTO V
VALUES (?, ?, ?)
```

What indexes ?

The Index Selection Problem 3

V(M, N, P);

Your workload is this

100000 queries:

```
SELECT *
FROM V
WHERE N=?
```

1000000 queries:

```
SELECT *
FROM V
WHERE N=? and P>?
```

100000 queries:

```
INSERT INTO V
VALUES (?, ?, ?)
```

A: V(N, P)

How does this index differ from:
1. Two indexes V(N) and V(P)?
2. An index V(P, N)?

The Index Selection Problem 4

V(M, N, P);

Your workload is this

1000 queries:

```
SELECT *
FROM V
WHERE N>? and N<?
```

100000 queries:

```
SELECT *
FROM V
WHERE P>? and P<?
```

What indexes ?

The Index Selection Problem 4

V(M, N, P);

Your workload is this

1000 queries:

```
SELECT *
FROM V
WHERE N>? and N<?
```

100000 queries:

```
SELECT *
FROM V
WHERE P>? and P<?
```

A: V(N) secondary, V(P) primary index

Two typical kinds of queries

```
SELECT *
FROM Movie
WHERE year = ?
```

- Point queries
- What data structure should be used for index?

```
SELECT *
FROM Movie
WHERE year >= ? AND
      year <= ?
```

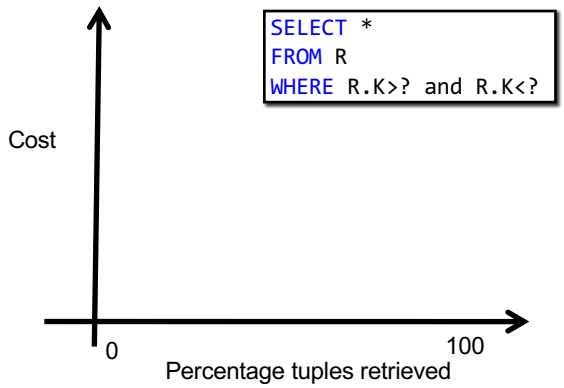
- Range queries
- What data structure should be used for index?

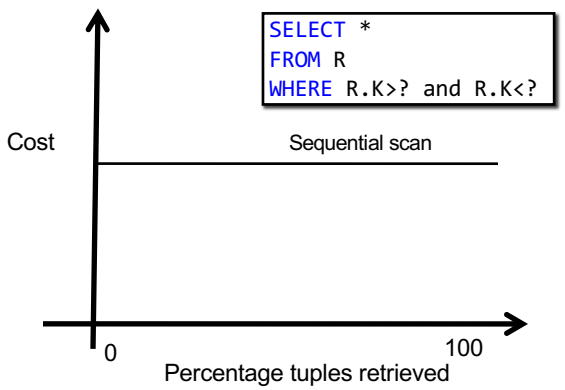
Basic Index Selection Guidelines

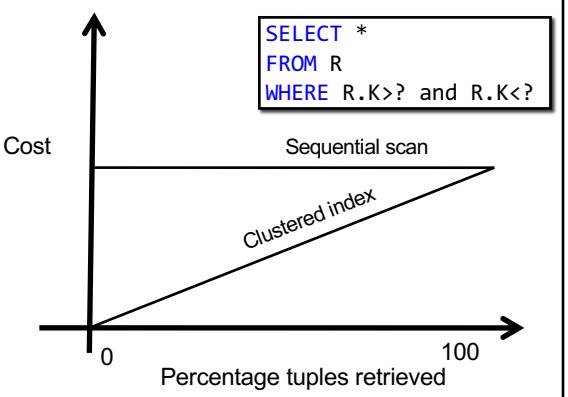
- Consider queries in workload in order of importance
- Consider relations accessed by query
  - No point indexing other relations
- Look at WHERE clause for possible search key
- Try to choose indexes that speed-up multiple queries

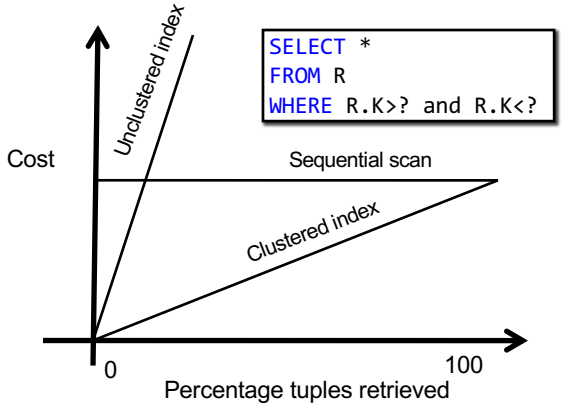
To Cluster or Not

- Range queries benefit mostly from clustering
- Covering indexes do *not* need to be clustered: they work equally well unclustered









Introduction to Database Systems CSE 344

Lecture 17:
Basics of Query Optimization and
Query Cost Estimation

Choosing Index is Not Enough

- To estimate the cost of a query plan, we still need to consider other factors:
  - How each operator is implemented
  - The cost of each operator
  - Let's start with the basics

Cost of Reading Data From Disk

Cost Parameters

- Cost = I/O + CPU + Network BW
  - We will focus on I/O in this class
- Parameters (a.k.a. statistics):
  - **B(R)** = # of blocks (i.e., pages) for relation R
  - **T(R)** = # of tuples in relation R
  - **V(R, a)** = # of distinct values of attribute a

Cost Parameters

- Cost = I/O + CPU + Network BW
  - We will focus on I/O in this class
- Parameters (a.k.a. statistics):
  - **B(R)** = # of blocks (i.e., pages) for relation R
  - **T(R)** = # of tuples in relation R
  - **V(R, a)** = # of distinct values of attribute a

When **a** is a key, **V(R,a) = T(R)**
When **a** is not a key, **V(R,a)** can be anything <= **T(R)**

Cost Parameters

- Cost = I/O + CPU + Network BW
  - We will focus on I/O in this class
- Parameters (a.k.a. statistics):
  - **B(R)** = # of blocks (i.e., pages) for relation R
  - **T(R)** = # of tuples in relation R
  - **V(R, a)** = # of distinct values of attribute a

When **a** is a key, **V(R,a) = T(R)**
When **a** is not a key, **V(R,a)** can be anything <= **T(R)**

- DBMS collects statistics about base tables must infer them for intermediate results

Selectivity Factors for Conditions

- A = c /* $\sigma_{A=c}(R)$ */
  - Selectivity = 1/V(R,A)
- A < c /* $\sigma_{A<c}(R)$*/
  - Selectivity = (c - min(R, A))/(max(R,A) - min(R,A))
- c1 < A < c2 /* $\sigma_{c1<A<c2}(R)$*/
  - Selectivity = (c2 – c1)/(max(R,A) - min(R,A))

Cost of Reading Data From Disk

- Sequential scan for relation R costs **B(R)**
- Index-based selection
  - Estimate selectivity factor **f** (see previous slide)
  - Clustered index: f***B(R)**
  - Unclustered index f***T(R)**

Note: we ignore I/O cost for index pages

Index Based Selection

- Example:

B(R) = 2000
T(R) = 100,000
V(R, a) = 20

cost of $\sigma_{a=v}(R) = ?$

- Table scan:
- Index based selection:

Index Based Selection

- Example: B(R) = 2000, T(R) = 100,000, V(R, a) = 20 — cost of $\sigma_{a=v}(R) = ?$
- Table scan: B(R) = 2,000 I/Os
- Index based selection:

Index Based Selection

- Example: B(R) = 2000, T(R) = 100,000, V(R, a) = 20 — cost of $\sigma_{a=v}(R) = ?$
- Table scan: B(R) = 2,000 I/Os
- Index based selection:
  - If index is clustered:
  - If index is unclustered:

Index Based Selection

- Example: B(R) = 2000, T(R) = 100,000, V(R, a) = 20 — cost of $\sigma_{a=v}(R) = ?$
- Table scan: B(R) = 2,000 I/Os
- Index based selection:
  - If index is clustered: B(R) * 1/V(R,a) = 100 I/Os
  - If index is unclustered:

Index Based Selection

- Example: B(R) = 2000, T(R) = 100,000, V(R, a) = 20 — cost of $\sigma_{a=v}(R) = ?$
- Table scan: B(R) = 2,000 I/Os
- Index based selection:
  - If index is clustered: B(R) * 1/V(R,a) = 100 I/Os
  - If index is unclustered: T(R) * 1/V(R,a) = 5,000 I/Os

Index Based Selection

- Example: B(R) = 2000, T(R) = 100,000, V(R, a) = 20 — cost of $\sigma_{a=v}(R) = ?$
- Table scan: B(R) = 2,000 I/Os
- Index based selection:
  - If index is clustered: B(R) * 1/V(R,a) = 100 I/Os
  - If index is unclustered: T(R) * 1/V(R,a) = 5,000 I/Os

Lesson: Don't build unclustered indexes when V(R,a) is small !

Cost of Executing Operators (Focus on Joins)

Outline

- **Join operator algorithms**
  - One-pass algorithms (Sec. 15.2 and 15.3)
  - Index-based algorithms (Sec 15.6)
- Note about readings:
  - In class, we discuss only algorithms for joins
  - Other operators are easier: read the book

Join Algorithms

- Hash join
- Nested loop join
- Sort-merge join

Hash Join

Hash join: R ⋈ S

- Scan R, build buckets in main memory
- Then scan S and join
- Cost: B(R) + B(S)
- Which relation to build the hash table on?

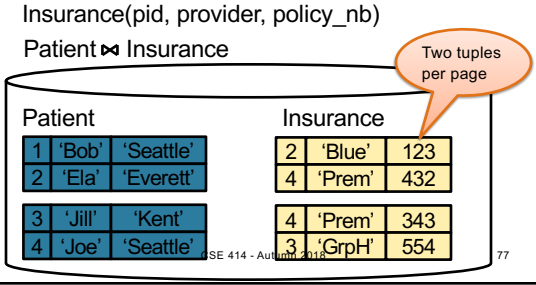
Hash Join

Hash join: R ⋈ S

- Scan R, build buckets in main memory
- Then scan S and join
- Cost: B(R) + B(S)
- Which relation to build the hash table on?
- One-pass algorithm when B(R) ≤ M
  - M = number of memory pages available

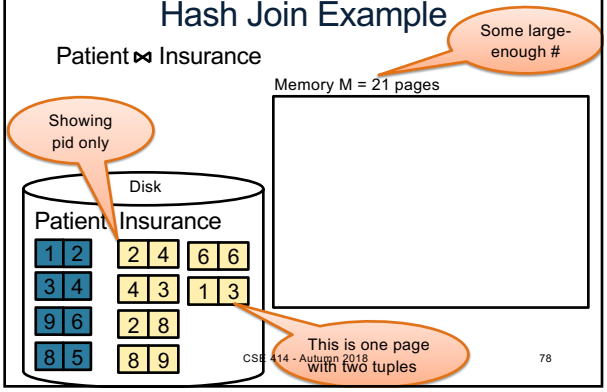
Hash Join Example

Patient(pid, name, address)
Insurance(pid, provider, policy_nb)
Patient ⋈ Insurance



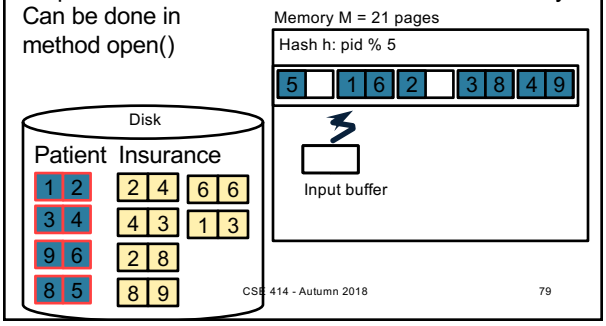
Hash Join Example

Patient ⋈ Insurance



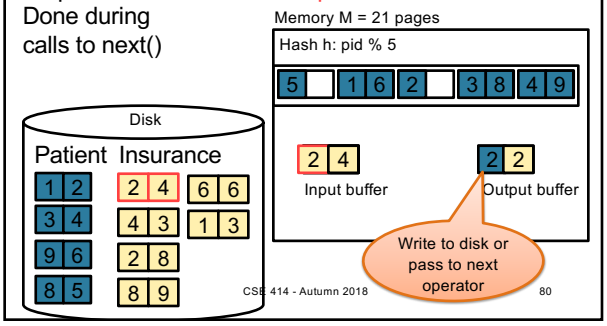
Hash Join Example

Step 1: Scan Patient and build hash table in memory
Can be done in method open()



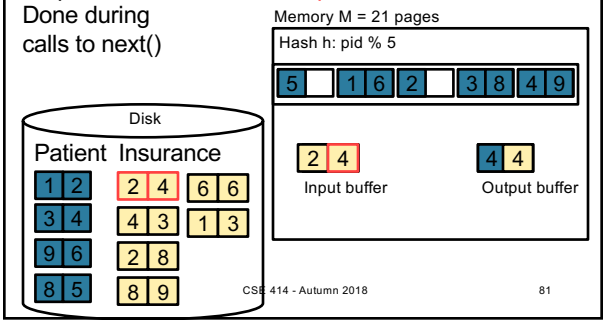
Hash Join Example

Step 2: Scan Insurance and probe into hash table
Done during calls to next()



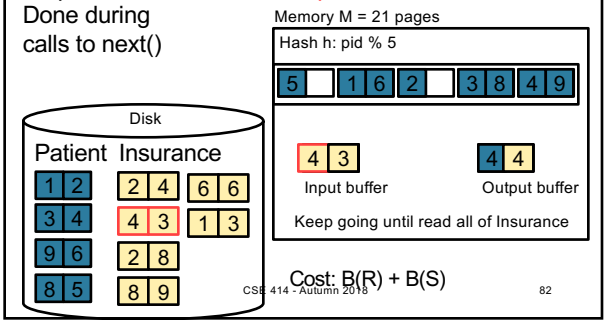
Hash Join Example

Step 2: Scan Insurance and probe into hash table
Done during calls to next()



Hash Join Example

Step 2: Scan Insurance and probe into hash table
Done during calls to next()



Cost: B(R) + B(S)

Nested Loop Joins

- Tuple-based nested loop R ⋈ S
- R is the outer relation, S is the inner relation

```
for each tuple t1 in R do
  for each tuple t2 in S do
    if t1 and t2 join then output (t1,t2)
```

What is the Cost?

Nested Loop Joins

- Tuple-based nested loop R ⋈ S
- R is the outer relation, S is the inner relation

```
for each tuple t1 in R do
  for each tuple t2 in S do
    if t1 and t2 join then output (t1,t2)
```

What is the Cost?

- Cost: B(R) + T(R) B(S)
- Multiple-pass since S is read many times

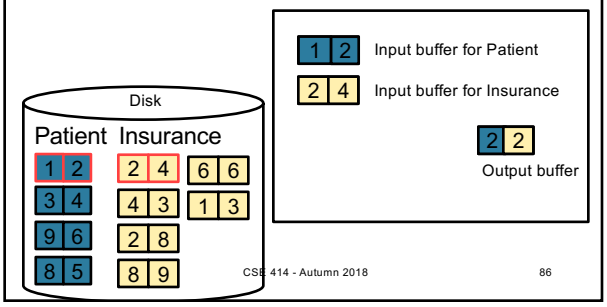
Page-at-a-time Refinement

```
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)
```

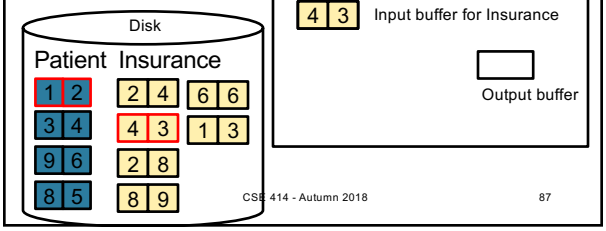
- Cost: B(R) + B(R)B(S)

What is the Cost?

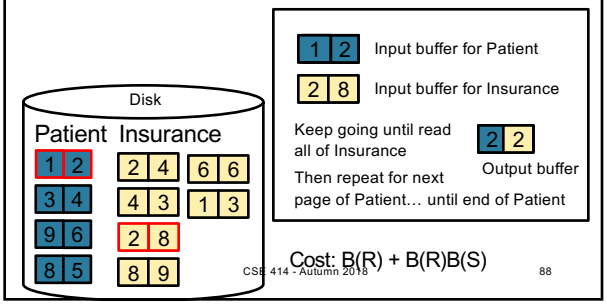
Page-at-a-time Refinement



Page-at-a-time Refinement



Page-at-a-time Refinement



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)

What is the Cost?

Sort-Merge Join

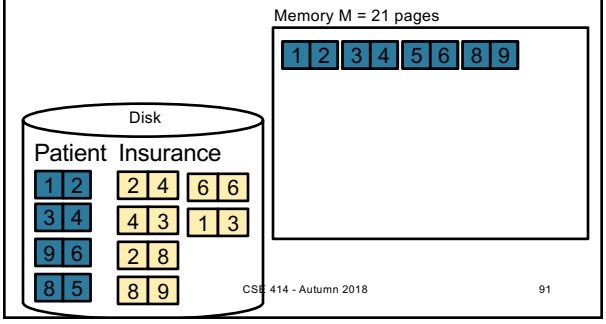
Sort-merge join: R ⋈ S

- 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

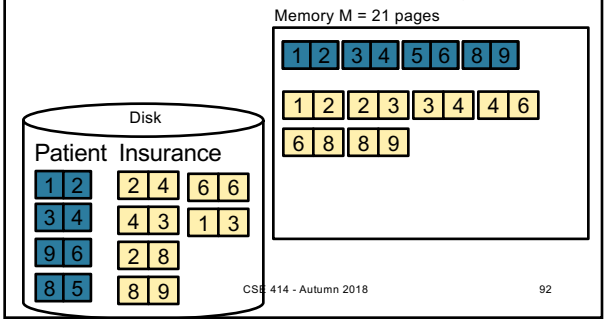
Sort-Merge Join Example

Step 1: Scan Patient and sort in memory



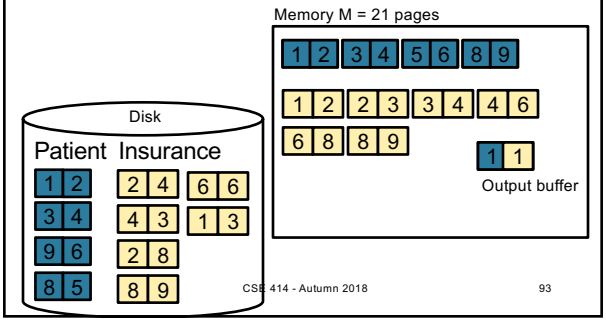
Sort-Merge Join Example

Step 2: Scan Insurance and sort in memory



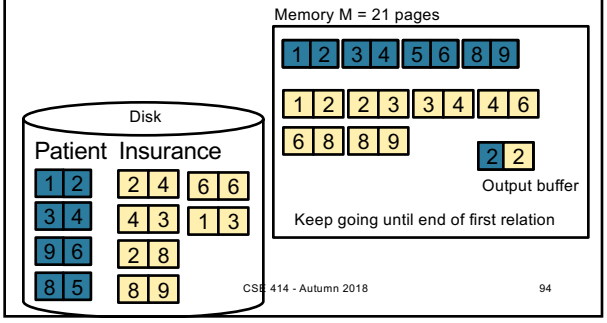
Sort-Merge Join Example

Step 3: Merge Patient and Insurance



Sort-Merge Join Example

Step 3: Merge Patient and Insurance



Index Nested Loop Join

R ⋈ S

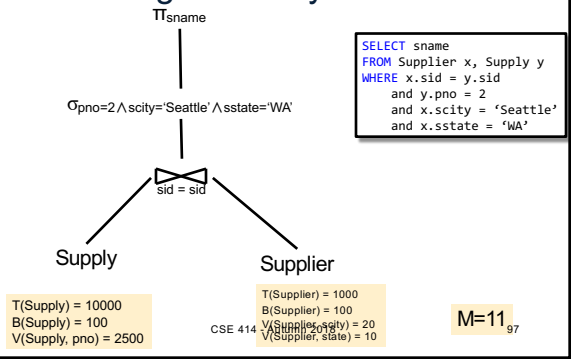
- Assume S has an index on the join attribute
- Iterate over R, for each tuple fetch corresponding tuple(s) from S

- Cost:
  - If index on S is clustered:
    B(R) + T(R) * (B(S) * 1/V(S,a))
  - If index on S is unclustered:
    B(R) + T(R) * (T(S) * 1/V(S,a))

Cost of Query Plans

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Logical Query Plan 1




```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```

T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11 97

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Logical Query Plan 1


```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```

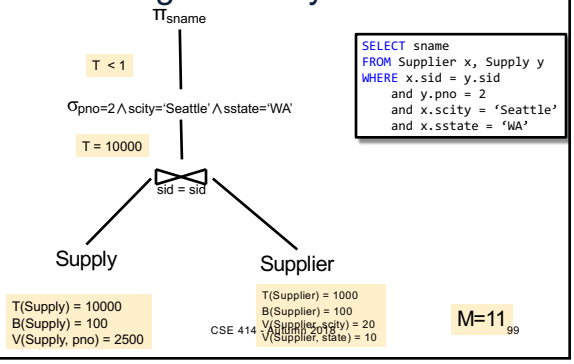
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11 98

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Logical Query Plan 1




```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```

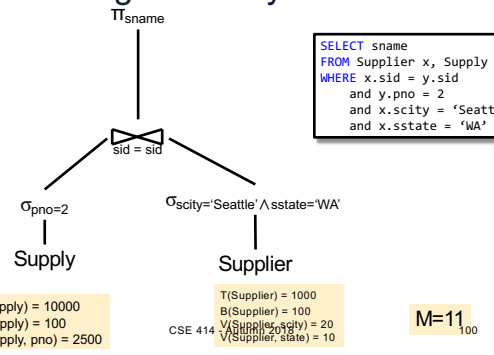
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11 99

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Logical Query Plan 2




```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```

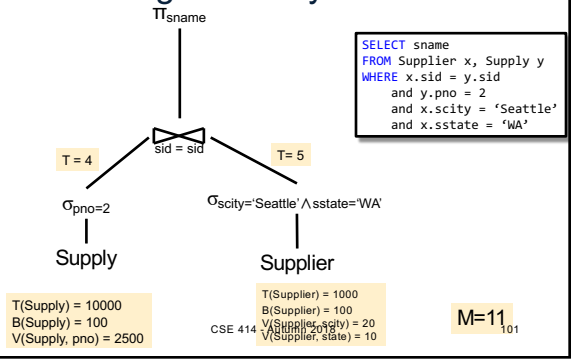
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11 100

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Logical Query Plan 2




```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```

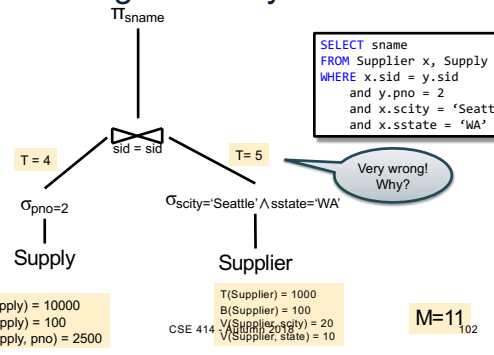
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11 101

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Logical Query Plan 2




```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```

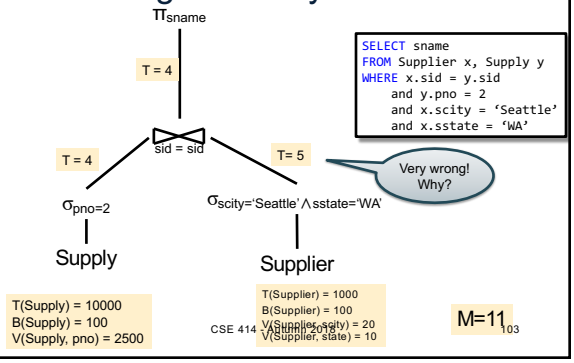
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11 102

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Logical Query Plan 2



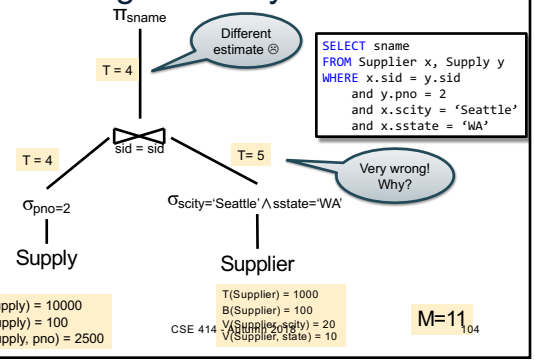
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Logical Query Plan 2



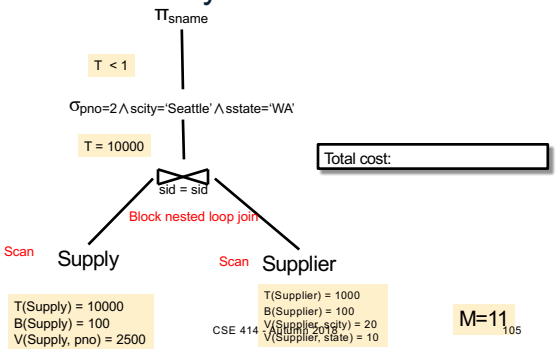
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Physical Plan 1



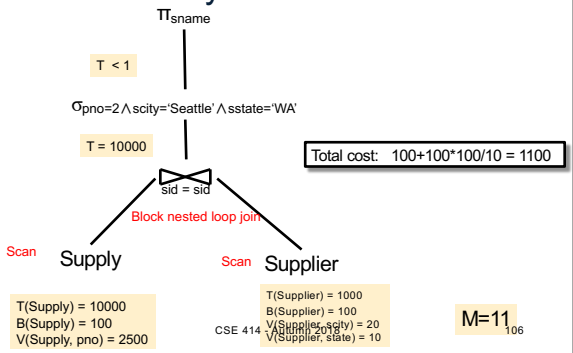
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Physical Plan 1



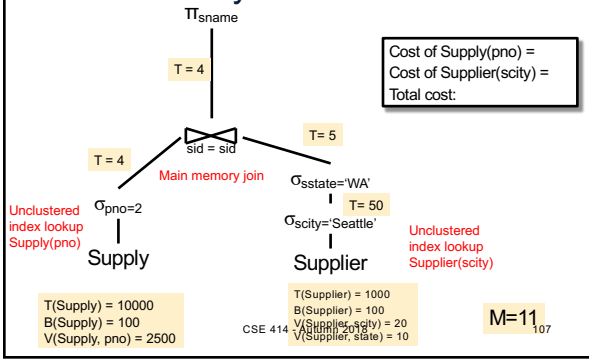
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Physical Plan 2



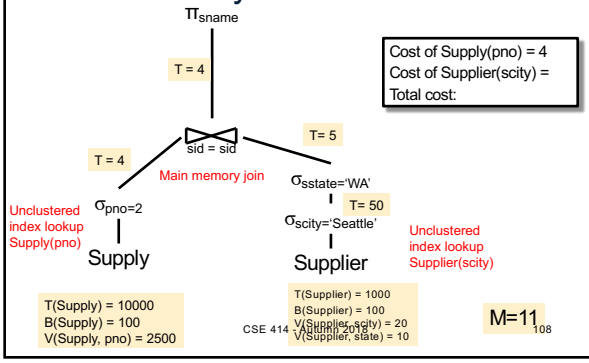
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Physical Plan 2



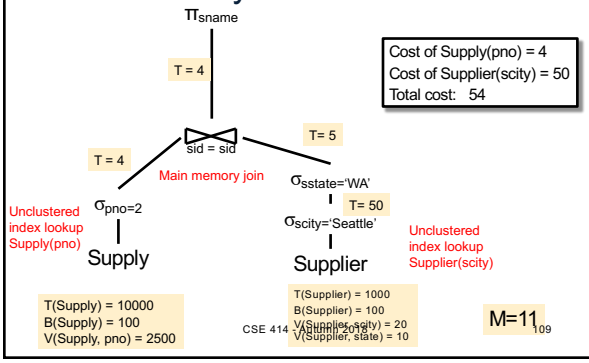
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

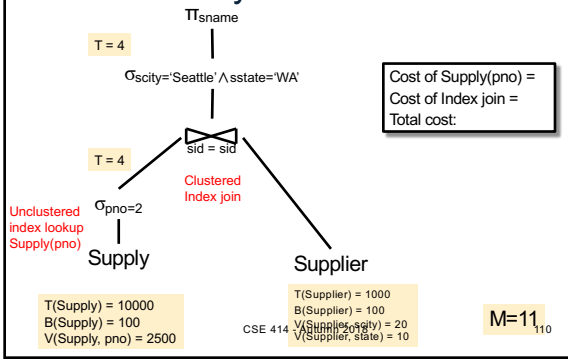
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Physical Plan 2



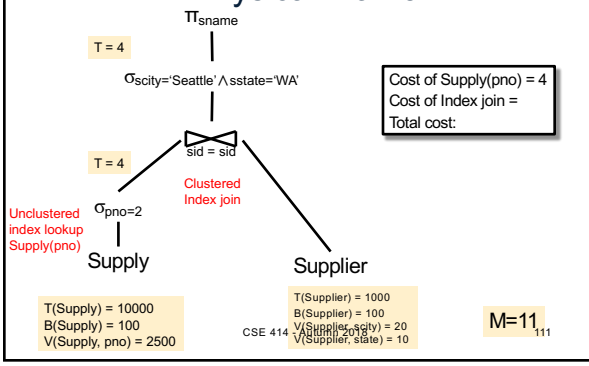
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Physical Plan 3



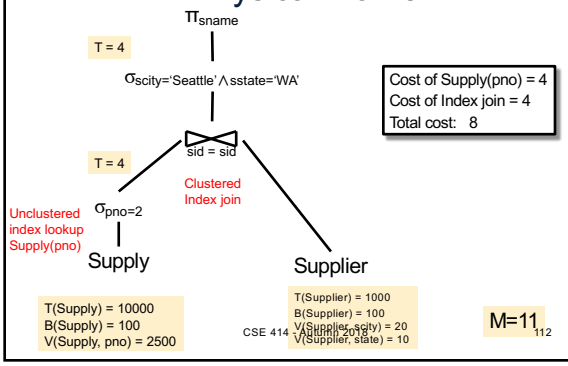
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Physical Plan 3



Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Physical Plan 3



Query Optimizer Summary

- 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
  - Choose plan with lowest cost
- This is called cost-based optimization