

# Introduction to Data Management

## CSE 344

### Lecture 8: SQL Wrap-up

# Announcements

- New web quiz out: due Tuesday, 11 pm
  - Short, covers nested queries
- Homework 3 due a week from Tuesday, **but**
  - Be absolutely sure to log on to the Azure cluster by this weekend and let us know if problems

# Review: Indexes

V(M, N);

Suppose we have queries like these:

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

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

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

Which of these indexes are helpful for each query?

1. Index on V(M)
2. Index on V(N)
3. Index on V(M,N)

# Review: Indexes

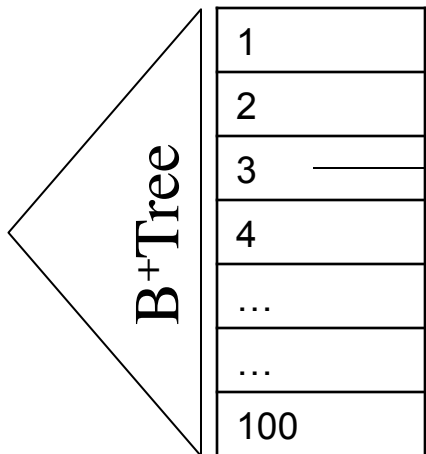
V(M, N);

Suppose V(M,N) contains 10,000 records:  
(1,1), (1,2), (1,3), ..., (1,100), (2,1),..., (100, 100)

```
SELECT *  
FROM V  
WHERE M=3
```

```
SELECT *  
FROM V  
WHERE N=5
```

```
SELECT *  
FROM V  
WHERE M=3 and N=5
```



List of pointers to records (3,1), (3,2), ..., (3,100)

Index on V(M)

# Review: Indexes

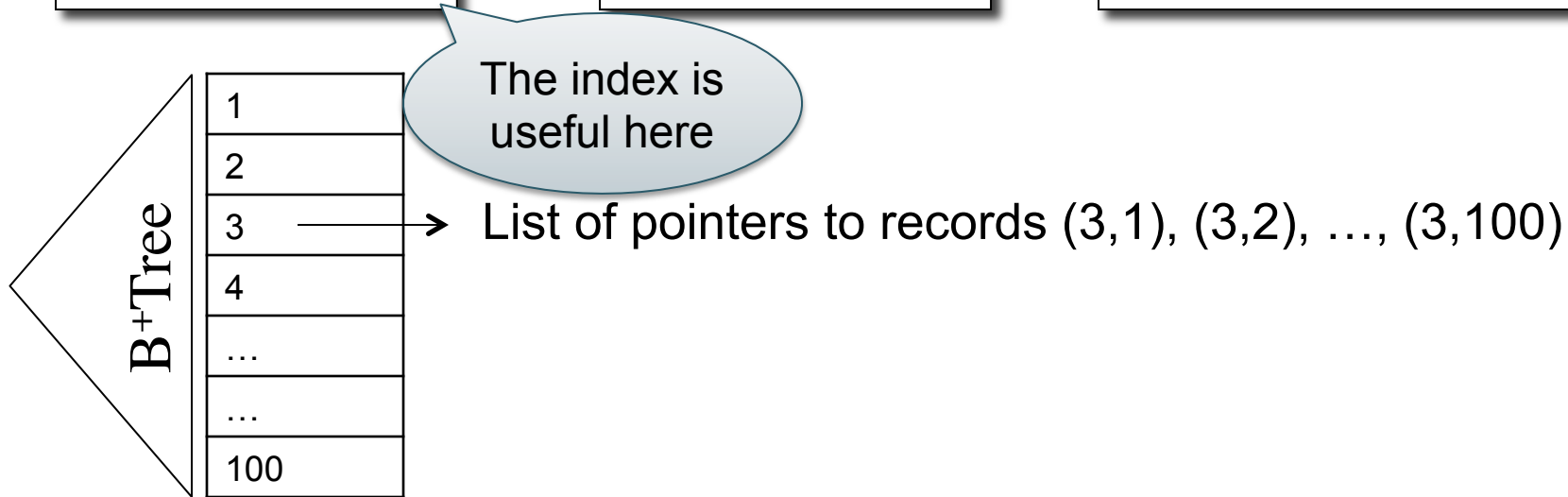
V(M, N);

Suppose V(M,N) contains 10,000 records:  
(1,1), (1,2), (1,3), ..., (1,100), (2,1),..., (100, 100)

```
SELECT *  
FROM V  
WHERE M=3
```

```
SELECT *  
FROM V  
WHERE N=5
```

```
SELECT *  
FROM V  
WHERE M=3 and N=5
```



Index on V(M)

# Review: Indexes

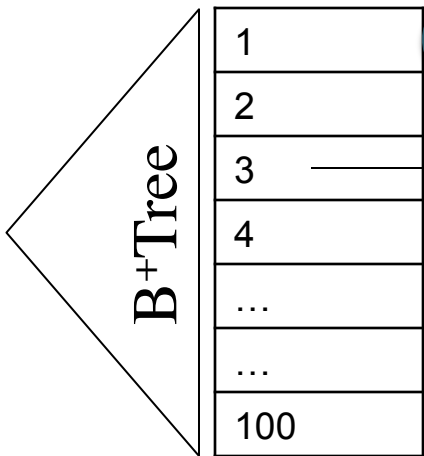
V(M, N);

Suppose V(M,N) contains 10,000 records:  
(1,1), (1,2), (1,3), ..., (1,100), (2,1),..., (100, 100)

```
SELECT *  
FROM V  
WHERE M=3
```

```
SELECT *  
FROM V  
WHERE N=5
```

```
SELECT *  
FROM V  
WHERE M=3 and N=5
```



The index is useful here

Useless here

List of pointers to records (3,1), (3,2), ..., (3,100)

Index on V(M)

# Review: Indexes

V(M, N);

Suppose V(M,N) contains 10,000 records:  
(1,1), (1,2), (1,3), ..., (1,100), (2,1),..., (100, 100)

SELECT \*  
FROM V  
WHERE M=3

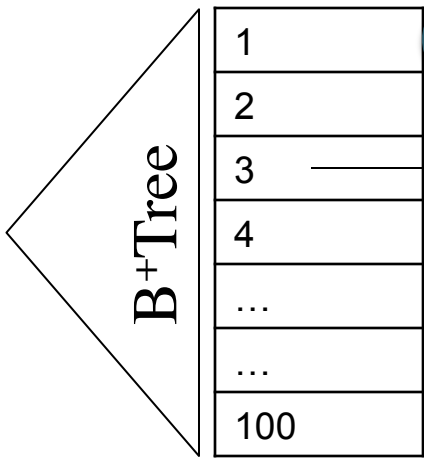
SELECT \*  
FROM V  
WHERE N=5

SELECT \*  
FROM V  
WHERE M=3 and N=5

The index is useful here

Useless here

Can we use it here?



List of pointers to records (3,1), (3,2), ..., (3,100)

Index on V(M)

# Review: Indexes

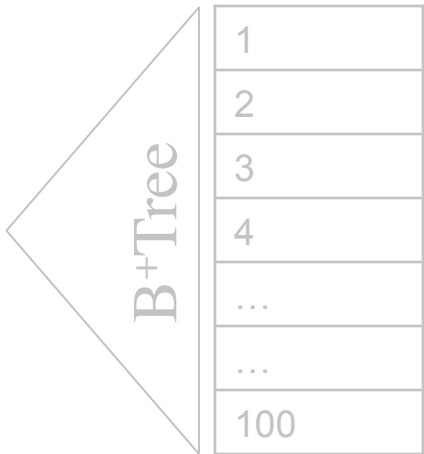
V(M, N);

Suppose V(M,N) contains 10,000 records:  
(1,1), (1,2), (1,3), ..., (1,100), (2,1),..., (100, 100)

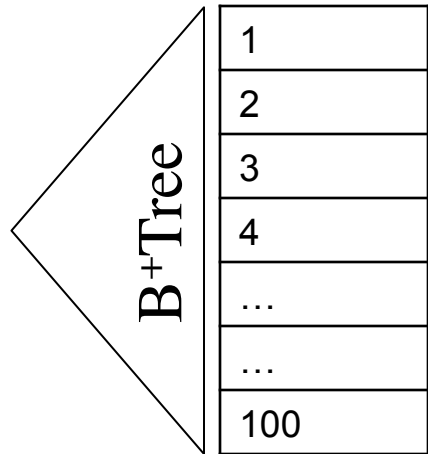
```
SELECT *  
FROM V  
WHERE M=3
```

```
SELECT *  
FROM V  
WHERE N=5
```

```
SELECT *  
FROM V  
WHERE M=3 and N=5
```



Index on V(M)



Index on V(N)

Where does this index help?



# Review: Indexes

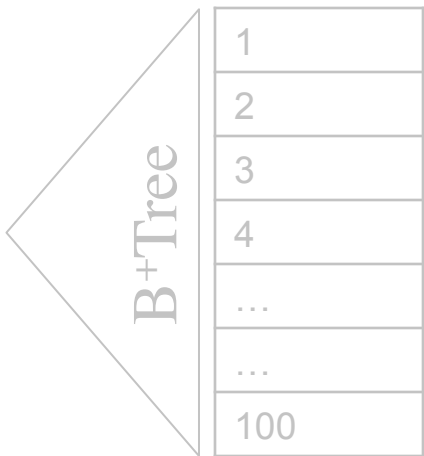
V(M, N);

Suppose V(M,N) contains 10,000 records:  
(1,1), (1,2), (1,3), ..., (1,100), (2,1),..., (100, 100)

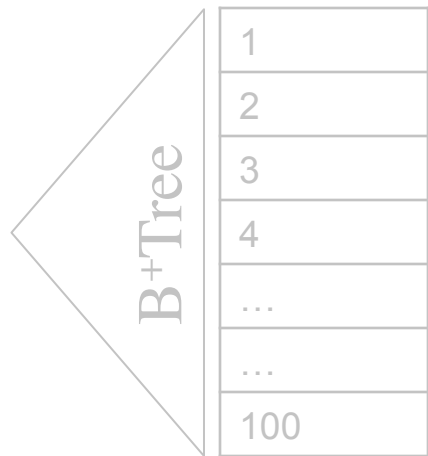
```
SELECT *  
FROM V  
WHERE M=3
```

```
SELECT *  
FROM V  
WHERE N=5
```

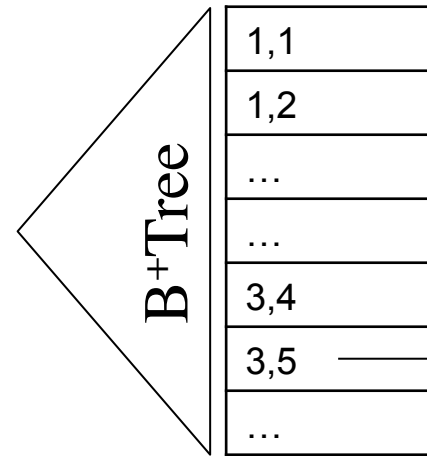
```
SELECT *  
FROM V  
WHERE M=3 and N=5
```



Index on V(M)



Index on V(N)



Index on V(M,N)

And this?

Single pointer to record (3,5)

# Review: Indexes

Suppose  $M$  is the primary key in  $V(\underline{M}, N)$ :

How do the two indexes  $V(M)$  and  $V(M,N)$  compare?

Consider their utility for these predicates:

- $M=5$
- $M=5$  and  $N=7$

# Nested Queries

- Subqueries can occur in every clause:
  - SELECT
  - FROM
  - WHERE
- When we must use nested subqueries:
  - Non-monotone queries
  - Queries making complex use of aggregates
  - “Finding witnesses”

# Practice these queries in SQL

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

Ullman's drinkers-bars-beers example

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

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

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

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

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

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

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

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

Product (pname, price, cid)  
Company(cid, cname, city)

# Unnesting Aggregates

Find the number of companies in each city

```
SELECT DISTINCT X.city, (SELECT count(*)  
                          FROM Company Y  
                          WHERE X.city = Y.city)  
FROM Company X
```

```
SELECT city, count(*)  
FROM Company  
GROUP BY city
```

Equivalent queries

Note: no need for **DISTINCT**  
(**DISTINCT** *is the same* as **GROUP BY**)

Product (pname, price, cid)  
Company(cid, cname, city)

## Unnesting Aggregates

Find the number of products made in each city

```
SELECT DISTINCT X.city, (SELECT count(*)  
                          FROM Product Y, Company Z  
                          WHERE Z.cid=Y.cid  
                          AND Z.city = X.city)  
FROM Company X
```

```
SELECT X.city, count(*)  
FROM Company X, Product Y  
WHERE X.cid=Y.cid  
GROUP BY X.city
```

NOT equivalent !  
You should know why!

Purchase(pid, product, quantity, price)

# GROUP BY v.s. Nested Queries

```
SELECT    product, Sum(quantity) AS TotalSales
FROM      Purchase
WHERE     price > 1
GROUP BY  product
```

```
SELECT DISTINCT x.product, (SELECT Sum(y.quantity)
                             FROM Purchase y
                             WHERE x.product = y.product
                             AND y.price > 1)
AS TotalSales
FROM      Purchase x
WHERE     x.price > 1
```

Why twice ?

Author(login,name)

Wrote(login,url)

## More Unnesting

Find authors who wrote  $\geq 10$  documents:



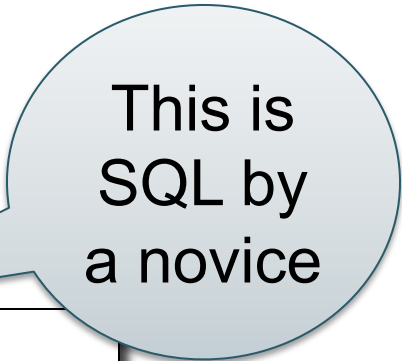
Author(login,name)

Wrote(login,url)

## More Unnesting

Find authors who wrote  $\geq 10$  documents:

Attempt 1: with nested queries



This is  
SQL by  
a novice

```
SELECT DISTINCT Author.name
FROM Author
WHERE (SELECT count(Wrote.url)
      FROM Wrote
      WHERE Author.login=Wrote.login)
      >= 10
```

Author(login,name)

Wrote(login,url)


## More Unnesting

Find authors who wrote  $\geq 10$  documents:

Attempt 1: with nested queries

Attempt 2: using GROUP BY and HAVING

```
SELECT Author.name
FROM Author, Wrote
WHERE Author.login=Wrote.login
GROUP BY Author.name
HAVING count(wrote.url) >= 10
```



This is  
SQL by  
an expert

Product (pname, price, cid)

Company(cid, cname, city)

## Finding Witnesses

For each city, find the most expensive product made in that city

Product (pname, price, cid)

Company(cid, cname, city)

## Finding Witnesses

For each city, find the most expensive product made in that city

Finding the maximum price is easy...

```
SELECT x.city, max(y.price)
FROM Company x, Product y
WHERE x.cid = y.cid
GROUP BY x.city;
```

But we need the *witnesses*, i.e. the products with max price

Product (pname, price, cid)

Company(cid, cname, city)

## Finding Witnesses

To find the witnesses, compute the maximum price in a subquery

```
SELECT DISTINCT u.city, v.pname, v.price
FROM Company u, Product v,
  (SELECT x.city, max(y.price) as maxprice
   FROM Company x, Product y
   WHERE x.cid = y.cid
   GROUP BY x.city) w
WHERE u.cid = v.cid
      and u.city = w.city
      and v.price=w.maxprice;
```

Product (pname, price, cid)

Company(cid, cname, city)

## Finding Witnesses

There is a more concise solution here:

```
SELECT u.city, v.pname, v.price
FROM Company u, Product v, Company x, Product y
WHERE u.cid = v.cid and u.city = x.city and x.cid = y.cid
GROUP BY u.city, v.pname, v.price
HAVING v.price = max(y.price);
```

Product (pname, price, cid)

Company(cid, cname, city)

## Finding Witnesses

And another one:

```
SELECT u.city, v.pname, v.price
FROM Company u, Product v
WHERE u.cid = v.cid
  and v.price >= ALL (SELECT y.price
                     FROM Company x, Product y
                     WHERE u.city=x.city
                        and x.cid=y.cid);
```

# Where We Are

- Motivation for using a DBMS for managing data
- SQL, SQL, SQL
  - Declaring the schema for our data (CREATE TABLE)
  - Inserting data one row at a time or in bulk (INSERT/.import)
  - Modifying the schema and updating the data (ALTER/UPDATE)
  - Querying the data (SELECT)
  - Tuning queries (CREATE INDEX)
- Next step: More knowledge of how DBMSs work
  - Client-server architecture
  - Relational algebra and query execution