Introduction to Data Management CSE 344

Lecture 9: SQL Wrap-up and RDBMs Architecture

Announcements

- Webquiz due on Monday, 1/28
- Homework 3 is posted: due on Wednesday, 2/6

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

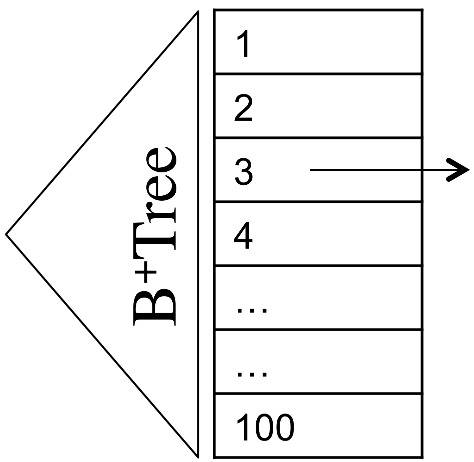
V(M, N);

Suppose V(M,N) contains 10,000 records:
(1,1), (1,2), …, (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);

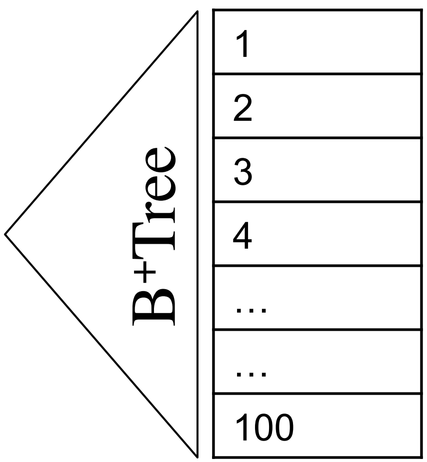
Suppose V(M,N) contains 10,000 records:
(1,1), (1,2), …, (100, 100)

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

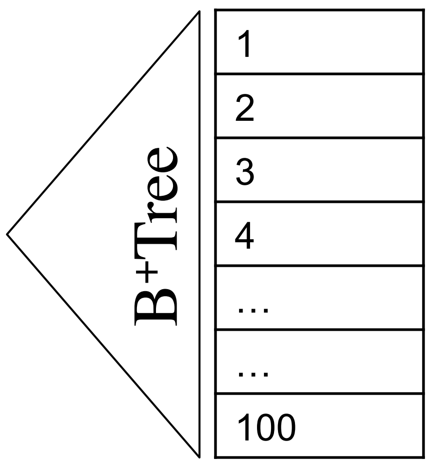
```
SELECT *
FROM V
WHERE N=5
```

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

How do we compute this query?



Index on V(M)



Index on V(N)

Review: Indexes

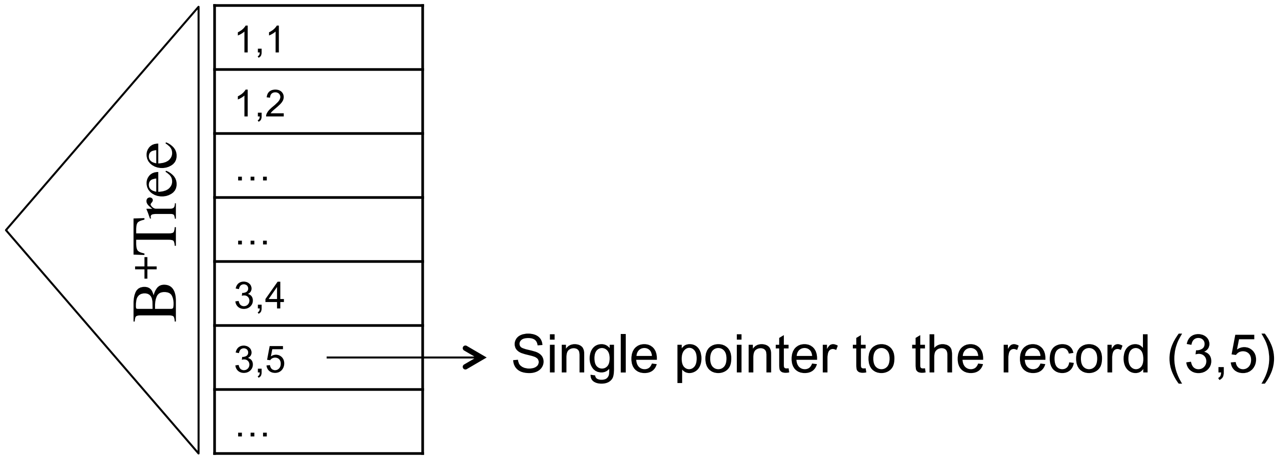
```
V(M, N);
```

Suppose V(M,N) contains 10,000 records: (1,1), (1,2), …, (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,N)

Review: Indexes

Discussion

- Why not create all three indexes V(M), V(N), V(M,N)?
- Suppose M is the primary key in V(<u>M</u>, N):
  V = {(1,1), (2,2), …, (10000, 10000)}
  How do the two indexes V(M) and V(M,N) compare? Consider their utility for evaluating the predicate M=5

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

Review: Subqueries in WHERE

Find all companies s.t. all their products have price < 200

same as:

Find all companies that make only products with price < 200

Universal quantifiers

Universal quantifiers are hard ! ☹

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

Review: Subqueries in WHERE

Find all companies s.t. all their products have price < 200

1. Find *the other* companies: i.e. s.t. some product ≥ 200

```
SELECT DISTINCT  C.cname
FROM     Company C
WHERE  C.cid IN (SELECT P.cid
                          FROM Product P
                          WHERE P.price >= 200)
```

2. Find all companies s.t. all their products have price < 200

```
SELECT DISTINCT  C.cname
FROM     Company C
WHERE  C.cid NOT IN (SELECT P.cid
                                FROM Product P
                                WHERE P.price >= 200)
```

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

Review: Subqueries in WHERE

Find all companies s.t. all their products have price < 200

Universal quantifiers

Using EXISTS:

```
SELECT DISTINCT  C.cname
FROM     Company C
WHERE NOT EXISTS (SELECT *
                  FROM Product P
                  WHERE P.cid = C.cid and P.price >= 200)
```

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

Review: Subqueries in WHERE

Find all companies s.t. all their products have price < 200

Universal quantifiers

Using ALL:

```
SELECT DISTINCT  C.cname
FROM     Company C
WHERE 200 > ALL  (SELECT price
                  FROM Product P
                  WHERE P.cid = C.cid)
```

Question for Database Fans and their Friends

- Can we unnest the *universal quantifier* query ?

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

Monotone Queries

- Definition A query Q is monotone if:
  - Whenever we add tuples to one or more input tables, the answer to the query will not lose any of of the tuples

Product

pname	price	cid
Gizmo	19.99	c001
Gadget	999.99	c003
Camera	149.99	c001

Company

cid	cname	city
c001	Sunworks	Bonn
c002	DB Inc.	Lyon
c003	Builder	Lodtz

Q →

A	B
149.99	Lodtz
19.99	Lyon

Is the mystery query monotone?

Product

pname	price	cid
Gizmo	19.99	c001
Gadget	999.99	c003
Camera	149.99	c001
iPad	499.99	c001

Company

cid	cname	city
c001	Sunworks	Bonn
c002	DB Inc.	Lyon
c003	Builder	Lodtz

Q →

A	B
149.99	Lyon
19.99	Lyon
19.99	Bonn
149.99	Bonn

Monotone Queries

- Theorem: A SELECT-FROM-WHERE query (without subqueries or aggregates) is monotone.

- Proof. We use the nested loop semantics: if we insert a tuple in a relation R_i, this will not remove any tuples from the answer

```
SELECT a1, a2, …, ak
FROM   R1 AS x1, R2 AS x2, …, Rn AS xn
WHERE  Conditions
```

```
for x1 in R1 do
    for x2 in R2 do
        .....
            for xn in Rn do
                if Conditions
                    output (a1,…,ak)
```

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

Monotone Queries

- The query:

Find all companies s.t. all their products have price < 200

is not monotone

pname	price	cid
Gizmo	19.99	c001

cid	cname	city
c001	Sunworks	Bonn

cname
Sunworks

pname	price	cid
Gizmo	19.99	c001
Gadget	999.99	c001

cid	cname	city
c001	Sunworks	Bonn

cname

- Consequence: we cannot write it as a SELECT-FROM-WHERE query without nested subqueries

Queries that must be nested

- Queries with universal quantifiers or with negation

- Queries that have complex aggregates

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.

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

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

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

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

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

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

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

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 price > 1)
                            AS TotalSales
FROM        Purchase x
WHERE       price > 1
```

Why twice ?

Product (<u>pname</u>, price, cid)
Company(<u>cid</u>, cname, city)

Unnesting Aggregates

Find the number of companies in each city

```
SELECT DISTINCT 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

What if there are no products for a city?

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
```

They are NOT equivalent ! (WHY?)

More Unnesting

Author(login,name)

Wrote(login,url)

- Find authors who wrote ≥ 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
```

More Unnesting

- Find all authors who wrote at least 10 documents:
- Attempt 2: SQL style (with GROUP BY)

```
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:

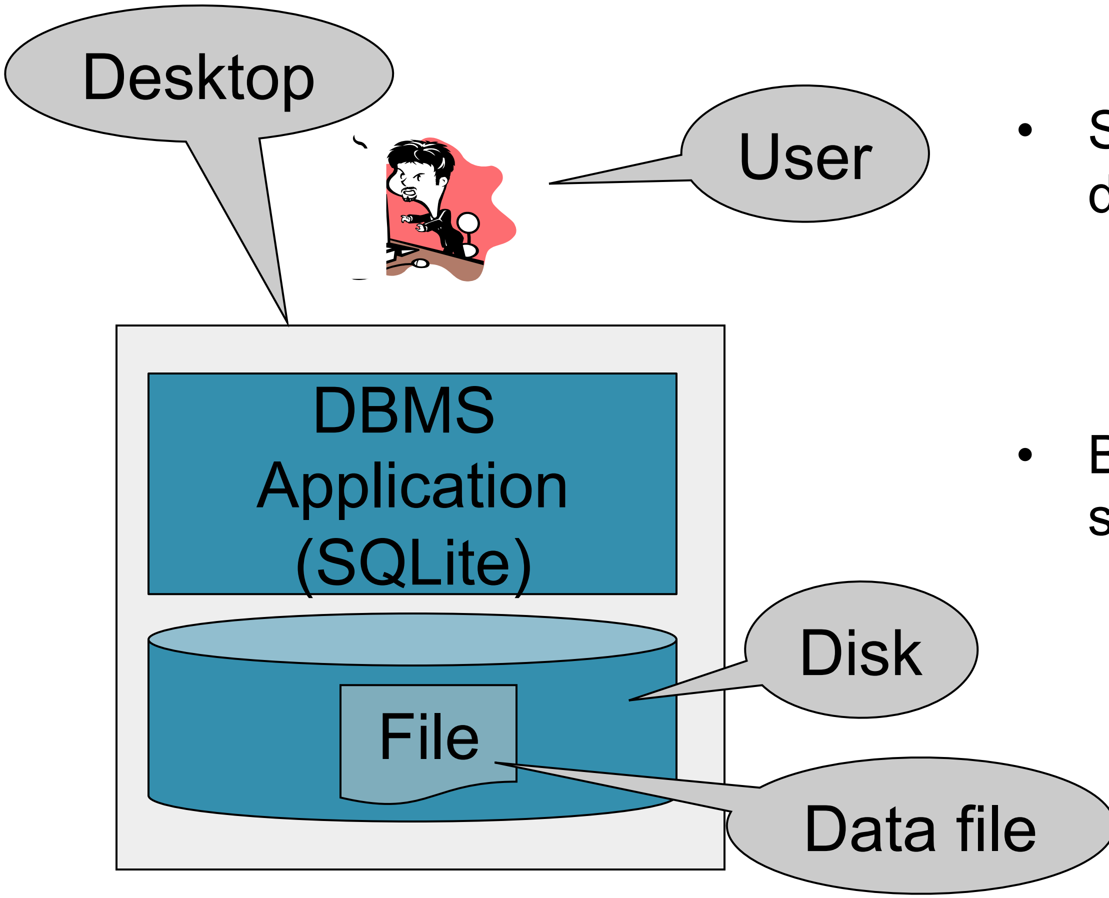
```
SELECT u.city, v.pname, v.price
FROM Company u, Product v
WHERE u.cid = v.cid
  and v.price >= ALL (SELECT y.price
                      FROM Company x, Product y
                      WHERE u.city=x.city
                            and x.cid=y.cid);
```

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)

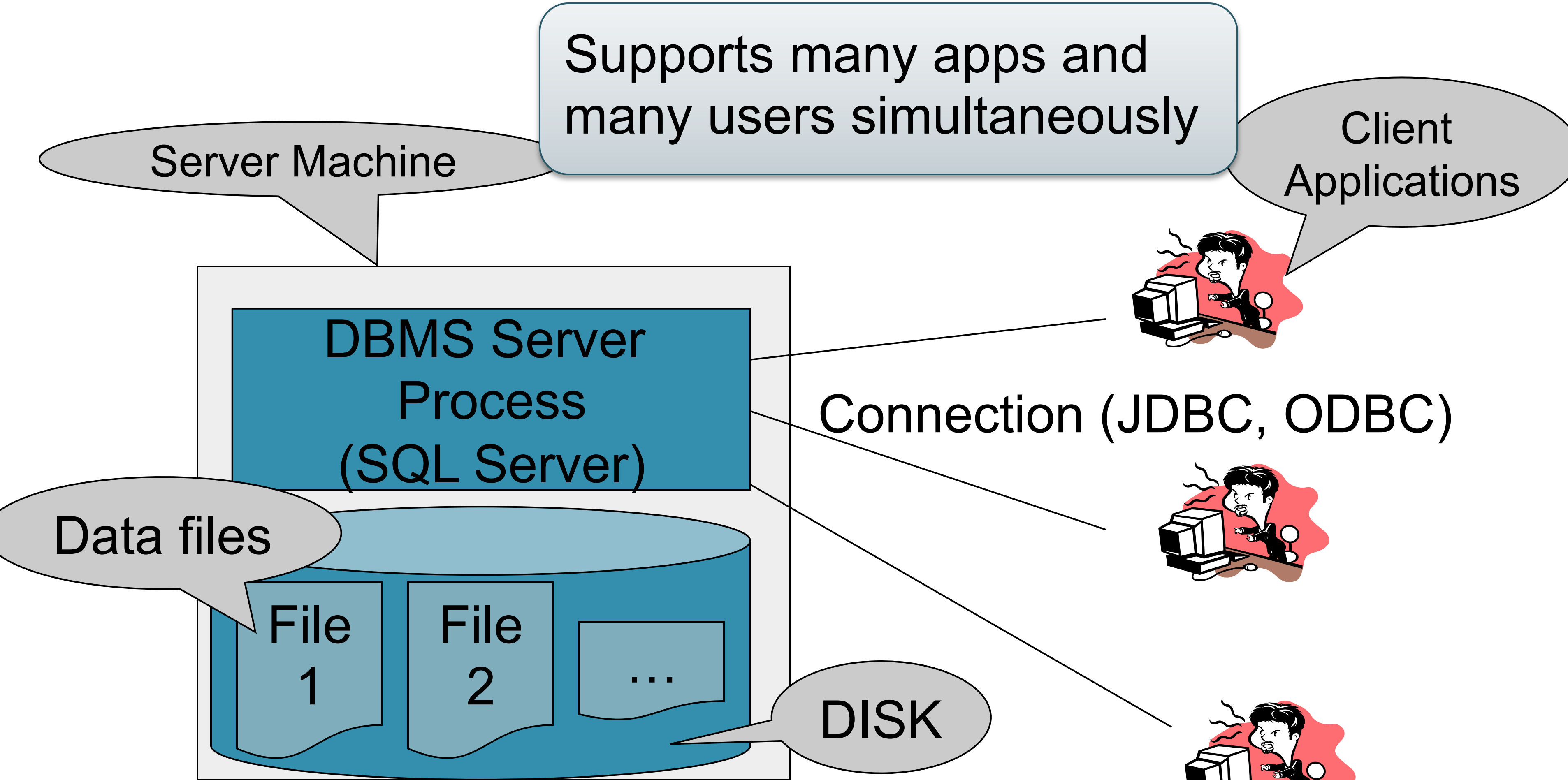
- Next step: More knowledge of how DBMSs work
  - Client-server architecture
  - Relational algebra and query execution

Data Management with SQLite



- So far, we have been managing data with SQLite as follows:
  - One data file
  - One user
  - One DBMS application
- But only a limited number of scenarios work with such model

Client-Server Architecture

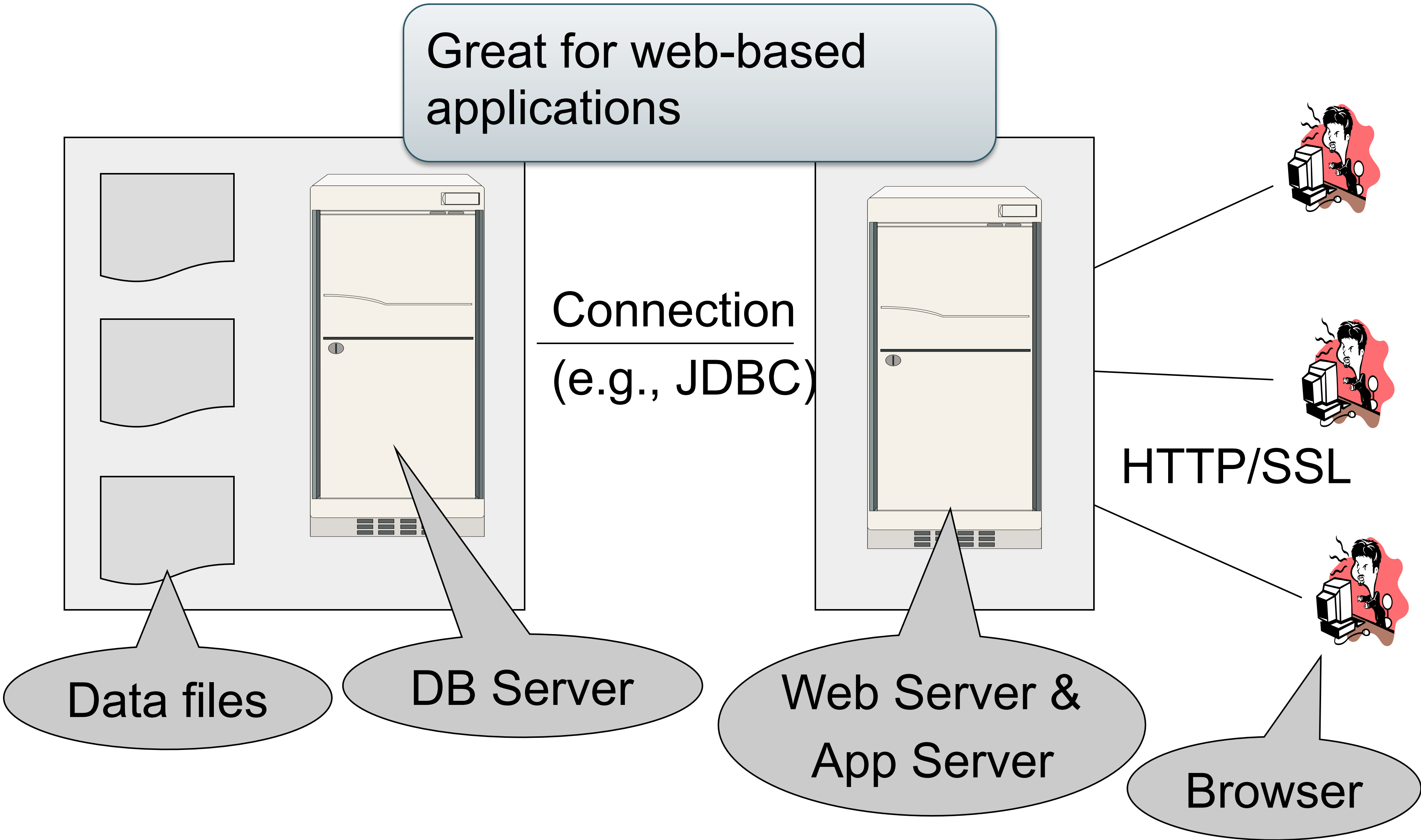


- One server running the database
- Many clients, connecting via the ODBC or JDBC (Java Database Connectivity) protocol

Client-Server Architecture

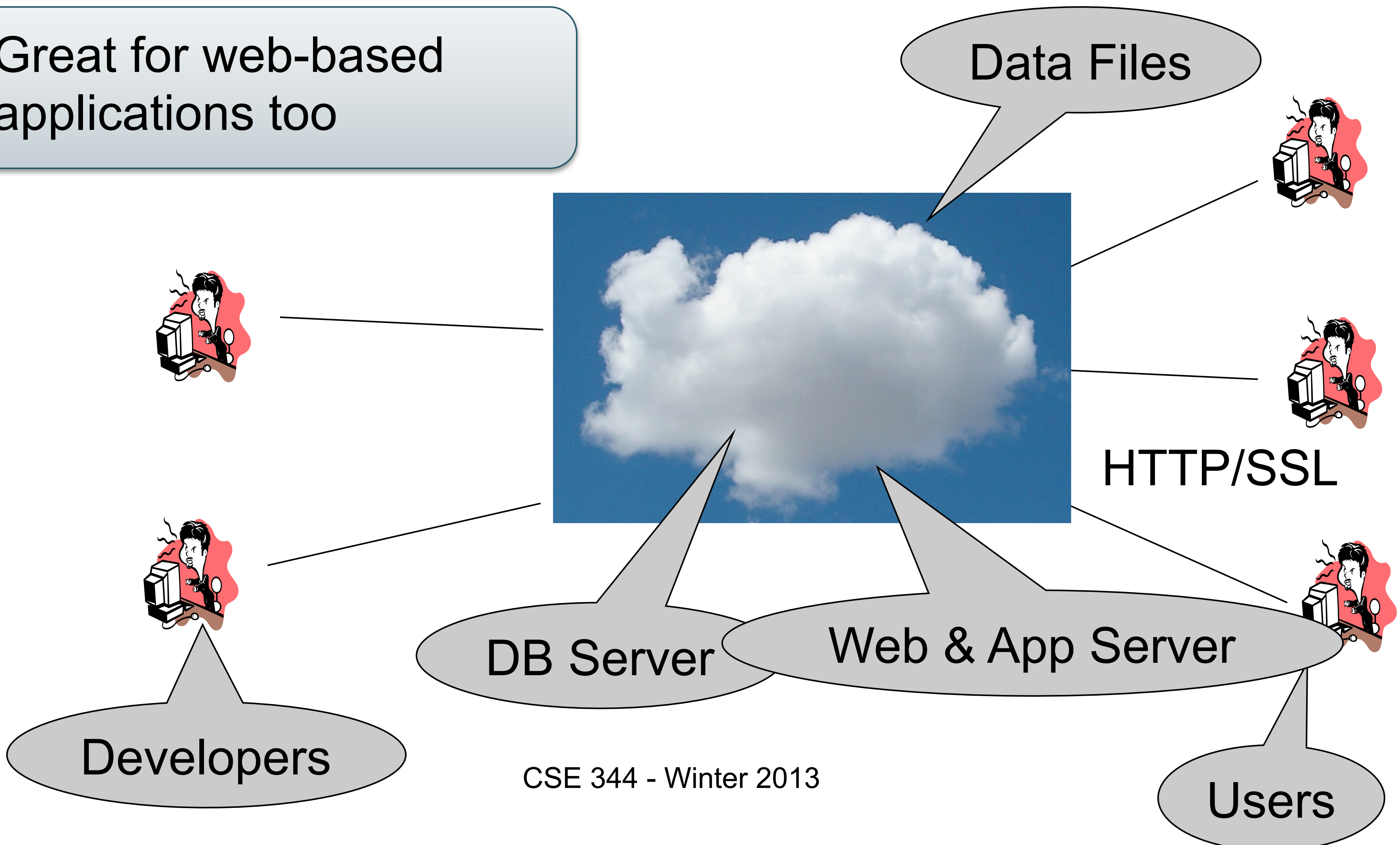
- One *server* that runs the DBMS (or RDBMS):
  - Your own desktop, or
  - Some beefy system, or
  - A cloud service (SQL Azure)
- Many *clients* run apps and connect to DBMS
  - Microsoft's Management Studio (for SQL Server), or
  - psql (for postgres)
  - Some Java program (HW5) or some C++ program
- Clients "talk" to server using JDBC/ODBC protocol

DBMS Deployment: 3 Tiers



DBMS Deployment: Cloud

Great for web-based applications too



Using a DBMS Server

1. Client application establishes connection to server
2. Client must authenticate self
3. Client submits SQL commands to server
4. Server executes commands and returns results