DATA516/CSED516
Scalable Data Systems and Algorithms

Lecture 1

Design of a Relational DBMS

Course Staff

- Instructor: Dan Suciu
  suciu@cs.washington.edu

- TA: Remy Wang
  remywang@cs.washington.edu

- TA: Zechariah Cheung
  zachcheu@gmail.com

Coarse Aims

- Study design of big data systems
  - Historical perspective
  - Sample of modern systems
  - Breadth of designs (relational, streaming, graph, etc.)
- Study key scalable data processing algorithms
- Gain hands-on experience with big data systems
  - Demonstrations and tutorials in sections
  - Assignments and projects

Coarse Content

- Query processing: single-sever, distributed
- MapReduce, legacy, successors
- Some important “Big data” algorithms
- Misc: streaming, column stores, graph engines

Course Format

- 5pm-7:50pm: Lectures
  - Discuss system architecture & algorithms
- 8pm-8:50pm: Hands-on tutorials
  - Learn how to use big data systems
  - Jump start your homeworks
  - Bring your laptop!

Grading (subject to change!)

- 15%: Reading assigned papers
  - Write short statement/review
- 60%: Homework assignments
  - Redshift Spark, Snowflake, others
- 25%: Final project

Project

Choose a topic:

- Don't worry about novelty!
- Highly recommended: Benchmark projects
  - Analyze the performance of some features
  - Compare the performance of different systems
  - Try to implement an interesting workload
- I will post a few ideas, but you are strongly encouraged to come up with your own

Project

1. Project proposal (1 page)
2. Project milestone (2-3 pages)
3. Project presentation (in class)
4. Project final report (4-5 pages)

Web Services

- HW1: Amazon Redshift – attend today's section!
- HW2: Spark/AWS
- HW3: Snowflake – see Remy's post
- HW4: mini-homeworks – stay tuned

Azure: optional, for the project

Communication

- Course webpage: all important stuff
  https://courses.cs.washington.edu/courses/csed516/20au/
- Discussion Board: ED. Say “hello”!
- Class email: only for important announcements

How to Turn In

https://gitlab.cs.washington.edu/

- Your own repository
- Pull to get homework instructions, starter files
- Push homework solutions, project reports

Reviews: we use google forms

- Typically around ½ page
- Goal is only for us to check that you have read the paper

Relational Database Management Systems

Quick Review

- Database is a collection of files
- Database management system (DBMS) is a piece of software to help manage that data
- History:
  - Origins in the 1960’s
  - Relational model 1970
  - First relational DBMSs (Ingres and System R): 1970’s
  - Parallel DBMSs: 1980’s

DBMS Functionality

1. Describe real-world entities in terms of a data model
2. Create & persistently store large datasets
3. Efficiently query & update
   1. Must handle complex questions about data
   2. Must handle sophisticated updates
   3. Performance matters
4. Change structure (e.g., add attributes)
5. Concurrency control: enable simultaneous updates
6. Crash recovery
7. Access control, security, integrity

Relational Data Model

- A Database is a collection of relations

- A Relation is a subset of $\mathbf{Dom_1}$ **x** $\mathbf{Dom_2}$ **x … x** $\mathbf{Dom_n}$
  - Where $\mathbf{Dom_i}$ is the domain of attribute **i**
  - **n** is number of attributes of the relation
  - A relation **R** is a set of tuples

- A Tuple t is an element of $\mathbf{Dom_1}$ **x** $\mathbf{Dom_2}$ **x … x** $\mathbf{Dom_n}$

Other names: relation = table; tuple = row

Discussion

- Rows in a relation:

  Data independence!

  - Ordering immaterial (a relation is a set)
  - All rows are distinct – set semantics
  - Query answers may have duplicates – bag semantics
- Columns in a tuple:

  Or is it?

  - Ordering is significant
  - Applications refer to columns by their names
- Domain of each column is a primitive type

Schema

- Relation schema: describes column heads
  - Relation name
  - Name of each field (or column, or attribute)
  - Domain of each field
  - The *arity* of the relation = # attributes

- Database schema: set of all relation schemas

Instance

- Relation instance: concrete table content
  - Set of tuples (also called records) matching the schema
  - The *cardinality* of the relation = # tuples
    (a.k.a. size)

- Database instance: set of all relation instances

What is the schema?
What is the instance?

Supplier

sno	sname	scity	sstate
1	s1	city 1	WA
2	s2	city 1	WA
3	s3	city 2	MA
4	s4	city 2	MA

What is the schema?
What is the instance?

Relation schema

Supplier(sno: integer, sname: string, scity: string, sstate: string)

Supplier

sno	**sname**	**scity**	**sstate**
1	s1	city 1	WA
2	s2	city 1	WA
3	s3	city 2	MA
4	s4	city 2	MA

instance

Relational Query Language

- Set-at-a-time:
  - Query inputs and outputs are relations

- Two variants of the query language:
  - Relational algebra: specifies order of operations
  - Relational calculus / SQL: declarative

Note

- We will review Relational Algebra and SQL today

- In addition: please review at home:
  - Review material from DATA514/CSED514

Structured Query Language: SQL

- Data definition language: DDL
  - Statements to create, modify tables and views
  - CREATE TABLE …,
    CREATE VIEW …,
    ALTER TABLE…
- Data manipulation language: DML
  - Statements to issue queries, insert, delete data
  - SELECT-FROM-WHERE…,
    INSERT…,
    UPDATE…,
    DELETE…

We focus on this

SQL Query

Basic form: (plus many many more bells and whistles)

```
SELECT  <attributes>
FROM    <one or more relations>
WHERE   <conditions>
```

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Quick Review of SQL

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Quick Review of SQL

```
SELECT DISTINCT  z.pno, z.pname
FROM       Supplier x, Supply y, Part z
WHERE      x.sno = y.sno
       and y.pno = z.pno
       and x.scity = 'Seattle'
       and y.price < 100
```

What does this query compute?

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Terminology

- Selection: return a subset of the rows:
  - SELECT * FROM Supplier
    WHERE scity = 'Seattle'

- Projection: return subset of the columns:
  - SELECT DISTINCT scity FROM Supplier;

- Join: refers to combining two or more tables
  - SELECT * FROM Supplier, Supply, Part …

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Self-Joins

Find the Parts numbers available both from suppliers in Seattle, and suppliers in Portland

Supplier(sno,sname,scity,sstate)

Supply(sno,pno,qty,price)

Part(pno,pname,psize,pcolor)

Self-Joins

Find the Parts numbers available both from suppliers in Seattle, and suppliers in Portland

```
SELECT DISTINCT  y1.pno
FROM    Supplier x1, Supplier x2, Supply y1, Supply y2
WHERE   x1.scity = 'Seattle'
        and x1.sno = y1.sno
        and x2.scity = 'Portland'
        and x2.sno = y2.sno
        and y1.pno = y2.pno
```

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Self-Joins

Find the Parts numbers available both from suppliers in Seattle, and suppliers in Portland

```
SELECT DISTINCT  y1.pno
FROM      Supplier x1, Supplier x2, Supply y1, Supply y2
WHERE     x1.scity = 'Seattle'
        and x1.sno = y1.sno
        and x2.scity = 'Portland'
        and x2.sno = y2.sno
        and y1.pno = y2.pno
```

Self-join

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Simple Analytics

For each part, compute its minimum and maximum price from all suppliers.

```
SELECT z.pno, z.pname, min(y.price) as p1, max(y.price) as p2
FROM     Supply y, Part z
WHERE    y.pno = z.pno
GROUP BY z.pno, z.pname
```

Terminology

- Online Analytical Processing (OLAP) a.k.a. Data Analytics queries
  - GROUP-BY + aggregates
  - No updates
  - Touch most of, or all the data
  - Very important in data science!

Data Science

Terminology

- Online Analytical Processing (OLAP) a.k.a. Data Analytics queries
  - GROUP-BY + aggregates
  - No updates
  - Touch most of, or all the data
  - Very important in data science!
- Online Transaction Processing (OLTP):
  - Point queries: return account 12345
  - Often have updates

Data Science

Other use of Relational Data

- Sparse vectors, matrics
- Graph databases

Sparse Matrics

$$A = \begin{bmatrix} 5 & 0 & -2 \\ 0 & 0 & -1 \\ 0 & 7 & 0 \end{bmatrix}$$

How can we represent it as a relation?

Sparse Matrics

$$A = \begin{bmatrix} 5 & 0 & -2 \\ 0 & 0 & -1 \\ 0 & 7 & 0 \end{bmatrix}$$

Row	Col	Val
1	1	5
1	3	-2
2	3	-1
3	2	7

Matrix Multiplication in SQL

$C = A \cdot B$

Matrix Multiplication in SQL

$$C = A \cdot B$$

$$C_{ik} = \sum_j A_{ij} \cdot B_{jk}$$

Matrix Multiplication in SQL

$$C = A \cdot B$$

$$C_{ik} = \sum_j A_{ij} \cdot B_{jk}$$

```
SELECT A.row, B.col, sum(A.val*B.val)
FROM A, B
WHERE A.col = B.row
GROUP BY A.row, B.col;
```

Discussion

- Matrix multiplication = join + group-by
- Many operations can be written in SQL
- E.g. try at home: write in SQL

  $$Tr(A \cdot B \cdot C)$$

  where the trace is defined as:

  $$Tr(X) = \sum_i X_{ii}$$

- Surprisingly, $A + B$ is a bit harder…

Matrix Addition in SQL

$$C = A + B$$

Matrix Addition in SQL

$$C = A + B$$

```
SELECT A.row, A.col, A.val + B.val as val
FROM   A, B
WHERE  A.row = B.row and A.col = B.col
```

Matrix Addition in SQL

$$C = A + B$$

```
SELECT A.row, A.col, A.val + B.val as val
FROM   A, B
WHERE  A.row = B.row and A.col = B.col
```

Why is this wrong?

Solution 1: Outer Joins

$$C = A + B$$

```
SELECT




FROM A full outer join B ON A.row = B.row and A.col = B.col;
```

Solution 1: Outer Joins

$$C = A + B$$

```
SELECT


 (CASE WHEN A.val is null THEN 0 ELSE A.val END) +
 (CASE WHEN B.val is null THEN 0 ELSE B.val END)  as val
FROM A full outer join B ON A.row = B.row and A.col = B.col;
```

Solution 1: Outer Joins

$$C = A + B$$

```
SELECT
 (CASE WHEN A.row is null THEN B.row ELSE A.row END) as row,

 (CASE WHEN A.val is null THEN 0 ELSE A.val END) +
 (CASE WHEN B.val is null THEN 0 ELSE B.val END)  as val
FROM A full outer join B ON A.row = B.row and A.col = B.col;
```

Solution 1: Outer Joins

$$C = A + B$$

```
SELECT
 (CASE WHEN A.row is null THEN B.row ELSE A.row END) as row,
 (CASE WHEN A.col is null THEN B.col ELSE A.col END) as col,
 (CASE WHEN A.val is null THEN 0 ELSE A.val END) +
 (CASE WHEN B.val is null THEN 0 ELSE B.val END)  as val
FROM A full outer join B ON A.row = B.row and A.col = B.col;
```

Discussion

- Outer joins: includes a tuple even if it doesn’t join with anything in the other table
- Left outer join, right outer join, full outer join – what do they mean?
- Note distinction between ON and WHERE

WHERE v.s. ON

Sparse vectors:

	1	2	3	4	5	6	7	8	9
X=	50	0	-30	60	-80	0	-90	10	0
Y=	-55	0	65	-15	0	35	-75	15	25

WHERE v.s. ON

Sparse vectors:

	1	2	3	4	5	6	7	8	9
X=	50	0	-30	60	-80	0	-90	10	0
Y=	-55	0	65	-15	0	35	-75	15	25

```
SELECT x.pos, x.val, y.val
FROM x left outer join y
  ON x.pos = y.pos and y.val > 0;
```

WHERE v.s. ON

Sparse vectors:

	1	2	3	4	5	6	7	8	9
X=	50	0	-30	60	-80	0	-90	10	0
Y=	-55	0	65	-15	0	35	-75	15	25

```
SELECT x.pos, x.val, y.val
FROM x left outer join y
  ON x.pos = y.pos and y.val > 0;
```

v.s.

```
SELECT x.pos, x.val, y.val
FROM x left outer join y
  ON x.pos = y.pos
WHERE y.val > 0;
```

WHERE v.s. ON

Sparse vectors:

	1	2	3	4	5	6	7	8	9
X=	50	0	-30	60	-80	0	-90	10	0
Y=	-55	0	65	-15	0	35	-75	15	25

```
SELECT x.pos, x.val, y.val
FROM x left outer join y
  ON x.pos = y.pos and y.val > 0;
```

x.pos	x.val	y.val
1	50	Null
3	-30	65
4	60	Null
5	-80	Null
7	-90	Null
8	10	15

v.s.

```
SELECT x.pos, x.val, y.val
FROM x left outer join y
  ON x.pos = y.pos
WHERE y.val > 0;
```

x.pos	x.val	y.val
3	-30	65
8	10	15

Solution 2: Group By

$$C = A + B$$

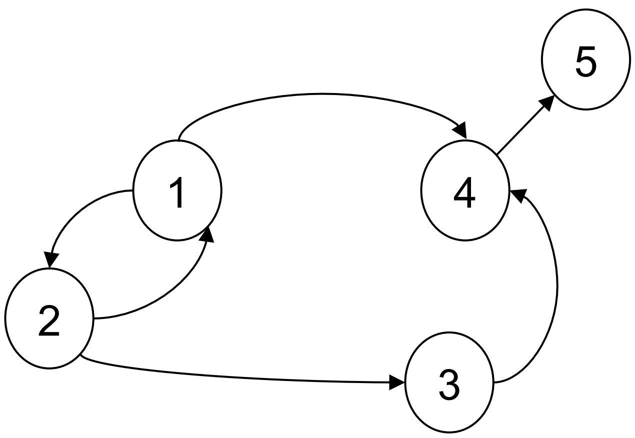
```
SELECT m.row, m.col, sum(m.val)
FROM (SELECT * FROM A
          UNION ALL
        SELECT * FROM B) as m
GROUP BY m.row, m.col;
```

Graph Databases

- Graph databases systems are a niche category of products specialized for processing large graphs
- E.g. Neo4J, TigerGraph
- A graph is a special case of a relation, and can be processed using SQL

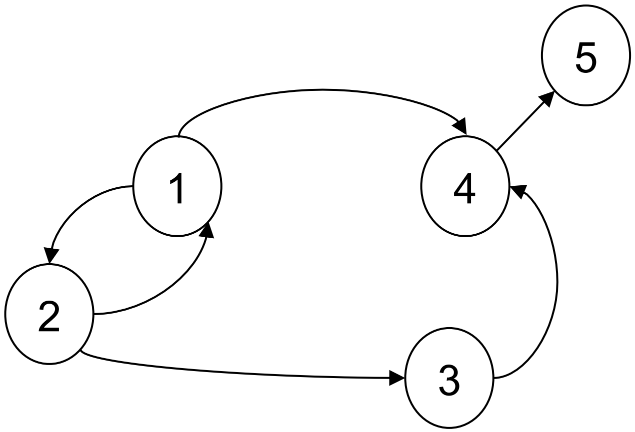
Graph Databases

A graph:



Graph Databases

A graph:



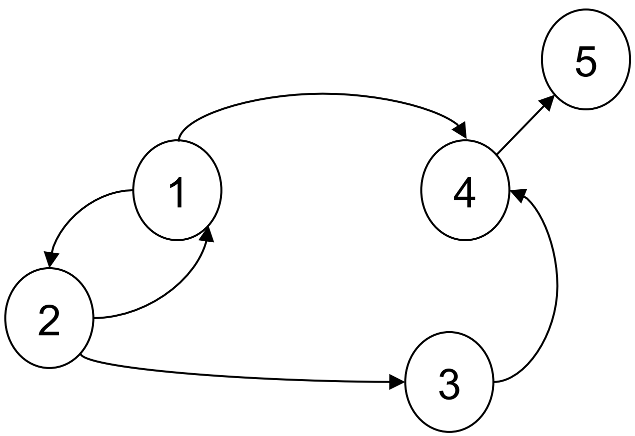
A relation:

Edge

src	dst
1	2
2	1
2	3
1	4
3	4
4	5

Graph Databases

A graph:



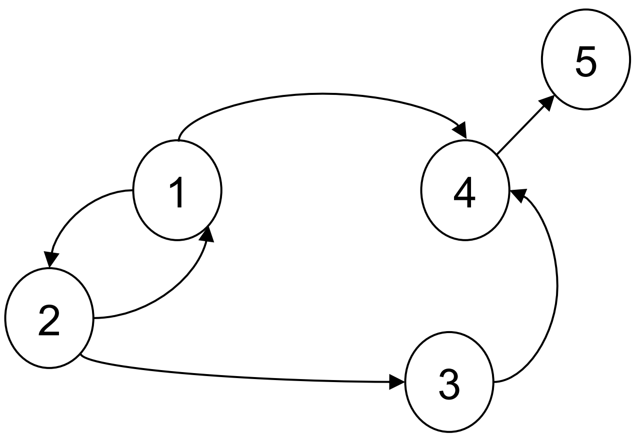
A relation: Edge

src	dst
1	2
2	1
2	3
1	4
3	4
4	5

Find nodes at distance 2: $\{(x,z) \mid \exists y\ Edge(x,y) \wedge Edge(y,z)\}$

Graph Databases

A graph:



A relation:

Edge

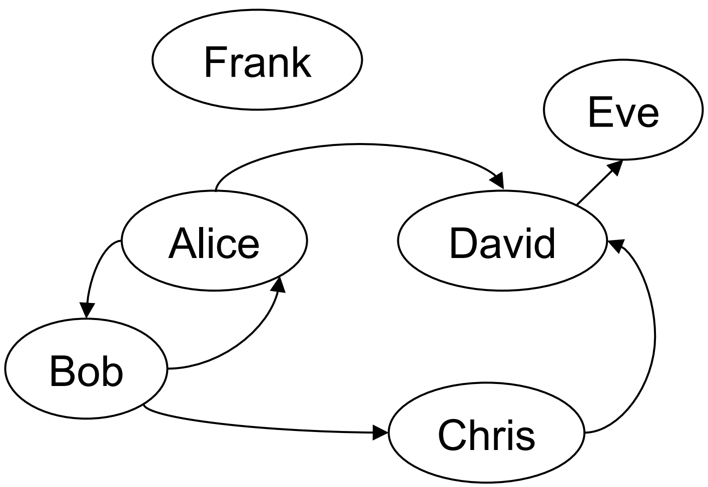
src	dst
1	2
2	1
2	3
1	4
3	4
4	5

Find nodes at distance 2: $\{(x, z) | \exists y\ Edge(x, y) \wedge Edge(y, z)\}$

```
SELECT DISTINCT e1.src as X, e2.dst as Z
FROM Edge e1, Edge e2
WHERE e1.dst = e2.src;
```

Other Representation

Representing nodes separately;
needed for “isolated nodes” e.g. Frank



Node

src
Alice
Bob
Chris
David
Eve
Frank

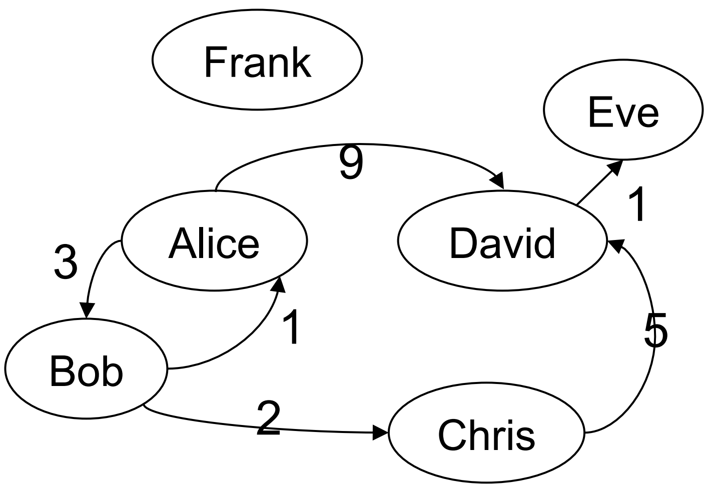
Edge

src	dst
Alice	Bob
Bob	Alice
Bob	Chris
Alice	David
Chris	David
David	Eve

Other Representation

Adding edge labels

Adding node labels…



Node

src
Alice
Bob
Chris
David
Eve
Frank

Edge

src	dst	weight
Alice	Bob	3
Bob	Alice	1
Bob	Chris	2
Alice	David	9
Chris	David	5
David	Eve	1

Limitations of SQL

- No recursion! Examples requiring recursion:
  - Gradient descent
  - Connected components in a graph
- Advanced systems *do* support recursion
- Practical solution: use some external driver, e.g. pyton

Example: Logistic Regression

Tom Mitchell: Machine Learning

Data

X1	X2	X3	Y
3	9	3	0
3	5	7	1
6	2	2	0
3	6	3	0
5	5	9	1
9	3	3	1
…	…	…	
…	…	…	

Example: Logistic Regression

Tom Mitchell: Machine Learning

Data

X1	X2	X3	Y
3	9	3	0
3	5	7	1
6	2	2	0
3	6	3	0
5	5	9	1
9	3	3	1
…	…	…	
…	…	…	

Switched
(following Mitchell)

$$P(Y=0|X) = \frac{1}{1+exp\left(w_0 + \sum_{i=1,3} w_i X_i\right)}$$

$$P(Y=1|X) = \frac{exp\left(w_0 + \sum_{i=1,3} w_i X_i\right)}{1+exp\left(w_0 + \sum_{i=1,3} w_i X_i\right)}$$

Example: Logistic Regression

Tom Mitchell: Machine Learning

Data

X1	X2	X3	Y
3	9	3	0
3	5	7	1
6	2	2	0
3	6	3	0
5	5	9	1
9	3	3	1
…	…	…	
…	…	…	

Switched
(following Mitchell)

$$P(Y=0|X)=\frac{1}{1+exp\left(w_0+\sum_{i=1,3}w_iX_i\right)}$$

$$P(Y=1|X)=\frac{exp\left(w_0+\sum_{i=1,3}w_iX_i\right)}{1+exp\left(w_0+\sum_{i=1,3}w_iX_i\right)}$$

Train weights w_0, w_1, w_2, w_3 to minimize loss:

$$L(w_0,\ldots,w_3)=\sum_{\ell=1,N}\left(Y^\ell\cdot\ln P\left(Y=1|X^\ell\right)+(1-Y^\ell)\cdot\ln P\left(Y=0|X^\ell\right)\right)$$

Example: Logistic Regression

Tom Mitchell: Machine Learning

Data

X1	X2	X3	Y
3	9	3	0
3	5	7	1
6	2	2	0
3	6	3	0
5	5	9	1
9	3	3	1
…	…	…	
…	…	…	

Gradient Descent:

$$w_i \leftarrow w_i + \eta \sum_{\ell=1,N} X_i^\ell \left(Y^\ell - P(Y=1|X^\ell)\right)$$

Example: Logistic Regression

Tom Mitchell: Machine Learning

Gradient Descent:

$$w_i \leftarrow w_i + \eta \sum_{\ell=1,N} X_i^{\ell}\left(Y^{\ell} - P(Y = 1|X^{\ell})\right)$$

Data

X1	X2	X3	Y
3	9	3	0
3	5	7	1
6	2	2	0
3	6	3	0
5	5	9	1
9	3	3	1
…	…	…	
…	…	…	

```
CREATE TABLE W (k int primary key, w0 real, w1 real, w2 real, w3 real);
INSERT INTO W VALUES (1, 0, 0, 0, 0);
```

Example: Logistic Regression

Tom Mitchell: Machine Learning

Data

X1	X2	X3	Y
3	9	3	0
3	5	7	1
6	2	2	0
3	6	3	0

Gradient Descent:

$$w_i \leftarrow w_i + \eta \sum_{\ell=1,N} X_i^\ell \left(Y^\ell - P(Y = 1 | X^\ell)\right)$$

```
CREATE TABLE W (k int primary key, w0 real, w1 real, w2 real, w3 real);
INSERT INTO W VALUES (1, 0, 0, 0, 0);
```

```
FROM data d, W
WHERE W.k=1
```

Example: Logistic Regression

Tom Mitchell: Machine Learning

Data

X1	X2	X3	Y
3	9	3	0
3	5	7	1
6	2	2	0
3	6	3	0

Gradient Descent:

$$w_i \leftarrow w_i + \eta \sum_{\ell=1,N} X_i^\ell \left(Y^\ell - P(Y = 1|X^\ell)\right)$$

```
CREATE TABLE W (k int primary key, w0 real, w1 real, w2 real, w3 real);
INSERT INTO W VALUES (1, 0, 0, 0, 0);
```

```
SELECT
  W.w0+0.01*sum(d.Y - 1 + 1/(1+exp(W.w0+W.w1*d.X1+W.w2*d.X2+W.w3*d.X3))) as w0,




FROM data d, W
WHERE W.k=1
```

Example: Logistic Regression

Tom Mitchell: Machine Learning

Data

X1	X2	X3	Y
3	9	3	0
3	5	7	1
6	2	2	0
3	6	3	0

Gradient Descent:

$$w_i \leftarrow w_i + \eta \sum_{\ell=1,N} X_i^\ell \left(Y^\ell - P(Y = 1 | X^\ell)\right)$$

```
CREATE TABLE W (k int primary key, w0 real, w1 real, w2 real, w3 real);
INSERT INTO W VALUES (1, 0, 0, 0, 0);
```

```
SELECT
  W.w0+0.01*sum(d.Y - 1 + 1/(1+exp(W.w0+W.w1*d.X1+W.w2*d.X2+W.w3*d.X3))) as w0,
  W.w1+0.01*sum(d.X1*(d.Y - 1 + 1/(1+exp(W.w0+W.w1*d.X1+W.w2*d.X2+W.w3*d.X3)))) as w1,



FROM data d, W
WHERE W.k=1
```

Example: Logistic Regression

Tom Mitchell: Machine Learning

Gradient Descent:

$$w_i \leftarrow w_i + \eta \sum_{\ell=1,N} X_i^{\ell}\left(Y^{\ell} - P(Y=1|X^{\ell})\right)$$

Data

X1	X2	X3	Y
3	9	3	0
3	5	7	1
6	2	2	[illegible]
3	6	3	[illegible]

```sql
CREATE TABLE W (k int primary key, w0 real, w1 real, w2 real, w3 real);
INSERT INTO W VALUES (1, 0, 0, 0, 0);
```

```sql
SELECT
  W.w0+0.01*sum(d.Y - 1 + 1/(1+exp(W.w0+W.w1*d.X1+W.w2*d.X2+W.w3*d.X3))) as w0,
  W.w1+0.01*sum(d.X1*(d.Y - 1 + 1/(1+exp(W.w0+W.w1*d.X1+W.w2*d.X2+W.w3*d.X3)))) as w1,
  W.w2+0.01*sum(d.X2*(d.Y - 1 + 1/(1+exp(W.w0+W.w1*d.X1+W.w2*d.X2+W.w3*d.X3)))) as w2,
  W.w3+0.01*sum(d.X3*(d.Y - 1 + 1/(1+exp(W.w0+W.w1*d.X1+W.w2*d.X2+W.w3*d.X3)))) as w3
FROM data d, W
WHERE W.k=1
```

Example: Logistic Regression

Tom Mitchell: [Machine Learning]

Data

X1	X2	X3	Y
3	9	3	0
3	5	7	1
6	2	2	[illegible]
3	6	3	[illegible]

Gradient Descent:

$$w_i \leftarrow w_i + \eta \sum_{\ell=1,N} X_i^{\ell}\left(Y^{\ell} - P(Y = 1 | X^{\ell})\right)$$

```
CREATE TABLE W (k int primary key, w0 real, w1 real, w2 real, w3 real);
INSERT INTO W VALUES (1, 0, 0, 0, 0);
```

```
SELECT
  W.w0+0.01*sum(d.Y - 1 + 1/(1+exp(W.w0+W.w1*d.X1+W.w2*d.X2+W.w3*d.X3))) as w0,
  W.w1+0.01*sum(d.X1*(d.Y - 1 + 1/(1+exp(W.w0+W.w1*d.X1+W.w2*d.X2+W.w3*d.X3)))) as w1,
  W.w2+0.01*sum(d.X2*(d.Y - 1 + 1/(1+exp(W.w0+W.w1*d.X1+W.w2*d.X2+W.w3*d.X3)))) as w2,
  W.w3+0.01*sum(d.X3*(d.Y - 1 + 1/(1+exp(W.w0+W.w1*d.X1+W.w2*d.X2+W.w3*d.X3)))) as w3
FROM data d, W
WHERE W.k=1
GROUP BY W.k, W.w0, W.w1, W.w2, W.w3;
```

Example: Logistic Regression

Tom Mitchell: Machine Learning

Data

X1	X2	X3	Y
3	9	3	0
3	5	7	1
6	2	2	0
3	6	3	0

Gradient Descent:

$$w_i \leftarrow w_i + \eta \sum_{\ell=1,N} X_i^{\ell}\left(Y^{\ell} - P(Y = 1|X^{\ell})\right)$$

```sql
CREATE TABLE W (k int primary key, w0 real, w1 real, w2 real, w3 real);
INSERT INTO W VALUES (1, 0, 0, 0, 0);
```

```sql
SELECT
  W.w0+0.01*sum(d.Y - 1 + 1/(1+exp(W.w0+W.w1*d.X1+W.w2*d.X2+W.w3*d.X3))) as w0,
  W.w1+0.01*sum(d.X1*(d.Y - 1 + 1/(1+exp(W.w0+W.w1*d.X1+W.w2*d.X2+W.w3*d.X3)))) as w1,
  W.w2+0.01*sum(d.X2*(d.Y - 1 + 1/(1+exp(W.w0+W.w1*d.X1+W.w2*d.X2+W.w3*d.X3)))) as w2,
  W.w3+0.01*sum(d.X3*(d.Y - 1 + 1/(1+exp(W.w0+W.w1*d.X1+W.w2*d.X2+W.w3*d.X3)))) as w3
FROM data d, W
WHERE W.k=1
GROUP BY W.k, W.w0, W.w1, W.w2, W.w3;
```

Update W, then repeat this
e.g. using python

Discussion

SQL in Data Science:

- Used primarily to prepare the data
  - ETL – Extract/Transform/Load
  - Join tables, process columns, filter rows
- Can also be used in training
  - Much less convenient than ML packages
  - But can be the best option if data is huge

SQL – Summary

- Very complex: >1000 pages,
  - No vendor supports full standard; (in practice, people use postgres as *de facto* standard)
  - Much more than DML
- It is a *declarative* language:
  - we say what we want
  - we don't say how to get it
- Relational algebra says how to get it

Relational Algebra

- Queries specified in an operational manner
  - A query gives a step-by-step procedure

- Relational operators
  - Take one or two relation instances as input
  - Return one relation instance as result
  - Easy to compose into relational algebra expressions

Five Basic Relational Operators

- Selection: $\sigma_{\text{condition}}(S)$
  - Condition is Boolean combination ($\wedge,\vee$) of atomic predicates (<, <=, =, ≠, >=, >)
- Projection: $\pi_{\text{list-of-attributes}}(S)$
- Union ($\cup$)
- Set difference (–),
- Cross-product/cartesian product ($\times$), Join: $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$

Other operators: anti-semijoin, renaming

Extended Operators of Relational Algebra

- Duplicate elimination (δ)
  - Since commercial DBMSs operate on multisets not sets
- Group-by/aggregate (ɣ)
  - Min, max, sum, average, count
  - Partitions tuples of a relation into "groups"
  - Aggregates can then be applied to groups
- Sort operator (τ)

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Logical Query Plans

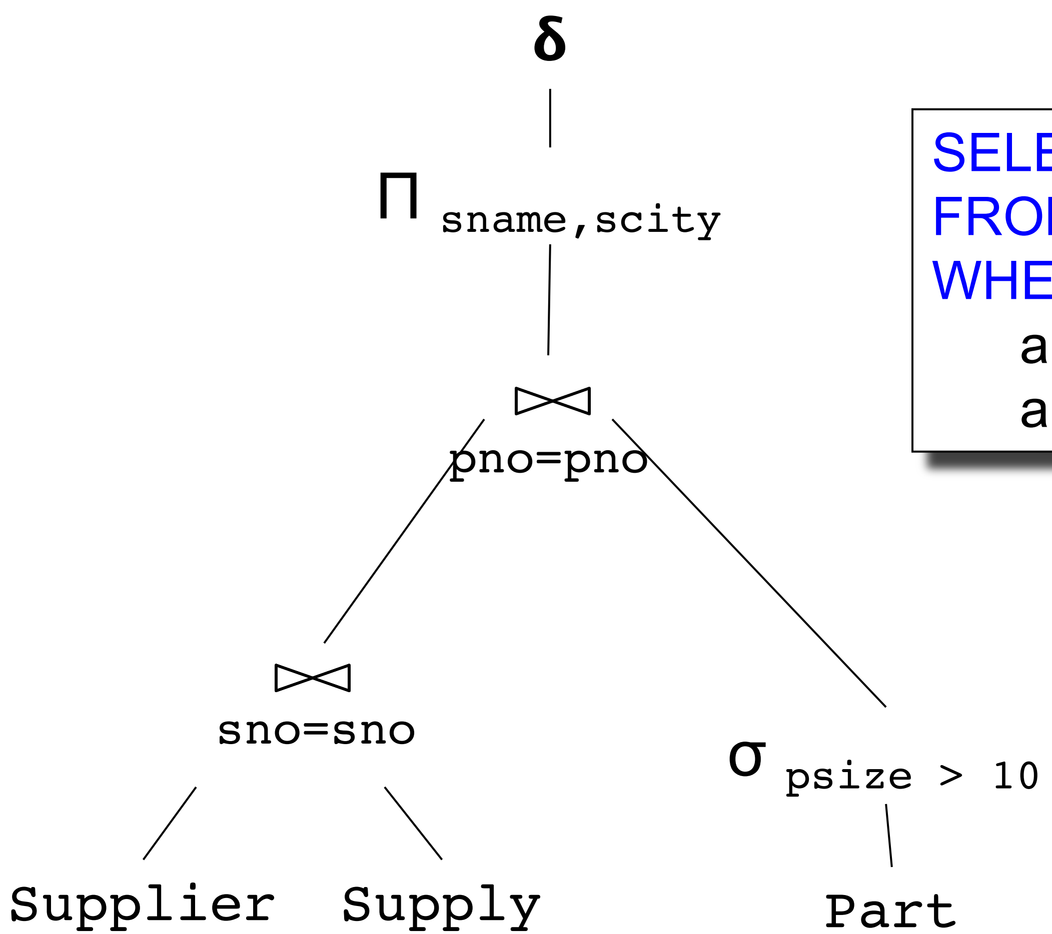
```
SELECT DISTINCT x.sname, x.scity
FROM Supplier x, Supply y, Part z
WHERE x.sno=y.sno
    and  y.pno=z.pno
    and z.psize > 10;
```

Supplier(sno,sname,scity,sstate)

Supply(sno,pno,qty,price)

Part(pno,pname,psize,pcolor)

Logical Query Plans




```sql
SELECT DISTINCT x.sname, x.scity
FROM Supplier x, Supply y, Part z
WHERE x.sno=y.sno
      and  y.pno=z.pno
      and z.psize > 10;
```

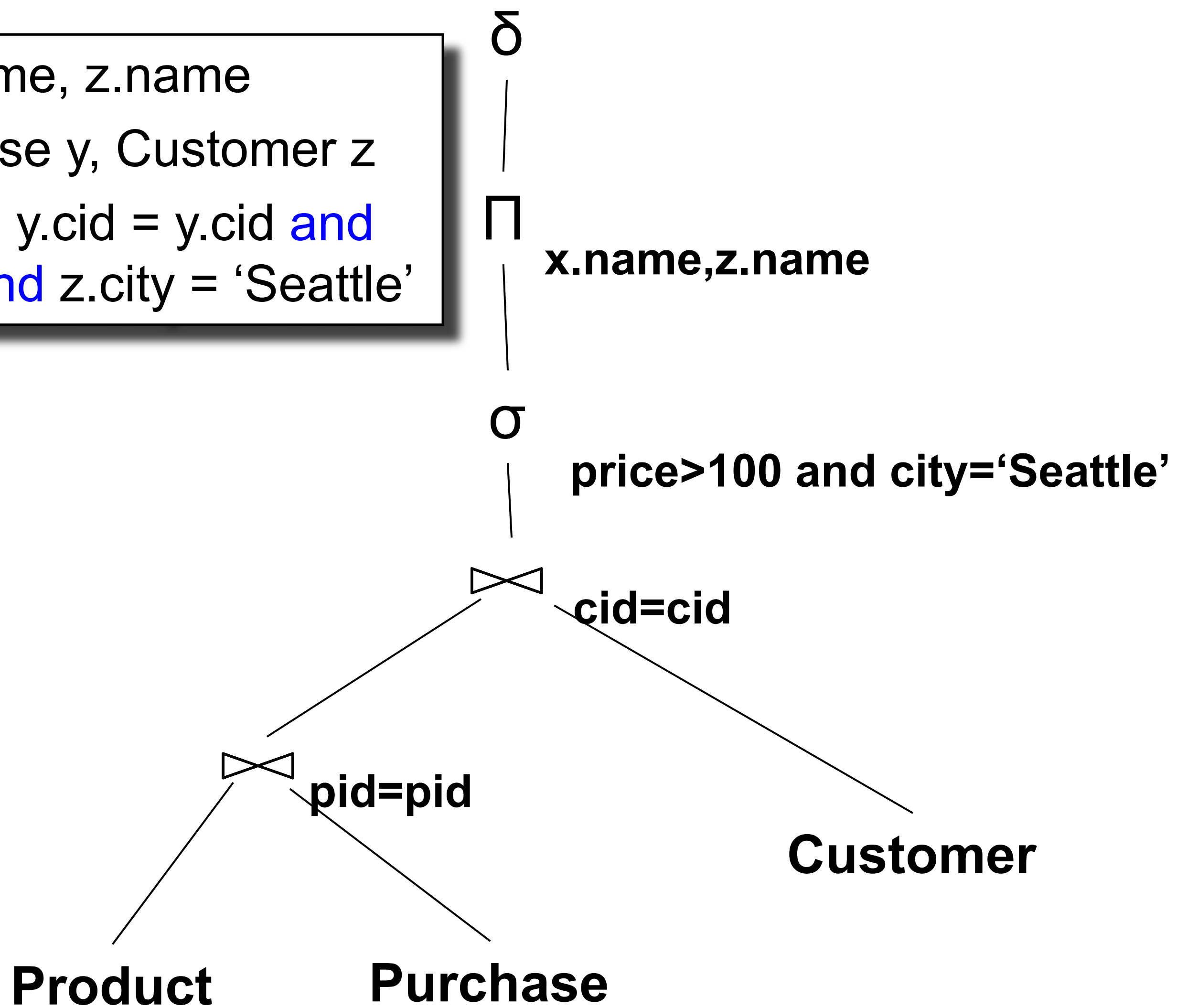
Query Optimizer

- Rewrite one relational algebra expression to a better one
- Very brief review now, more details next lecture

Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

Optimization

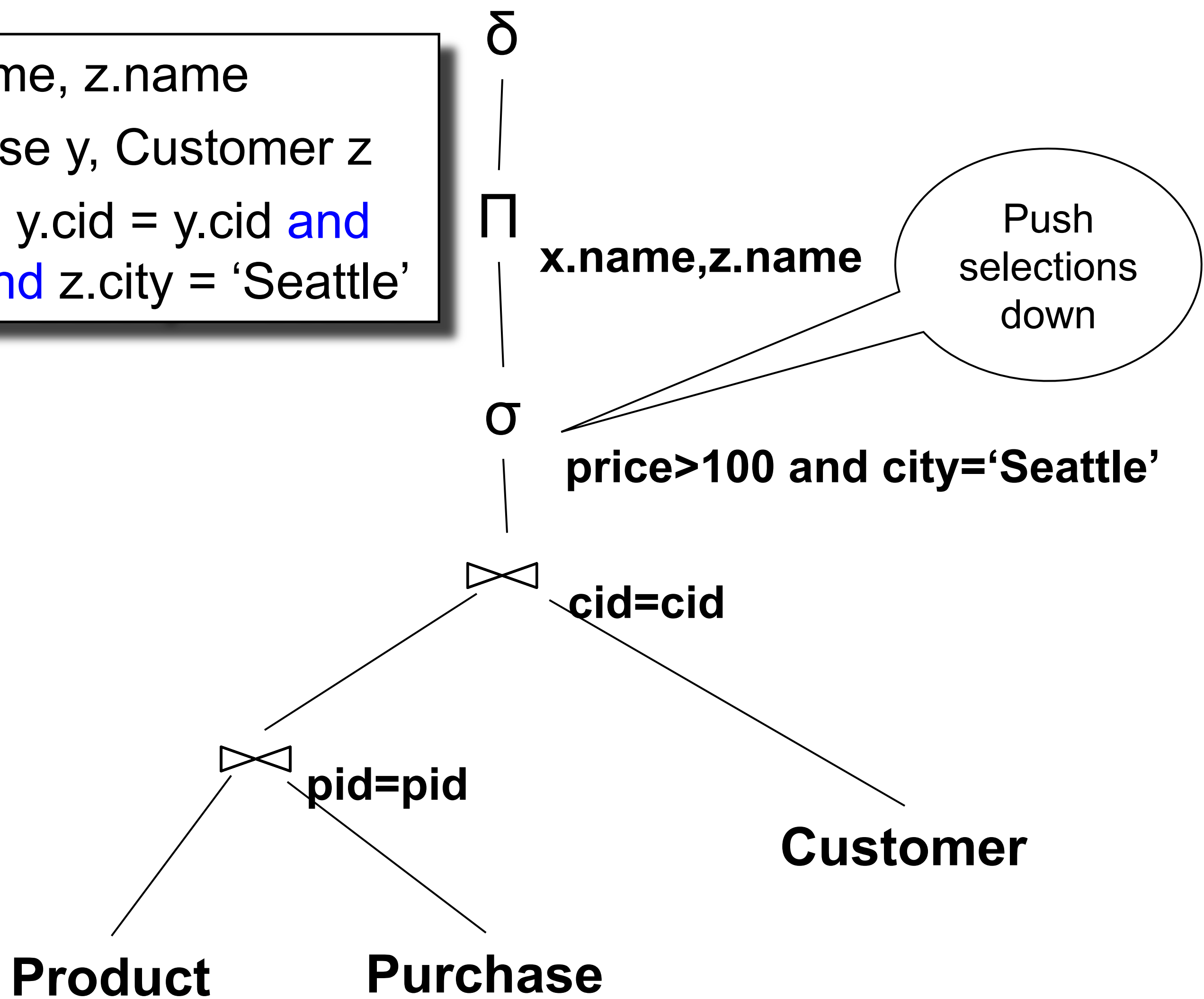
```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = y.cid and
          x.price > 100 and z.city = 'Seattle'
```



Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

Optimization

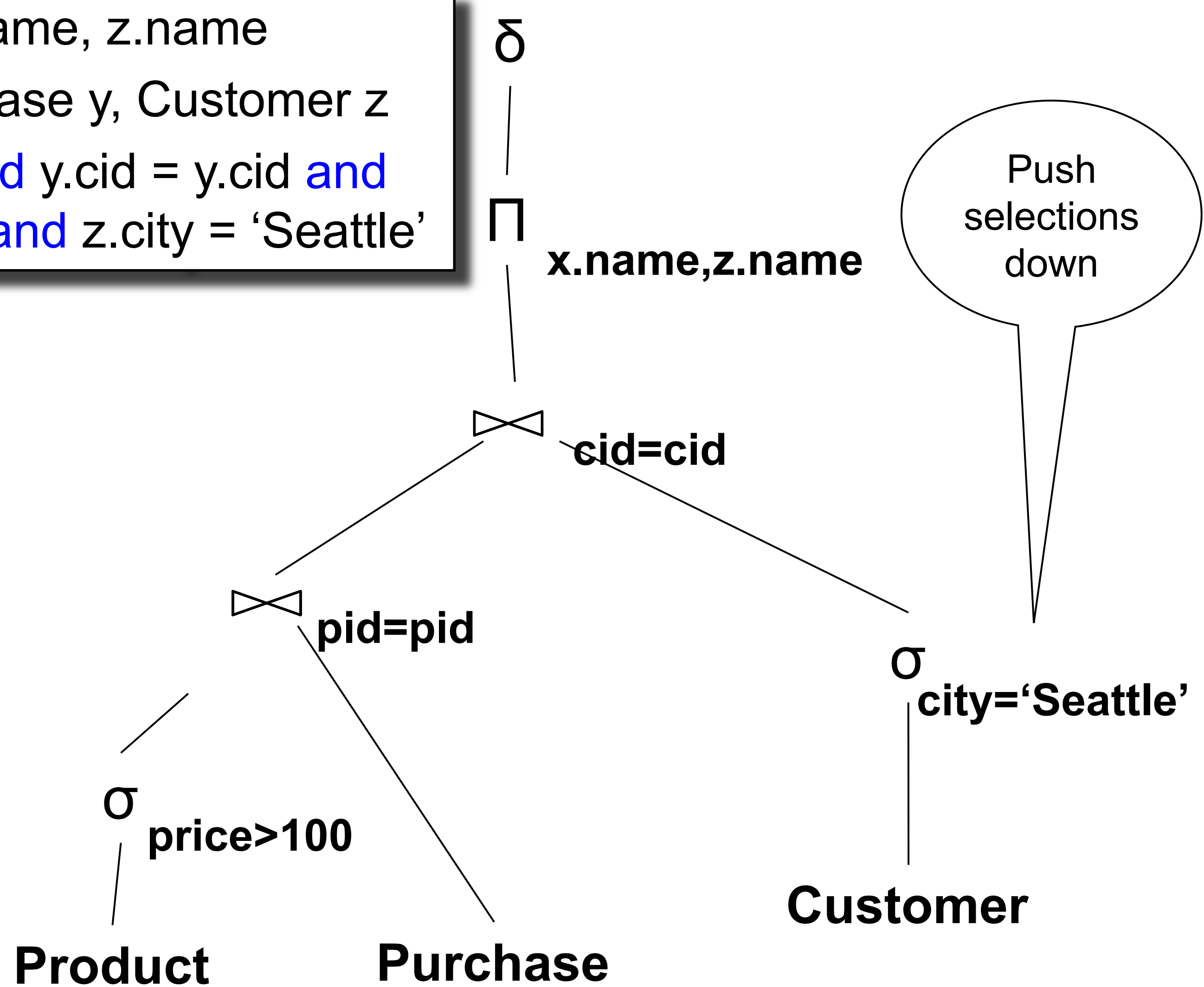
```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = y.cid and
        x.price > 100 and z.city = 'Seattle'
```



Product(<u>pid</u>, name, price)
Purchase(<u>pid, cid</u>, store)
Customer(<u>cid</u>, name, city)

Optimization

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = y.cid and
          x.price > 100 and z.city = 'Seattle'
```



More about this next lecture

Relational Model -- Summary

- Schema v.s. Data

Relational Model -- Summary

- Schema v.s. Data
- Data is *normalized* (what is that?)

Relational Model -- Summary

- Schema v.s. Data
- Data is ***normalized*** (what is that?)
  - 1st NF: relations are flat (also unordered)
  - BCNF (or 3rd or 4th NF…): split large table into many small (why?), need to join back

Relational Model -- Summary

- Schema v.s. Data
- Data is *normalized* (what is that?)
  - 1st NF: relations are flat (also unordered)
  - BCNF (or 3rd or 4th NF…): split large table into many small (why?), need to join back
  - (Consequence: joins are *really* important)

Relational Model -- Summary

- Schema v.s. Data
- Data is *normalized* (what is that?)
  - 1st NF: relations are flat (also unordered)
  - BCNF (or 3rd or 4th NF…): split large table into many small (why?), need to join back
  - (Consequence: joins are *really* important)
- Query language is SQL, or something equivalent, like relational algebra

Benefits of Relational Model

- Physical data independence
  - Can change how data is organized on disk without affecting applications
- Logical data independence
  - Can change the logical schema without affecting applications (not 100%... consider updates)

Physical Data Independence

Supplier

sno	sname	scity	sstate
1	s1	city 1	WA
2	s2	city 1	WA
3	s3	city 2	MA
4	s4	city 2	MA

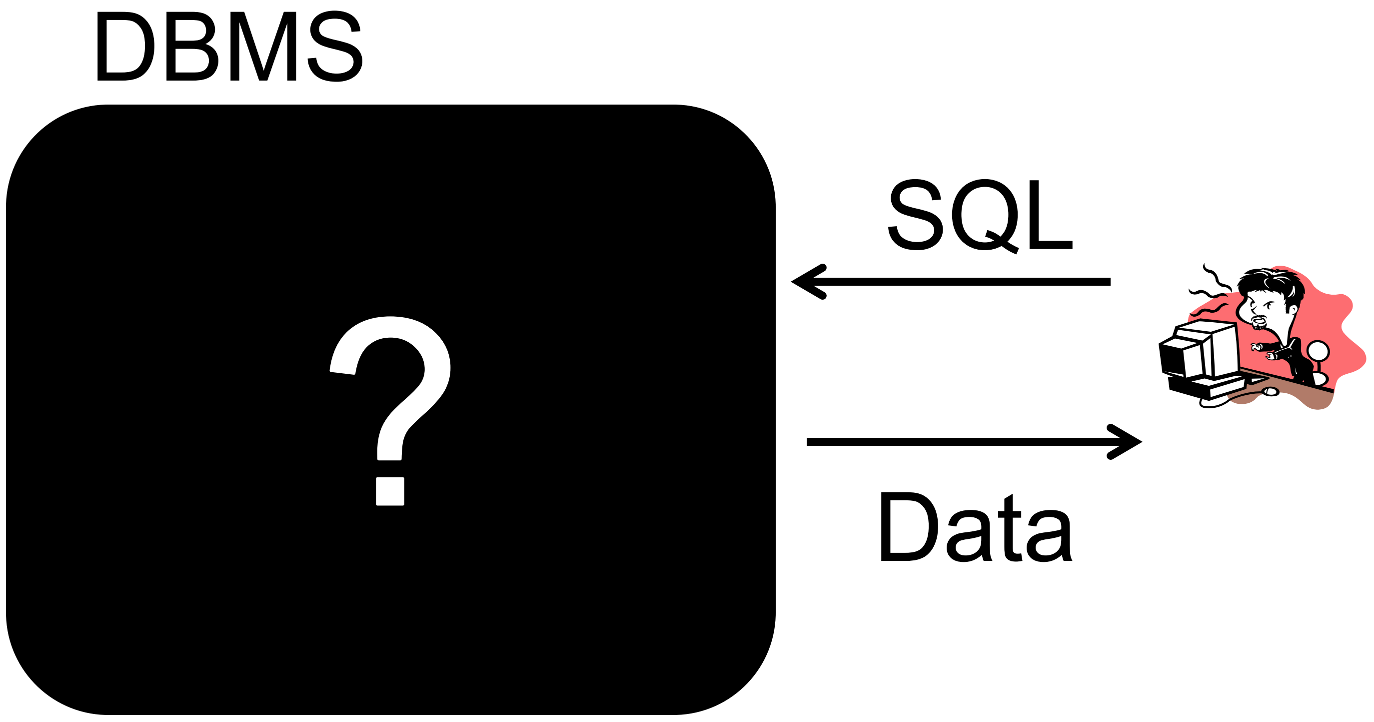
```
SELECT DISTINCT sname
FROM Supplier
WHERE scity = 'Seattle'
```

How is the data stored on disk?
(e.g. row-wise, column-wise)

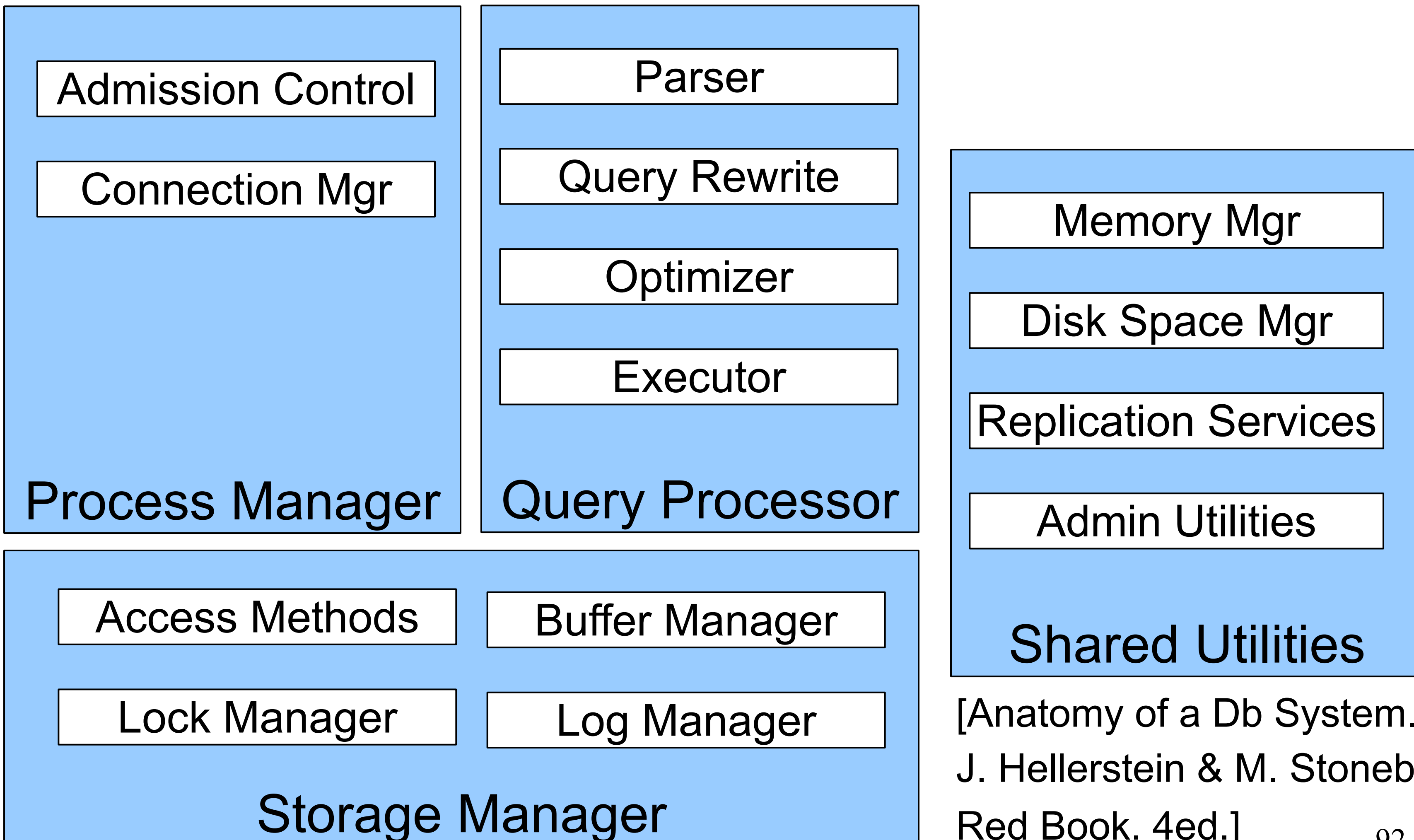
The SQL query works the same, regardless of the answers to these questions

Is there an index on scity?
(e.g. no index, unclustered index, clustered index)

How to Implement a Relational DBMS?



DBMS Architecture



[Anatomy of a Db System.
J. Hellerstein & M. Stonebraker.
Red Book. 4ed.]

Storage Manager

Disks

- Data resides persistently on *<u>disks</u>*
  - Your local disk, or a Network Attached Storage (NAS), or Amazon’s S3
- For processing, data must reside in *<u>main memory</u>*

Disks v.s. Main Memory

Disk

- Unit of data =1 block
  4KB or 8KB or 16KB
- Access time* = seek time + rotational latency + transfer rate
  ≈ 12ms + 5ms + 150MB/s
  - Random access ≈ 17 ms
  - Sequential access ≈ 50μs
- Organization:
  - Heap file
  - Index file

Main memory

- Unit of data =1byte
- Access time** = 50ns
- Organization:
  - Lists, arrays, hash tables, …

* https://en.wikipedia.org/wiki/Hard_disk_drive_performance_characteristics
** https://en.wikipedia.org/wiki/Dynamic_random-access_memory#Memory_timing

Disks v.s. Main Memory

Disk

- Unit of data =1 block
  4KB or 8KB or 16KB
- Access time* = seek time + rotational latency + transfer rate
  ≈ 12ms + 5ms + 150MB/s
  - Random access ≈ 17 ms
  - Sequential access ≈ 50μs
- Organization:
  - Heap file
  - Index file

Orders of magnitude slower

Main memory

- Unit of data =1byte
- Access time** = 50ns
- Organization:
  - Lists, arrays, hash tables, …

* https://en.wikipedia.org/wiki/Hard_disk_drive_performance_characteristics
** https://en.wikipedia.org/wiki/Dynamic_random-access_memory#Memory_timing

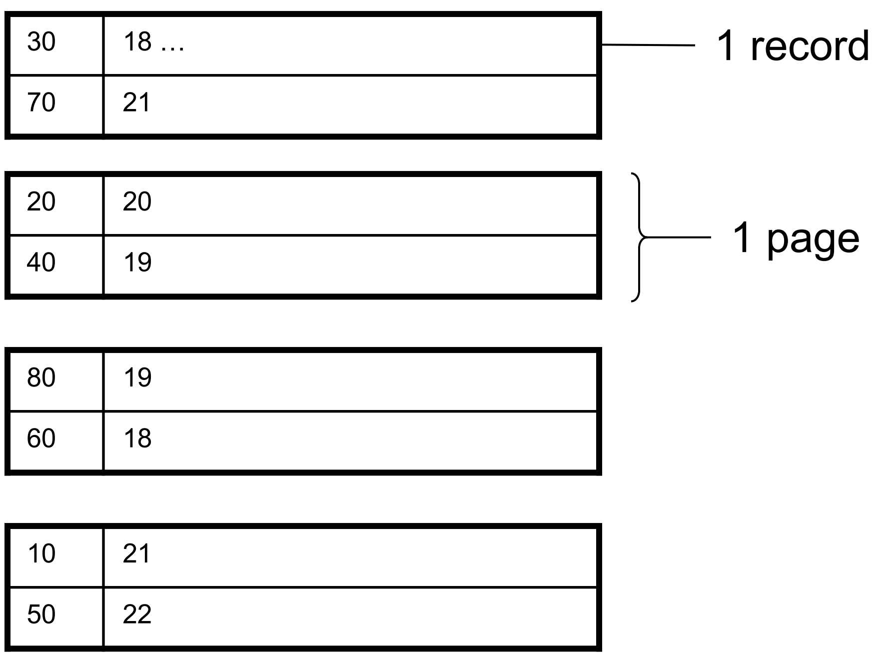
Heap File

Data on disk is stored in *files*

Files consist of *pages* filled with *records*

A *heap file* is not sorted on any attribute

`Student(sid: int, age: int, …)`

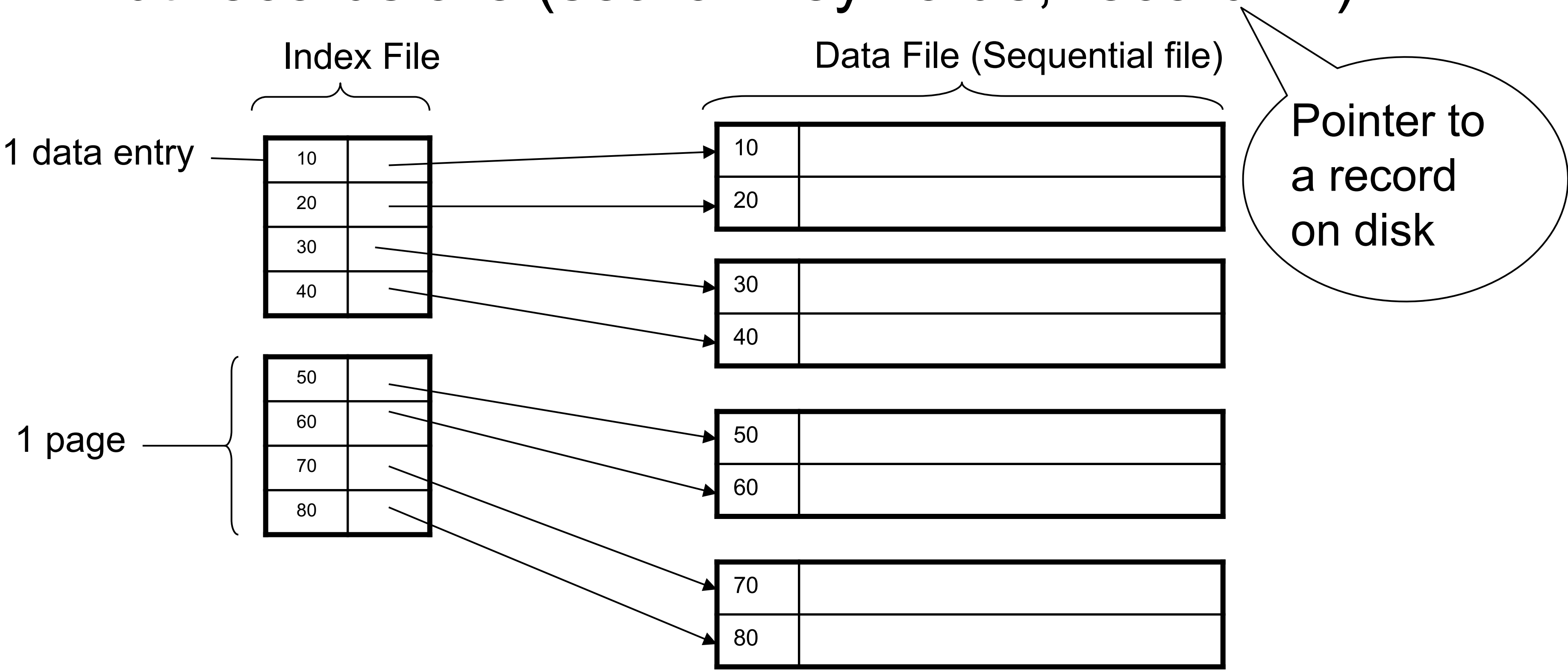


Index

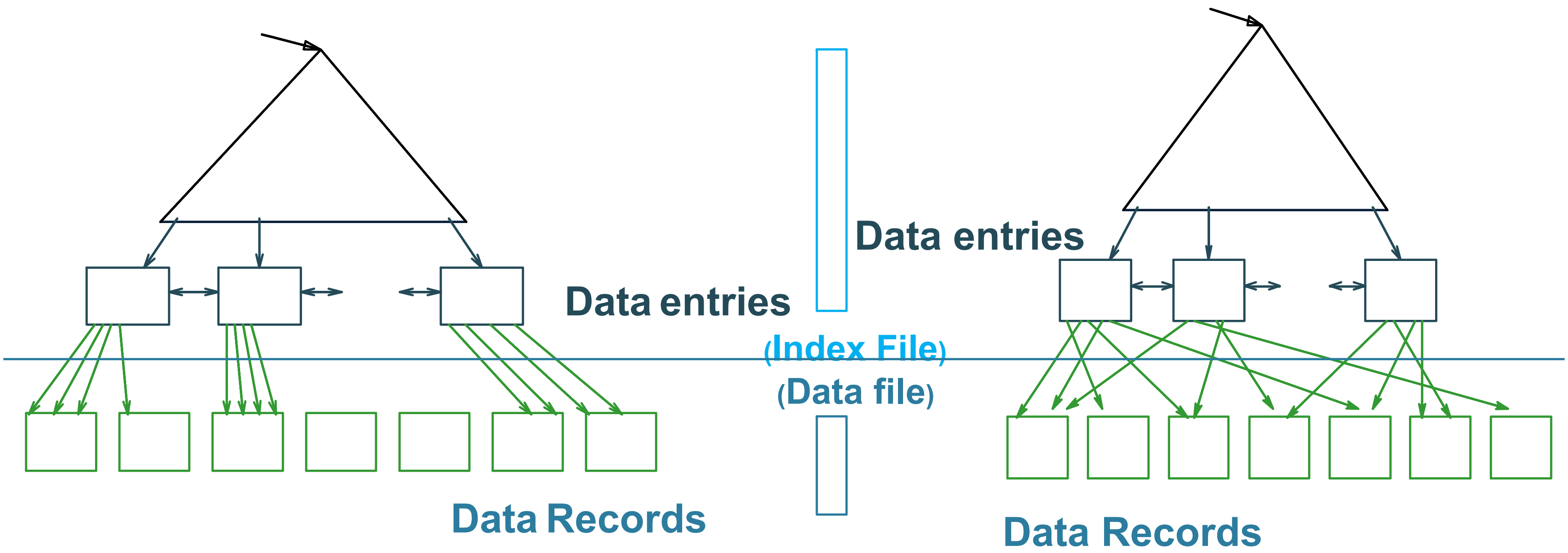
An index is a data structure stored on disk

It is stored in a file consisting of pages & records

But records are (search key value, record ID)



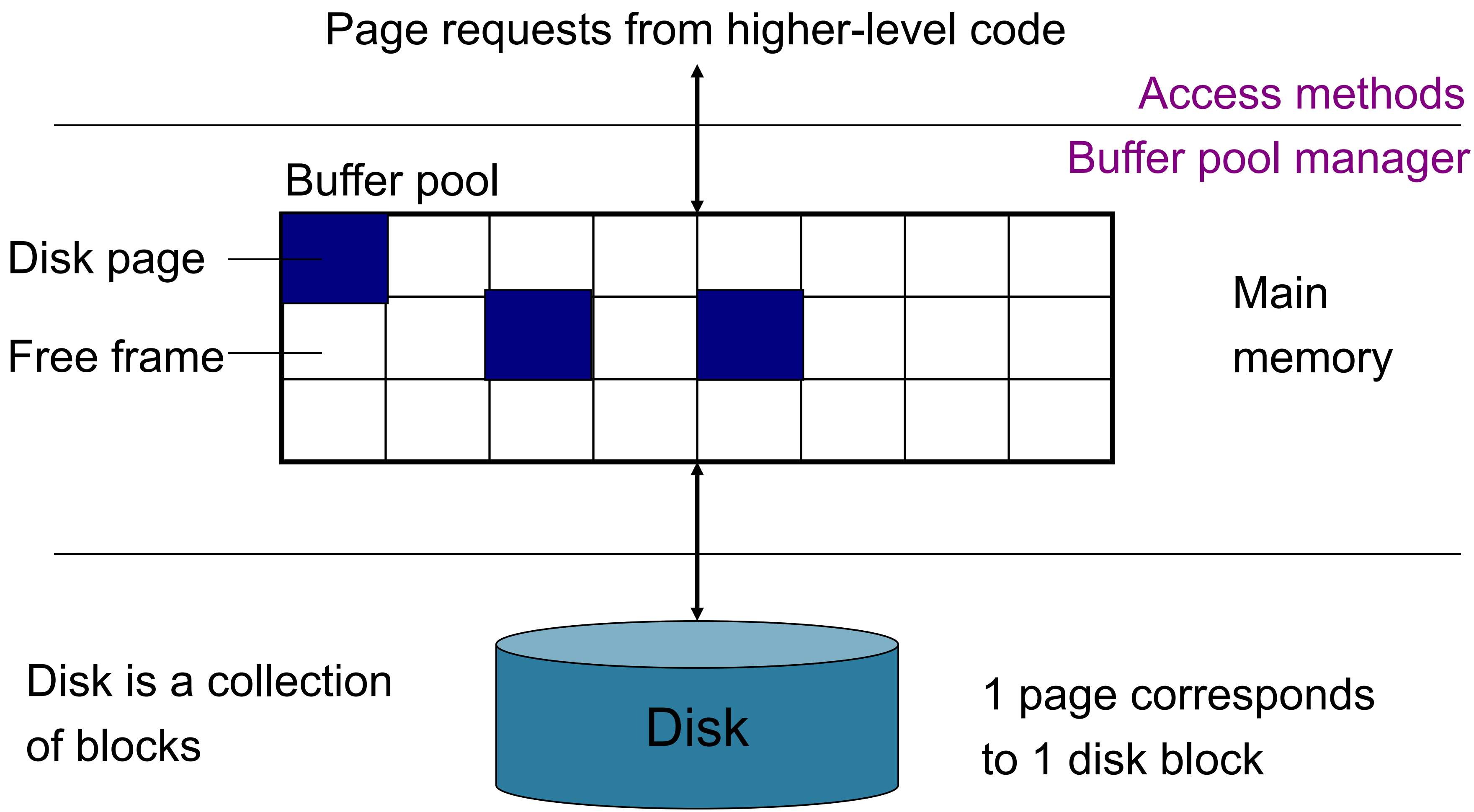
Clustered vs. Unclustered Index



CLUSTERED **UNCLUSTERED**

Clustered = records close in index are close in data

Buffer Manager



Discussion

- Disks are necessary both for persistent storage, *and* in order to process data larger than main memory
- They are slow! Buffer pool mitigates this
- "*Cold* v.s. *Hot* execution": first time you execute the query it is slow; if you repeat it, then it is faster (WHY?)

Discussion

- Main idea for Distributed data processing: if we distribute the data to many servers, then the data will fit in main memory
- For that reason, they often do not implement "*out of core*" algorithm
- Our focus in this course is on distributed data processing, not on out-of-core.

Summary

- RDMBS are complex systems
- Need to know some of their basics inner workings in order to understand query performance

Next week: we start review query processing, optimization, and start discussion distributed query processing