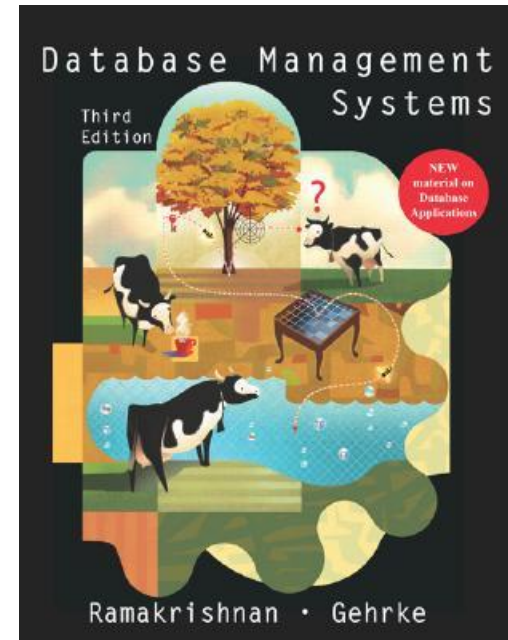


Using a DBMS

Shan-Hung Wu & DataLab
CS, NTHU

Assigned Reading

- [Java concurrency](#)
- "Database Management Systems," 3ed, by Ramakrishnan



Coverage

- Java concurrency
- Chaps 2 and 3 on how to store your data into a DBMS
 - ER model and relational model
- Chaps 4 and 5 on queries
 - SQL language (DDL and DML)
 - Relational algebra
- Chap 19* on how to store your data *well*
 - Easy maintenance
 - Answering most queries fast

Coverage

- Java concurrency
- Chaps 2 and 3 on how to store your data into a DBMS
 - ER model and relational model
- Chaps 4 and 5 on queries
 - SQL language (DDL and DML)
 - Relational algebra
- Chap 19* on how to store your data *well*
 - Easy maintenance
 - Answering most queries fast

Starting a New Thread

```
public class HelloRunnable implements Runnable {  
    @Override  
    public void run() {  
        System.out.println("Hello from a thread!");  
    }  
  
    public static void main(String args[]) {  
        (new Thread(new HelloRunnable())).start();  
    }  
}
```

or

```
public class HelloThread extends Thread {  
    @Override  
    public void run() {  
        System.out.println("Hello from a thread!");  
    }  
  
    public static void main(String args[]) {  
        (new HelloThread()).start();  
    }  
}
```

What Happened?

```
public class HelloRunnable implements Runnable {  
    @Override  
    public void run() {  
        System.out.println("Hello from a thread!");  
    }  
  
    public static void main(String args[]) {  
        (new Thread(new HelloRunnable())).start();  
    }  
}
```

- A new stack is allocated for `run()`, in addition to that of `main()`
- Your CPU spends time on executing `run()` ***in parallel with*** `main()`

Multiple Stacks, Single Heap

- The heap in memory scheme?
 - Stores objects
 - *Shared by all threads*
- Can two threads access the same object? Yes
- How? Passing the same object to their constructors

```
public static void main(String args[]) {  
    Counter counter = ...;  
    (new Thread(new HelloRunnableA(counter))).start(); // thread A  
    (new Thread(new HelloRunnableB(counter))).start(); // thread B  
}
```

Concurrent Access

- Given the same object `counter`
- Suppose both threads execute in `run()`:

```
...  
int c = counter.get();  
c++; // c--;  
counter.set(c);
```

```
class Counter {  
    private int c = 0;  
    public void set(int c) {  
        this.c = c;  
    }  
    public int get () {  
        return c;  
    }  
}
```

- Thread A's result will be lost if
 1. Thread A: Get c
 2. Thread B: Get c
 3. Thread A: Increment retrieved value; result is 1
 4. Thread B: Decrement retrieved value; result is -1
 5. Thread A: Set result in c; c is now 1.
 6. Thread B: Set result in c; c is now -1.

Synchronization

```
public class SynchronizedCounter {  
    private int c = 0;  
    public synchronized void set(int c) {  
        this.c = c;  
    }  
    public synchronized int get() {  
        return c;  
    }  
}
```

- Same as...

- Only one thread can enter sync. block of an obj. at a time
- Problem solved?

```
public class SynchronizedCounter {  
    private int c = 0;  
    public void set(int c) {  
        synchronized(this) { this.c = c; }  
    }  
    public int get() {  
        synchronized(this) { return c; }  
    }  
}
```

Still Wrong!

- Two threads in `run()`:

```
... // counter is a SynchronizedCounter instance
int c = counter.get();
c++; // c--;
counter.set(c);
```

- Thread A's result will still be lost if
 1. Thread A: Get c
 2. Thread B: Get c
 3. Thread A: Increment retrieved value; result is 1
 4. Thread B: Decrement retrieved value; result is -1
 5. Thread A: Set result in c; c is now 1.
 6. Thread B: Set result in c; c is now -1.

Synchronization at Right Place

- Solution1: callers lock `counter` during the entire increment/decrement period:

```
synchronized(counter) {  
    int c = counter.get();  
    c++; // or c--;  
    counter.set(c);  
}
```

- Solution2: callee provides atomic methods

```
public class SynchronizedCounter {  
    private int c = 0;  
    public void synchronized increment() {  
        c++;  
    }  
    public int get() {  
        return c;  
    }  
}
```

Blocking and Waiting States

- Threads are **blocked** outside a critical section if someone is in
- Thread *A* in a critical section of `o` can stop and enter the **waiting** state by calling `o.wait()`
 - Gives up the lock, so some other blocking thread *B* can enter the critical section
 - If *B* calls `o.notifyAll()`, *A* competes for the lock again and resume

Wrap `wait()` in a Loop

- It's a good practice to wrap `wait()` in a loop to prevent bugs
- Queue length: 10

Threads A, B:

```
// enqueue
synchronized(queue) {
    while(queue.size() == 10) {
        queue.wait();
    }
    queue.add(...);
    queue.notifyAll();
}
```

Threads C, D:

```
// dequeue
synchronized(queue) {
    while(queue.size() == 0) {
        queue.wait();
    }
    ... = queue.remove();
    queue.notifyAll();
}
```

Coverage

- Java concurrency
- Chaps 2 and 3 on how to store your data into a DBMS
 - ER model and relational model
- Chaps 4 and 5 on queries
 - SQL language (DDL and DML)
 - Relational algebra
- Chap 19* on how to store your data *well*
 - Easy maintenance
 - Answering most queries fast

DBMS \neq Database

- A database is a collection of your data stored in a computer
- A DBMS (DataBase Management System) is a software that manages databases

Storing Data

- Let's say, you have data in memory to store
- How does data in memory (heap) look like?
 - Objects
 - References to objects
- Objects formatted by classes you defined
- Could we store these objects and references directly?

Data Model

- Definition: A ***data model*** is a framework for describing the structure of databases in a DBMS
- Common data models: ***ER model*** and ***relational model***
- A DBMS supporting the relational model is called the relational DBMS

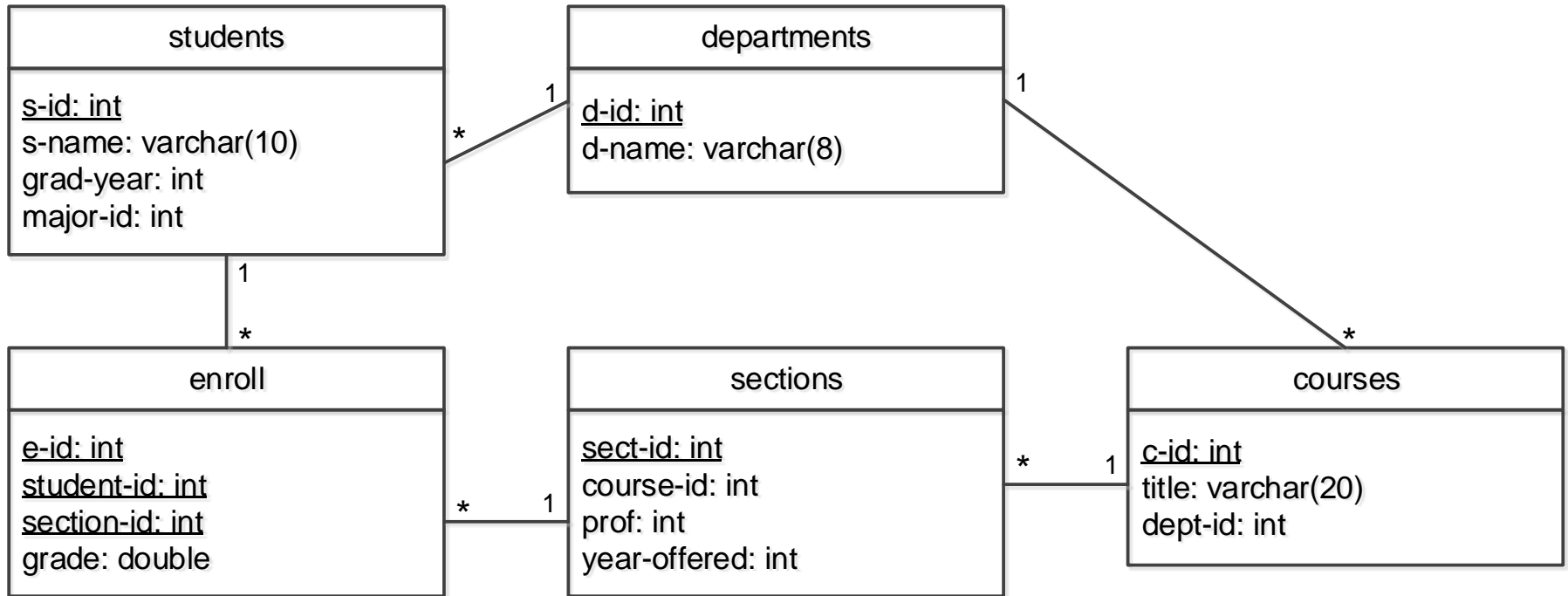
Why ER Model?

- Allows thinking your data in OOP way
- **Entity**
 - An object (or instance of a class)
 - With attributes
- **Entity group**
 - A class
 - Must define the ID attribute for each entity
- **Relationship** between entities
 - References (“has-a” relationship)
 - Could be 1-1, 1-many, and many-many

Why Relational Model?

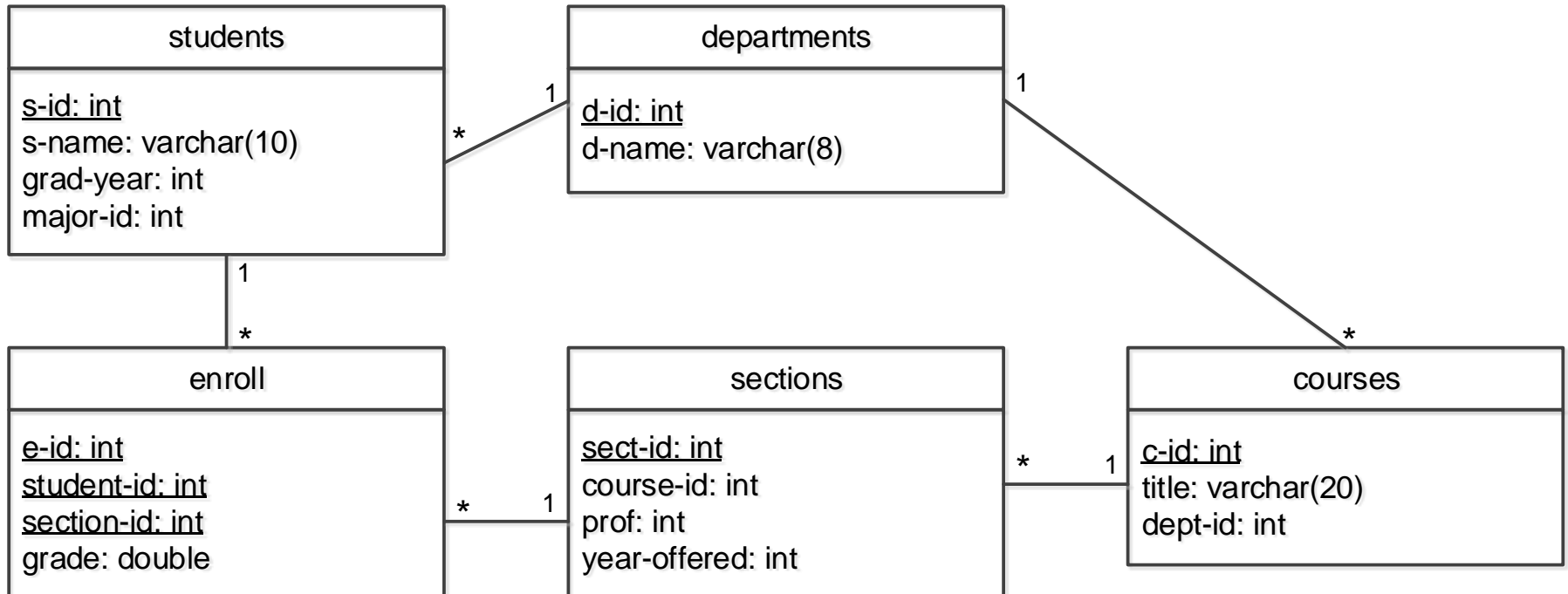
- To realize an ER model using a collocation of tables/*relations*
 - Simplifies data management and query processing
- Still logic (not how your data stored physically)

Example: Student DB



- Relation (table)
 - Realization of 1) an entity group via table; or 2) a relationship
 - **Fields/attributes** as columns
 - **Records/tuples** as rows

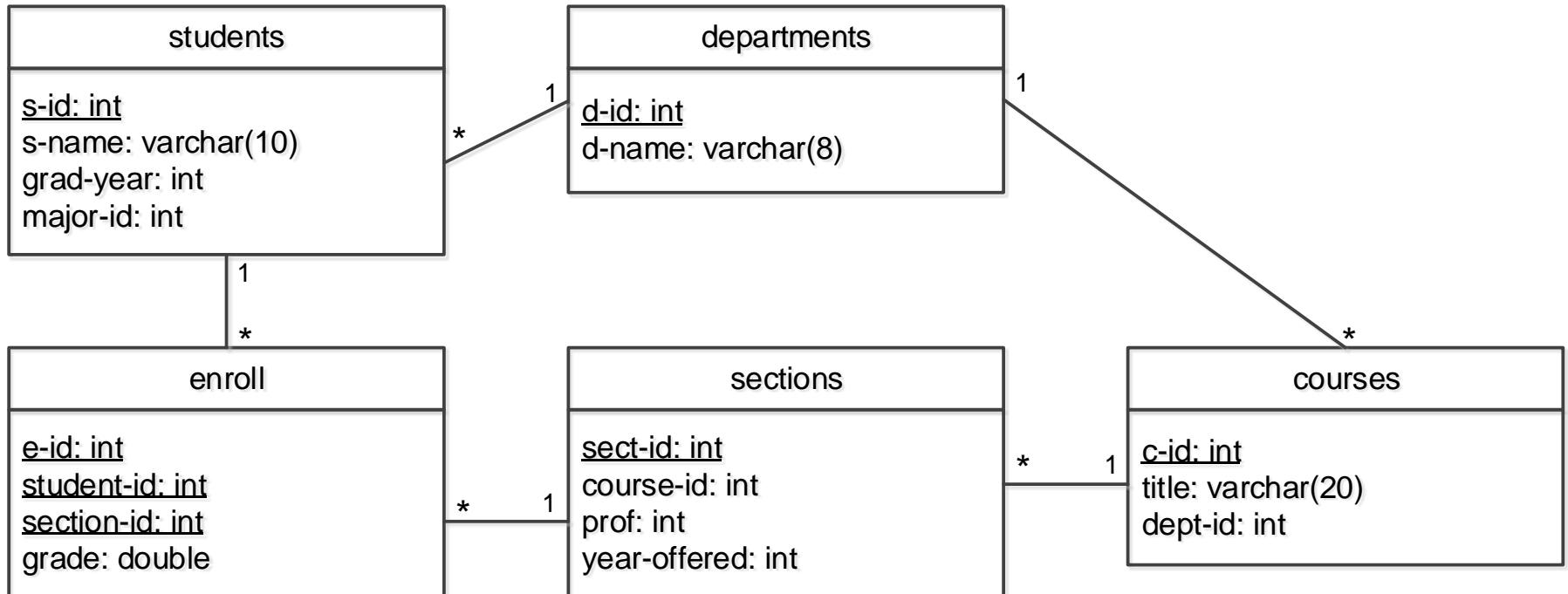
Example: A student DB



- **Primary Key**

- Realization of ID via a group of fields

Example: A student DB



- **Foreign key**
 - Realization of relationship
 - A record can point to the primary key of the other record
 - Only 1-1 and 1-many
 - Intermediate relation is needed for many-many

Schema

- Definition: A *schema* is the structure of a particular database
- The schema of a relation/table is its fields and field types

Coverage

- Java concurrency
- Chaps 2 and 3 on how to store your data into a DBMS
 - ER model and relational model
- Chaps 4 and 5 on queries
 - SQL language (DDL and DML)
 - Relational algebra
- Chap 19 on how to store your data *well*
 - Easy maintenance
 - Answering most queries fast

Queries

- Data Definition Language (DDL) on schema
 - CREATE TABLE ...
 - ALTER TABLE ...
 - DROP TABLE ...
- Data Manipulation Language (DML) on records
 - INSERT INTO ... VALUES ...
 - SELECT ... FROM ... WHERE ...
 - UPDATE ... SET ... WHERE ...
 - DELETE FROM ... WHERE ...

Data Model and Queries (1/3)

Q1: among a set of blog pages, find those pages written by Steven Sinofsky after 2011

Step1: structure your data by following the *relational data model*

- Identify *records* (e.g., web pages, authors, etc.) with the same *fields* in your data and place them into respective *tables*

blog_pages

| blog_id | url | created | author_id |
|---------|----------------|------------|-----------|
| 33981 | ms.com/... | 2012/10/31 | 729 |
| 33982 | apache.org/... | 2012/11/15 | 4412 |

← record

field



users

| user_id | name | balance |
|---------|-----------------|---------|
| 729 | Steven Sinofsky | 10,235 |
| 730 | Picachu | NULL |

Data Model and Queries (2/3)

Q1: among a set of blog pages, find those pages written by Steven Sinofsky after 2011

```
CREATE TABLE blog_pages (  
    blog_id INT NOT NULL AUTO_INCREMENT,  
    url VARCHAR(60),  
    created DATETIME,  
    author_id INT);
```

```
INSERT INTO blog_pages (url, created, author_id)  
VALUES ('ms.com/...', 2012/09/18, 729);
```

blog_pages

| blog_id | url | created | author_id |
|---------|----------------|------------|-----------|
| 33981 | ms.com/... | 2012/10/31 | 729 |
| 33982 | apache.org/... | 2012/11/15 | 4412 |

Data Model and Queries (3/3)

Q1: among a set of blog pages, find those pages written by Steven Sinofsky after 2011

Step2: issue queries

```
SELECT b.blog_id
FROM blog_pages b, users u
WHERE b.author_id=u.user_id
      AND u.name='Steven Sinofsky'
      AND b.created >= 2011/1/1;
```

How Is a Query Answered?

```
SELECT b.blog_id
```

```
FROM blog_pages b, users u
```

```
WHERE b.author_id=u.user_id  
      AND u.name='Steven Sinofsky'  
      AND b.created >= 2011/1/1;
```

product(b, u)

| blog_id | url | created | author_id | user_id | name | balance |
|---------|-----|------------|-----------|---------|-----------------|---------|
| 33981 | ... | 2009/10/31 | 729 | 729 | Steven Sinofsky | 10,235 |
| 33981 | ... | 2009/10/31 | 729 | 730 | Picachu | NULL |
| 33982 | ... | 2012/11/15 | 4412 | 729 | Steven Sinofsky | 10,235 |
| 33982 | ... | 2012/11/15 | 4412 | 730 | Picachu | NULL |
| 41770 | ... | 2012/10/20 | 729 | 729 | Steven Sinofsky | 10,235 |
| 41770 | ... | 2012/10/20 | 729 | 730 | Picachu | NULL |

b

| blog_id | url | created | author_id |
|---------|-----|------------|-----------|
| 33981 | ... | 2009/10/31 | 729 |
| 33982 | ... | 2012/11/15 | 4412 |
| 41770 | ... | 2012/10/20 | 729 |

u

| user_id | name | balance |
|---------|-----------------|---------|
| 729 | Steven Sinofsky | 10,235 |
| 730 | Picachu | NULL |

How Is a Query Answered?

```
SELECT b.blog_id
```

```
FROM blog_pages b, users u
```

```
WHERE b.author_id=u.user_id  
      AND u.name='Steven Sinofsky'  
      AND b.created >= 2011/1/1;
```

select(p, where...)

| blog_id | url | created | author_id | user_id | name | balance |
|---------|-----|------------|-----------|---------|-----------------|---------|
| 41770 | ... | 2012/10/20 | 729 | 729 | Steven Sinofsky | 10,235 |



p = product(b, u)

| blog_id | url | created | author_id | user_id | name | balance |
|---------|-----|------------|-----------|---------|-----------------|---------|
| 33981 | ... | 2009/10/31 | 729 | 729 | Steven Sinofsky | 10,235 |
| 33981 | ... | 2009/10/31 | 729 | 730 | Picachu | NULL |
| 33982 | ... | 2012/11/15 | 4412 | 729 | Steven Sinofsky | 10,235 |
| 33982 | ... | 2012/11/15 | 4412 | 730 | Picachu | NULL |
| 41770 | ... | 2012/10/20 | 729 | 729 | Steven Sinofsky | 10,235 |
| 41770 | ... | 2012/10/20 | 729 | 730 | Picachu | NULL |

How Is a Query Answered?

```
SELECT b.blog_id
```

```
FROM blog_pages b, users u
```

```
WHERE b.author_id=u.user_id  
      AND u.name='Steven Sinofsky'  
      AND b.created >= 2011/1/1;
```

project(s, select...)

| blog_id |
|---------|
| 41770 |

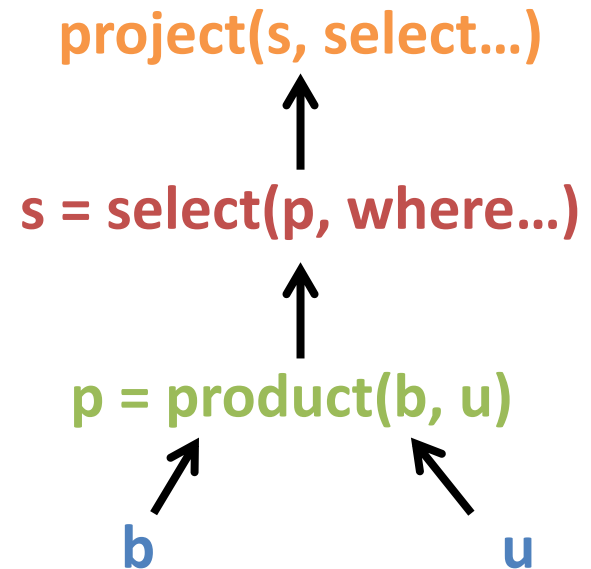


s = select(p, where...)

| blog_id | url | created | author_id | user_id | name | balance |
|---------|-----|------------|-----------|---------|-----------------|---------|
| 41770 | ... | 2012/10/20 | 729 | 729 | Steven Sinofsky | 10,235 |

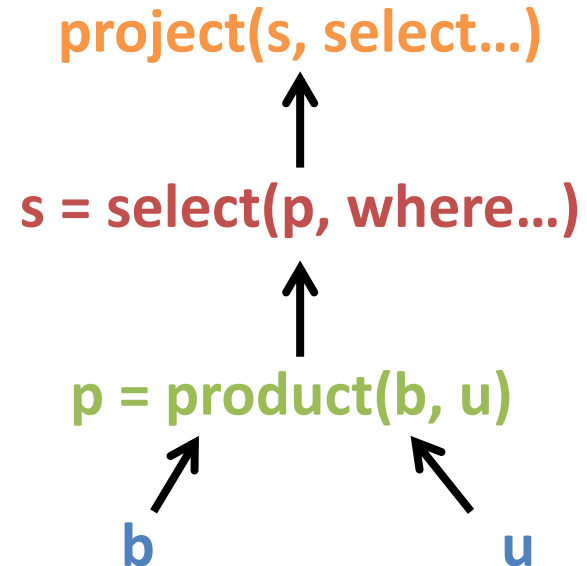
Query Algebra

- Operators
 - Product, select, project, join, group-by, etc.
- Operands
 - Tables, output of other operators, predicates, etc.



Query Plan

- A tree that answers a query



- ***Not unique!***
- A DBMS automatically seeks for the best query plan

Coverage

- Java concurrency
- Chaps 2 and 3 on how to store your data into a DBMS
 - ER model and relational model
- Chaps 4 and 5 on queries
 - SQL language (DDL and DML)
 - Relational algebra
- Chap 19* on how to store your data ***well***
 - Easy maintenance
 - Answering most queries fast

How Good are Your Data?

- Let's say, if you want to track the topics of a blog page
- Is this a good table?

blog_pages

| blog_id | url | created | author_id | topic | topic_admin |
|---------|----------------|------------|-----------|-------------|-------------|
| 33981 | ms.com/... | 2012/10/31 | 729 | programming | 5638 |
| 33981 | ms.com/... | 2012/10/31 | 729 | databases | 5649 |
| 33982 | apache.org/... | 2012/11/15 | 4412 | programming | 5638 |
| 33982 | apache.org/... | 2012/11/15 | 4412 | os | 7423 |

Insertion Anomaly

blog_pages

| blog_id | url | created | author_id | topic | topic_admin |
|---------|----------------|------------|-----------|-------------|-------------|
| 33981 | ms.com/... | 2012/10/31 | 729 | programming | 5638 |
| 33981 | ms.com/... | 2012/10/31 | 729 | databases | 5649 |
| 33982 | apache.org/... | 2012/11/15 | 4412 | programming | 5638 |
| 33982 | apache.org/... | 2012/11/15 | 4412 | os | 7423 |

| | | | | | |
|-------|----------------|------------|------|--|--|
| 33983 | apache.org/... | 2013/02/15 | 7412 | | |
|-------|----------------|------------|------|--|--|




- A blog cannot be inserted without knowing all fields of topics (except setting them to null)

Update Anomaly

blog_pages

| blog_id | url | created | author_id | topic | topic_admin |
|---------|----------------|------------|-----------|------------------|-------------|
| 33981 | ms.com/... | 2012/10/31 | 729 | <i>win prog.</i> | 5638 |
| 33981 | ms.com/... | 2012/10/31 | 729 | databases | 5649 |
| 33982 | apache.org/... | 2012/11/15 | 4412 | programming | 5638 |
| 33982 | apache.org/... | 2012/11/15 | 4412 | os | 7423 |




- If you forget to update all duplicated cells, you get inconsistent data

Deletion Anomaly

blog_pages

| blog_id | url | created | author_id | topic | topic_admin |
|---------|----------------|------------|-----------|--------------------|-------------|
| 33981 | ms.com/... | 2012/10/31 | 729 | <i>programming</i> | <i>5638</i> |
| 33981 | ms.com/... | 2012/10/31 | 729 | databases | 5649 |
| 33982 | apache.org/... | 2012/11/15 | 4412 | programming | 5638 |
| 33982 | apache.org/... | 2012/11/15 | 4412 | os | 7423 |



- Deleting topics force you to delete the blog fields too

Normalization

- Avoids these anomaly through schema normalization
 - 3rd normal form
 - BCNF normal form
- Idea: break your one, big table into multiple small, modular tables
 - Reuse tables
 - Avoid bias towards any particular pattern of querying