Background

vanilladb.org

Why do you need a database system?

To store data,

why not just use a file system?

Advantages of a Database System

- It answers ***queries*** fast

  Q1: among a set of blog pages, find those pages written by Ash Ketchum after 2011

  Q2: among a set of employers, increase the salary by 20% for those who have worked longer then 4 years

- Queries (from multiple users) can execute ***concurrently*** without affecting each other
- It ***recovers*** from crash
  - No corrupt data after restart

Data Model and Queries (1/3)

Q1: among a set of blog pages, find those pages written by Ash Ketchum after 2011

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

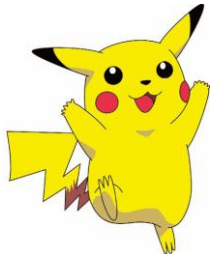
- Identify ***records*** (e.g., web pages, authors, etc.) with the same ***fields*** in your data and place them into respective ***tables***

blog_pages

blog_id	url	created	author_id
33981	pokemon.com/…	2012/10/31	729
33982	apache.org/…	2012/11/15	4412

← record

field ↓



users

user_id	name	balance
729	Ash Ketchum	10,235
730	Picachu	NULL

Data Model and Queries (2/3)

Q1: among a set of blog pages, find those pages written by Ash Ketchum 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 ('pokemon.com/...', 2012/09/18, 729);
```

blog_pages

blog_id	url	created	author_id
33981	pokemon.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 Ash Ketchum 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='Ash Ketchum'
        AND b.created >= 2011/1/1;
```

Advantages of a Database System

- It answers ***queries*** fast

  Q1: among a set of web pages, find those pages written by Ash Ketchum after 2011

  Q2: among a set of employers, increase the salary by 20% for those who have worked longer then 4 years

- Queries (from multiple users) can execute ***concurrently*** without affecting each other
- It ***recovers*** from crash
  - No corrupt data after restart

Transactions (1/3)

- Each query, by default, is placed in a ***transaction*** (***tx*** for short) automatically

```
BEGIN TRANSACTION;
    SELECT b.blog_id
        FROM blog_pages b, users u
        WHERE b.author_id=u.user_id
            AND u.name='Ash Ketchum'
            AND b.created >= 2011/1/1;
COMMIT TRANSACTION;
```

Transactions (2/3)

- You can group multiple queries in a transaction optionally
- For example, Steven wants to donate $100 to Picachu:

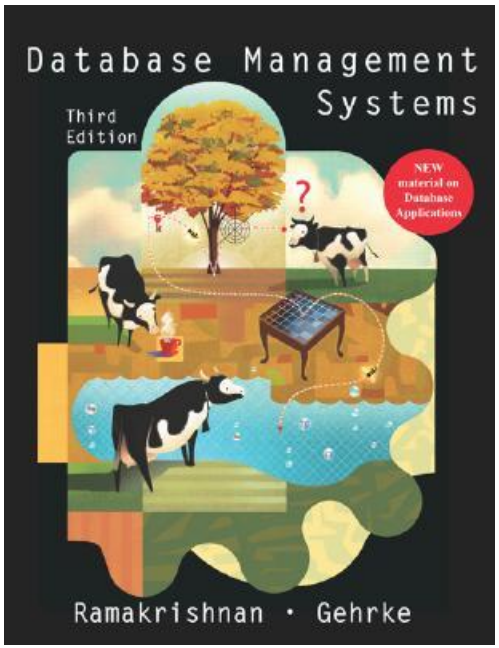
```
BEGIN TRANSACTION;
    UPDATE users
        SET balance=blance-100
        WHERE name='Ash Ketchum';
    UPDATE users
        SET balance=blance+100
        WHERE name='Picachu';
COMMIT TRANSACTION;
```

Transactions (3/3)

- A database ensures the ***ACID*** properties of transactions
- ***Atomicity***
  - All operations in a transaction either succeed (transaction commits) or fail (transaction rollback) together
- ***Consistency***
  - After/before each transaction (which commits or rollback), your data do not violate any rule you have set
  - E.g., blog_pages.author_id must be a valid users.user_id
- ***Isolation***
  - Multiple transactions can run concurrently, but cannot interfere with each other
- ***Durability***
  - Once a transaction commits, any change it made lives in DB permanently (unless overridden by other transactions)

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

Staring a New Thread

```
public class HelloRunnable implements Runnable {
    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 {
    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 {
    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 in the memory scheme
- Your CPU spends time on executing the code in `run()`

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

Thread Interference

- Given the same object o
- Suppose two threads execute

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

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

- Memory scheme?

Still Wrong!

- Solution1: the caller locks ○ during the entire increment/decrement period:

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

- Solution2: callee provides atomic methods

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


Blocking and Waiting

- Threads are ***blocked*** outside a critical section if some other is in
- A thread *A* in a critical section of `o` can give up the lock by calling `o.wait()`
  - So, some other blocking thread *B* can be in
  - *A* can regain the lock by `o.notifyAll()` (called by other threads)

```
while (c == 10) { // c has upper bound
   o.wait();
}
C++;
o.set(c);
```

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

Note

- DBMS ≠ 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
- What’s the data in memory (heap) look like?
  - Objects
  - References to objects
  - You define classes, the blueprint
- 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

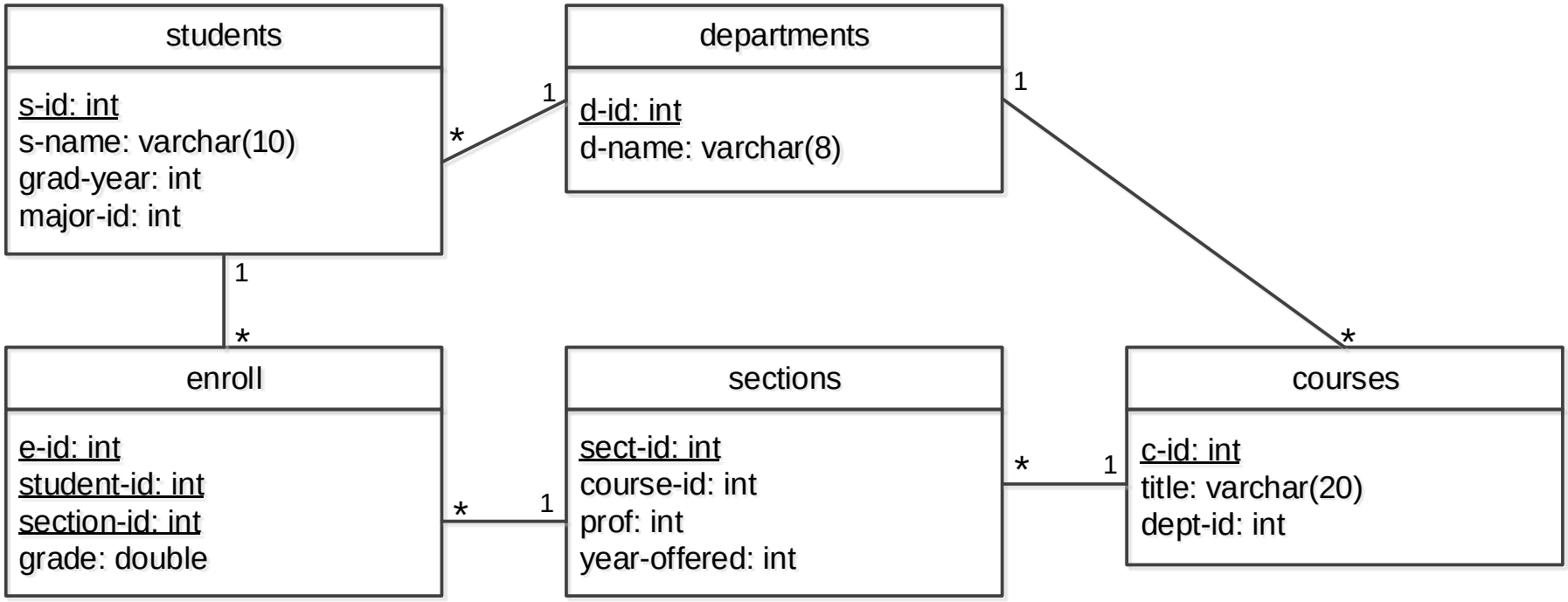
Why ER Model?

- 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
- ***Relationship*** between entities
  - References (has-a relationship)
  - Could be 1-1, 1-many, and many-many

Why Relational Model?

- A realization of ER model
  - Allows queries to be defined and answered
  - Still logic (not how your data stored physically)
- ***Relation***
  - Realization of 1) an entity group via table; or 2) a relationship
  - ***Fields/attributes*** as columns
  - ***Records/tuples*** as rows
- ***Primary Key***
  - Realization of ID via a group of fields
- ***Foreign key***
  - Realization of relationship
  - A record can have the primary key fields of the other record it refers to
  - Only 1-1 and 1-many
  - Intermediate relation is needed for many-many

Example: A student DB



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 Ash Ketchum 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

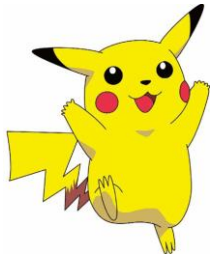
blog_id	url	created	author_id
33981	pokemon.com/…	2012/10/31	729
33982	apache.org/…	2012/11/15	4412

← record

field ↓

users

user_id	name	balance
729	Ash Ketchum	10,235
730	Picachu	NULL



Data Model and Queries (2/3)

Q1: among a set of blog pages, find those pages written by Ash Ketchum 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 ('pokemon.com/...', 2012/09/18, 729);
```

blog_pages

blog_id	url	created	author_id
33981	pokemon.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 Ash Ketchum 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='Ash Ketchum'
        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='Ash Ketchum'
        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	Ash Ketchum	10,235
33981	…	2009/10/31	729	730	Picachu	NULL
33982	…	2012/11/15	4412	729	Ash Ketchum	10,235
33982	…	2012/11/15	4412	730	Picachu	NULL
41770	…	2012/10/20	729	729	Ash Ketchum	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	Ash Ketchum	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='Ash Ketchum'
          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	Ash Ketchum	10,235

p = product(b, u)

blog_id	url	created	author_id	user_id	name	balance
33981	…	2009/10/31	729	729	Ash Ketchum	10,235
33981	…	2009/10/31	729	730	Picachu	NULL
33982	…	2012/11/15	4412	729	Ash Ketchum	10,235
33982	…	2012/11/15	4412	730	Picachu	NULL
41770	…	2012/10/20	729	729	Ash Ketchum	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='Ash Ketchum'
        AND b.created >= 2011/1/1;
```

project(s, select…)

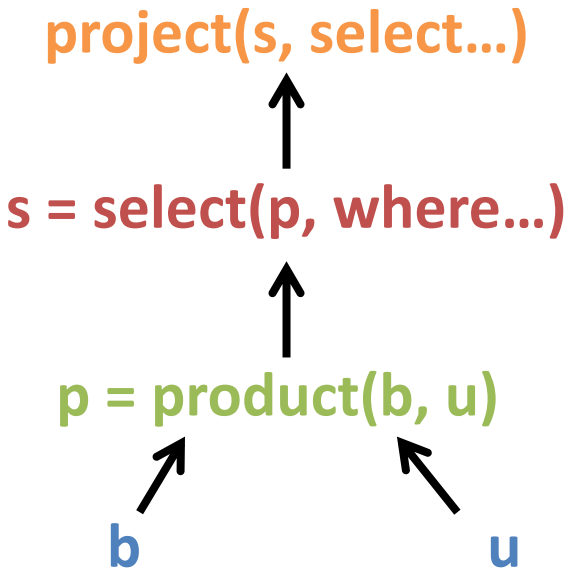
blog_id
41770

s = select(p, where…)

blog_id	url	created	author_id	user_id	name	balance
41770	…	2012/10/20	729	729	Ash Ketchum	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	pokemon.com/…	2012/10/31	729	programming	5638
33981	pokemon.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	pokemon.com/…	2012/10/31	729	programming	5638
33981	pokemon.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	pokemon.com/…	2012/10/31	729	Java prog.	5638
33981	pokemon.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	pokemon.com/…	2012/10/31	729	programming	5638
33981	pokemon.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