

Introduction to Database Systems

Shan-Hung Wu
CS, NTHU

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 Steven Sinofsky after 2011
 - Q2: among a set of employers, increase the salary by 20% for those who have worked longer than 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 Steven Sinofsky after 2011

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

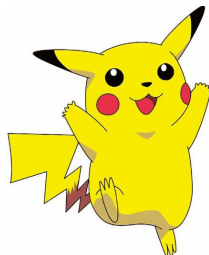
- Identify *records* (e.g., web pages, authors, etc.) with the same *fields* in your data and place them into respective *tables*

blog_pages

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

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;
```

Advantages of a Database System

- It answers *queries* fast
 - Q1: among a set of web pages, find those pages written by Steven Sinofsky after 2011
 - Q2: among a set of employers, increase the salary by 20% for those who have worked longer than 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='Steven Sinofsky'  
            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 Pikachu:

```
BEGIN TRANSACTION;  
    UPDATE users  
        SET balance=balance-100  
        WHERE name='Steven Sinofsky';  
    UPDATE users  
        SET balance=balance+100  
        WHERE name='Pikachu';  
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)

So, why do you need
a cloud database system?

Definition



- A **cloud database** is a database designed to run in the cloud
 - Manages data of tremendous applications (called **tenants**)
- Is MySQL a cloud database?
 - I can run MySQL in a Amazon EC2 VM instance
- No

What's the Difference

- Ideally, in addition to all features provided by a traditional database, a cloud database should ensure **SAE**:
 - **high Scalability**
 - High max. throughput (measured by Tx/Query per second/minute)
 - **high Availability**
 - Stay on all the time, despite of server/network failure
 - **Elasticity**
 - Add/shutdown physical/virtual servers dynamically based on the current workload and/or capability

The evolving database landscape

451 Research

Relational

Analytic

Hadoop Piccolo Teradata Aster Netezza ParAccel SAP Sybase IQ
 Hadapt Infobright EMC Greenplum IBM InfoSphere
 HPCC RainStor Teradata Calpont Action VectorWise HP Vertica

Non-relational

NoSQL

MarkLogic

Versant

McObject

Progress

Objectivity

DataStax Enterprise
 Castle Acunu

Citrusleaf
 BerkeleyDB Cassandra HBase
 Oracle NoSQL
 RethinkDB
 HandlerSocket*

Riak Redis-to-go
 SimpleDB
 DynamoDB

LevelDB
 Redis
 Membrain
 Voldemort

Couchbase
Key value

Big tables

App Engine

Datastore

Redis-to-go

SimpleDB

DynamoDB

Iris

Couch Lab

Couchbase

MongoDB

CouchDB

Lotus Notes

InterSystems

Graph

Neo4J
 InfiniteGraph
 OrientDB
 DEX

NuvolaBase

-as-a-Service

Cloudant

RavenDB

MongoHQ

MongoLab

MongoCloud

MongoDB

CouchDB

Lotus Notes

InterSystems

Document

-as-a-Service

FathomDB
 Amazon RDS
 Database.com
 Postgres Plus Cloud
 ClearDB
 Rackspace MySQL Cloud
 Google Cloud SQL
 SQL Azure

Action Ingres
 EnterpriseDB
 SAP Sybase ASE

-as-a-Service

StormDB
 Xeround

Tokutek

Storage engines

MySQL Cluster

ScaleDB

ScaleBase

ScaleArc

Scale

Galera

CodeFutures

NewSQL

NuoDB VoltDB **New databases**

MemSQL JustOneDB SQLFire

Drizzle Akiban Translattice

SchoonerSQL Clustrix

GenieDB

ScaleArc

ParElastic

Continuent

ScaleBase

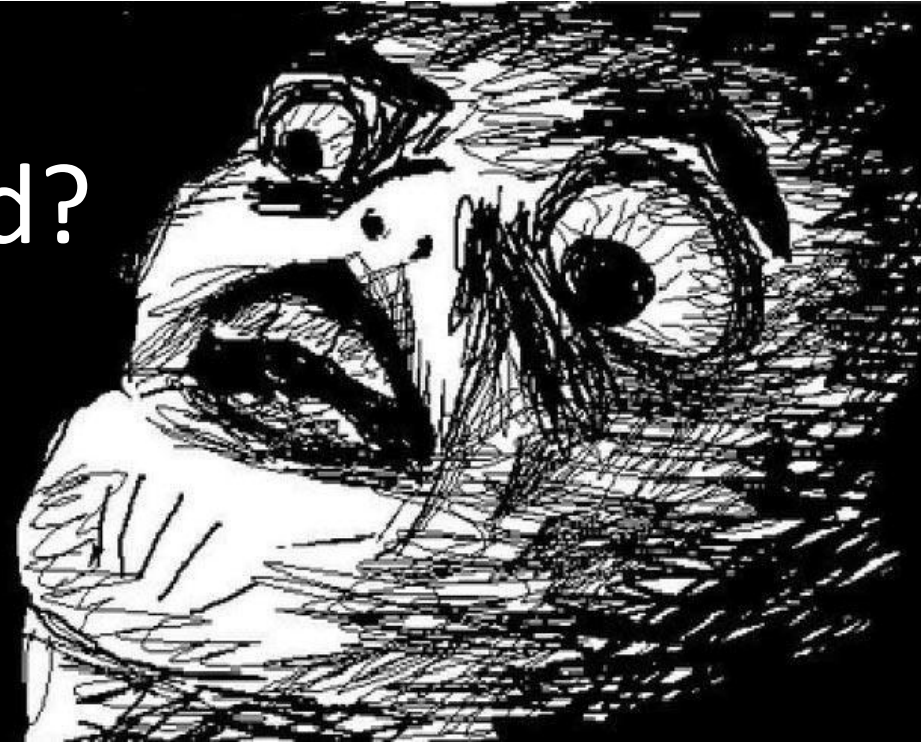
Clustering/sharding

Operational

Starcounter

InterSystems

Why So Complicated?



Full DB functions + SAE = a goal no one can achieve (currently)

- Even with Oracle 11g + SPARC SuperCluster
 - 30,249,688 TPC-C transactions per minute
 - \$30+ million USDyou loss elasticity

Primer Goal of This Class

- To guide you through the design trade-offs of existing database systems in the cloud
- **NOT** a comprehensive survey
- **NOT** on how to use the cloud databases

Goals

- To use a DB system (in only 2 weeks)
- To understand how to *write* a DB system
 - Architecture
 - Trade-offs

Prerequisites

- Data structure
- Good programming skill
 - OOP (in Java)
 - Multi-threaded programming
 - Project management tools like Git

Syllabus

- [Here](#)
 - Subject to change
- Mon: physical + **video** lecture
- Thu: labs (TA time)
 - Explain your new assignment
 - Review your pass assignment
- Homework every 2 weeks
 - Not only code, but also reports
 - Summarizing ***diff from TA's code*** and ***your observations***

Grading

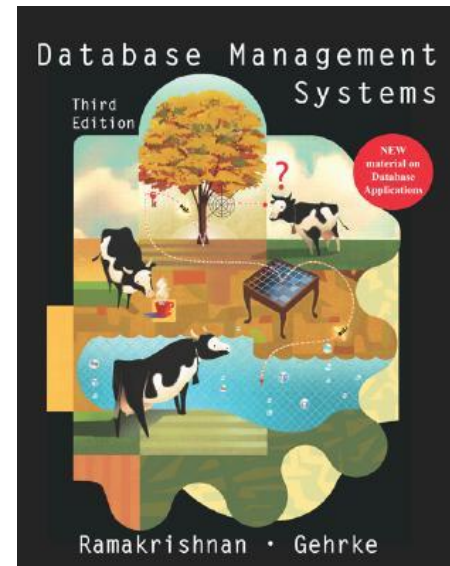
- Homework (x5): 50%
- SQL & Java quiz: 15%
 - *On next Thu (2/24)*
- Midterm exam: 15%
- Final project: 20%
- Q/A Bonus: up to 5%

Resources

- Text Book
 - Lecture notes
 - Reference links
- Course page
 - <http://www.cs.nthu.edu.tw/~shwu>
- TODO
 - Register your team

About the Quiz...

- Coverage:
 - Java concurrency
 - How to use a DBMS (SQL language)
- On **2/24**
- Assigned readings:
 - [Java Concurrency Tutorial](#)
 - [SQL Tutorial](#)
 - ~~Chaps 2 and 3 on ER & relational models~~



Questions?

FAQ (1/2)

- Do I need to write programs in this course?
 - A lot!
 - ***We will give extensive coding assignments***
- Do I need to write code with others?
 - Yes, 1~3 students a team

FAQ (2/2)

- Do we need to come to the class?
 - No, as long as you can pass
- Is this a light-loading class or heavy-loading class?
 - Should be *heavy* to most students
 - *Reserve time, otherwise you will have high chance to fail*