

Databases

SYSC3020

Databases: agenda

- Relational Databases
 - Properties (design goals)
 - Relational Data Modeling
 - Queries: SQL
 - ACID properties
 - Data Warehousing
- NoSQL databases
 - Design Goals
 - Different models

Relational Databases

- Large (GBs of data)
- Safe and reliable
 - Must resist system crashes, power outages, hacking...
 - Handle multiple concurrent access
- Efficient: fast query processing
- Convenient:
 - declarative queries
 - Interoperability standards (ODBC)

The Relational Model

- Data as tables

moodle_users					
id	username	firstname	lastname	email	password
1	alicesmith	Alice	Smith	a@bbc.co	himum!99
2	bobjones	Robert	Jones	bob@a.ca	hidad%90
3	charlie	Charles	Chan	ccd98@a.b	mycat01#
4	evajoli	Eva	Joly	ej@cc.ca	mydog!99

The Relational Model

- Data as tables

The diagram illustrates a table named 'moodle_users'. The table has a light blue header row and four data rows. Annotations include: 'table name' pointing to the header, 'fixed column headers' pointing to the header cells, and 'rows of data' pointing to the data rows.

moodle_users					
id	username	firstname	lastname	email	password
1	alicesmith	Alice	Smith	a@bbc.co	himum!99
2	bobjones	Robert	Jones	bob@a.ca	hidad%90
3	charlie	Charles	Chan	ccd98@a.b	mycat01#
4	evajoli	Eva	Joly	ej@cc.ca	mydog!99

The Relational Model

- Data as tables

relational schema

table name

fixed column headers

moodle_users					
id	username	firstname	lastname	email	password
1	alicesmith	Alice	Smith	a@bbc.co	himum!99
2	bobjones	Robert	Jones	bob@a.ca	hidad%90
3	charlie	Charles	Chan	ccd98@a.b	mycat01#
4	evajoli	Eva	Joly	ej@cc.ca	mydog!99

rows of data

The Relational Model

- Data as tables

moodle_users						
id	username	firstname	lastname	email	password	
1						
2	socialwiki_pages					
	id	title	content	timestamp	parentid	userid
	1	Course Overview	“this course is ...”	2014-05-01	null	3
	5	Software Quality	“ stuff”	2014-05-01	null	3
	12	Software Quality	“ stuff and edits”	2014-05-01	5	6

The Relational Model

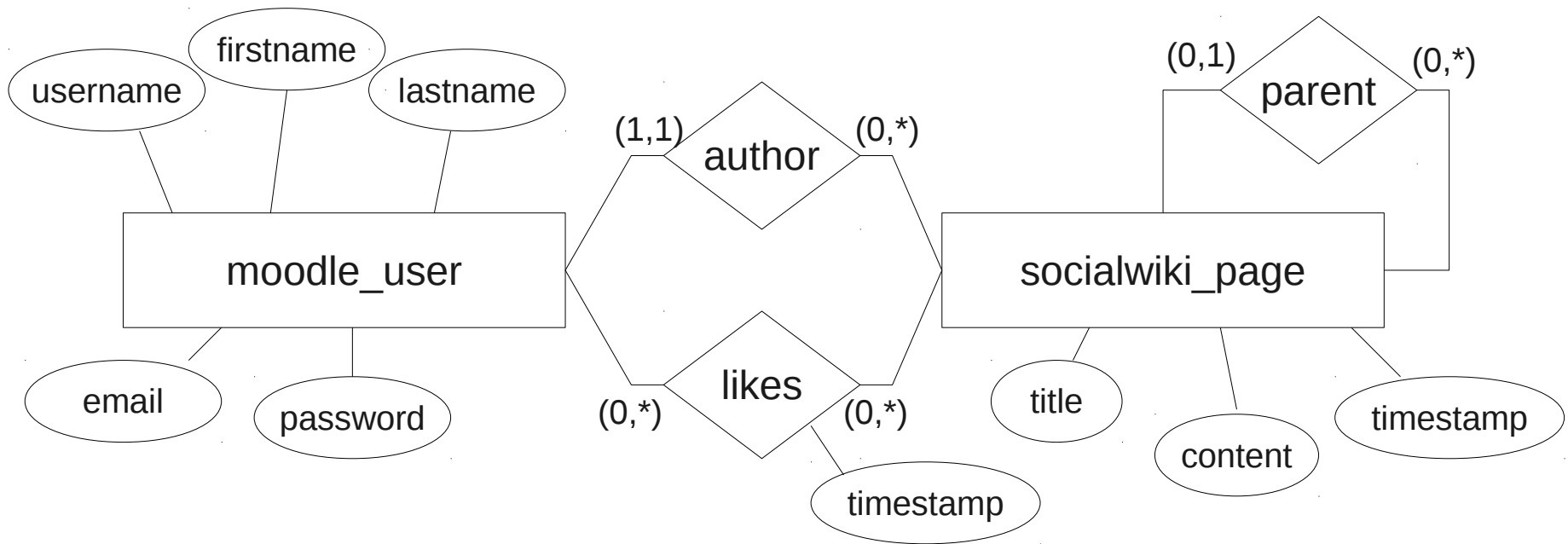
- Data as tables

moodle_users						
id	username	firstname	lastname	email	password	
1						
2						
3						
4						

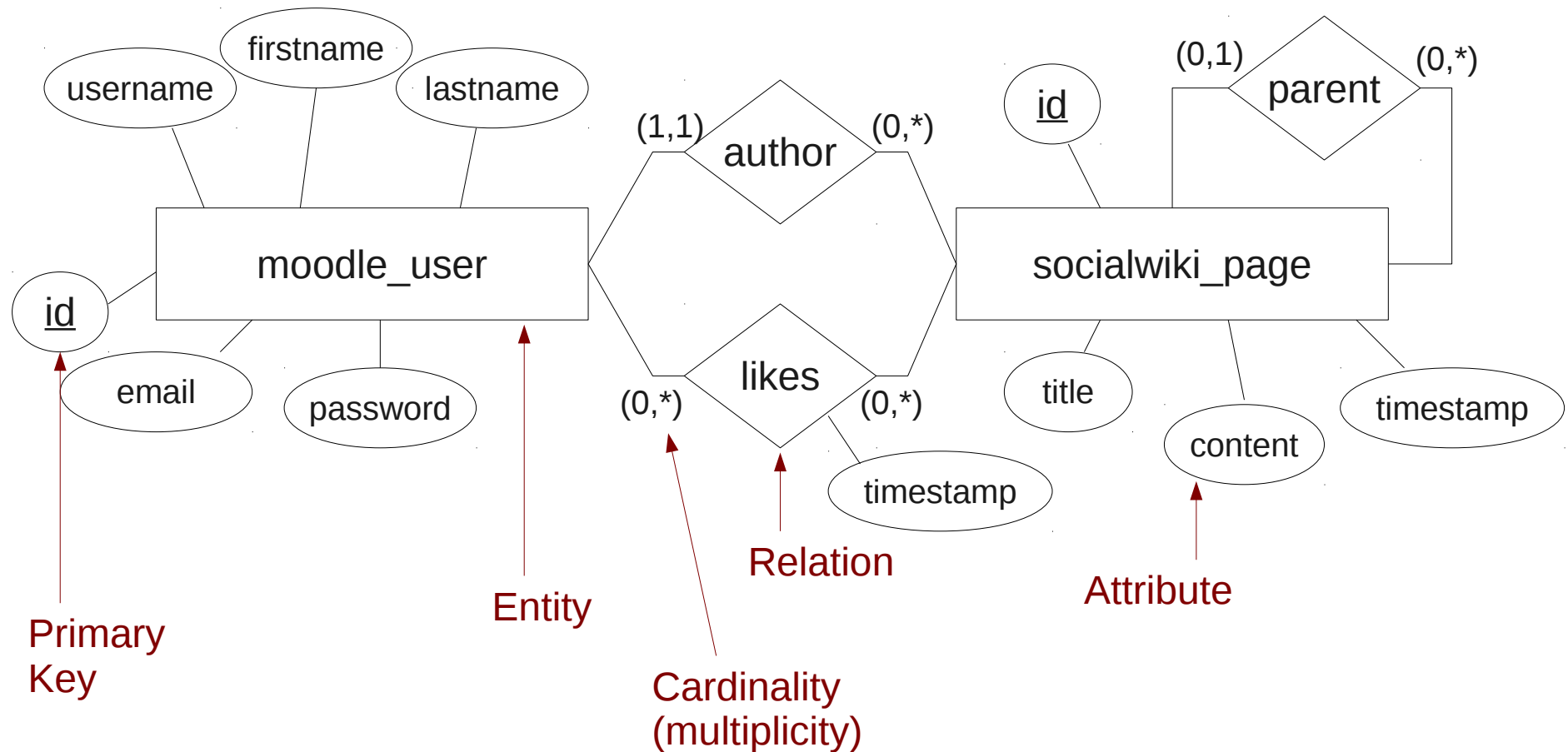
socialwiki_pages						
id	title	content	timestamp	parentid	userid	
1	Cou				3	
5	Soft				3	
12	Soft				6	

socialwiki_likes				
id	userid	pageid	timestamp	
1	3	5	2014-05-03 12:24	
2	3	6	2014-05-03 15:50	
3	5	5	2014-05-20 08:40	
4	7	5	2014-05-22 18:00	
5	7	2	2014-05-22 18:01	

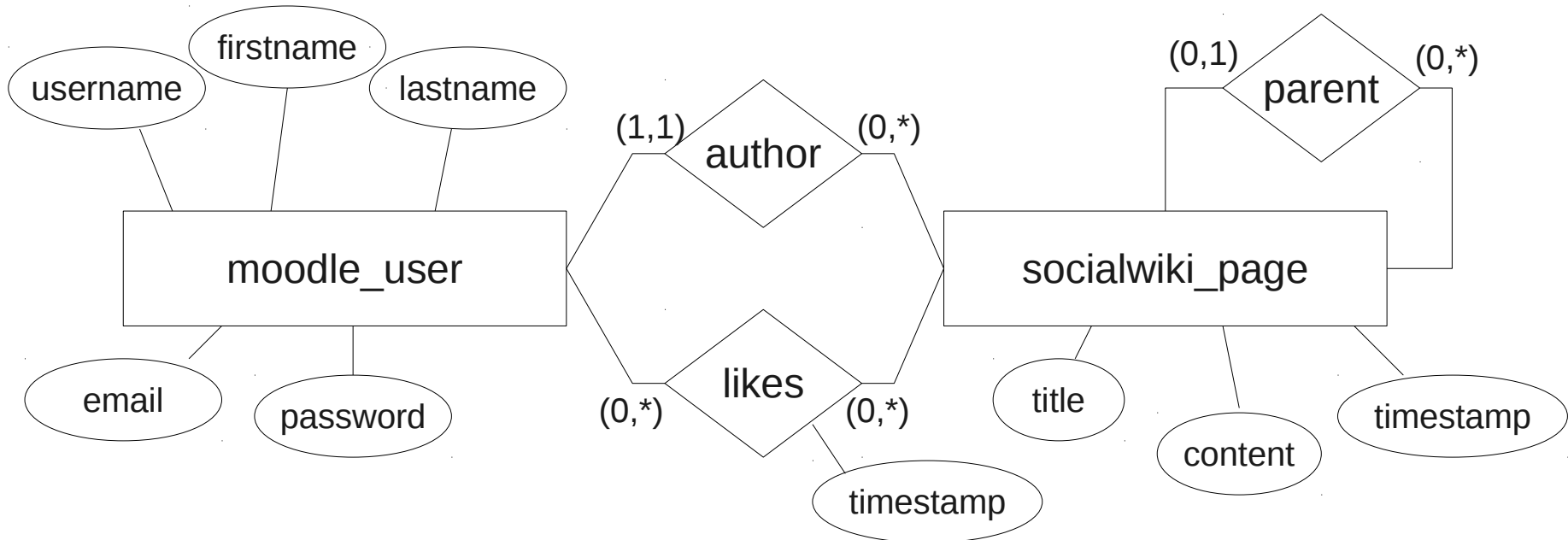
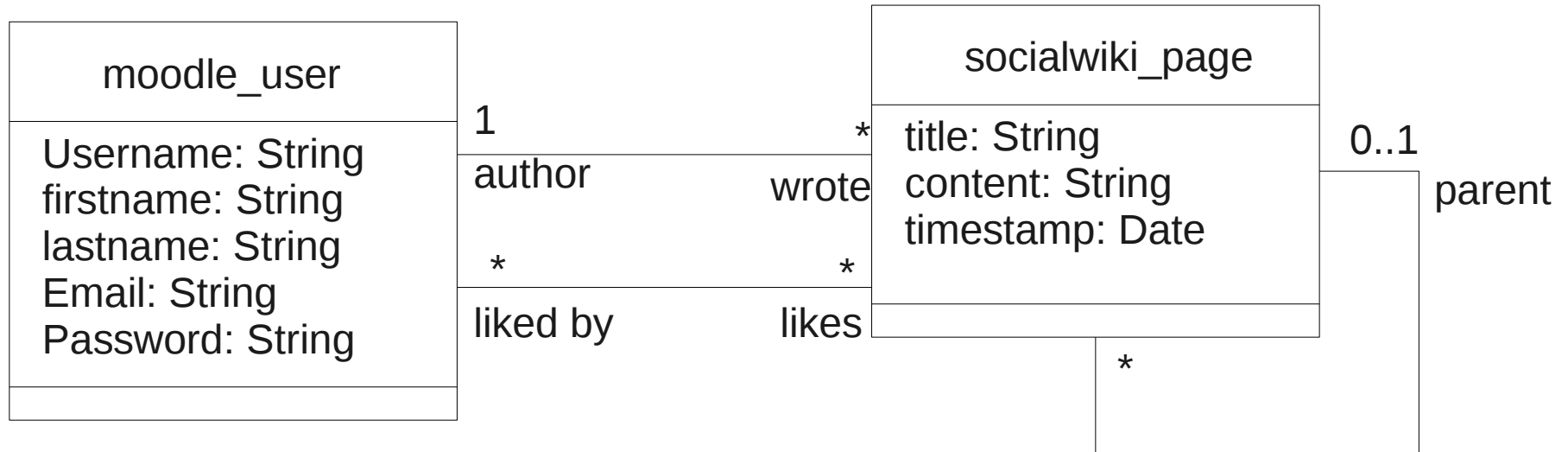
The Entity-Relation Model



The Entity-Relation Model



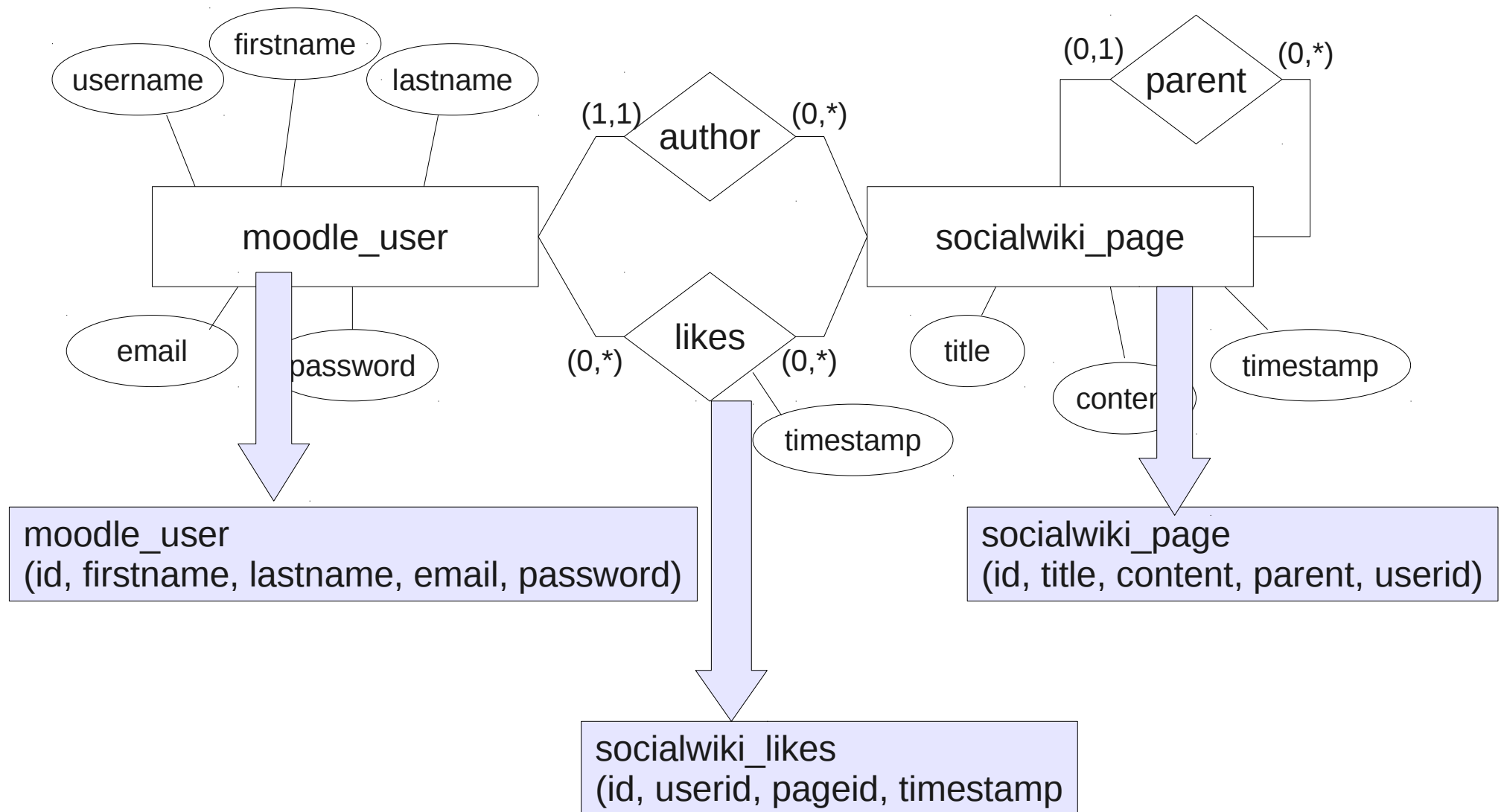
The Entity-Relation Model



ER model to relational schema

- Systematic process
- Simplified guidelines:
 - Entity => table
 - Attributes => columns
 - Introduce *unique id* attribute for each table
 - 1-to-many relation => column on the “many” side
 - Many-to-many relation => table
 - Relation attributes => columns

ER model to relational schema



Relational Queries: SQL

CRUD: create, retrieve, update, delete.

- Create:
 - **Create** tables, columns...
 - **Insert** rows into tables
- Retrieve
 - **Select** (retrieve) rows of data
- Update
 - **Update** (modify) rows
- Delete
 - **Drop** tables
 - **Delete** rows

SQL retrieval queries

- Based on relational algebra = strong mathematical foundations
- Declarative queries: what you want, not how to get it
- 3 main aspects
 - filter rows (relational algebra: selection)
 - filter columns (relational algebra: projection)
 - combine tables (relational algebra: join or cartesian product)

Relational Queries: SQL

- Based on relational algebra = strong mathematical foundations
- Declarative queries: what you want, not how to get it
- What you can do:
 - filter rows (relational algebra: selection)
 - filter columns (relational algebra: projection)
 - combine tables (relational algebra: join or cartesian product)

get all users who have authored a page with title “software quality”:

```
SELECT moodle_users.username
FROM    moodle_users, socialwiki_pages
WHERE   moodle_users.id=socialwiki_pages.userid
         AND socialwiki_pages.title = “Software Quality”
```


Transactions

Requirements:

- Concurrent database access
- Resilience to system failures

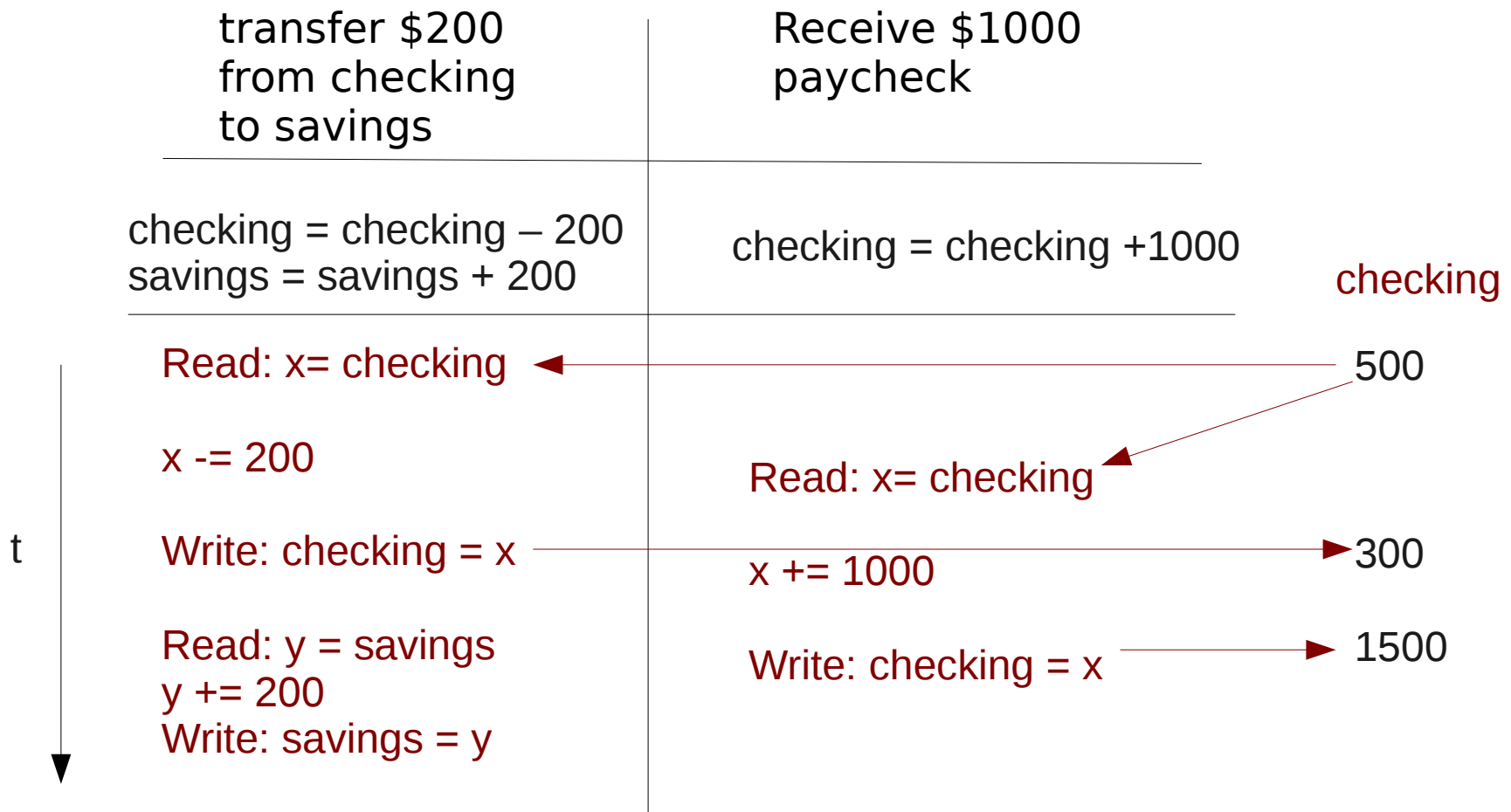
Transactions

- Bank DB: Transaction examples:

	transfer \$200 from checking to savings	Receive \$1000 paycheck
	checking = checking – 200 savings = savings + 200	checking = checking +1000
t ↓	Read: x= checking x -= 200 Write: checking = x Read: y = savings y += 200 Write: savings = y	Read: x= checking x += 1000 Write: checking = x

Transactions

- Bank DB: Transaction examples:



Transactions

- Bank DB: Transaction examples:

transfer \$200 from checking to savings	Receive \$1000 paycheck
checking = checking - 200 savings = savings + 200	checking = checking + 1000
<div> <div>t</div> <div> <div>Read: x= checking</div> <div>x -= 200</div> <div>Write: checking = x</div> <div>-- CRASH --</div> <div>Read: y = savings</div> <div>y += 200</div> <div>Write: savings = y</div> </div> </div>	<div> <div>Read: z= checking</div> <div>z += 1000</div> <div>Write: checking = z</div> </div>

ACID

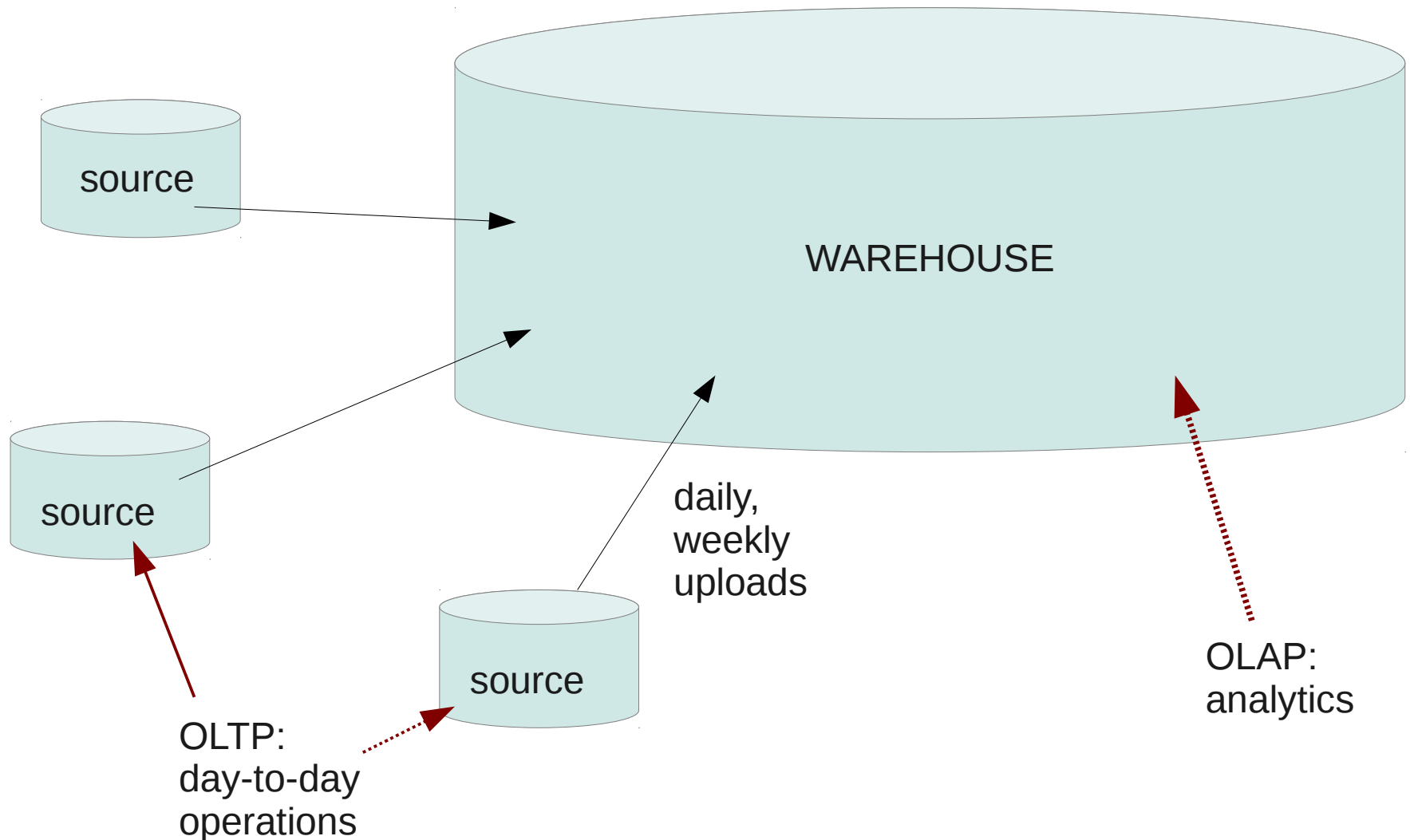
- “transactional processing” = ACID properties
- Atomicity (of transactions)
- Consistency (of DB states)
- Isolation (concurrency management)
- Durability (data not lost over crashes...)

Consistency

2 kinds of consistency:

- Integrity constraints respected
- Consistency across replicated data

Data Warehousing



Data Warehousing

- OLTP: online transaction processing
 - Day-to-day operations
 - Simple, frequent queries
 - Data must be up-to-date
- OLAP: online analytical processing
 - Knowledge discovery, machine learning... “business intelligence”
 - Slow, complex queries over very large amounts of data
 - Can be slightly outdated
 - Massive infrastructures (clusters)
 - “Dimensional” data models (time, products, regions...)

Databases: agenda

- Relational Databases
 - Properties
 - Relational Data Modeling
 - Queries: SQL
 - ACID Properties
 - Data Warehousing
- NoSQL databases
 - Design Goals
 - Different models

NOSQL databases

- Amazon, Google, Facebook, Twitter...
- “Internet age” performance requirements:
 - PB+ (1 petabyte = 10^{15} bytes = 1M GB) of data
 - Very high frequencies of operations, on logically centralized data

=> Limits of traditional (relational) DBs.
- Elasticity

NOSQL databases

- Amazon, Google, Facebook, Twitter...
- “Internet age” performance requirements:
 - PB+ (1 petabyte = 10^{15} bytes) of data
 - Very high frequencies of operations, on logically centralized data
 - => Limits of traditional (relational) DBs.
- CAP theorem:
 - Consistency, Availability, Partition tolerance: pick any 2
- Elasticity
- Need for parallel processing = MapReduce...

NOSQL databases

- CAP theorem:
 - Consistency, Availability, Partition tolerance: pick any 2
- Amazon: an extra 1/10 second latency => sales drop 1%
- Google: 1/2 second increase in latency => traffic drops 20%

=> **Availability** is key

- relaxed view on consistency

NOSQL database Models

- Key-Value stores
- BigTable
- Document
- Graph

Key-Value Stores

- SimpleDB (Amazon)
- BerkeleyDB
- Voldemort (LinkedIn, now open-source)
- “HashMaps”
- Distributed !
- Key-Value storage is the principle underlying most NoSQL DBs
- Main motivation: $\log(n)$ efficiency to retrieve data

Columnar databases (BigTable)

- BigTable (Google)
- Cassandra (Apache)
- Hypertable

- Key -> compound (multiple) values
- Tables with many columns
- Rows are versioned
- Again: efficiency, scalability, eventual consistency

Document databases

- CouchDB
- MongoDB

```
16524: {  
  "Subject": "I like Plankton",  
  "Author": "Rusty",  
  "PostedDate": "5/23/2006",  
  "Tags": ["plankton", "baseball", "decisions"],  
  "Body": "I decided today that I don't like  
          baseball. I like plankton."  
}
```

- No Schema needed
- Access optimized for documents (through key)

Graph databases

- Neo4J
 - AllegroGraph RDF Store
-
- Nodes and edges with properties
 - Queries: graph traversal
 - May be implemented by K-V stores, document DBs...

MapReduce

- Term popularized by Google ca. 2005
- Open-source frameworks: Apache Hive/Hadoop
- Supported by other NOSQL DBs
- Perform complex computations in parallel across thousands of machines
- Express the problem as 2-step computation: map - reduce

MapReduce

- Typical problem structure:
 - Read a lot of data
 - **Map**: extract something you care about from each record (**1 to 1**)
 - Shuffle, sort
 - **Reduce**: aggregate, summarize, filter, transform (**many to 1**)
- Outline stays the same, express algorithm by specifying **Map** and **reduce** functions

MapReduce

Map = apply **f** to each of the input set

Reduce = apply **f** to the input set as a whole

- **Map:** $(k, v) \rightarrow (k', v')$
- **Reduce:** $(k, \langle v \rangle^*) \rightarrow (k', v')^*$

Functional programming

Python: Map, Reduce, Filter

```
from math import sqrt  
map(sqrt, [1,4,9,16])           # ==> [1.0, 2.0, 3.0, 4.0]
```

```
map(str.lower, ['A','b','C'])   # ==> ['a', 'b', 'c']
```

```
filter(lambda x: x % 2 == 0, [1,2,3,4,5]) # ==> [2, 4]
```

```
reduce(lambda s,x: s+str(x), [1,2,3,4], "") # ==> '1234'
```