CS2102 Lecture 1
Introduction

Database Management System (DBMS)

- What is a DBMS?
  - Software for managing large persistent data
- Advantages of using DBMS
  - Data Independence
  - Efficient Data Access
  - Data Integrity & Security
  - Data Administration
  - Transaction Management
    - Concurrent Access & Crash Recovery
  - Query language

Traditional Data Processing: File Processing Techniques

```
initialize some book-keeping information I
open data file F
while (F is not empty)
        read next record r from F
        if (r satisfies some condition) then
             do something with r
        update I if necessary
do something with I if necessary
close file F
```

Study of DBMS

- Database design
  - How to model the data requirements of applications
  - How to organize data using a DBMS
  - Topics: relational model, ER model, schema refinement
- Database programming
  - How to create, query, and update a database
  - How to specify data constraints
  - How to use SQL in applications
  - Topics: SQL, relational algebra/calculus
- DBMS implementation
  - How to build a DBMS (Covered in CS3223 & CS4224)

Describing Data in a DBMS

- A DBMS allows users to define and query data in terms of a *data model*
- A data model is a collection of concepts for describing data
- A schema is a description of the structure of a database using a data model
- A schema instance is the content of the database at a particular time

Data Models

- Network Model (e.g., General Electric's IDS (1964))
- Hierarchical Model (e.g., IBM's IMS (1966))
- Relational Model
  - **Commercial RDBMS**: IBM DB2, Microsoft SQL Server, Oracle, SAP ASE, etc.
  - **Open-source RDBMS**: MariaDB, MySQL, SQLite, etc.
- Object-oriented Model (e.g., ObjectStore 1988)
- Object-relational Model (e.g., Postgres 1986)
- etc.

Relational Database Systems

(Image: Software Engineering Daily)

Relational Data Model

- Introduced by **Edgar Codd** of IBM Research Laboratory in 1970
- Data is modeled using relations (tables with rows & columns)

Students

studentId	*name*	*birthDate*	*cap*
3118	Alice	1999-12-25	3.8
1423	Bob	2000-05-27	4.3
5609	Carol	1999-06-11	4.0

Degree/Arity = Number of columns

Cardinality = Number of rows

- Each relation has a definition called a relation schema
  - Schema specifies attributes and data constraints
  - Data constraints include domain constraints

Students (*studentId*: **integer**, *name*: **text**, *birthDate*: **date**, *cap*: **numeric**)

- Each row in a relation is called a tuple/record; it has one component for each attribute of relation

(1423, ‘Bob’, 2000-05-27, 4.3)

Relational Data Model (cont.)

- Domain - a set of atomic values (e.g., integer, numeric, text)
- Let domain(A_i) denote the domain of an attribute A_i (set of possible values for A_i)
- Each value of attribute A_i is either a value in domain(A_i) or null
- null is a special value used to indicate that the value is either not applicable or unknown
- A relation is a set of tuples
  - Consider a relation schema $R(A_1, A_2, \cdots, A_n)$ with n attributes $A_1, \cdots, A_n$
  - Each instance of schema R is a relation which is a subset of $\{(a_1, a_2, \cdots, a_n) \mid a_i \in domain(A_i) \cup \{null\}\}$

Relational Data Model (cont.)

- Consider the relation schema **Lectures(course, day, hour)**
  - domain(course) = {'cs101', 'cs203', 'cs305'}
  - domain(day) = {1, 2, 3, 4, 5}
  - domain(hour) = {8, 10, 12, 14, 16}
- Each instance of `Lectures` is a subset of

{'cs101', 'cs203', 'cs305', null} $\times$ {1, 2, 3, 4, 5, null} $\times$ {8, 10, 12, 14, 16, null}

course	*day*	*hour*
cs101	1	8
cs101	1	10
cs101	1	12
cs101	1	14
cs101	1	16
cs101	2	8
⋮	⋮	⋮
null	null	16
null	null	null

Relational Data Model (cont.)

- A relational database schema consists of a set of relation schemas

Students	(*studentId*: **integer**, *name*: **text**, *birthDate*: **date**, *cap*: **numeric**)
Courses	(*courseId*: **integer**, *name*: **text**, *credits*: **integer**)
Enrols	(*sid*: **integer**, *cid*: **integer**, *grade*: **numeric**)

- A relational database is a collection of tables

Students

studentId	*name*	*birthDate*	*cap*
3118	Alice	1999-12-25	3.8
1423	Bob	2000-05-27	4.3
5609	Carol	1999-06-11	4.0

Courses

courseId	*name*	*credits*
101	Programming in C	5
112	Discrete Mathematics	4
204	Analysis of Algorithms	4
311	Database Systems	5

Enrols

sid	*cid*	*grade*
3118	101	5.0
3118	112	4.0
3118	204	3.0
1423	112	4.5
......	......	......

- Relational database schema = relational schemas + data constraints

Relation/Database Schema/Instance

- Relation schema

  Students (*studentId*: **integer**, *name*: **text**, *birthDate*: **date**, *cap*: **numeric**)

- Database schema

Students	(*studentId*: **integer**, *name*: **text**, *birthDate*: **date**, *cap*: **numeric**)
Courses	(*courseId*: **integer**, *name*: **text**, *credits*: **integer**)
Enrols	(*sid*: **integer**, *cid*: **integer**, *grade*: **numeric**)

- Relation (or relation instance)

- Database (or database instance)

Students

studentId	*name*	*birthDate*	*cap*
3118	Alice	1999-12-25	3.8
1423	Bob	2000-05-27	4.3
5609	Carol	1999-06-11	4.0

Courses

courseId	*name*	*credits*
101	Programming in C	5
112	Discrete Mathematics	4
204	Analysis of Algorithms	4
311	Database Systems	5

Enrols

sid	*cid*	*grade*
3118	101	5.0
3118	112	4.0
3118	204	3.0
1423	112	4.5
. . . . . .	. . . . . .	. . . . . .

Integrity Constraints (ICs)

- Integrity constraint: a condition that restricts the data that can be stored in database instance
  - Specified when schema is defined
  - ICs are checked when relations are updated
- A legal relation instance is a relation that satisfies all specified ICs.
- A DBMS enforces ICs - allows only legal instances to be stored

Integrity Constraints (ICs) (cont.)

- Without any additional integrity constraints, each instance of $R(A_1, \cdots, A_n) \subseteq \{(a_1, a_2, \cdots, a_n) \mid a_i \in domain(A_i) \cup \{null\}\}$

Students

studentId	*name*	*birthDate*	*cap*
3118	Alice	1999-12-25	3.8
1423	Bob	2000-05-27	4.3
5609	Carol	1999-06-11	6.5
1423	Dave	2000-10-05	3.7

Courses

courseId	*name*	*credits*
101	Programming in C	5
112	Discrete Mathematics	4
204	Analysis of Algorithms	4
null	Compiler Design	4
311	Database Systems	5

Enrols

sid	*cid*	*grade*
3118	101	5.0
3118	112	4.0
3118	202	3.0
1423	112	3.7
5609	101	4.5

Types of Integrity Constraints

- Domain constraints restrict attribute values of relations
- Key constraints
- Foreign key constraints
- Other general constraints

Key Constraints

- A superkey is a subset of attributes in a relation that uniquely identifies its tuples
  - No two distinct tuples of a relation have the same values in all attributes of superkey
- **Example**: Which of the following is a superkey for the relation Students (studentId, name, birthDate, cap)?
  - {studentId}
  - {name}
  - {birthDate}
  - {cap}
  - {studentId, name}
  - {studentId, birthDate}
  - {studentId, cap}
  - {name, birthDate}
  - {name, cap}
  - {birthDate, cap}
  - {studentId, name, birthDate}
  - {studentId, name, cap}
  - {studentId, birthDate, cap}
  - {name, birthDate, cap}
  - {studentId, name, birthDate, cap}

Key Constraints (cont.)

- A key is a superkey that satisfies the additional property:
  - No *proper subset* of the key is a superkey
- Thus, a key is a minimal subset of attributes in a relation that uniquely identifies its tuples
- **Example**: Which of the following is a key for the relation Students (studentId, name, birthDate, cap)?
  - {studentId}
  - {name}
  - {birthDate}
  - {cap}
  - {studentId, name}
  - {studentId, birthDate}
  - {studentId, cap}
  - {name, birthDate}
  - {name, cap}
  - {birthDate, cap}
  - {studentId, name, birthDate}
  - {studentId, name, cap}
  - {studentId, birthDate, cap}
  - {name, birthDate, cap}
  - {studentId, name, birthDate, cap}

Key Constraints (cont.)

- Key attribute values cannot be *null*
- A relation could have multiple keys called candidate keys
- One of the candidate keys is selected as the primary key
- **Example**:
  - `Accounts` (acctId, name, email, address)
  - Accounts has two candidate keys: {*acctId*} and {email}
  - Any one of them could be selected as the primary key

Key Constraints (cont.)

- Consider the relation schema

  Lectures (cname, pname, day, hour)

- cname is a course taught by professor pname at time given by day & hour
- `Lectures` satisfies these constraints:
  - day $\in \{1, 2, 3, 4, 5\}$
  - hour $\in \{8, 10, 12, 14, 16\}$
  - At any time, each professor is teaching at most one course
  - Each course is taught by exactly one professor
  - Each course could have multiple lectures

cname	**pname**	**day**	**hour**
cs101	alice	1	10
cs101	alice	3	14
cs200	bob	2	8
ma300	bob	1	10

Foreign Key Constraints

- A subset of attributes in a relation is a foreign key if it refers to the primary key of a second relation

Enrolls

sid	*cid*	*grade*
1	**204**	3.0
1	**101**	5.0
2	**204**	3.0
3	**101**	4.0
3	**112**	4.0

Referencing relation

Courses

courseId	*name*	*credits*
101	Programming in C	5
112	Discrete Mathematics	4
204	Analysis of Algorithms	4
311	Database Systems	5

Referenced relation

- *cid* is a foreign key in `Enrolls` that refers to the primary key *courseId* in `Courses`
- Foreign key constraint: each foreign key value in referencing relation must either (1) appear as primary key value in referenced relation or (2) be a null value

Foreign Key Constraints (cont.)

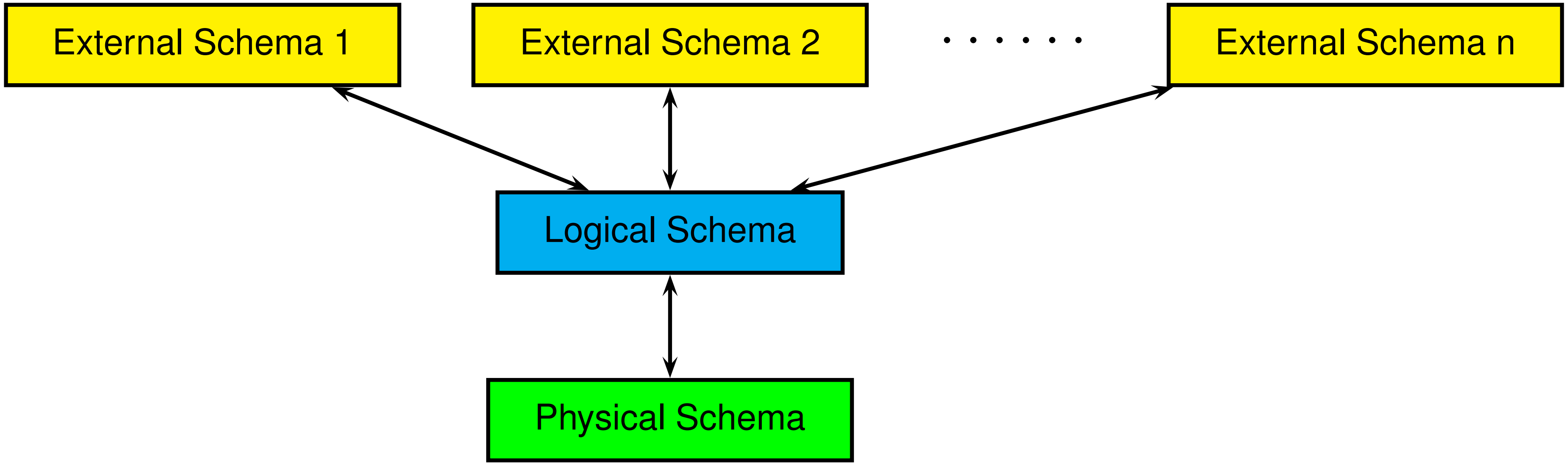
- The referencing & referenced relations could be the same relation
- **Example**: Each employee has at most one manager

Employees

eid	*ename*	*managerid*
001	Alice	*null*
002	Bob	**001**
003	Carol	**001**
007	Dave	*null*
008	Eve	**007**

- Constraints on `Employees` table:
  - *eid* is the primary key
  - *managerid* is a foreign key that refers to *eid*
- Foreign key constraints are also known as referential integrity constraints

Levels of Data Abstraction



- Data in DBMS is described at three levels of abstractions
- **Logical Schema** - logical structure of data in DBMS
- **Physical Schema** - how the data described by logical schema is physically organized in DBMS
- **External Schema** - A customized view of logical schema for a group of users or an individual user

External Schema Example

- Consider the following logical database schema:

Students	(*studentId*, *sname*, *birthDate*, *cap*)
Profs	(*profId*, *pname*, *email*, *office*)
Courses	(*courseId*, *cname*, *credits*, *profId*, *lectureTime*)
Enrols	(*sid*, *cid*, *grade*)

- External schema for Alice:

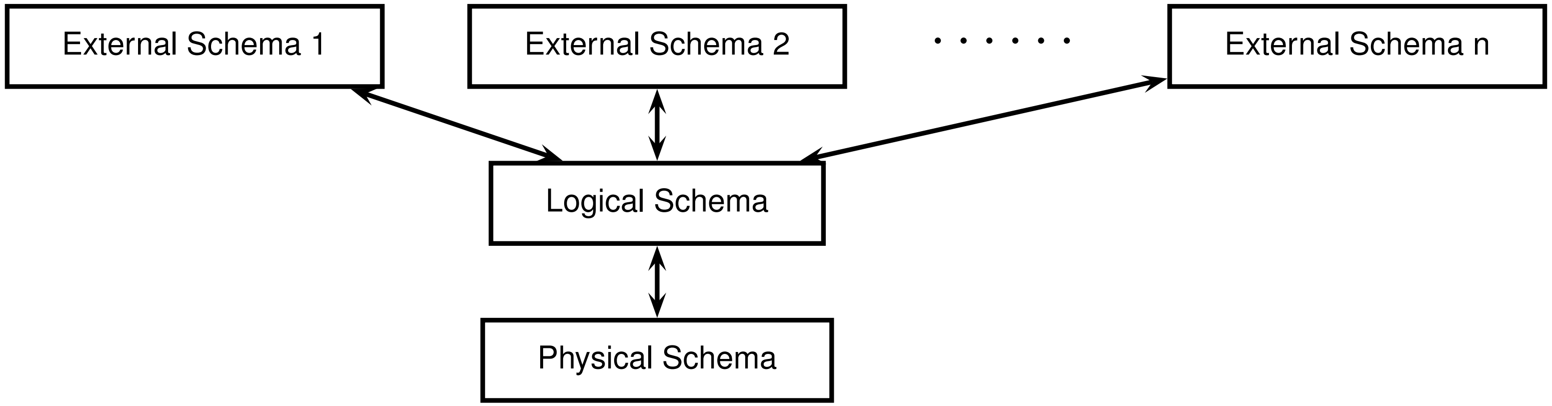
CourseInfo	(*cname*, *pname*, *lectureTime*, *totalEnrollment*)

- External schema for Bob:

StudentInfo	(*studentId*, *sname*)
CourseInfo	(*courseId*, *cname*, *credits*, *profId*, *lectureTime*)
EnrollInfo	(*sid*, *cid*, *cname*, *pname*, *lectureTime*)

Data Independence

- Insulate users/applications from changes in how data is structured and stored
- Data independence is achieved via the three levels of abstraction



- **Physical data independence** - protection from changes in physical schema
- **Logical data independence** - protection from changes in logical schema
- Data independence is an important advantage of using DBMS!

Transactions

- Abstraction for representing a logical unit of work
- **ACID Properties**
  - Atomicity: Either all the effects of a transaction are reflected in the database or none are
  - Consistency: The execution of a transaction in isolation preserves the consistency of the database
  - Isolation: The execution of a transaction is isolated from the effects of other concurrent transaction executions
  - Durability: The effects of a committed transaction persists in the database even in the presence of system failures

Transaction Example

Transfer(X, Y, amount)

```
fromBal := read balance from X's account
if fromBal ≥ amount then
        toBal := read balance from Y's account
        update Y's balance to toBal + amount
        update X's balance to fromBal - amount
end if
```

Serial Transaction Executions

Two possible serial executions of Transfer(1,2,100) & Transfer(2,1,100)

(1): fromBal := read 1's balance
toBal := read 2's balance
Update 2's balance to toBal + 100
Update 1's balance to fromBal - 100
fromBal := read 2's balance
toBal := read 1's balance
Update 1's balance to toBal + 100
Update 2's balance to fromBal - 100

(2): fromBal := read 2's balance
toBal := read 1's balance
Update 1's balance to toBal + 100
Update 2's balance to fromBal - 100
fromBal := read 1's balance
toBal := read 2's balance
Update 2's balance to toBal + 100
Update 1's balance to fromBal - 100

Concurrent Transaction Executions

A concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

fromBal := read 1's balance
toBal := read 2's balance
Update 2's balance to toBal + 100
fromBal := read 2's balance
toBal := read 1's balance
Update 1's balance to fromBal - 100
Update 1's balance to toBal + 100
Update 2's balance to fromBal - 100

Example 1

A concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

Initial Database State: (1, \$400), (2, \$2000)

Transfer(1,2,100)

fromBal:
toBal:

fromBal := read 1's balance
toBal := read 2's balance
Update 2's balance to toBal + 100

fromBal := read 2's balance
toBal := read 1's balance

Accounts

(1, \$400)
(2, \$2000)

Update 1's balance to fromBal - 100

Update 1's balance to toBal + 100
Update 2's balance to fromBal - 100

Transfer(2,1,100)

fromBal:
toBal:

Example 1

A concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

Initial Database State: (1, $400), (2, $2000)

Transfer(1,2,100)

fromBal: 400
toBal:

fromBal := read 1's balance
toBal := read 2's balance
Update 2's balance to toBal + 100

fromBal := read 2's balance
toBal := read 1's balance

Update 1's balance to fromBal - 100

Update 1's balance to toBal + 100
Update 2's balance to fromBal - 100

Accounts

(1, $400)
(2, $2000)

Transfer(2,1,100)

fromBal:
toBal:

Example 1

A concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

Initial Database State: (1, \$400), (2, \$2000)

fromBal := read 1's balance
toBal := read 2's balance
Update 2's balance to toBal + 100

fromBal := read 2's balance
toBal := read 1's balance

Update 1's balance to fromBal - 100

Update 1's balance to toBal + 100
Update 2's balance to fromBal - 100

Transfer(1,2,100)

fromBal: 400
toBal: 2000

Accounts

(1, \$400)
(2, \$2000)

Transfer(2,1,100)

fromBal:
toBal:

Example 1

A concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

Initial Database State: (1, $400), (2, $2000)

Transfer(1,2,100)

fromBal: 400
toBal: 2000

fromBal := read 1's balance
toBal := read 2's balance
Update 2's balance to toBal + 100

fromBal := read 2's balance
toBal := read 1's balance

Update 1's balance to fromBal - 100

Update 1's balance to toBal + 100
Update 2's balance to fromBal - 100

Accounts

(1, $400)
(2, $2100)

Transfer(2,1,100)

fromBal:
toBal:

Example 1

A concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

Initial Database State: (1, $400), (2, $2000)

Transfer(1,2,100)

fromBal: 400
toBal: 2000

fromBal := read 1's balance
toBal := read 2's balance
Update 2's balance to toBal + 100

fromBal := read 2's balance
toBal := read 1's balance

Update 1's balance to fromBal - 100

Update 1's balance to toBal + 100
Update 2's balance to fromBal - 100

Accounts

(1, $400)
(2, $2100)

Transfer(2,1,100)

fromBal: 2100
toBal:

Example 1

A concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

Initial Database State: (1, $400), (2, $2000)

Transfer(1,2,100)

fromBal: 400
toBal: 2000

fromBal := read 1's balance
toBal := read 2's balance
Update 2's balance to toBal + 100

fromBal := read 2's balance
toBal := read 1's balance

Update 1's balance to fromBal - 100

Update 1's balance to toBal + 100
Update 2's balance to fromBal - 100

Accounts

(1, $400)
(2, $2100)

Transfer(2,1,100)

fromBal: 2100
toBal: 400

Example 1

A concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

Initial Database State: (1, $400), (2, $2000)

Transfer(1,2,100)

fromBal: 400
toBal: 2000

fromBal := read 1's balance
toBal := read 2's balance
Update 2's balance to toBal + 100

fromBal := read 2's balance
toBal := read 1's balance

Update 1's balance to fromBal - 100

Update 1's balance to toBal + 100
Update 2's balance to fromBal - 100

Accounts

(1, $300)
(2, $2100)

Transfer(2,1,100)

fromBal: 2100
toBal: 400

Example 1

A concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

Initial Database State: (1, $400), (2, $2000)

Transfer(1,2,100)

fromBal: 400
toBal: 2000

fromBal := read 1's balance
toBal := read 2's balance
Update 2's balance to toBal + 100

fromBal := read 2's balance
toBal := read 1's balance

Update 1's balance to fromBal - 100

Update 1's balance to toBal + 100
Update 2's balance to fromBal - 100

Accounts

(1, $500)
(2, $2100)

Transfer(2,1,100)

fromBal: 2100
toBal: 400

Example 1

A concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

Initial Database State: (1, \$400), (2, \$2000)

Transfer(1,2,100)

fromBal: 400
toBal: 2000

fromBal := read 1's balance
toBal := read 2's balance
Update 2's balance to toBal + 100

fromBal := read 2's balance
toBal := read 1's balance

Update 1's balance to fromBal - 100

Update 1's balance to toBal + 100
Update 2's balance to fromBal + 100

Accounts

(1, \$500)
(2, \$2000)

Transfer(2,1,100)

fromBal: 2100
toBal: 400

Example 2

Another concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

Initial Database State: (1, $400), (2, $2000)

Transfer(1,2,100)

fromBal:	
toBal:	

fromBal := read 1's balance
toBal := read 2's balance
Update 2's balance to toBal + 100

fromBal := read 2's balance

Update 1's balance to fromBal - 100

toBal := read 1's balance
Update 1's balance to toBal + 100
Update 2's balance to fromBal - 100

Accounts

(1, $400)
(2, $2000)

Transfer(2,1,100)

fromBal:	
toBal:	

Example 2

Another concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

Initial Database State: (1, $400), (2, $2000)

fromBal := read 1's balance
toBal := read 2's balance
Update 2's balance to toBal + 100

fromBal := read 2's balance

Update 1's balance to fromBal - 100

toBal := read 1's balance
Update 1's balance to toBal + 100
Update 2's balance to fromBal - 100

Transfer(1,2,100)

fromBal: 400
toBal:

Accounts

(1, $400)
(2, $2000)

Transfer(2,1,100)

fromBal:
toBal:

Example 2

Another concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

Initial Database State: (1, $400), (2, $2000)

Transfer(1,2,100)

fromBal:	400
toBal:	2000

fromBal := read 1's balance
toBal := read 2's balance
Update 2's balance to toBal + 100

fromBal := read 2's balance

Update 1's balance to fromBal - 100

toBal := read 1's balance
Update 1's balance to toBal + 100
Update 2's balance to fromBal - 100

Accounts

(1, $400)
(2, $2000)

Transfer(2,1,100)

fromBal:	
toBal:	

Example 2

Another concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

Initial Database State: (1, $400), (2, $2000)

Transfer(1,2,100)

fromBal: 400
toBal: 2000

fromBal := read 1's balance
toBal := read 2's balance
Update 2's balance to toBal + 100

fromBal := read 2's balance

Update 1's balance to fromBal - 100

toBal := read 1's balance
Update 1's balance to toBal + 100
Update 2's balance to fromBal - 100

Accounts

(1, $400)
(2, $2100)

Transfer(2,1,100)

fromBal:
toBal:

Example 2

Another concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

Initial Database State: (1, \$400), (2, \$2000)

Transfer(1,2,100)

fromBal: 400	
toBal: 2000	

fromBal := read 1's balance
toBal := read 2's balance
Update 2's balance to toBal + 100

fromBal := read 2's balance

Update 1's balance to fromBal - 100

toBal := read 1's balance
Update 1's balance to toBal + 100
Update 2's balance to fromBal - 100

Accounts

(1, \$400)	
(2, \$2100)	

Transfer(2,1,100)

fromBal: 2100	
toBal:	

Example 2

Another concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

Initial Database State: (1, $400), (2, $2000)

Transfer(1,2,100)

fromBal: 400
toBal: 2000

fromBal := read 1's balance
toBal := read 2's balance
Update 2's balance to toBal + 100

fromBal := read 2's balance

Update 1's balance to fromBal - 100

toBal := read 1's balance
Update 1's balance to toBal + 100
Update 2's balance to fromBal - 100

Accounts

(1, $300)
(2, $2100)

Transfer(2,1,100)

fromBal: 2100
toBal:

Example 2

Another concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

Initial Database State: (1, \$400), (2, \$2000)

Transfer(1,2,100)

fromBal: 400	
toBal: 2000	

fromBal := read 1's balance
toBal := read 2's balance
Update 2's balance to toBal + 100

fromBal := read 2's balance

Update 1's balance to fromBal - 100

toBal := read 1's balance
Update 1's balance to toBal + 100
Update 2's balance to fromBal - 100

Accounts

(1, \$300)
(2, \$2100)

Transfer(2,1,100)

fromBal: 2100
toBal: 300

Example 2

Another concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

Initial Database State: (1, $400), (2, $2000)

Transfer(1,2,100)

fromBal: 400
toBal: 2000

fromBal := read 1’s balance
toBal := read 2’s balance
Update 2’s balance to toBal + 100

fromBal := read 2’s balance

Update 1’s balance to fromBal - 100

toBal := read 1’s balance
Update 1’s balance to toBal + 100
Update 2’s balance to fromBal - 100

Accounts

(1, $400)
(2, $2100)

Transfer(2,1,100)

fromBal: 2100
toBal: 300

Example 2

Another concurrent execution of Transfer(1,2,100) & Transfer(2,1,100)

Initial Database State: (1, $400), (2, $2000)

Transfer(1,2,100)

fromBal: 400
toBal: 2000

fromBal := read 1's balance
toBal := read 2's balance
Update 2's balance to toBal + 100

fromBal := read 2's balance

Update 1's balance to fromBal - 100

toBal := read 1's balance
Update 1's balance to toBal + 100
Update 2's balance to fromBal + 100

Accounts

(1, $400)
(2, $2000)

Transfer(2,1,100)

fromBal: 2100
toBal: 300

Serializable Transaction Executions

- A concurrent execution of a set of transactions is serializable if this execution is equivalent to some serial execution of the same set of transactions
- A concurrent execution is **equivalent** to a serial execution (of the same set of transactions) if
  - both executions produce the same final database state, and
  - every read operation in one execution returns the same value as the corresponding read operation in the other execution
- A serial execution is trivially serializable

Revisiting Examples

Serial Execution 1

fromBal := read 1's balance
toBal := read 2's balance
Update 2's balance to toBal + 100
Update 1's balance to fromBal - 100
fromBal := read 2's balance
toBal := read 1's balance
Update 1's balance to toBal + 100
Update 2's balance to fromBal - 100

Serial Execution 2

fromBal := read 2's balance
toBal := read 1's balance
Update 1's balance to toBal + 100
Update 2's balance to fromBal - 100
fromBal := read 1's balance
toBal := read 2's balance
Update 2's balance to toBal + 100
Update 1's balance to fromBal - 100

Concurrent Execution 1

fromBal := read 1's balance
toBal := read 2's balance
Update 2's balance to toBal + 100
fromBal := read 2's balance
toBal := read 1's balance
Update 1's balance to fromBal - 100
Update 1's balance to toBal + 100
Update 2's balance to fromBal - 100

Concurrent Execution 2

fromBal := read 1's balance
toBal := read 2's balance
Update 2's balance to toBal + 100
fromBal := read 2's balance
Update 1's balance to fromBal - 100
toBal := read 1's balance
Update 1's balance to toBal + 100
Update 2's balance to fromBal - 100

Summary

- DBMS used to store, update, and query data
- Relational data model
  - Tabular representation of data
  - Integrity constraints specify restrictions on data based on application semantics
- Levels of data abstraction provide data independence
- Transactions simply application development