Relational Databases &Transaction Processing
The Big Picture

CSE 305 – Principles of Database Systems

Paul Fodor

Stony Brook University

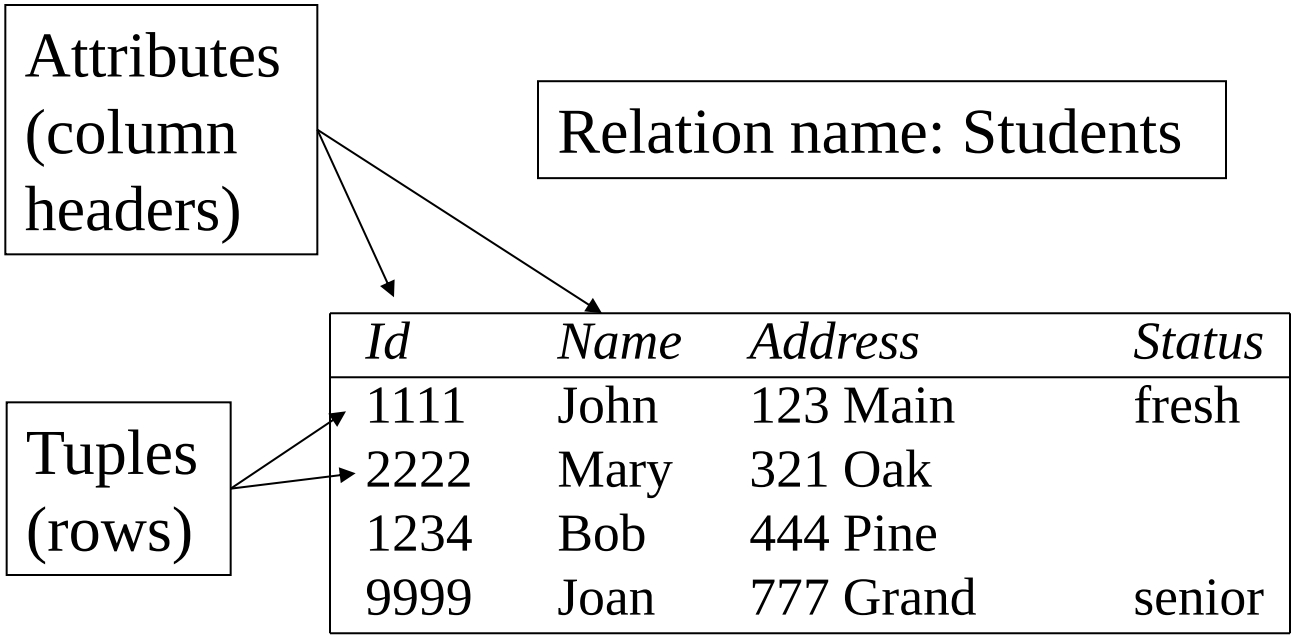
http://www.cs.stonybrook.edu/~cse305

What is a Data Model?

- What is a Data Model?
  - Mathematical representation of data
    - Examples:
      - relational model = tables;
      - semistructured model = trees/graphs.
  - Determines the operations on the data.
  - Determines the constraints on the language.

Relational Databases

- We are particularly interested in relational databases.
- Data is stored in tables.



Table

- Set of rows (**no duplicates**)
- Each *row* describes a different entity
- Each *column* states a particular fact about each entity
  - Each column has an associated *domain*

Id	*Name*	*Address*	*Status*
1111	John	123 Main	fresh
2222	Mary	321 Oak	
1234	Bob	444 Pine	
9999	Joan	777 Grand	senior

  - Domain of *Status* = {fresh, soph, junior, senior}

Relation

- Mathematical entity corresponding to a table
  - row ~ *tuple*
  - column ~ *attribute*
- Values in a tuple are *related* to each other
  - John lives at 123 Main
- Relation **R** can be thought of as predicate **R**
  - **R**(x,y,z) is true iff tuple (x,y,z) is in **R**

Relation

- Why Relations?
  - Very simple model.
  - Often matches how we think about data.
  - Abstract model that underlies SQL, the most important database language today.

Schemas

- *Relation schema* = relation name and attribute list.
  - Optionally: types of attributes.
  - Example: Students(id, name, address, status) or Students(id:int, name:string, address:string, status:string{fresh, soph, junior, senior})
    - Underline = *key* (tuples cannot have the same value in all key attributes). Excellent example of a constraint.
- *Database* = collection of relations.
- *Database schema* = set of all relation schemas in the database

Operations

- Also, operations on relations are precisely defined:
  - Take relation(s) as argument, produce new relation as result
  - Unary (*e.g.*, delete certain rows)
  - Binary (*e.g.*, union, Cartesian product)
- Corresponding operations defined on tables as well
- Using mathematical properties, *equivalence* can be decided
  - Important for *query optimization:*

$$op1(T1, op2(T2)) \stackrel{?}{=} op3(op4(T1), T2)$$

Structured Query Language: SQL

- Language for manipulating tables
- *Declarative* – Statement specifies *what* needs to be obtained, *not how* it is to be achieved (*e.g.*, how to access data, the order of operations)
- Due to declarativity of SQL, DBMS determines evaluation strategy
  - This greatly simplifies application programs
  - *But DBMS is not infallible*: programmers should have an idea of strategies used by DBMS, so they can design better tables, indices, statements, in such a way that DBMS can evaluate statements efficiently.

Structured Query Language: SQL

- Example:

  SELECT *<attribute list>*
  FROM *<table list >*
  WHERE *<condition>*

- Language for constructing a new table from argument table(s).
  - FROM indicates source tables
  - WHERE indicates which *rows* to retain
    - It acts as a filter
  - SELECT indicates which *columns/attributes* to extract from the retained rows
    - This is *Projection*
- The result is a table.

SQL Example

```
SELECT Name
FROM  Student
WHERE Id > 4999
```

Id	*Name*	*Address*	*Status*
1234	John	123 Main	fresh
5522	Mary	77 Pine	senior
9876	Bill	83 Oak	junior

Student

Name
Mary
Bill

Result

SQL Examples

```
SELECT Id, Name FROM Student

SELECT Id, Name FROM Student
    WHERE Status = 'senior'

SELECT * FROM Student
    WHERE Status = 'senior'

SELECT COUNT(*) AS CountSeniors FROM Student
    WHERE Status = 'senior'
```

the result is a table with one column and one row

More Complex Example

- **Goal:** extract a table in which each row names a senior student and gives a course taken and grade
  - It combines information from two tables:
    - Student: *Id*, *Name*, *Address*, *Status*
    - Transcript: *StudId*, *CrsCode*, *Semester*, *Grade*

```
SELECT Name, CrsCode, Grade
FROM  Student, Transcript
WHERE  Transcript.StudId = Student.Id
        AND Status = 'senior'
```

Join

SELECT *a1, b1*
FROM T1, T2
WHERE *a2 = b2*

T1

a1	*a2*	*a3*
A	1	xxy
B	17	rst

T2

b1	*b2*
3.2	17
4.8	17

FROM T1, T2
yields: Cartesian Product

a1	*a2*	*a3*	*b1*	*b2*
A	1	xxy	3.2	17
A	1	xxy	4.8	17
B	17	rst	3.2	17
B	17	rst	4.8	17

WHERE *a2 = b2 (filter)*
yields:

B	17	rst	3.2	17
B	17	rst	4.8	17

SELECT *a1, b1 (projection)*
yields result:

B	3.2
B	4.8

Modifying Tables

UPDATE *Student*
SET *Status* = ‘soph’
WHERE *Id* = 111111111

INSERT INTO Student (*Id*, *Name*, *Address*, *Status*)
VALUES (999999999, ‘Bill’, ‘432 Pine’, ‘senior’)

DELETE FROM Student
WHERE *Id* = 111111111

Creating Tables

- SQL is primarily a query language, for getting information from a database.
- But SQL also includes a *data-definition* component for describing database schemas.

```
CREATE TABLE Student (
   Id  INTEGER,
   Name CHAR(20),
   Address  CHAR(50),
   Status  CHAR(10),
   PRIMARY KEY (Id) )
```

Constraint

- To delete a relation:

```
DROP TABLE <name>;
```

Creating Tables

- Elements of Table Declarations
  - Most basic element: an attribute and its type.
  - The most common types are:
    - INT or INTEGER (synonyms).
    - REAL or FLOAT (synonyms).
    - CHAR(n) = fixed-length string of n characters.
    - VARCHAR(n) = variable-length string of up to n characters.
      - Strings require single quotes.
      - Two single quotes = real quote, e.g., 'Paul''s grade'.
    - Any value can be NULL.

Dates and Times

- DATE and TIME are types in SQL.
  - The form of a date value is:

    DATE 'yyyy-mm-dd'

    - Example: `DATE '2016-09-10'` for Sept. 10, 2016.
  - The form of a time value is:

    TIME 'hh:mm:ss'

  with an optional decimal point and fractions of a second following.

    - Example: `TIME '15:30:02.5'` = two and a half seconds after 3:30PM.

Transactions

- Many enterprises use databases to store information about their state
  - *E.g.*, balances of all depositors
- The occurrence of a real-world event that changes the enterprise state requires the execution of a program that changes the database state in a corresponding way
  - *E.g.*, balance must be updated when you withdraw
- A *transaction* is a program that accesses the database in response to real-world events

Transactions

- Many activities (transactions) at the database at all times.
- Must not confuse actions, e.g., two withdrawals from the same account must **each debit the account**.

Transactions

- Transactions are not just ordinary programs
- Additional requirements are placed on transactions (and particularly their execution environment) that go beyond the requirements placed on ordinary programs.
  - **A**tomicity
  - **C**onsistency
  - **I**solation
  - **D**urability

ACID properties

Atomicity

- The system must ensure that the transaction either runs to completion (i.e., *commits*) **or**, if it does not complete, has no effect at all (as if it had never been started) (i.e., *aborts*).
  - This is not true of ordinary programs. A hardware or software failure could leave files partially updated.
  - The TP monitor has the responsibility of ensuring that whatever partial changes the transaction has made to the database are undone (i.e., *rolled back*)

Integrity Constraints

- Or Consistency Constraints.
- Rules of the enterprise generally **limit the occurrence of certain real-world events**.
  - Student cannot register for a course if current number of registrants = maximum allowed
- Correspondingly, **allowable database states are restricted**.
  - All database states must have:

    current_registrations <= *maximum_registrations*
- These limitations are expressed as *integrity constraints*, which are **assertions** that **must be satisfied by the database state**.

Consistency

- A transaction must access and update the database in such a way that it preserves all database integrity constraints.
- The transaction designer must ensure that:

  IF the database is in a state that satisfies all integrity constraints when execution of a transaction is started

  THEN when the transaction completes:

  - All integrity constraints are once again satisfied (constraints can be violated in intermediate states)

Consistency

- Examples:
  - The database contains the Id of each student:
    - IC1: The Id of each student must be unique.
  - The database contains a list of prerequisites for each course and, for each student, a list of completed courses.
    - IC2: A student cannot register for a course without having taken all prerequisite courses.

Consistency

- The database contains the maximum number of students allowed to take each course and the number of students who are currently registered for each course.
  - IC3: The number of students registered for each course cannot be greater than the maximum number allowed for that course.

Consistency

- It might be possible to determine the number of students registered for a course from the database in two ways: the number is stored as a count in the course, and computed from the information describing each student by counting the number of student records that indicate that the student is registered for (or enrolled in) the course
  - IC4: the two determinations must yield the same result.

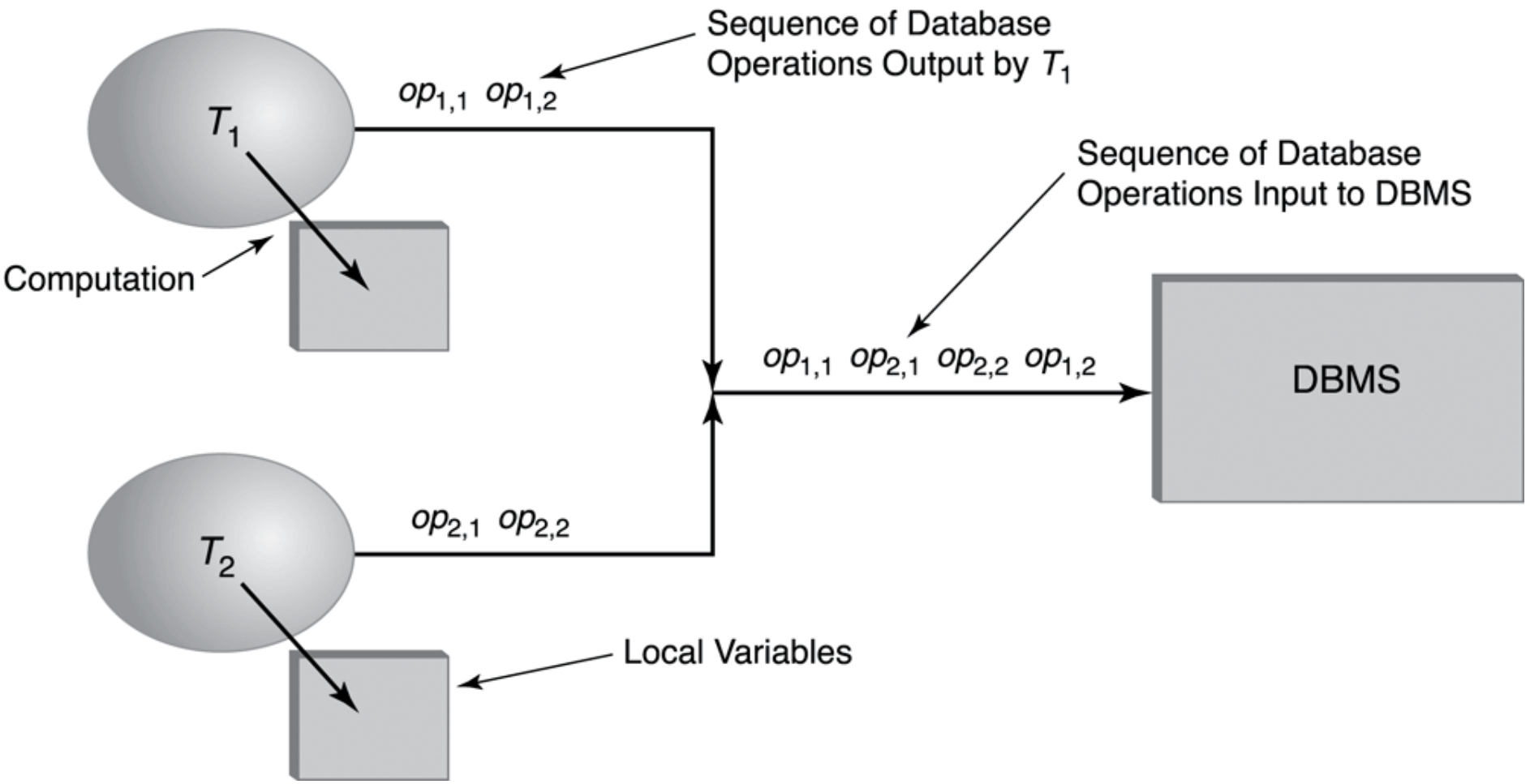
Isolation

- A set of transactions is executed *sequentially*, or *serially*, if one transaction in the set is executed to completion before another is started.
  - If all transactions are consistent and the database is initially in a consistent state, serial execution maintains consistency.
  - But serial execution is *inadequate* from a performance perspective

Isolation

- *Concurrent execution* is when multiple transactions are executed simultaneously.
  - different transactions are effectively interleaved in time

Concurrent Transaction Execution



Isolation

- Concurrent (interleaved) execution of a set of transactions offers performance benefits, but might not be correct.
- **Example**: Two students execute the course registration transaction at about the same time (*cur_reg* is the number of current registrants)

T_1: read(*cur_reg* : 29) write(*cur_reg* : 30)

T_2: read(*cur_reg* : 29) write(*cur_reg* : 30)

time →

Result: Database state no longer corresponds to real-world state, integrity constraint violated.

Lost update: one of the increments has been lost.

Isolation

- *Isolation* = Even though transactions are executed concurrently, the overall effect of the schedule must be the same as if the transactions had executed serially in some order.
- The effect of concurrently executing a set of transactions must be the same as if they had executed serially in some order
  - The execution is thus *not* serial, but *serializable*

Isolation

- Serializable execution has better performance than serial, but performance might still be inadequate.
  - Database systems offer several isolation levels with different performance characteristics (but some guarantee correctness only for certain kinds of transactions – not in general)

Durability

- The system must ensure that once the transaction commits, its effects remain in the database even if the computer, or the medium on which the database is stored, subsequently crashes.
  - Example: if a student successfully registers for a course, he/she expects the system to remember that he/she is registered even if the system later crashes.

ACID Properties

- The transaction monitor is responsible for ensuring atomicity, (the requested level of) isolation and durability.
  - Hence it provides the abstraction of failure-free, non-concurrent environment, greatly simplifying the task of the transaction designer.
- The transaction designer is responsible for ensuring the consistency of each transaction, but doesn't need to worry about concurrency (because of isolation) and system failures (because of atomicity and durability).

ACID Properties

- Atomicity = Each transaction is executed completely or not at all.
- Consistency = Each transaction maintains database consistency.
- Isolation = The concurrent execution of a set of transactions has the same effect as some serial execution of that set.
- Durability = The effects of committed transactions are permanently recorded in the database.