A Closer Look

Chapter 2

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	soph
1234	bob	444 pine	soph
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

Operations

- 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

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

Structured Query Language: SQL

- Language for manipulating tables
- Declarative - Statement specifies goal, not how it is to be achieved (*e.g.*, indices to use, order of operations); DBMS determines evaluation strategy
  - – Simplifies application program
- But programmers should have an idea of strategies used by DBMS so they can design tables, indices, statements, in such a way that DBMS can evaluate statements efficiently

Structured Query Language (SQL)

SELECT <attribute list>
FROM <table(s)>
WHERE <condition>

- Language for constructing a new table from argument table(s).
  - FROM indicates argument(s)
  - WHERE indicates which rows to retain
    - Selection (filter)
  - SELECT indicates which columns to extract from retained rows
    - Projection
- The result is a table.

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

Examples

SELECT Id, Name FROM Student

SELECT Id, Name FROM Student
WHERE Status = ‘senior’

SELECT * FROM Student
WHERE Status = ‘senior’

SELECT COUNT(*) FROM Student
WHERE Status = ‘senior’
(result is not a table in this case)

Example

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

SELECT Name, CrsCode, Grade
FROM Student, Transcript
WHERE StudId = 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:

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
yields:

B	17	rst	3.2	17
B	17	rst	4.8	17

SELECT a1, b1
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

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

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 deposit
- A transaction is a program that accesses the database in response to real-world events

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

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

Consistency

- Transaction designer must ensure that, assuming the database is in a state that satisfies all integrity constraints when execution of a transaction is started, when the transaction completes:
  - All integrity constraints are once again satisfied
    - Constraints might be violated in intermediate states
  - New database state satisfies specifications of transaction

Atomicity

- A real-world event either happens or does not happen.
  - Student either registers or does not register.
- Similarly, the system must ensure that either the transaction runs to completion (*commits*) or, if it does not complete, it has no effect at all (*aborts*).
  - Not true of ordinary programs. A hardware or software failure could leave files partially updated.

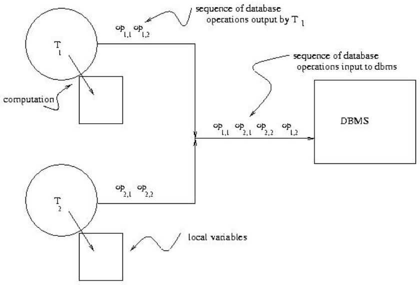
Durability

- The system must ensure that once a transaction commits its effect on the database state is not lost in spite of subsequent failures.
  - Not true of ordinary systems. For example, a media failure after a program terminates could cause the file system to be restored to a state that preceded the execution of the program.

Isolation

- Deals with the execution of multiple transactions concurrently.
- If the initial database state is consistent and accurately reflects the real-world state, then the *serial* execution of a set of consistent transactions preserves these properties.
- Hence, serial execution preserves consistency. But it is inadequate from a performance perspective.

Concurrent Transaction Execution


sequence of database operations output by T 1
op1,1 op1,2
T1
sequence of database operations input to dbms
computation
op1,1 op2,1 op2,2 op1,2
DBMS
op2,1 op2,2
T2
local variables

Isolation

- Concurrent (interleaved) execution of a set of transactions offers performance benefits, but might not be correct.
- Example: course registration (*cur_reg* is number of current registrants)

T_1: *r(cur_reg : 29)* *w(cur_reg : 30)*
T_2: *r(cur_reg : 29) w(cur_reg : 30)*
time →

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

Isolation

- The effect of concurrently executing a set of transactions must be the same as if they had executed serially (*serializable*) in some order.
- Serializable has better performance than serial, but performance might still be inadequate. Database systems offer several isolation levels with different performance characteristics -- but beware!

ACID Properties

- The transaction monitor is responsible for ensuring atomicity, durability, and (the requested level of) isolation.
  - 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.