A Closer Look

Underlying Concepts of Databases
and
Transaction Processing

1

Databases

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

2

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

Column

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

3

Table – Another Example

Course	*Max*	*Current*	*Room*
CS482	30	34	SH113
CS170	50	45	SH102
CS272	30	45	SH115

4

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**

5

Relation – Quick Reminder

- A *relation* R on the sets $S_1, S_2, \ldots, S_n$, denoted by $R \subseteq S_1 \times S_2 \times \ldots \times S_n$ is a set of tuples of the form $(p_1, p_2, \ldots, p_n)$ where p_i is a member of S_i for $i=1,\ldots,n$.
- $S_1 \times S_2 \times \ldots \times S_n$ denotes the Cartesian product of $S_1, S_2, \ldots, S_n$

6

Relation – Example

- For $S_1 = \{1,2,3,4\}$, $S_2 = \{a,b,c,d\}$, the following are examples of relations on S_1 and S_2
  - $R_1 = \{(1,a),(2,b),(3,c)\}$
  - $R_1 = \{\}$ (also written as $\emptyset$, called the empty set)
  - $R_3 = \{(1,a),(2,b),(3,c),(4,d)\}$

  If S_1 is a set of student IDs, S_2 is the set of grade, the relation R1, R2, and R3 represent the grade sheet in different semesters.

7

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:*

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

8

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

9

Structured Query Language (SQL)

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* to extract from retained rows
    - Projection
- The result is a table.

10

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

11

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

12

Examples

SELECT *Id*, *Name* FROM Student

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

SELECT * FROM Student
WHERE *Status* = 'senior'

result is a table with one column and one row

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

13

More Complex 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'*

14

Modifying Tables

SQL allows users to
• update
• insert
• delete
rows from tables

15

Modifying Tables

UPDATE *Student*
SET *Status* = 'soph'
WHERE *Id* = 1234

Column name

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

Student

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

Result

Change the value of column 'Status' of student 1234 to 'soph'

16

Modifying Tables

INSERT INTO Student (*Id*, *Name*, *Address*, *Status*)
VALUES (9999, 'Bill', '432 Pine', 'senior')

Id	*Name*	*Address*	*Status*
1234	John	123 Main	soph
5522	Mary	77 Pine	senior
9876	Bill	83 Oak	junior
9999	Bill	432 Pine	senior

New row

Result

17

Modifying Tables

DELETE FROM Student
WHERE *Id* = 9876

Id	*Name*	*Address*	*Status*
1234	John	123 Main	soph
5522	Mary	77 Pine	senior

Result

18

Creating Tables

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

Constraint: explained later

19

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

20

Transactions

- Transactions are <u>not just ordinary programs</u>
- 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

  (explained next)

21

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.

22

Consistency

- 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)
  - New database state satisfies specifications of transaction

23

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*).
  - This is not true of ordinary programs. A hardware or software failure could leave files partially updated.

24

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.

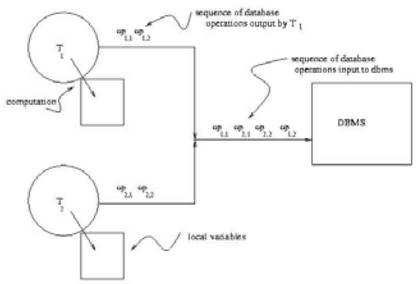
25

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* (one after another) execution of a set of consistent transactions preserves consistency.
- But serial execution is *inadequate* from a performance perspective.

26

Concurrent Transaction Execution



27

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.

28

Isolation

- 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*
- 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)

29

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, but doesn't need to worry about concurrency and system failures.

30