A Closer Look

Underlying Concepts of Databases and Transaction Processing

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

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Address</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>John</td>
<td>123 Main</td>
<td>fresh</td>
</tr>
<tr>
<td>2222</td>
<td>Mary</td>
<td>321 Oak</td>
<td>soph</td>
</tr>
<tr>
<td>1234</td>
<td>Bob</td>
<td>444 Pine</td>
<td>soph</td>
</tr>
<tr>
<td>9999</td>
<td>Joan</td>
<td>777 Grand</td>
<td>senior</td>
</tr>
</tbody>
</table>

- 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:
    \[ \text{op1(T1,op2(T2))} \equiv \text{op3(op2(T1),T2)} \]

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)

- **SELECT** <attribute list>
  - 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.

**Example**

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

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Address</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>John</td>
<td>123 Main</td>
<td>fresh</td>
</tr>
<tr>
<td>5522</td>
<td>Mary</td>
<td>77 Pine</td>
<td>senior</td>
</tr>
<tr>
<td>9876</td>
<td>Bill</td>
<td>83 Oak</td>
<td>junior</td>
</tr>
</tbody>
</table>

**Result**

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
</tr>
<tr>
<td>Bill</td>
</tr>
</tbody>
</table>

**Examples**

```sql
SELECT Id, Name FROM Student
```

- 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

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

**Join**

```sql
SELECT a1, b1
FROM T1, T2
WHERE a2 = b2
```

<table>
<thead>
<tr>
<th>a1</th>
<th>a2</th>
<th>a3</th>
<th>b1</th>
<th>b2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>xxy</td>
<td>3.2</td>
<td>17</td>
</tr>
<tr>
<td>B</td>
<td>17</td>
<td>rst</td>
<td>4.8</td>
<td>17</td>
</tr>
</tbody>
</table>

**Modifying Tables**

```sql
UPDATE Student
SET Status = 'soph'
WHERE Id = 111111111
```

```sql
INSERT INTO Student (Id, Name, Address, Status)
VALUES (999999999, 'Bill', '432 Pine', 'senior')
```

```sql
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.  
  – Atomicity  
  – Consistency  
  – Isolation  
  – Durability  
  (explained next)  

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  
  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

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

Concurrent Transaction Execution

- 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 (\(\text{cur\_reg}\) is the number of current registrants)
  - \(T_1: \text{read}(\text{cur\_reg}: 29)\) write(\(\text{cur\_reg}: 30\))
  - \(T_2: \text{read}(\text{cur\_reg}: 29)\) write(\(\text{cur\_reg}: 30\))

  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 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)

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.