A Closer Look
Underlying Concepts of Databases and Transaction Processing

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

<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>

Table – Another Example

<table>
<thead>
<tr>
<th>Course</th>
<th>Max</th>
<th>Current</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS482</td>
<td>30</td>
<td>34</td>
<td>SH113</td>
</tr>
<tr>
<td>CS170</td>
<td>50</td>
<td>45</td>
<td>SH102</td>
</tr>
<tr>
<td>CS272</td>
<td>30</td>
<td>45</td>
<td>SH115</td>
</tr>
</tbody>
</table>

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 – 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 $S_1, S_2, \ldots, S_n$.
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 \( R_1 \), \( R_2 \), and \( R_3 \) represent the grade sheet in different semesters.

Operations

- 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:
    \[ \text{op1}(T_1, \text{op2}(T_2)) = \text{op3}(\text{op2}(T_1), T_2) \]

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)

- Language for constructing a new table from argument table(s).
  - \text{FROM} indicates source tables
  - \text{WHERE} indicates which \textit{rows} to retain
    - It acts as a filter
  - \text{SELECT} indicates which \textit{columns} to extract from retained rows
  - Projection
  - The result is a table.

Example

\[
\begin{align*}
\text{SELECT} & \quad \text{Name} \quad \text{FROM} \quad \text{Student} \\
& \text{WHERE} \quad \text{Id} > 4999
\end{align*}
\]

\[
\begin{array}{|c|c|c|c|}
\hline
\text{Id} & \text{Name} & \text{Address} & \text{Status} \\
\hline
1234 & John & 123 Main & fresh \\
5522 & Mary & 77 Pine & senior \\
9876 & Bill & 83 Oak & junior \\
\hline
\end{array}
\]

\[
\begin{array}{|c|}
\hline
\text{Name} \\
\hline
Mary \\
Bill \\
\hline
\end{array}
\]

\[
\text{Student}
\]

Join

\[
\begin{align*}
\text{SELECT} \quad a_1, b_1 \\
\text{FROM} \quad T_1, T_2 \\
\text{WHERE} \quad a_2 = b_2
\end{align*}
\]

\[
\begin{array}{|c|c|c|}
\hline
a_1 & a_2 & a_3 \\
\hline
A & 1 & xxy \\
B & 17 & rst \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
b_1 & b_2 \\
\hline
3.2 & 17 \\
4.8 & 17 \\
\hline
\end{array}
\]

FROM \( T_1, T_2 \) yields:

\[
\begin{array}{|c|c|c|c|}
\hline
a_1 & a_2 & a_3 & b_1 \\
\hline
A & 1 & xxy & 3.2 \\
A & 1 & xxy & 4.8 \\
B & 17 & rst & 3.2 \\
B & 17 & rst & 4.8 \\
\hline
\end{array}
\]

WHERE \( a_2 = b_2 \) yields:

\[
\begin{array}{|c|c|c|}
\hline
b_1 & b_2 \\
\hline
3.2 & 17 \\
4.8 & 17 \\
\hline
\end{array}
\]

\[
\begin{align*}
\text{SELECT} & \quad a_1, b_1 \\
\text{FROM} \quad T_1, T_2 \\
\text{WHERE} \quad a_2 = b_2
\end{align*}
\]

\[
\begin{array}{|c|c|c|}
\hline
b_1 & b_2 \\
\hline
3.2 & 17 \\
4.8 & 17 \\
\hline
\end{array}
\]

\[
\text{Result}
\]
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 a table with one column and one row

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'
```

Modifying Tables

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

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

<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>soph</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>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>soph</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>

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

<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>soph</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>
<tr>
<td>9999</td>
<td>Bill</td>
<td>432 Pine</td>
<td>senior</td>
</tr>
</tbody>
</table>

New row

Result

```
DELETE FROM Student
WHERE Id = 9876
```

<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>soph</td>
</tr>
<tr>
<td>5522</td>
<td>Mary</td>
<td>77 Pine</td>
<td>senior</td>
</tr>
</tbody>
</table>

Result
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.

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.
  - $T_1$: read(cur_reg : 29) write(cur_reg : 30)
  - $T_2$: read(cur_reg : 29) write(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.