CSE 510
Web Data Engineering

Database Design

cse@buffalo
How to Design a Database and Avoid Bad Decisions

• With experience...
• Learn in CSE462 normalization rules of database design
• Think **entities and relationships** – translate to relations
E/R-Based Design

- **Entity**: Classes, Students
  - **Attributes** of Classes: Name, Number, DateCode, StartTime, EndTime
  - **Attributes** of Students: FirstName, LastName, UBID
  - **Relationship**: Enrollment
  - **Attribute of Enrollment**: Credits
• For every entity, create corresponding table
  – Include an ID attribute even if not in E/R
• For every relationship, create table
  – For each referenced entity $E_i$ include foreign key attribute referencing ID of $E_i$
Example

- **Movies**
  - Title
  - Year
  - Length

- **StarsIn**
  - Stars

- **Owns**

- **Stars**
  - Name
  - Address

- **Studios**
  - Name
  - Address
A studio has contracted with a particular star to act in a particular movie.
Relationships with Roles

Movies

Year
Title
Length

Original
Sequel
SequelOf

[Diagram showing relationships between movies and sequels]
“Subclassing”

- Title
- Year
- Length

Movies

- StarsIn
  - Name
  - Address

Cartoons

- Voices

IsA
Transaction Management

• **Transaction:** Collection of actions that maintain the consistency of the database if ran to completion & isolated

• **Goal:** Guarantee integrity and consistency of data despite
  – Concurrency
  – Failures

• Concurrency Control

• Recovery
Concurrency & Failure Problems

- Consider the “John & Mary” checking & savings account
  - C: checking account balance
  - S: savings account balance
- Check-to-Savings transfer transaction moves $X$ from C to S
  - If it runs in the system alone and to completion, the total sum of C and S stays the same

\[
\begin{align*}
\text{C2S (X=100)} \\
\text{Read(C)} \\
C := C - 100 \\
\text{Write(C)} \\
\text{Read(S)} \\
S := S + 100 \\
\text{Write(S)}
\end{align*}
\]
Failure Problem & Recovery Module’s Goal

- Database is in inconsistent state after machine restarts
- It is not the developer’s problem to account for crashes
- Recovery module guarantees that all or none of a transaction happens and its effects become “durable”

C2S(x=100)
Read(C)
C:=C-100
Write(C)

CPU Halts
Read(S)
S:=S+100
Write(S)
Concurrent Problem & Concurrency Control Module’s Goals

• If multiple transactions run in sequence, the resulting database is consistent

• Serial schedules
  – De facto correct

Serial Schedule:

Read(C)
C:=C+100
Write(C)
Read(S)
S:=S-100
Write(S)

Read(C)
C:=C+50
Write(C)
Read(S)
S:=S-50
Write(S)
Concurrency Problem & Concurrency Control Module’s Goals

- Databases allow transactions to run in parallel

Good Schedule with Concurrency
Read(C)
C:=C+100
Write(C)

Read(S)
S:=S-100
Write(S)

Read(C)
C:=C+50
Write(C)

Read(S)
S:=S-50
Write(S)
Concurrency Problem & Concurrency Control Module’s Goals

Bad Schedule with Concurrency
Read(C)
C:=C+100
Write(C)
C:=C+50
Write(C)
Read(S)
S:=S-50
Write(S)
Read(S)
S:=S-100
Write(S)

• “Bad” interleaved schedules may leave database in inconsistent state
• Developer should not have to account for parallelism
• Concurrency control module guarantees serializability
  – only schedules equivalent to serial ones happen