ICS 321 Fall 2010

The Database Language SQL (iv)

Asst. Prof. Lipyeow Lim

Information & Computer Science Department

University of Hawaii at Manoa
Insertion

**INSERT INTO** R(A1, A2, ...)  
**VALUES** (v1, v2, ...);

**INSERT INTO** Studio(name)  
**SELECT DISTINCT** studioname  
**FROM** Movies  
**WHERE** studioname **NOT IN**  
**(SELECT name**  
**FROM** Studio);

- If inserting results from a query, query must be evaluated prior to actual insertion
Deletion

- Deletion specified using a where clause.
- To delete a specific tuple, you need to use the primary key or candidate keys.

```
DELETE FROM R
WHERE <condition>;
```

```
DELETE FROM StarsIn
WHERE movieTitle = 'The Maltese Falcon' AND MovieYear = 1942 AND starName='Sydney Greenstreet';
```
Updates

UPDATE R
SET <new value assignments>
WHERE <condition>;

UPDATE MovieExec
SET name='Pres. ' || name
WHERE cert# IN ( 
   SELECT presC#
   FROM Studio );

• Tuples to be updated are specified using a where clause.
• To update a specific tuple, you need to use the primary key or candidate keys.
Airline Reservation Example

Flights ( fltNo , fltDate , seatNo , seatStatus )

To view available seats:

SELECT seatNo
FROM Flights
WHERE fltNo = 123 AND fltDate = DATE '2008-12-25'
   AND seatStatus = ' available ' ;

To reserve a particular seat:

UPDATE Flights
SET seatStatus = 'occupied'
WHERE fltNo = 123 AND fltDate = DATE '2008-12-25'
   AND seatNo = '22A';
Transactions

• A transaction is the DBMS’s abstract view of a user program: a sequence of reads and writes.
  – Eg. User 1 views available seats and reserves seat 22A.

• A DBMS supports multiple users, ie, multiple transactions may be running concurrently.
  – Eg. User 2 views available seats and reserves seat 22A.
  – Eg. User 3 views available seats and reserves seat 23D.
Concurrent Execution

- DBMS tries to execute transactions concurrently – why?

<table>
<thead>
<tr>
<th>Schedule 1</th>
<th>Schedule 2</th>
<th>Schedule 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>U1</td>
<td>U2</td>
<td>U1</td>
</tr>
<tr>
<td>Finds 22A empty</td>
<td>Finds 22A empty</td>
<td>Finds 22A empty</td>
</tr>
<tr>
<td>Reserves 22A</td>
<td>Reserves 22A</td>
<td>Reserves 22A</td>
</tr>
</tbody>
</table>

10/25/2010

Lipyew Lim -- University of Hawaii at Manoa
ACID Properties

4 important properties of transactions

• **Atomicity**: all or nothing
  – Users regard execution of a transaction as atomic
  – No worries about incomplete transactions

• **Consistency**: a transaction must leave the database in a good state
  – Semantics of consistency is application dependent
  – The user assumes responsibility

• **Isolation**: a transaction is isolated from the effects of other concurrent transaction

• **Durability**: Effects of completed transactions persists even if system crashes before all changes are written out to disk
Atomicity

• A transaction might *commit* after completing all its actions, or it could *abort* (or be aborted by the DBMS) after executing some actions.

• A very important property guaranteed by the DBMS for all transactions is that they are *atomic*. That is, a user can think of a Xact as always executing all its actions in one step, or not executing any actions at all.
  
  – DBMS logs all actions so that it can *undo* the actions of aborted transactions.
Example (Atomicity)

- The first transaction is transferring $100 from B’s account to A’s account.
- The second is crediting both accounts with a 6% interest payment.
- There is no guarantee that T1 will execute before T2 or vice-versa, if both are submitted together. However, the net effect must be equivalent to these two transactions running serially in some order.

T1: BEGIN
    A = A + 100
    B = B - 100
    END

T2: BEGIN
    A = 1.06 * A
    B = 1.06 * B
    END
T1: BEGIN
A=A+100
B=B-100
END

T1: BEGIN
Read A from disk
A=A+100
Write A to disk

Read B from disk
B=B-100
Write B to disk
END

T1: BEGIN
R(A)
W(A)
R(B)
W(B)
END
### Serial Executions

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>A = A + 100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B = B - 100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A = 1.06 * A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B = 1.06 * B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A = 100, B = 200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A = 200, B = 100</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
A = 100, B = 200
A = 200, B = 200
A = 200, B = 100
A = 212, B = 100
A = 212, B = 106
```
## Example (Serializability)

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>Equivalent States</th>
</tr>
</thead>
<tbody>
<tr>
<td>A = A + 100</td>
<td>A = A + 100</td>
<td>A = 100, B = 200</td>
</tr>
<tr>
<td>B = B - 100</td>
<td>A = 1.06 * A</td>
<td>A = 200, B = 200</td>
</tr>
<tr>
<td></td>
<td>B = 1.06 * B</td>
<td>A = 212, B = 200</td>
</tr>
<tr>
<td>A = A + 100</td>
<td>A = 1.06 * A</td>
<td>A = 212, B = 106</td>
</tr>
<tr>
<td></td>
<td>B = 1.06 * B</td>
<td></td>
</tr>
</tbody>
</table>

**Equivalent States:**
- A = 100, B = 200
- A = 200, B = 200
- A = 212, B = 200
- A = 212, B = 100
- A = 212, B = 106
- A = 212, B = 112
Scheduling Transactions

• **Serial schedule:** Schedule that does not interleave the actions of different transactions.

• **Equivalent schedules:** For any database state, the effect (on the set of objects in the database) of executing the first schedule is identical to the effect of executing the second schedule.

• **Serializable schedule:** A schedule that is equivalent to some serial execution of the transactions.

(Note: If each transaction preserves consistency, every serializable schedule preserves consistency.)