Transactions: Serializable and Other Isolation Levels
Three Phenomenon

- **Dirty Read: \( w_1(x), r_2(x), a_1 \)**
  - Transaction T1 writes, T2 reads it, then T1 aborts; T2 has a dirty read.
  - T1 reserves a seat for trip plan.
    Rest of the trip is not ok; T1 aborts.
    T2 reads during the time else wait too long.
    T2 has a dirty read.

- **Non-repeatable read (Fuzzy read):**
  \( r_1[x], w_2[x], c_2, r_1[x] \)
  - Transaction T1 reads, T2 writes and commits, then T1 reads again. T1 has a unrepeatable read.
  - T1 checks seat availability on several flights, decides on a particular flight, read it again and seat is gone.
    Same may occur for on-line enrollment.

- **Phantom read: \( r_1[P] \ldots w_2[y \text{ in } P] \ldots (c_1 \text{ or } a_1) \)**
  \( w_2: \) ANSI standard specifies only insert but write makes it general.
P: set of rows satisfying a where-clause

Transaction T1 reads a set of rows from a table (satisfying a where-clause), T2 inserts a row which satisfies the where-clause, if T1 repeats, it will see a phantom row, a row that was not there before.
Four ANSI SQL Isolation Levels

- ANSI SQL defines four isolation levels in terms of the three phenomenon:

<table>
<thead>
<tr>
<th>Isolation level</th>
<th>Dirty Read</th>
<th>Non-repeatable Read</th>
<th>Phantom Read</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Uncommitted</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Read Committed</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Repeatable Read</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Serializable</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

- Less throughput with increasing isolation level.

- More consistency with increasing isolation level.
Implementation of Isolation Levels by Locks

• Definitions:

  – Well-formed reads (WFR): reads preceded by shared locks.
  – Well-formed writes (WFW): writes preceded by exclusive locks.
  – Long duration locks (LDR, LDW): locks held until after commit or abort.
  – Short duration locks (SDR, SDW, SDPR): locks not long duration locks.
    (locks released immediately after the operations)
  – Predicate lock (PR): lock on all data items satisfying the where clause.
• **Implementation:**

<table>
<thead>
<tr>
<th>ISOLATION LEVEL</th>
<th>READ LOCKS</th>
<th>WRITE LOCKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Uncommitted</td>
<td>none required</td>
<td>WFW, LDW</td>
</tr>
<tr>
<td>Read Committed</td>
<td>WFR, SDR</td>
<td>WFW, LDW</td>
</tr>
<tr>
<td></td>
<td>SDPR</td>
<td></td>
</tr>
<tr>
<td>Repeatable Read</td>
<td>WFR, LDR</td>
<td>WFW, LDW</td>
</tr>
<tr>
<td></td>
<td>SDPR</td>
<td></td>
</tr>
<tr>
<td>Serializable</td>
<td>WFR, LDR</td>
<td>WFW, LDW</td>
</tr>
<tr>
<td></td>
<td>LDPR</td>
<td></td>
</tr>
</tbody>
</table>

• Why always LDW? If we allow SDW (Short Duration Write) it may produce dirty write (i.e., transactions write may be lost) which is not acceptable.

w1[x]...w2[x] ... (c1 or a1) and (c2 or a2) in any order

Here transaction 1 writes then unlocks (short duration write lock), then transaction 2 writes and commits, then transaction 1 commits. Transaction 1’s write is lost.
• Why SDR and SDPR for READ COMMITTED? This prevents dirty read. This guarantees there is no write lock on the data item that is being read. If we read without requiring a lock, there may be a write lock existing on the data item (i.e., some one is writing on the data item) another transaction reads it because no read lock is required for read, transaction which did the write aborts.

• We do not require predicate lock for READ COMMITTED because there is no conflict with record that has been inserted yet. As a result there is no dirty read.

• Why LDR for REPEATABLE READ? If a transaction uses SDR, that is, after reading as soon as the read lock is released, another transaction can change the data using write lock and then write and commit, the first transaction then can read the same data item again creating non-repeatable read.

• ANSI ISOLATION level (which is implementation independent) is not exactly same as LOCK-based isolation level as given above but they are close.

• Why SDPR for REPEATABLE READ? Prevents dirty read however because of of short duration pred-
icate lock, it does not guarantee repeatability for a
group of records (predicate based), thereby creat-
ing phantom read. Note that phenomenon ”Non-
repeatable read” does not apply to predicate and re-
peatability for predicate is covered by the phenomenon
”phantom read.”
EXAMPLE:

What are the ANSI SQL isolation levels for the following two transaction-schedules:

1. S1: $R_1(A)W_1(A)R_2(A)R_1(B)C_1C_2$
   It has a dirty read. It is read uncommitted

2. S2: $R_1(A)R_2(A)R_2(B)W_2(B)C_1C_2$
   Serializable.
Oracle Isolation Levels

- Three Levels:
  - Transaction read committed
  - Transaction serializable
  - Read only (transactions with only reads, not a standard)
- Default is read committed.
- Two modes:
  - Transaction Level
  - Session Level
  - Can run both levels concurrently.
Implementation

- Read Committed:
  
  - Long duration locks: Locks held until commit or rollback.
  
  - Note there is no read locks.

  Statement level read consistency:
  
  * Single point in time: All data returned by an sql statement comes from a single point in time, time the query begins (instead of sql retrieving tuples over a period).
  
  * Only data, committed before this time is visible to the query.
  
  * Note other transactions may be writing on the same data during this time.

- Serializable:

  - Long duration locks: Locks held until commit or rollback.

  - Note there is no read locks.

  Transaction level read consistency:

  * Single point in time: All data returned by sql statements within a transaction comes from a
single point in time, time the transaction begins begins (instead of SQLs retrieving tuples over a period).

* Only data, committed before this time is visible to the query.
* Note other transactions may be writing on the same data during this time.

- For Read:
  - Read Committed transaction: sees data written by transactions committed before the statement started.
  - No dirty read.
  - Serializable transaction: Sees data written by transactions committed before the transaction started.
  - No repeatable reads.

- Oracle generates an error when a serializable transaction tries to update or delete data modified by a transaction that commits after the serializable transaction begins.
• For Write (DML):
  Row level locks are exclusive/ table level locks are shared:

  Locks are kept until commit.

• Read committed transaction:

  Transaction waits on conflicting lock and acquires it when released.

• Serializable:

  If other transactions try to get exclusive lock after this transaction has a lock, later transaction aborts (note the later transaction might have read it already with no read locks).
Example:
schedule: \( r_1[x], w_1[x], r_2[x], c_1, c_2 \)
Schedule is not allowed in oracle because of statement level read consistency. T2 waits until t1 aborts or commits.
Correct sequence: \( r_1[x], w_1[x], c_1, r_2[x], c_2 \)
schedule: \( r_1[x], r_2[x], w_1[x], w_2[x], r_1[x], c_1, c_2 \)
(Note read does not need a lock in oracle.)
T1 has write lock. Assume T2 is RC
T2 waits until t1 commits or aborts, t2 gets write lock.
Correct sequence: \( r_1[x], r_2[x], w_1[x], c_1, w_2[x], c_2 \)
T2 serializable: t1 has write lock.
\( r_1[x], r_2[x], w_1[x], w_2[x] \)
t2 fails because t1 has committed a change since the t2 began.

Implemented by
- Multiversion concurrency control
  Maintains time stamp and rollback segment
- automatic locks:
  row locks and table locks.
Choosing an Isolation level

• Based on application performance needs and consistency needs.

• Oracle’s Read committed transactions:
  – Many high performance environments with high transaction arrival rates and can tolerate repeatable and phantom reads
  – Very low transaction arrival rate (low risk of incorrect results)
  – Few applications require reissue of the same query twice resulting in unrepeateable read and phantom read which are ok.

• Oracle’s serializable transactions:
  – Where transaction executes a query
more than once and unrepeatable read or phantom reads are not desired..

– Low chance that two concurrent transactions will modify the same rows.

– Relatively long-running transactions are primarily read-only.

– Suitable for large databases with short transactions that update only a few rows.